МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

ИСО

МЕЖДУНАРОДНАЯ
ОРГАНИЗАЦИЯ
ПО СТАНДАРТИЗАЦИИ

МЭК

МЕЖДУНАРОДНАЯ
ЭЛЕКТРОТЕХНИЧЕСКАЯ
КОМИССИЯ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.6

ЯЗЫК МККТТ ВЫСОКОГО УРОВНЯ (CHILL)

РЕКОМЕНДАЦИЯ Z.200

МЕЖДУНАРОДНЫЙ СТАНДАРТ
ISO/IEC 9496

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

Справочный номер
ISO/IEC 9496: 1989 (E)

МЕЖДУНАРОДНЫЙ СОЮЗ ЭЛЕКТРОСВЯЗИ

МККТТ

МЕЖДУНАРОДНЫЙ
КОНСУЛЬТАТИВНЫЙ КОМИТЕТ
ПО ТЕЛЕГРАФИИ И ТЕЛЕФОНИИ

ИСО

МЕЖДУНАРОДНАЯ
ОРГАНИЗАЦИЯ
ПО СТАНДАРТИЗАЦИИ

МЭК

МЕЖДУНАРОДНАЯ
ЭЛЕКТРОТЕХНИЧЕСКАЯ
КОМИССИЯ

СИНЯЯ КНИГА

ТОМ X — ВЫПУСК X.6

ЯЗЫК МККТТ ВЫСОКОГО УРОВНЯ (CHILL)

РЕКОМЕНДАЦИЯ Z.200

МЕЖДУНАРОДНЫЙ СТАНДАРТ
ISO/IEC 9496

IX ПЛЕНАРНАЯ АССАМБЛЕЯ
МЕЛЬБУРН, 14–25 НОЯБРЯ 1988 ГОДА

ISBN 92-61-03804-2

Справочный номер
ISO/IEC 9496: 1989 (E)

Printed in Russia

СОДЕРЖАНИЕ КНИГИ МККТТ, ДЕЙСТВУЮЩЕЙ ПОСЛЕ IX ПЛЕНАРНОЙ АССАМБЛЕИ (1988 г.)

СИНЯЯ КНИГА

ВЫПУСК III.6 — Передача по линии нетелефонных сигналов. Передача сигналов звукового и телевизионного вещания. Рекомендации серий H и J (Исследовательская комиссия XV).

ВЫПУСК III.7 — Цифровая сеть интегрального обслуживания (ЦСИО) — Общая структура, услуги и возможности обслуживания. Рекомендации I.110—I.257 (Исследовательская комиссия XVIII).

ВЫПУСК III.8 — Цифровая сеть интегрального обслуживания (ЦСИО) — Общесетевые аспекты и функции, интерфейсы "пользователь — сеть" ЦСИО. Рекомендации I.310—I.470 (Исследовательская комиссия XVIII).

ВЫПУСК III.9 — Цифровая сеть с интеграцией служб (ЦСИС) — Межсетевые стыки и принципы технической эксплуатации. Рекомендации I.500—I.605 (Исследовательская комиссия XVIII).

Том IV

ВЫПУСК IV.1 — Общие принципы технической эксплуатации: техническая эксплуатация международных систем передачи и международных телефонных каналов. Рекомендации M.10—M.782 (Исследовательская комиссия IV).

ВЫПУСК IV.2 — Техническая эксплуатация международных телеграфных, фототелеграфных и арендованных каналов. Техническая эксплуатация международной телефонной сети общего пользования. Техническая эксплуатация морских спутниковых систем и систем передачи данных. Рекомендации M.800—M.1375 (Исследовательская комиссия IV).

ВЫПУСК IV.3 — Техническая эксплуатация международных каналов звукового и телевизионного вещания. Рекомендации серии N (Исследовательская комиссия IV).

ВЫПУСК IV.4 — Требования к измерительной аппаратуре. Рекомендации серии O (Исследовательская комиссия IV).

Том V — Качество телефонной передачи. Рекомендации серии P (Исследовательская комиссия XII).

Том VI

ВЫПУСК VI.1 — Общие Рекомендации по телефонной коммутации и сигнализации. Функции и информационные потоки для служб в ЦСИС. Дополнения. Рекомендации Q.1—Q.118 *bis* (Исследовательская комиссия XI).

ВЫПУСК VI.2 — Требования к системам сигнализации № 4 и № 5. Рекомендации Q.120—Q.180 (Исследовательская комиссия XI).

ВЫПУСК VI.3 — Требования к системе сигнализации № 6. Рекомендации Q.251—Q.300 (Исследовательская комиссия XI).

ВЫПУСК VI.4 — Требования к системам сигнализации R1 и R2. Рекомендации Q.310—Q.490 (Исследовательская комиссия XI).

ВЫПУСК VI.5 — Цифровые местные, транзитные, комбинированные и международные станции в интегральных цифровых сетях и смешанных аналого-цифровых сетях. Дополнения. Рекомендации Q.500—Q.554 (Исследовательская комиссия XI).

ВЫПУСК VI.6 — Взаимодействие систем сигнализации. Рекомендации Q.601—Q.699 (Исследовательская комиссия XI).

ВЫПУСК VI.7 — Требования к системе сигнализации № 7. Рекомендации Q.700—Q.716 (Исследовательская комиссия XI).

ВЫПУСК VI.8 — Требования к системе сигнализации № 7. Рекомендации Q.721—Q.766 (Исследовательская комиссия XI).

ВЫПУСК VI.9 — Требования к системе сигнализации № 7. Рекомендации Q.771—Q.795 (Исследовательская комиссия XI).

ВЫПУСК VI.10 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), уровень звена данных. Рекомендации Q.920 и Q.921 (Исследовательская комиссия XI).

ВЫПУСК VI.11 — Цифровая абонентская система сигнализации № 1 (ЦАС 1), сетевой уровень, управление "пользователь—сеть". Рекомендации Q.930—Q.940 (Исследовательская комиссия XI).

ВЫПУСК VI.12 — Сухопутная подвижная сеть общего пользования. Взаимодействие с ЦСИС и коммутируемой телефонной сетью общего пользования. Рекомендации Q.1000—Q.1032 (Исследовательская комиссия XI).

ВЫПУСК VI.13 — Сухопутная подвижная сеть общего пользования. Подсистема подвижного применения и стыки. Рекомендации Q.1051—Q.1063 (Исследовательская комиссия XI).

ВЫПУСК VI.14 — Взаимодействие с системами подвижной спутниковой связи. Рекомендации Q.1100—Q.1152 (Исследовательская комиссия XI).

Том VII

ВЫПУСК VII.1 — Телеграфная передача. Рекомендации серии R. Оконечное оборудование телеграфных служб. Рекомендации серии S (Исследовательская комиссия IX).

ВЫПУСК VII.2 — Телеграфная коммутация. Рекомендации серии U (Исследовательская комиссия IX).

ВЫПУСК VII.3 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.0—T.63 (Исследовательская комиссия VIII).

ВЫПУСК VII.4 — Процедуры аттестационных испытаний для Рекомендаций по телетексу. Рекомендация T.64 (Исследовательская комиссия VIII).

ВЫПУСК VII.5 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T. 65—T.101, T.150—T.390 (Исследовательская комиссия VIII).

ВЫПУСК VII.6 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.400—T.418 (Исследовательская комиссия VIII).

ВЫПУСК VII.7 — Оконечное оборудование и протоколы для телематических служб. Рекомендации T.431—T.564 (Исследовательская комиссия VIII).

Том VIII

ВЫПУСК VIII.1 — Передача данных по телефонной сети. Рекомендации серии V (Исследовательская комиссия XVII).

ВЫПУСК VIII.2 — Сети передачи данных: службы и услуги, стыки. Рекомендации X.1—X.32 (Исследовательская комиссия VII).

ВЫПУСК VIII.3 — Сети передачи данных: передача, сигнализация и коммутация, сетевые аспекты, техническая эксплуатация и административные предписания. Рекомендации X.40—X.181 (Исследовательская комиссия VII).

ВЫПУСК VIII.4 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Модель и система обозначений, определение служб. Рекомендации X.200—X.219 (Исследовательская комиссия VII).

ВЫПУСК VIII.5 — Сети передачи данных: взаимосвязь открытых систем (ВОС) — Требования к протоколам, аттестационные испытания. Рекомендации X.220—X.290 (Исследовательская комиссия VII).

ВЫПУСК VIII.6 — Сети передачи данных: взаимодействие между сетями, подвижные системы передачи данных, межсетевое управление. Рекомендации X.300—X.370 (Исследовательская комиссия VII).

ВЫПУСК VIII.7 — Сети передачи данных: системы обработки сообщений. Рекомендации X.400—X.420 (Исследовательская комиссия VII).

ВЫПУСК VIII.8 — Сети передачи данных: справочная служба. Рекомендации X.500—X.521 (Исследовательская комиссия VII).

Том IX — Защита от помех. Рекомендации серии K (Исследовательская комиссия V). Конструкция, прокладка и защита кабелей и других элементов линейного оборудования. Рекомендации серии L (Исследовательская комиссия VI).

Том X

ВЫПУСК X.1 — Язык функциональной спецификации и описания. Критерии применения методов формальных описаний. Рекомендация Z.100 с Приложениями A, B, C и E, Рекомендация Z.110 (Исследовательская комиссия X).

ВЫПУСК X.2 — Приложение D к Рекомендации Z.100: Руководство для пользователей языка SDL (Исследовательская комиссия X).

ВЫПУСК X.3 — Приложение F.1 к Рекомендации Z.100: Формальное определение языка SDL. Введение (Исследовательская комиссия X).

ВЫПУСК X.4 — Приложение F.2 к Рекомендации Z.100: Формальное определение языка SDL. Статическая семантика (Исследовательская комиссия X).

ВЫПУСК X.5 — Приложение F.3 к Рекомендации Z.100: Формальное определение языка SDL. Динамическая семантика (Исследовательская комиссия X).

ВЫПУСК X.6 — Язык МККТТ высокого уровня (CHILL). Рекомендация Z.200 (Исследовательская комиссия X).

ВЫПУСК X.7 — Язык "человек—машина" (MML). Рекомендации Z.301—Z.341 (Исследовательская комиссия X).

ПРИМЕЧАНИЕ

Рекомендация Z.200 была подготовлена Международным консультативным комитетом по телеграфии и телефонии (МККТТ) Международного союза электросвязи (МСЭ) и принята в соответствии со специальной "ускоренной процедурой" в качестве Международного стандарта ИСО/МЭК 9496 ИСО (Международной организацией по стандартизации) и МЭК (Международной электротехнической комиссией) через посредство их Объединенного технического комитета (ISO/IEC JTC 1, *Information technology*).

Текст Рекомендации Z.200 МККТТ является стандартом ИСО/МЭК 9496.

ЯЗЫК ВЫСОКОГО УРОВНЯ МККТТ (CHILL)

(Женева, 1988 г.)

СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ

В настоящей Рекомендации описан язык программирования высокого уровня МККТТ, называемый CHILL; это — сокращение от CCITT High Level Language (Язык Высокого Уровня МККТТ).

В последующих подразделах данной главы представлены некоторые основания для разработки указанного языка и приведены его характеристики.

Относительно информации, относящейся к вводному материалу и материалу по обучению, связанной с языком CHILL, читателю следует обращаться к руководствам МККТТ: "Введение в CHILL" и "Руководство пользователя CHILL".

В Руководстве МККТТ "Формальное описание CHILL" приводится описание языка CHILL в точной математической форме (основанное на представлении VDM).

1.1 ОБЩИЕ ПОЛОЖЕНИЯ

Язык CHILL — это язык со строгим контролем типов, с блочной структурой, разработанный для реализации больших и сложных вложенных систем.

Язык CHILL:

- увеличивает надежность и динамическую эффективность программ посредством обширных статических проверок программ;
- обеспечивает достаточную гибкость и мощность при разработке прикладных программ необходимого диапазона применения, а также при эксплуатации различного оборудования;
- обеспечивает средства модульной разработки больших систем;
- поддерживает реализацию программ реального масштаба времени посредством встроенных примитивов параллельного выполнения и контроля времени;
- позволяет генерировать высокоэффективные объектные программы;
- является легким для изучения и использования.

Средства выражения, имеющиеся в языке, позволяют инженерам выбрать соответствующие конструкции из широкого набора средств, так чтобы реализация в результате могла бы наиболее точно соответствовать исходной спецификации.

Поскольку в языке CHILL делается четкое различие между статическими и динамическими объектами, то почти весь семантический контроль может быть выполнен во время компиляции. Это имеет очевидные выгоды при выполнении программ. Нарушение динамических правил языка CHILL приводит к динамическим исключениям, которые обрабатываются соответствующей программой обработки исключений (однако, генерация таких неявных проверок является необязательной, до пор тех пока программа обработки исключений явно задана пользователем).

Язык CHILL позволяет писать программы независимо от используемой машины. Сам язык является машинно-независимым языком; однако в отдельных системах компиляции могут потребоваться объекты, определяемые конкретным применением. Следует заметить, что программы, содержащие такие объекты, не будут, в общем, мобильными.

1.2 ОБЗОР ЯЗЫКА

CHILL-программа содержит три основные части:

- описание объектов данных;
- описание действий, подлежащих выполнению над объектами данных;
- описание структуры программы.

Объекты данных описываются с помощью операторов данных (операторов описания и операторов определения); действия описываются с помощью операторов действия, а структура программы определяется операторами компоновки программы.

Обрабатываемыми объектами данных в языке CHILL являются значения и ячейки, в которых хранятся эти значения. Действия задают операции, выполняемые над объектами данных, а также порядок, в котором значения заносятся для хранения в ячейки и выбираются из ячеек. Структура программы определяет время жизни и видимость объектов данных.

Язык CHILL предусматривает обширный статический контроль использования данных в определенном контексте.

В нижеследующих разделах дается краткое изложение различных понятий языка CHILL. Каждый раздел представляет собой введение в соответствующую главу с тем же названием, которая подробно описывает определенное понятие.

1.3 ВИДЫ И КЛАССЫ

Ячейка имеет некоторый приписанный ей вид. Вид ячейки определяет набор значений, которые могут храниться в ней, а также другие свойства, связанные с ячейкой (заметим, что не все свойства ячейки можно определить одним ее видом). Свойствами ячейки являются: размер, внутренняя структура, возможность автономного чтения содержимого, ссылаемость и т.д. Свойствами значений являются: внутреннее представление, упорядочение, применимые операции и т.д.

Значение имеет приписанный ему класс. Виды ячеек, которые могут содержать данное значение, определяются классом значения.

В языке CHILL имеются следующие категории видов:

дискретные виды:	целый, символьный, булевский, перечислимые (символические) виды и их диапазоны;
множественные виды:	наборы элементов некоторого дискретного вида;
ссылочные виды:	закрепленные ссылки, свободные ссылки и ряды, используемые в качестве ссылок на ячейки;
составные виды:	строковый, массивный и структурный виды;
процедурные виды:	процедуры, рассматриваемые в качестве обрабатываемых объектов данных;
экземплярные виды:	идентификации процессов;
синхронизационные виды:	событийный и буферный виды для синхронизации процессов и связи между ними;
обменные виды:	ассоциативный, доступный и текстовый виды для операций ввода—вывода;
временные виды:	продолжительный и абсолютно-временной виды для контроля времени.

В языке CHILL предусмотрены обозначения для набора стандартных видов. Виды, определяемые программой, могут вводиться с помощью определений видов. Некоторые конструкции языка имеют так называемый приписанный динамический вид. Динамический вид — это вид, ряд свойств которого может быть определен только динамически. Динамические виды — это всегда параметризованные виды с динамическими параметрами. Вид, не являющийся динамическим, называется статическим видом.

Классы в языке CHILL не имеют обозначений. Они вводятся в метаязык только для описания статических и динамических контекстовых условий.

1.4 ЯЧЕЙКИ И ДОСТУП К НИМ

Ячейки являются (абстрактными) местами, где могут храниться значения или из которых значения могут быть получены. Для получения или хранения значения ячейка должна иметь доступ.

Для доступа к ячейке используются имена, определяемые операторами описания. Такими операторами описания являются:

1. описания ячеек;
2. описания опознавателей-ячеек.

Оператор описания ячейки создает ячейки и устанавливает имена доступа ко вновь созданным ячейкам. Оператор описания опознавателя-ячейки устанавливает новые имена доступа для ячеек, созданных где-нибудь в другом месте.

Кроме описаний ячеек, новые ячейки могут быть созданы в результате вызовов встроенных программ *GETSTACK* или *ALLOCATE*, выдающих ссылочные значения (см. ниже) на вновь созданную ячейку.

Ячейка может быть **ссылочной**. Это означает, что для данной ячейки существует соответствующее ссылочное значение. Это ссылочное значение получается в результате операции ссылки, применяемой к ссылочной ячейке. Ссылаемая ячейка получается в результате разыменования ссылочного значения. В языке CHILL требуется, чтобы одни ячейки были **ссылочными**, а другие не были таковыми, однако в последнем случае решение вопроса о том, являются ли ячейки **ссылочными** или нет, зависит от реализации. Ссылаемость должна быть статически определяемым свойством ячеек.

Ячейка может иметь **неизменяемый** вид, что означает, что доступ к ней может осуществляться только с целью получения значения, а не с целью хранения в ней нового значения (за исключением инициализации).

Ячейка может быть составной, что означает, что ячейка имеет субъячейки, доступ к которым осуществляется порознь. Субъячейка не обязательно должна быть **ссылочной**. Считается, что ячейка, содержащая по крайней мере одну **неизменяемую** субъячейку, обладает **свойством неизменяемости**. Методами доступа, выдающими субъячейки (или субзначения), являются индексирование и вырезание строк и массивов (подстроки, подмассивы), а также выбор структур.

Ячейка имеет приписанный ей вид. Если этот вид динамический, то ячейка называется ячейкой динамического вида.

Следующие свойства ячейки, хотя и определяемые статически, не являются частью вида:

ссылаемость: существует или не существует ссылочное значение для ячейки;

класс памяти: является ли данная ячейка распределяемой статически;

региональность: описана ли данная ячейка внутри некоторого региона.

1.5 ЗНАЧЕНИЯ И ОПЕРАЦИИ НАД НИМИ

Значения являются основными объектами, над которыми определены конкретные операции. Значение может быть либо определенным значением (в смысле языка CHILL), либо **неопределенным** (в смысле языка CHILL). Употребление неопределенного значения в специфицированном контексте приводит к неопределенной ситуации (в смысле языка CHILL), и программа считается ошибочной.

Язык CHILL разрешает использовать ячейки в тех контекстах, где требуются значения. В этом случае ячейка доступна для получения содержащегося в ней значения.

Значение имеет приписанный ему класс. **Строгие** значения — это значения, имеющие, кроме их класса, также приписанный им вид. В этом случае значение — это всегда одно из значений, определемых видом. Класс используется для проверки совместимости, а вид — для описания свойств значения. В некоторых контекстах требуется, чтобы эти свойства были известны; в этом случае требуется **строгое** значение.

Значение может быть **константой**; в этом случае оно обозначает некоторое независимое от реализации дискретное значение, известное во время компиляции. Значение может быть **постоянным**; в этом случае оно вырабатывает одно и то же значение, то есть его требуется вычислить только один раз. Когда в контексте требуется **постоянное** значение или **константа**, то предполагается, что значение вычисляется перед выполнением программы и, следовательно, не может генерировать динамическое исключение. Значение может быть **внутрирегиональным**; в этом случае оно может как-то ссылаться на ячейки, описанные внутри региона. Значение может быть составным, то есть содержать субзначения.

Операторы определения синонимов устанавливают новые имена для обозначения **постоянных** выражений.

1.6 ДЕЙСТВИЯ

Действия составляют алгоритмическую часть CHILL-программы.

Оператор присваивания сохраняет (вычисленное) значение в одной или нескольких ячейках. Вызов процедуры активизирует процедуру, вызов встроенной программы активизирует встроенную программу (встроенная программа — это процедура, определение которой в языке CHILL не является необходимым, а параметры и механизм получения результата в которой могут быть более общими). Для возврата результата и/или установления результата или при вызове процедуры используются действия возврата и получения результата.

Для управления потоком последовательных действий язык CHILL предусматривает следующие управляющие операторы:

условный оператор	для двойного ветвления;
оператор выбора	для множественного ветвления. Выбор ветви может основываться на нескольких значениях, подобно таблице решений;
оператор цикла	для итерации или заключения в скобки;
оператор выхода	для выхода из сложного оператора или модуля, выполненного структурным образом;
оператор причины	возбуждает конкретное исключение;
оператор перехода	для безусловного перехода к помеченной точке программы.

Операторы действия и операторы данных могут быть сгруппированы в модуль или блок begin-end, образуя (составное) действие.

Для управления потоком параллельных действий язык CHILL предусматривает операторы запуска, останова, задержки, продолжения, посылки, выбора задержки и варианта получения и выражения получения и запуска.

1.7 ВВОД И ВЫВОД

Средства ввода—вывода в языке CHILL обеспечивают возможности взаимодействия с множеством устройств внешней среды.

В эталонной модели ввода—вывода имеются три состояния. В свободном состоянии взаимодействие с внешней средой не происходит.

Посредством операции *ASSOCIATE* можно войти в состояние обработки файла. В этом состоянии имеются ячейки ассоциативного вида, обозначающие объекты внешней среды. С помощью встроенных программ можно читать и модифицировать атрибуты ассоциаций, определяемые в языке, то есть **существующий, читаемый, возможный для записи, индексируемый, последовательный** и **переменный**. Создание и исключение файла также совершаются в состоянии обработки файла.

Посредством операции *CONNECT* ячейка доступного вида связывается с ячейкой ассоциативного вида, и осуществляется вход в состояние переноса данных. Операция *CONNECT* позволяет указать позицию **базового** индекса в файле. В состоянии переноса данных можно проверить различные атрибуты ячеек доступного вида, а также использовать операции переноса данных *READRECORD* и *WRITERECORD*.

Посредством операций переноса текста значения в языке CHILL могут быть представлены в удобочитаемом виде для передачи в файл или из файла, а также в ячейку или из ячейки.

1.8 ОБРАБОТКА ИСКЛЮЧЕНИЙ

Динамические семантические условия языка CHILL — это те условия (не контекстно-свободные), которые, в общем случае, не могут быть определены статически. (Решение о том, генерировать код или нет для тестирования динамических условий во время прогона программы, если соответствующая программа обработки исключений явно не описана, зависит от реализации.) Нарушение динамических семантических правил приводит к исключениям, касающимся времени выполнения; однако если реализация может определить статически, что динамическое условие будет нарушено, то программа может быть отброшена.

Исключения также могут быть вызваны выполнением оператора причины или, при некотором условии, выполнением оператора контроля. Если в данной точке программы имеет место исключение, то управление передается соответствующей программе обработки данного исключения, если таковая описана (то есть она имеет имя), и описывается. Статически можно определить, описана или нет программа обработки исключения в данной точке. Если программа обработки исключения явно не задана, то управление может быть передано программе обработки исключений, определяемой реализацией.

Исключения имеют имя, являющееся либо именем, определяемым в языке CHILL, либо именем исключения, определяемым реализацией, либо именем исключения, определяемым программой. Заметим, что когда программа обработки исключений описывается по имени исключения, то должно проверяться соответствующее динамическое условие.

1.9 КОНТРОЛЬ ВРЕМЕНИ

Средства контроля времени в языке CHILL дают возможность реагировать на ход времени во внешней среде.Процессы в языке CHILL могут **прерываться** в течение выполнения только в точные **прерываемые** моменты. Когда это случается, управление передается к соответствующей программе обработки исключений.

Программы могут обнаруживать окончание периода времени или могут синхронизироваться с абсолютной точкой времени или точными интервалами времени без накопления погрешностей. Встроенные программы времени предназначены для преобразования значений времени и продолжительности в целые значения, для постановки процесса в состояние ожидания и для обнаружения конца контроля времени.

1.10 СТРУКТУРА ПРОГРАММ

Операторами структуры программы являются блок begin-end, модуль, процедура, процесс и регион. Операторы структуры программы обеспечивают средства управления временем жизни ячеек и видимостью имен.

Время жизни ячейки — это время, в течение которого ячейка существует в программе. Ячейки могут быть описаны явно (в описании ячейки) или генерироваться (в результате вызова встроенных программ *GETSTACK* или *ALLOCATE*), или же они могут быть описаны неявно либо генерироваться в результате использования конструкций языка.

Считается, что имя **видимо** в определенной точке программы, если оно может быть использовано в этой точке. Область действия имени объединяет все точки, где оно **видимо**, то есть где обозначаемый объект идентифицируется данным именем.

Блоки begin-end определяют видимость имен и время жизни ячеек.

Модули ограничивают видимость имен для защиты от несанкционированного использования. С помощью операторов видимости можно управлять видимостью имен в различных частях программы.

Процедура — это (возможно, параметризованная) подпрограмма, которая может быть активизирована (вызвана) в различных местах программы. Она может возвращать значение (процедура возврата значения) или ячейку (процедура возврата ячейки) или же не вырабатывать никакого результата. В последнем случае процедура может быть вызвана только в операторе вызова процедуры.

Процессы и регионы обеспечивают средства параллельного исполнения.

Полная CHILL-программа — это список модулей или регионов, окруженных определением (воображаемого) процесса. Этот внешний процесс запускается системой, под управлением которой выполняется программа.

Конструкции облегчают различные способы модульной разработки программы. Модуль спецификации и регион спецификации используются для определения статических свойств модуля программы; контекст используется для определения статических свойств занимаемых имен. С помощью средств удаленной спецификации можно дополнительно описать тот текст программного модуля, который находится где-нибудь в другом месте.

1.11 ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ

Язык CHILL допускает параллельное выполнение программных блоков. Процесс — это блок, выполняемый параллельно. Вычисление выражения запуска вызывает создание нового процесса по указанному определению процесса. Процесс затем считается выполняемым параллельно с тем процессом, который его запустил. Язык CHILL допускает существование одновременно одного или нескольких процессов с одинаковыми или различными определениями. Оператор останова, выполненный некоторым процессом, является причиной его завершения.

Процесс всегда находится в одном из двух состояний: он может быть либо активным, либо задержанным. Переход из активного состояния в задержанное называется задержкой процесса; обратный переход называется реактивацией процесса. Причиной задержки процесса могут служить действия задержки по событиям, действия приема сигналов или приема в буферы или действия посылки в буферы. Причиной реактивации процесса могут служить действия задержки по событиям, действия посылки в буферы или посылки сигналов, действия приема в буферы.

Буферы и события — это ячейки с ограниченным использованием. Операторы посылки, получения и варианты получения определены на буферах; операторы задержки, выбора задержки и продолжения определены на событиях. Буферы — это средства синхронизации и передачи информации между процессами. События используются только в целях синхронизации. Сигналы определяются операторами определения сигналов. Они обозначают функции составления и разложения списков значений, передаваемых между процессами. Операторы посылки и операторы варианта получения служат целям передачи списка значений и синхронизации.

Регион — это модуль специального вида. Он используется для обеспечения взаимоисключающего доступа к структурам данных, совместно используемым несколькими процессами.

1.12 ОБЩИЕ СЕМАНТИЧЕСКИЕ СВОЙСТВА

Семантические (не контекстно-свободные) условия языка CHILL — это условия совместимости видов и классов (проверка вида) и условия видимости (проверка области действия). Правила для видов определяют, каким образом могут использоваться имена; правила для области действия определяют, где могут использоваться имена.

Правила для видов формулируются в терминах требований совместимости между видами, классами, а также между видами и классами. Требования совместимости между видами и классами, а также между самими видами определяются в терминах отношений эквивалентности между видами. Если рассматриваются динамические виды, то проверка вида является частично динамической.

Правила для области действия определяют видимость имен посредством структуры программы и операторов явной видимости. Операторы явной видимости влияют на область действия упомянутых имен, а также на, возможно, **неявные** имена из упомянутых имен. Используемые в программе имена имеют место их описания или определения. Это место называется определяющим вхождением имени. Места, где имя используется, называются вхождениями с использованием имени.

1.13 ВАРИАНТЫ РЕАЛИЗАЦИИ

В языке CHILL допускаются целые виды, определяемые реализацией: встроенные программы, имена **процессов**, программы обработки исключений и имена исключений.

Определяемый реализацией целый вид должен обозначаться именем **вида**, определяемым реализацией. Считается, что это имя устанавливается в операторе определения нового вида, который не описан в языке CHILL. Распространение существующих арифметических операций, устанавливаемых языком CHILL, на целые виды, определяемые реализацией, допустимо в рамках синтаксических и семантических правил языка CHILL. Примерами целых видов, определяемых реализацией, могут служить длинные целые и короткие целые.

Встроенная программа — это процедура, определение которой на языке CHILL не является необходимым и которая может иметь более общую схему передачи параметров и результатов по сравнению с процедурами языка CHILL.

Имя встроенного **процесса** — это имя **процесса**, определение которого на языке CHILL не является необходимым и который может иметь более общую схему передачи параметров по сравнению с процессами языка CHILL. Процесс языка CHILL может объединяться с встроенным процессом или запускать такие процессы.

Определяемая реализацией программа обработки исключений — это программа обработки исключений, присоединенная к определению процесса. Если управление передается к этой программе после того, как появляется исключение, то состав предпринимаемых действий определяется реализацией. Нарушение определяемого реализацией динамического условия является причиной определяемого реализацией исключения.

2 ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

2.1 МЕТАЯЗЫК

Описание языка CHILL состоит из двух частей:

- описание контекстно-свободного синтаксиса;
- описание семантических условий.

2.1.1 Описание контекстно-свободного синтаксиса

Контекстно-свободный синтаксис описывается с помощью расширенной формы Бекуса—Наура (БНФ). Синтаксические категории обозначаются одним или несколькими словами, написанными символами, имеющими курсивное начертание, и заключенными в угловые скобки (< и >). Такое обозначение называется нетерминальным символом. Для каждого нетерминального символа в соответствующем разделе синтаксиса дается некоторое правило вывода. Правило вывода для нетерминального символа состоит из данного нетерминального символа слева от символа :: = и одной или нескольких конструкций, состоящих из нетерминальных и/или терминальных символов в правой части. Эти конструкции отделяются вертикальной чертой (|) для обозначения альтернативных выводов для данного нетерминального символа.

Иногда нетерминальный символ включает некоторую подчеркнутую часть. Эта подчеркнутая часть не является частью контекстно-свободного описания, а определяет семантическую категорию (см. раздел 2.1.2).

Синтаксические элементы могут группироваться с помощью фигурных скобок ({ и }). Повторение элементов внутри фигурных скобок указывается звездочкой (*) или знаком плюс ($^+$). Звездочка означает, что группа элементов не является обязательной и в дальнейшем может быть повторена любое число раз; знак плюс означает, что наличие группы элементов является обязательным и в дальнейшем она может быть повторена произвольное число раз. Например, $\{ A \}^*$ означает, что имеется последовательность элементов A, в том числе и пустая, в то время как $\{ A \}^+$ означает любую последовательность элементов A, но не пустую. Если синтаксические элементы группируются с использованием квадратных скобок ([и]), то эта группа не является обязательной. В группах, выделенных фигурными или квадратными скобками, может иметься одна или несколько вертикальных черт, указывающих на наличие альтернативных синтаксических элементов.

Между строгим синтаксисом, при котором семантические условия задаются непосредственно, и производным синтаксисом имеются различия. Производный синтаксис рассматривается как расширение строгого синтаксиса, а семантика производного синтаксиса косвенно объясняется в терминах соответствующего строгого синтаксиса.

Следует отметить, что описание контекстно-свободного синтаксиса выбирается для соответствия описанию семантики в данном документе, а не для соответствия некоторому отдельному алгоритму синтаксического анализа (например, имеются некоторые неоднозначности, связанные с контекстно-свободным синтаксисом и введенные для ясности). Эти неоднозначности устраняются путем использования семантической категории синтаксических элементов.

2.1.2 Описание семантики

Для каждой синтаксической категории (нетерминального символа) описание семантики приводится в следующих подразделах: **Семантика**, **Статические свойства**, **Динамические свойства**, **Статические условия** и **Динамические условия**.

Раздел **Семантика** описывает понятия, обозначенные синтаксическими категориями (то есть их значение и поведение).

Раздел **Статические свойства** рассматривает статически определяемые семантические свойства синтаксической категории. Эти свойства используются при формулировке статических и/или динамических условий в тех разделах, где используется данная синтаксическая категория.

Раздел **Динамические свойства** определяет свойства синтаксической категории, которые известны только динамически.

Раздел **Статические условия** описывает контекстно-зависимые, статически проверяемые условия, которые должны выполняться при использовании синтаксической категории. Некоторые статические условия выражаются в самом синтаксисе с помощью подчеркнутой части в нетерминальном символе (см. раздел 2.1.1). Такое использование требует, чтобы нетерминальный символ относился к указанной семантической категории. Например, < *булевское выражение* > идентично в контекстно-свободном смысле нетерминальному символу < *выражение* >, но семантически требуется, чтобы *выражение* было булевского класса.

Раздел **Динамические условия** описывает контекстно-зависимые условия, которые должны выполняться в течение периода выполнения программы. В некоторых случаях условия являются статическими при отсутствии динамических видов. В этих случаях условие упоминается в разделе **Статические условия**, но обращение к нему имеет место в разделе **Динамические условия**. В других случаях динамические условия могут проверяться статически; в реализации это может трактоваться как нарушение статического условия.

При описании семантики используется различный шрифт, а именно: курсив (без < и >) используется для указания на синтаксические объекты; соответствующие термины, имеющие прямое начертание, указывают на соответствующие семантические объекты (например, *ячейка* обозначает ячейка). Жирное начертание используется для наименования семантических свойств; иногда это свойство может быть выражено как синтаксически, так и семантически (например, предложение "*выражение* является **постоянным**" означает то же самое, что и предложение "*выражение* является *постоянным выражением*").

Если не указано иное, то семантика, свойства и условия, описанные в подразделе синтаксической категории остаются в силе независимо от контекста, в котором в других разделах может встретиться эта синтаксическая категория.

Свойства синтаксической категории A, имеющей правило вывода в виде A :: = B, где B — синтаксическая категория, являются теми же, что и у B, если не указано иное.

2.1.3 Примеры

Среди большинства синтаксических разделов имеется раздел **Примеры**, содержащий один или несколько примеров определенных синтаксических категорий. Эти примеры взяты из ряда примеров программ, содержащихся в добавлении D. Ссылки указывают на то синтаксическое правило, с помощью которого получен пример и из какого примера взята синтаксическая категория.

Например, *6.20* (*d* + *5*)*/5* (*1.2*) указывает на пример терминальной строки (*d* + *5*)*/5*, полученной по правилу (*1.2*) соответствующего синтаксического раздела, взятой из примера программы № *6*, строка *20*.

2.1.4 Правила связывания в метаязыке

Иногда описание семантики упоминает строки **специальных** простых имен (см. добавление C). Эти строки **специальных** простых имен всегда используются со своим значением, установленным в языке CHILL и поэтому на них не оказывают влияния правила связывания, используемые в реальной CHILL-программе.

2.2 СЛОВАРЬ

Программы представляются с использованием набора символов языка CHILL (см. добавление A). Алфавит представляется синтаксической категорией < *символ* >, из которой можно получить в качестве терминального порождения любой символ набора символов языка CHILL.

Лексическими элементами языка CHILL являются:

- специальные символы;
- строки простых имен;
- константы.

Кроме лексических элементов, имеются также комбинации специальных символов. Специальные символы и комбинации специальных символов перечислены в добавлении B.

Строки простых имен образуются согласно следующему синтаксису:

Синтаксис:

< *строка простого имени* > :: = *(1)*
< *буква* > {< *буква* > |< *цифра* > | _ }* (1.1)

< *буква* > :: = *(2)*
A | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* | *J* | *K* | *L* | *M* *(2.1)*
| *N* | *O* | *P* | *Q* | *R* | *S* | *T* | *U* | *V* | *W* | *X* | *Y* | *Z* *(2.2)*
| *a* | *b* | *c* | *d* | *e* | *f* | *g* | *h* | *i* | *j* | *k* | *l* | *m* *(2.3)*
| *n* | *o* | *p* | *q* | *r* | *s* | *t* | *u* | *v* | *w* | *x* | *y* | *z* *(2.4)*

< *цифра* > :: = *(3)*
0 | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* *(3.1)*

Семантика: Символ подчеркивания (_) составляет часть строки простого имени; например, строка простого имени *life_time* отличается от строки простого имени *lifetime*. Прописные и строчные буквы различаются, например, *Status* и *status* — это две различные строки простых имен.

Язык имеет некоторое число строк **специальных** простых имен с предопределенными значениями (см. добавление С). Некоторые из них являются **резервированными**, то есть их нельзя использовать в других целях.

Строки **специальных** простых имен в программном модуле представлены все либо прописными, либо строчными буквами. Строки **резервированных** простых имен резервируются только в выбранном представлении (например, если выбрано представление строчными буквами, то **row** является зарезервированным, а **ROW** — нет).

Статические условия: *Строка простого имени* может не быть одной из строк **резервированных** простых имен (см. добавление С.1).

2.3 ИСПОЛЬЗОВАНИЕ ПРОБЕЛОВ

Любой лексический элемент или комбинация специальных символов заканчиваются пробелом. Лексические элементы также заканчиваются первым символом, который не может быть частью лексического элемента. Например, *IFBTHEN* будет рассматриваться как *строка простого имени*, а не как начало оператора **IF** *B* **THEN**, //* будет рассматриваться как символ конкатенации (//) с последующей звездочкой (*), а не как символ деления (/) с последующей открывающей скобкой комментария (/*).

2.4 КОММЕНТАРИИ

Синтаксис:

< комментарий > :: = *(1)*
< комментарий в скобках > *(1.1)*
| *< комментарий в конце строки >* *(1.2)*

< комментарий в скобках > :: = *(2)*
*/ * < символьная строка > * /* *(2.1)*

< комментарий в конце строки > :: = *(3)*
- - < символьная строка > < конец-строки > *(3.1)*

< символьная строка > :: = *(4)*
{ *< символ >* }* (4.1)

Примечание. — *Конец-строки* обозначает конец строки с комментарием.

Семантика: *Комментарий* предоставляет информацию пользователю программы. Комментарий не влияет на семантику программы.

Комментарий может быть введен во все места, где разрешены пробелы в качестве разделителей.

Комментарий в скобках заканчивается первым вхождением специальной последовательности: */. *Комментарий в конце строки* заканчивается первым вхождением конца строки.

Примеры:

4.1 */* из набора алгоритмов для CACM, № 93 */* *(2.1)*

2.5 СПЕЦИФИКАТОРЫ ФОРМАТА

Спецификаторы формата BS (возврат позиции), CR (возврат каретки), FF (перевод страницы), HT (горизонтальное табурлирование), LF (перевод строки) и VT (вертикальное табулирование) набора символов языка CHILL (см. добавление А, позиции FE_0—FE_5) не упоминаются в описании контекстно-свободного синтаксиса языка CHILL. При использовании они, как и пробелы, служат разделителями. Пробелы и спецификаторы формата не могут встречаться в лексических элементах (за исключением символьных строковых констант).

2.6 ДИРЕКТИВЫ КОМПИЛЯТОРА

Синтаксис:

< предложение директивы > :: = *(1)*
< >< директива > { , < директива > } < >* **(1.1)**

< директива > :: = *(2)*
< директива реализации > *(2.1)*

Семантика: Предложение директивы передает информацию компилятору. Данная информация специфицирована в формате, определенном реализацией.

Директива реализации не должна влиять на семантику программы, то есть программа с директивами реализации корректна в смысле языка CHILL тогда и только тогда, если она корректна без этих директив.

Предложение директивы завершается первым вхождением символа окончания директивы (< >). *Директива* может содержать любой символ из набора символов (см. добавление А).

Статические свойства: *Предложение директивы* может быть вставлено на любое место, где разрешены пробелы. Оно имеет такой же разделительный эффект, как пробел. Используемые в *предложении директивы* имена подчиняются схеме связывания имен, определенной реализацией, не влияющей на правила связывания имен в языке CHILL (см. раздел 12.2).

2.7 ИМЕНА И ИХ ОПРЕДЕЛЯЮЩИЕ ВХОЖДЕНИЯ

Синтаксис:

< имя > :: = *(1)*
< строка имени > *(1.1)*

< строка имени > :: = *(2)*
< строка простого имени > *(2.1)*
| *< строка префиксного имени >* *(2.2)*

< строка префиксного имени > :: = *(3)*
< префикс > ! < строка простого имени > *(3.1)*

< префикс > :: = *(4)*
*< простой префикс > { !< простой префикс >}** **(4.1)**

< простой префикс > :: = *(5)*
< строка простого имени > *(5.1)*

< определяющее вхождение > :: = *(6)*
< строка простого имени > *(6.1)*

< список определяющих вхождений > :: = *(7)*
*< определяющее вхождение > {,< определяющее вхождение > }** **(7.1)**

< имя поля > :: = *(8)*
< строка простого имени > *(8.1)*

< определяющее вхождение имени поля > :: = *(9)*
< строка простого имени > *(9.1)*

< список определяющих вхождений имен полей > :: = *(10)*
*< определяющее вхождение имени поля > { , < определяющее вхождение имени поля > }** **(10.1)**

< имя исключения > :: = *(11)*
< строка простого имени > *(11.1)*
| *< строка префиксного имени >* *(11.2)*

< имя ссылки на текст > :: = *(12)*
< строка простого имени > *(12.1)*
| *< строка префиксного имени >* *(12.2)*

Семантика: Имена в программе определяют объекты. Для данного вхождения *имени* (формально: вхождение терминального порождения *имени*) в программе правила связывания, приведенные в разделе 12.2, приводят к *определяющим вхождениям* (формально: вхождениям терминальных порождений *определяющего вхождения*), с которыми **связано** это *имя* (вхождение *имени*). *Имя* обозначает объект, определенный или описанный *определяющими вхождениями*. (Несколько *определяющих вхождений имени* может иметься только в случае *имен элементов перечисления* или *имен* с **квазиопределяющими** *вхождениями*.) Считается, что *определяющие вхождения* определяют *имя* и *имя* является вхождением с использованием имени, созданного *определяющим вхождением*, с которым **связано** данное имя. Самая правая *строка простого имени* и строка *имени* для данного имени совпадают.

Подобно этому *имена полей* **связаны** с *определяющими вхождениями имен полей* и обозначают поля (структурного вида), устанавливаемые этими *определяющими вхождениями имен полей*.

Имена исключений используются для идентификации программ обработки исключений в соответствии с правилами главы 8.

Имена ссылок на текст используются для идентификации описаний блоков исходного текста способом, зависящим от реализации, согласно правилам раздела 10.10.1.

Когда имя **связано** с несколькими *определяющими вхождениями*, тогда каждое из этих *определяющих вхождений*, с которым *имя* **связано**, устанавливает или описывает один и тот же объект (см. правила в разделах 10.10 и 12.2.2).

Определение обозначений: Для данной *строки имени* NS и строки символов P, являющейся либо *префиксом*, либо пустой, результат снабжения строки имени NS префиксом P, записанный как P ! NS, определяется следующим образом:

- если P пустая, то P ! NS есть NS;
- в противном случае P ! NS — это строка имени, полученная конкатенацией всех символов в P, префиксной операции и всех символов в NS.

Например, если P есть "*q* ! *r*", а NS есть "*s* ! *n*", тогда P ! NS есть "*q* ! *r* ! *s* ! *n*".

Статические свойства: Каждая *строка простого имени* имеет приписанную строку **канонического** имени, которая сама является *строкой простого имени*. *Строка имени* имеет приписанную строку **канонического** имени, которая, в случае:

- если *строка имени* — это *строка простого имени*, является строкой **канонического** имени для данной *строки простого имени*;
- если *строка имени* — это *строка префиксного имени*, является конкатенацией слева направо всех *строк простых имен* в *строке имени*, разделенных префиксными операциями, то есть содержащиеся пробелы, комментарии и спецификаторы формата (если имеются) выбрасываются.

В остальной части данного документа:

- строка имени для *имени*, *имени исключения* или *имени ссылки на текст* используется для обозначения строки **канонического** имени для *строки имени* в этом *имени*, *имени исключения* или *имени ссылки на текст* соответственно;
- строка имени для *определяющего вхождения*, *имени поля* или *определяющего вхождения имени поля* используется для обозначения строки **канонического** имени для *строки простого имени* в этом *определяющем вхождении*, *имени поля* или *определяющего вхождения имени поля* соответственно.

Правила связывания таковы, что:

- *имена* со *строкой простого имени* **связаны** с *определяющими вхождениями* с одной и той же *строкой имени*;
- *имена* со *строкой префиксного имени* **связаны** с *определяющими вхождениями* с одной и той же *строкой* имени, что и самая правая *строка простого имени* в *строке префиксного имени* данного *имени*;
- *имена полей* **связаны** с *определяющими вхождениями имени поля* с той же самой *строкой имени*, что и *имена полей*.

Имя наследует все статические свойства, приписанные имени *определяющим вхождением*, с которым оно **связано**. *Имя поля* наследует все статические свойства, приписанные имени поля, устанавливаемому *определяющим вхождением имени поля*, с которым оно **связано**.

3 ВИДЫ И КЛАССЫ

3.1 ОБЩИЕ ПОЛОЖЕНИЯ

Каждая ячейка имеет некоторый вид, каждое значение имеет некоторый класс. Вид ячейки определяет тот набор значений, которые может содержать ячейка, методы доступа к этой ячейке и операции, допустимые над этими значениями. Класс значения — это средство определения видов ячеек, которые могут содержать данное значение. Некоторые значения являются **строгими**. **Строгое** значение имеет связанные с ним класс и вид. **Строгие** значения необходимы в тех контекстах значений, где требуется информация о видах.

3.1.1 Виды

В языке CHILL имеются статические виды (то есть виды, все статические свойства которых определяются статически) и динамические виды (то есть виды, некоторые свойства которых известны только во время выполнения программы). Динамические виды — это всегда параметризованные виды с динамическими параметрами.

Статические виды являются терминальными порождениями *вида* синтаксической категории.

В данном документе введены имена виртуальных **видов** для описания видов, явно не обозначенных в тексте программы. В таких случаях перед именем **вида** ставится амперсанд (знак &).

Виды также параметризуются значениями, неявно обозначенными в тексте программы.

3.1.2 Классы

В языке CHILL классы не имеют обозначений.

Любое значение в CHILL-программе имеет класс одного из следующих видов, имеющихся в языке CHILL:

- для вида M существует класс M-значения. Все значения такого класса и только они являются **строгими**, а связанный со значением вид — это M;
- для вида M существует M-производный класс;
- для любого вида M существует M-ссылочный класс;
- **нулевой** класс;
- **полный** класс.

Два последних класса — это постоянные классы, то есть они не зависят от вида M. Считается, что класс является динамическим тогда и только тогда, если это класс M-значения, M-производный класс или M-ссылочный класс, где M — динамический вид.

3.1.3 Свойства видов и классов и отношения между ними

Виды в языке CHILL обладают свойствами. Это могут быть наследственные и ненаследственные свойства. Наследственное свойство передается от определяющего вида к имени **вида**, определяемому видом. Ниже приведен перечень свойств всех видов (определения свойств видов, за исключением первого, приведены в разделе 12.1):

- вид обладает **новизной** (определение дано в разделах 3.2.2, 3.2.3 и 3.3);
- вид обладает **свойством неизменяемости** (свойством **только-чтения**);
- вид может быть **параметризованным**;
- вид может обладать **свойством ссылаемости**;
- вид может обладать **свойством теговой параметризации**;
- вид может обладать **свойством беззначимости**.

Классы в языке CHILL могут обладать следующими свойствами (которые определены в разделе 12.1):

- класс может иметь **корневой** вид;
- один или несколько классов могут иметь **результирующий класс**.

Операции в языке CHILL определяются видами и классами ячеек и значений. Это выражается правилами проверки видов (раздел 12.1) в виде отношений между видами и классами. Существуют следующие отношения:

- два вида могут быть **подобными**;
- два вида могут быть **v-эквивалентными**;
- два вида могут быть **эквивалентными**;
- два вида могут быть **l-эквивалентными**;
- два вида могут быть **похожими**;
- два вида могут быть **связаны новизной**;
- два вида могут быть **совместимыми по чтению**;
- два вида могут быть **динамическими совместимыми по чтению**;
- два вида могут быть **динамическими эквивалентными**;
- вид может быть **ограничиваемым** по виду;
- вид может быть **совместимым** с классом;
- класс может быть **совместимым** с классом.

3.2 ОПРЕДЕЛЕНИЯ ВИДОВ

3.2.1. Общие положения

Синтаксис:

< определение вида > :: = *(1)*
< список определяющих вхождений > = < определяющий вид > *(1.1)*

< определяющий вид > :: = *(2)*
< вид > *(2.1)*

Производный синтаксис: *Определение вида*, где *список определяющих вхождений* содержит несколько *определяющих вхождений*, получено из нескольких определений вида (по одному на каждое определяющее вхождение), разделенных запятыми, с одним и тем же *определяющим видом*. Например:

NEWMODE *dollar, pound = INT*;

получено из:

NEWMODE *dollar = INT, pound = INT*;

Семантика: Определение вида задает имя, обозначающее описанный вид. Определения видов встречаются в операторах определения нового вида и синонимического вида. Синонимический вид является **синонимическим** с его определяющим видом. Новый вид не является **синонимическим** с его определяющим видом. Различие определяется в терминах свойства **новизны**, используемого при проверке видов (см. раздел 12.1).

Статические свойства: *Определяющее вхождение* в *определении вида* определяет имя **вида**.

Предопределенные имена **видов** и имена целых **видов**, определяемых реализацией (если имеются, см. раздел 3.4.2), также являются именами **видов**.

Имя **вида** имеет **определяющий** вид, являющийся *определяющим видом* в *определении вида*, которое его устанавливает. (Для предопределенных имен **видов** и имен **видов**, определяемых реализацией, этот **определяющий** вид является виртуальным.) Наследственные свойства имени **вида** те же, что и у его **определяющего** вида.

Набор рекурсивных определений — это набор таких определений видов или определений синонимических видов (см. раздел 5.1), что *определяющий вид* в каждом *определении вида* или *постоянное значение* или *вид* в каждом *определении синонима* являются (или содержат непосредственно) именем **вида** или именем **синонима**, задаваемыми определением из набора определений.

Набор определений рекурсивных видов — это набор рекурсивных определений, имеющих только определения видов. (Любой набор рекурсивных определений должен быть набором определений рекурсивных видов; см. раздел 5.1.)

Считается, что любой вид, являющийся именем вида или содержащий имя **вида**, устанавливаемое в наборе определений рекурсивного вида, обозначает рекурсивный вид. Цепь в наборе определений рекурсивного вида является списком имен **видов**, причем каждое имя индексируется маркером, так что:

- все имена в цепи имеют различное определение;
- для каждого имени его преемник является его определяющим видом или непосредственно входит в него (преемником последнего имени является первое имя);
- маркер однозначно указывает позицию имени в определяющем виде его предшественника (предшественник первого имени является последним именем).

(Примеры: **NEWMODE** *M* = **STRUCT** (*i M*, *n* **REF** *M*); содержит две части: $\{M_i\}$ и $\{M_n\}$.)

Цепь **надежна** тогда и только тогда, когда по крайней мере одно из ее имен содержится в *ссылочном виде*, *рядовом виде* или *процедурном виде* на маркируемом месте.

Статические условия: Для любого набора определений рекурсивных видов все его цепи должны быть **надежными**. (Первая цепь в примере, приведенном выше, не является **надежной**.)

Примеры:

1.15 *operand_mode = INT* *(1.1)*
3.3 *complex* = **STRUCT** *(re,im INT)* *(1.1)*

3.2.2 Определения синонимических видов

Синтаксис:

< *оператор определения синонимического вида* > :: = *(1)*
SYNMODE < *определение вида* > { , < *определение вида* > }* ; (1.1)

Семантика: Оператор определения синонимического вида определяет имена **видов**, **синонимических** с их определяющим видом.

Статические свойства: *Определяющее вхождение* в *определении вида оператора определения синонимического вида* устанавливает имя **синонимического вида** (являющееся также именем **вида**). Считается, что имя **синонимического вида** является **синонимическим** с видом M (и наоборот, M является **синонимическим** с именем **синонимического вида**) тогда и только тогда, когда:

- либо M является **определяющим** видом имени **синонимического вида**;
- либо **определяющий** вид имени **синонимического вида** сам является именем **синонимического вида**, которое является **синонимическим** с M.

Новизна имени **синонимического вида** является новизной его **определяющего** вида.

Если **определяющий** вид — ограниченный вид, то **родительский** вид имени **синонима** является родительским видом его **определяющего** вида. Если **определяющий** вид — **изменяющийся** строковый вид, то **компонентный** вид имени **синонима** является компонентным видом его **определяющего** вида.

Примеры:

6.3 **SYNMODE** *month* = **SET** *(jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec);* *(1.1)*

3.2.3 Определение новых видов

Синтаксис:

< *оператор определения нового вида* > :: = *(1)*
NEWMODE < *определение вида* > { , < *определение вида* > }*; (1.1)

Семантика: Оператор определения нового вида устанавливает имена **видов**, которые не являются **синонимическими** с их определяющим видом.

Статические свойства: *Определяющее вхождение* в *определении вида оператора определения нового вида* устанавливает имя **нового вида** (являющееся также именем **вида**).

Новизна имени **нового вида** — это *определяющее вхождение*, определяющее его. Если **определяющий** вид имени **нового вида** — это ограниченный вид, тогда *&имя* виртуального вида вводится как **родительский** вид имени **нового вида**. **Определяющий** вид *&имени* — это **родительский** вид ограниченного вида, а **новизна** *&имени* — это новизна имени **нового вида**.

Если **определяющий** вид — это **изменяющийся** строковый вид, тогда *&имя* виртуального вида вводится как **компонентный** вид имени **нового вида**. Определяющий вид *&имени* является **компонентным** видом **изменяющегося** строкового вида, а **новизна** *&имени* — это новизна имени **нового вида**.

Если *определяющее вхождение* определения вида — это **квазиопределяющее** *вхождение*, то **новизна** является **квазиновизной**, в противном случае это **действительная новизна**.

Статические условия: Если **новизна** — это **квазиновизна**, то только одна **действительная новизна** должна быть **связана новизной** с ней.

Примеры:

11.6 **NEWMODE** *line* = *INT (1:8);* *(1.1)*
11.12 **NEWMODE** *board* = **ARRAY** *(line)* **ARRAY** *(column) square;* *(1.1)*

3.3 КЛАССИФИКАЦИЯ ВИДОВ

Синтаксис:

< *вид* > :: = *(1)*
[**READ**] < *несоставной вид* > (1.1)
| [**READ**] < *составной вид* > (1.2)

< *несоставной вид* > :: = *(2)*
< *дискретный вид* > *(2.1)*
| < *множественный вид* > *(2.2)*
| < *ссылочный вид* > *(2.3)*
| < *процедурный вид* > *(2.4)*
| < *экземплярный вид* > *(2.5)*
| < *синхронизационный вид* > *(2.6)*
| < *обменный вид* > *(2.7)*
| < *временной вид* > *(2.8)*

Семантика: Вид определяет набор значений и операции, допустимые над этими значениями. Вид может быть **неизменяемым** видом, показывающим, что ячейка этого вида не может быть доступной для хранения этого значения. Вид имеет **новизну**, показывающую, был ли введен вид посредством оператора определения нового вида или нет.

Статические свойства: Вид обладает следующими наследственными свойствами и является:

- **неизменяемым** видом, если это явный или неявный **неизменяемый** вид;
- явным **неизменяемым** видом, если описан атрибут **READ** или это **параметризованный** массивный вид, **параметризованный** строковый вид или **параметризованный** структурный вид, где содержащиеся в нем имя **начального** массивного вида, имя **начального** строкового вида или имя **начального вариантного** структурного вида являются, соответственно, **неизменяемыми** видами.

- неявным **неизменяемым** видом, если он не является явным **неизменяемым** видом и если:

  — это вид **элемента неизменяемого** массивного вида (см. раздел 3.12.3);

  — это вид **поля неизменяемого** структурного вида или вид поля **признака параметризованного** структурного вида (см. раздел 3.12.4).

Вид обладает теми же свойствами, что и его *несоставной вид* или *составной вид*. В последующих разделах определяются свойства предопределенных имен **видов** и *видов*, не являющихся *именами вида*; свойства *имен вида* определены в разделе 3.2. **Неизменяемые** виды имеют те же свойства, что и соответствующие им не-**неизменяемые** виды, за исключением **свойства неизменяемости** (см. раздел 12.1.1.1).

Вид обладает следующими ненаследственными свойствами, к которым относятся:

- **Новизна**, которая либо **nil**, либо *определяющее вхождение* в *определении вида оператора определения нового вида*. **Новизна** вида, не являющаяся *именем вида* (ни *именем вида* **READ**), определяется следующим образом:

  — если это **параметризованный** структурный вид, **параметризованный** массивный вид или **параметризованный** структурный вид, то его **новизна** — это новизна его **начального** строкового вида, **начального** массивного вида или **начального вариантного** структурного вида соответственно;

  — если это ограниченный вид, то его **новизна** — это новизна его **родительского** вида;

  — в противном случае его **новизна** — **nil**.

  Новизна вида, являющегося *именем вида* (*именем вида READ*), определяется в разделах 3.2.2 и 3.2.3.

- **Размер**, являющийся значением, вырабатываемым *SIZE* (*&M*), где *&M* — имя виртуального **синонимического** вида, которое является **синонимическим** с *видом*.

3.4 ДИСКРЕТНЫЕ ВИДЫ

3.4.1 Общие положения

Синтаксис:

```
< дискретный вид > :: =                                   (1)
        < целый вид >                                     (1.1)
      | < булевский вид >                                 (1.2)
      | < символьный вид >                                (1.3)
      | < перечислимый вид >                              (1.4)
      | < ограниченный вид >                              (1.5)
```

Семантика: Дискретные виды определяют наборы и поднаборы упорядоченных значений.

3.4.2 Целые виды

Синтаксис:

```
< целый вид > :: =                                        (1)
        < имя целого вида >                               (1.1)
```

Предопределенные имена: Имя *INT* предопределено как имя **целого вида**.

Семантика: Целый вид определяет набор целых значений со знаком в границах, определяемых реализацией, в пределах которых определены обычное упорядочение и арифметические операции (см. раздел 5.3). В реализации могут определяться другие целые виды с различными границами (например, *LONG_INT*, *SHORT_INT*, ...), которые могут быть использованы в качестве **родительских** видов для диапазонов (см. раздел 13.2). Внутреннее представление целого значения является само целым значением.

Статические свойства: Целый вид обладает следующими наследственными свойствами, к которым относятся:

- **Верхняя граница** и **нижняя граница**, являющиеся константами, обозначающими соответственно наибольшее и наименьшее значения, определяемые целым видом. Они определяются реализацией.

- **Количество значений**, равное: **верхняя граница — нижняя граница** + 1.

Примеры:

1.5 INT *(1.1)*

3.4.3 Булевские виды

Синтаксис:

< булевский вид > :: = *(1)*
< имя булевского вида > *(1.1)*

Предопределенные имена: Имя *BOOL* является предопределенным в качестве имени **булевского вида**.

Семантика: Булевский вид определяет логические значения истинности (*TRUE* или *FALSE*) с обычными булевскими операциями (см. раздел 5.3). Внутренними представлениями для *FALSE* и *TRUE* являются целые значения 0 и 1 соответственно. Это представление определяет порядок значений.

Статические свойства: Булевский вид обладает следующими наследственными свойствами, к которым относятся:

- **Верхняя граница**, которая есть *TRUE*, и **нижняя граница**, которая есть *FALSE*.

- **Количество значений**, равное двум.

Примеры:

5.4 BOOL *(1.1)*

3.4.4 Символьные виды

Синтаксис:

< символьный вид > :: = *(1)*
< имя символьного вида > *(1.1)*

Предопределенные имена: Имя *CHAR* предопределено как имя **символьного вида**.

Семантика: Символьный вид определяет значения символов согласно набору символов языка CHILL (см. добавление A). Этот алфавит определяет порядок символов и целых значений, являющихся их внутренними представлениями.

Статические свойства: Символьный вид обладает следующими наследственными свойствами, к которым относятся:

- **Верхняя граница** и **нижняя граница**, являющиеся символьными константами, обозначающими соответственно наибольшее и наименьшее значения, определяемые именем *CHAR*.
- **Количество значений**, равное 256.

Примеры:

8.4 CHAR *(1.1)*

3.4.5 Перечислимые виды

Синтаксис:

< перечислимый вид > :: = *(1)*
SET *(< список перечисления >)* *(1.1)*
| *< имя перечислимого вида >* *(1.2)*

< список перечисления > :: = *(2)*
< нумеруемый список перечисления > *(2.1)*
| *< ненумеруемый список перечисления >* *(2.2)*

< нумеруемый список перечисления > :: = *(3)*
< нумеруемый элемент перечисления > { , *< нумеруемый элемент перечисления >* }* *(3.1)*

< нумеруемый элемент перечисления > :: = *(4)*
< определяющее вхождение > = < целое постоянное выражение > *(4.1)*

< ненумеруемый список перечисления > :: = *(5)*
< элемент перечисления > { , *< элемент перечисления >* }* *(5.1)*

< элемент перечисления > :: = *(6)*
< определяющее вхождение > *(6.1)*

Семантика: Перечислимый вид определяет набор именованных и неименованных значений. Именованные значения обозначаются именами, определяемыми *определяющими вхождениями* в *списке перечисления*; неименованные значения — это все прочие значения. Внутренним представлением именованных значений является связанное с ними целое значение. Это представление определяет порядок значений.

Статические свойства: *Определяющее вхождение* в *списке перечисления* определяет имя **элемента перечисления**. Имя **элемента перечисления** имеет приписанный **перечислимый** вид, являющийся данным перечислимым видом.

Перечислимый вид имеет следующие наследственные свойства:

- Он имеет набор имен **элементов перечисления**, являющийся набором имен, устанавливаемых *определяющими вхождениями* в его *списке перечисления*.
- Каждое имя **элемента перечисления** перечислимого вида имеет соответствующее значение внутреннего представления, которое в случае *нумеруемого элемента перечисления* является значением, вырабатываемым находящимся в этом элементе *целым постоянным выражением*; в противном случае это одно из значений 0, 1, 2 и т.д. согласно его позиции в *ненумеруемом списке перечисления*. Например, в **SET** (*a*,*b*), *a* имеет значение представления 0, *b* — значение представления 1.
- Это **верхняя граница** и **нижняя граница**, являющиеся его именами элементов перечисления соответственно с наибольшим и наименьшим значениями представления.
- **Количество значений** равно наибольшему из значений, соответствующих именам **элементов перечисления**, плюс единица.
- Это **нумеруемый** перечислимый вид, если его *список перечисления* является *нумеруемым списком перечисления*; в противном случае это **ненумеруемый** перечислимый вид.

Статические условия: Для каждой пары *целых постоянных выражений* e_1, e_2 в *списке перечисления NUM* (e_1) и *NUM* (e_2) должны выдавать различные неотрицательные результаты.

Примеры:

11.7	**SET** *(occupied, free)*	*(1.1)*
6.3	*month*	*(1.2)*

3.4.6 Ограниченные виды

Синтаксис:

< ограниченный вид > :: =	*(1)*
< имя дискретного вида > (< постоянный диапазон >)	*(1.1)*
RANGE *(< постоянный диапазон >)*	*(1.2)*
BIN *(< целое постоянное выражение >)*	*(1.3)*
< имя ограниченного вида >	*(1.4)*
< постоянный диапазон > :: =	*(2)*
<нижняя граница > : < верхняя граница >	*(2.1)*
< нижняя граница > :: =	*(3)*
< дискретное постоянное выражение >	*(3.1)*
< верхняя граница > :: =	*(4)*
< дискретное постоянное выражение >	*(4.1)*

Производный синтаксис: Запись **BIN** (*n*) является производной от *INT* (*0 : 2^n — 1*), например, **BIN** (*2 + 1*) соответствует *INT* (*0 : 7*).

Семантика: Ограниченный вид определяет набор диапазона значений между границами, специфицированными (включенными) в *постоянный диапазон*. Диапазон состоит из специального **родительского** вида, который определяет операции и порядок значений диапазона.

Статические свойства: Ограниченный вид, имеет следующее ненаследственное свойство: **родительский** вид, определяемый следующим образом:

- Если ограниченный вид представлен в форме:

  < имя дискретного вида > (< постоянный диапазон >),

  тогда **родительский** вид — это *имя дискретного вида*, если *имя дискретного вида* не является ограниченным видом; в противном случае это **родительский** вид *имени дискретного вида*.

- Если ограниченный вид представлен в форме:

  RANGE *(< постоянный диапазон >),*

  тогда **родительский** вид является **корневым** видом **результирующего класса** из классов *верхней границы* и *нижней границы* в *постоянном диапазоне*.

- Если ограниченный вид — это *имя ограниченного вида*, являющееся именем **синонимического** вида, тогда его **родительский** вид является родительским видом **определяющего** вида имени **синонимического** вида; в противном случае это имя **нового вида**, и тогда его **родительский** вид — это виртуально введенный **родительский** вид (см. раздел 3.2.3).

Ограниченный вид имеет следующие наследственные свойства, к которым относятся:

- **Верхняя граница** и **нижняя граница**, которые являются константами, обозначающими значения, вырабатываемые *верхней границей* и *нижней границей*, соответственно, в *постоянном диапазоне*.

- **Количество значений**, равное значению, получаемому из *NUM* (*U*)– *NUM* (*L*) + 1, где *U* и *L* обозначают, соответственно, **верхнюю границу** и **нижнюю границу** ограниченного вида.

- **Нумеруемый** ограниченный вид, если его **родительский** вид является **нумеруемым** перечислимым видом.

Статические условия: Классы *верхней границы* и *нижней границы* должны быть **совместимыми** и оба класса должны быть **совместимыми** с *именем дискретного вида*, если он описан.

Нижняя граница должна вырабатывать значение, которое меньше или равно значению, вырабатываемому *верхней границей*, и оба значения должны принадлежать набору значений, определяемых *именем дискретного вида*, если оно описано.

Целое постоянное выражение в случае **BIN** должно вырабатывать неотрицательное значение.

Примеры:

9.5	*INT (2:max)*	*(1.1)*
11.12	*line*	*(1.4)*

3.5 МНОЖЕСТВЕННЫЕ ВИДЫ

Синтаксис:

< множественный вид > :: =	*(1)*
POWERSET *< элементный вид >*	*(1.1)*
| *< имя множественного вида >*	*(1.2)*
< элементный вид > :: =	*(2)*
< дискретный вид >	*(2.1)*

Семантика: Множественный вид определяет значения, являющиеся наборами значений его элементного вида. Множественные значения охватывают все поднаборы элементного вида. Над множественными значениями определены обычные теоретико-множественные операции (см. раздел 5.3).

Статические свойства: Множественный вид обладает следующим наследственным свойством:

- Это **элементный** вид, являющийся данным *элементным видом*.

Примеры:

8.4	**POWERSET** *CHAR*	*(1.1)*
9.5	**POWERSET** *INT (2:max)*	*(1.1)*
9.6	*number_list*	*(1.2)*

3.6 ССЫЛОЧНЫЕ ВИДЫ

3.6.1 Общие положения

Синтаксис:

< ссылочный вид > :: =	*(1)*
< закрепленный ссылочный вид >	*(1.1)*
| *< свободный ссылочный вид >*	*(1.2)*
| *< рядовый вид >*	*(1.3)*

Семантика: Ссылочный вид определяет ссылки (адреса или дескрипторы) на **ссылочные** ячейки. Согласно определению закрепленные ссылки касаются ячеек данного статического вида; свободные ссылки могут касаться ячеек любого статического вида; ряды относятся к ячейкам динамического вида.

Операция разыменования (снятия ссылки) определена над ссылочными значениями (см. разделы 4.2.3, 4.2.4 и 4.2.5), вырабатывающими ссылаемую ячейку.

Два ссылочных значения равны тогда и только тогда, когда они ссылаются на одну и ту же ячейку, либо, когда оба значения не ссылаются на ячейку (то есть они оба — значение *NULL*).

3.6.2 Закрепленные ссылочные виды

Синтаксис:

< закрепленный ссылочный вид > :: = *(1)*
REF *< ссылаемый вид >* *(1.1)*
| *< имя закрепленного ссылочного вида >* *(1.2)*

< ссылаемый вид > :: = *(2)*
< вид > *(2.1)*

Семантика: Закрепленный ссылочный вид определяет ссылочные значения на ячейки определенного ссылаемого вида.

Статические свойства: Закрепленный ссылочный вид обладает следующим наследственным свойством:

- **Ссылаемый** вид является данным *ссылаемым видом*.

Примеры:

10.42 **REF** *cell* *(1.1)*

3.6.3 Свободные ссылочные виды

Синтаксис:

< свободный ссылочный вид > :: = *(1)*
< имя свободного ссылочного вида > *(1.1)*

Предопределенные имена: Имя *PTR* предопределено как имя **свободного ссылочного вида**.

Семантика: Свободный ссылочный вид определяет ссылочные значения на ячейки любого статического вида.

Примеры:

19.8 *PTR* *(1.1)*

3.6.4 Рядовые виды

Синтаксис:

< рядовый вид > :: = *(1)*
ROW *< строковый вид >* *(1.1)*
| **ROW** *< массивный вид >* *(1.2)*
| **ROW** *< вариантный структурный вид >* *(1.3)*
| *< имя рядового вида >* *(1.4)*

Семантика: Рядовый вид определяет ссылочные значения на ячейки динамического вида (ячейки некоторого параметризованного вида со статически неизвестными параметрами).

Рядовое значение может отсылать к:

- ячейке строк со статически неизвестной **длиной строки**;
- ячейке массива со статически неизвестной **верхней границей**;
- ячейке параметризованной структуры со статически неизвестными параметрами.

Статические свойства: Рядовый вид обладает следующим наследственным свойством:

- Это **ссылаемый начальный** вид, являющийся *строковым видом*, *массивным видом* или *вариантным структурным видом* соответственно.

Статические условия: *Вариантный структурный вид* должен быть **параметризованным**.

Примеры:

8.6 **ROW CHARS** *(max)* *(1.1)*

3.7 ПРОЦЕДУРНЫЕ ВИДЫ

Синтаксис:

< процедурный вид > :: = *(1)*
PROC *(* [*< список параметров >*] *)* [*< спецификация результата >*]
[**EXCEPTIONS** *(< список исключений >)*] [**RECURSIVE**] *(1.1)*
| *< имя процедурного вида >* *(1.2)*

< список параметров > :: = *(2)*
< спецификация параметра > { , *< спецификация параметра >* }* *(2.1)*

< спецификация параметра > :: = *(3)*
< вид > [*< атрибут параметра >*] *(3.1)*

< атрибут параметра > :: = *(4)*
IN | **OUT** | **INOUT** | **LOC** [**DYNAMIC**] *(4.1)*

< спецификация результата > :: = *(5)*
RETURNS *(< вид >* [*< атрибут результата >*]*)* *(5.1)*

< атрибут результата > :: = *(6)*
[**NONREF**] **LOC** [**DYNAMIC**] *(6.1)*

< список исключений > :: = *(7)*
< имя исключения > { , *< имя исключения >* }* *(7.1)*

Семантика: Процедурный вид определяет (**общие**) значения процедур, то есть объекты, обозначаемые именами **общих процедур**, являющихся именами, определяемыми в операторах определения процедур. Значения процедур указывают на блоки программ в динамическом контексте. Процедурные виды позволяют оперировать с процедурой динамически, например, передавать ее в качестве параметра в другие процедуры, посылать ее как значение сообщения в буфер, заносить для хранения в ячейку и т.д.

Значения процедур можно вызывать (см. раздел 6.7).

Два значения процедуры равны тогда и только тогда, когда они обозначают одну и ту же процедуру в одном и том же динамическом контексте, или оба значения не обозначают никакую процедуру (то есть они оба — значения *NULL*).

Статические свойства: Процедурный вид обладает следующими наследственными свойствами:

- Он имеет список **спецификаций параметров**, при этом каждая спецификация содержит вид и, возможно, атрибут параметра. **Спецификации параметров** определяются *списком параметров*.
- Он имеет необязательную **спецификацию результата**, состоящую из вида и необязательного атрибута результата. **Спецификация результата** определяется синтаксической категорией *спецификация результата*.
- Он содержит, возможно, пустой список имен **исключений**, упоминаемых в *списке исключений*.
- Он обладает **рекурсивностью**, то есть он **рекурсивен**, если описан атрибут **RECURSIVE**; в противном случае **рекурсивность** или **нерекурсивность** определяется по умолчанию реализацией.

Статические условия: Все имена из *списка исключений* должны быть различными.

Только в том случае, если атрибут **LOC** описан в *спецификации параметра* или *спецификации результата*, содержащийся там *вид* обладает **свойством беззначимости**.

Если атрибут **DYNAMIC** описан в *спецификации параметра* или *спецификации результата*, то содержащийся там *вид* должен быть **параметризованным**.

3.8 ЭКЗЕМПЛЯРНЫЕ ВИДЫ

Синтаксис:

< экземплярный вид > :: = *(1)*
< имя экземплярного вида > *(1.1)*

Предопределенные имена: Имя *INSTANCE* предопределено в качестве имени **экземплярного вида**.

Семантика: Экземплярный вид определяет значения, идентифицирующие процессы. В результате создания процесса (см. разделы 5.2.14, 6.13 и 11.1) в качестве идентификации созданного процесса вырабатывается единственное значение экземпляра.

Два значения экземпляра равны тогда и только тогда, когда они идентифицируют один и тот же процесс, либо оба значения не идентифицируют никакой процесс (то есть они оба — значения *NULL*).

Примеры:

15.39 INSTANCE *(1.1)*

3.9 СИНХРОНИЗАЦИОННЫЕ ВИДЫ

3.9.1 Общие положения

Синтаксис:

< синхронизационный вид > :: = *(1)*
< событийный вид > *(1.1)*
| *< буферный вид >* *(1.2)*

Семантика: Синхронизационный вид является средством синхронизации и коммуникации между процессами (см. главу 11). Язык CHILL не предусматривает выражений для обозначения значения, определяемого синхронизационным видом. Следовательно, над этими значениями не определяются никакие операции.

3.9.2 Событийные виды

Синтаксис:

< событийный вид > :: = *(1)*
EVENT [*(< длина события >)*] *(1.1)*
| *< имя событийного вида >* *(1.2)*

< длина события > :: = *(2)*
< целое постоянное выражение > *(2.1)*

Семантика: Ячейка событийного вида является средством обеспечения синхронизации между процессами. Над ячейками событийного вида определены оператор продолжения, оператор задержки и оператор выбора задержки, которые описаны в разделах 6.15, 6.16 и 6.17 соответственно.

Длина события описывает максимальное количество процессов, которые могут стать задержанными по ячейке события; это количество может быть неограниченным при отсутствии спецификации *длины события*.

Статические свойства: Событийный вид обладает следующим наследственным свойством:

- Он имеет необязательную **длину события**, которая является значением, вырабатываемым *длиной события*.

Статические условия: *Длина события* должна вырабатывать положительное значение.

Примеры:

14.10 **EVENT** *(1.1)*

3.9.3 Буферные виды

Синтаксис:

< буферный вид > :: = *(1)*
BUFFER [*(* *< длина буфера >* *)*] *< буферный элементный вид >* *(1.1)*
| *< имя буферного вида >* *(1.2)*

< длина буфера > :: = *(2)*
< целое постоянное выражение > *(2.1)*

< буферный элементный вид > :: = *(3)*
< вид > *(3.1)*

Семантика: Ячейка буферного вида является средством синхронизации и коммуникации между процессами. Над буферными ячейками определены оператор посылки, оператор варианта получения и выражение получения, описанные в разделах 6.18, 6.19 и 5.3.9 соответственно.

Длина буфера описывает максимальное количество значений, которые можно хранить в событийной ячейке; это количество может быть неограниченным при отсутствии спецификации *длины буфера*.

Статические свойства: Буферный вид обладает следующими наследственными свойствами и имеет:

- необязательную **длину буфера**, являющуюся значением, вырабатываемым *длиной буфера*;
- **буферный элементный** вид, являющийся *буферным элементным видом*.

Статические условия: *Длина буфера* должна вырабатывать неотрицательное значение.

Буферный элементный вид не должен обладать **свойством беззначимости.**

Примеры:

16.30 **BUFFER** *(1) user_messages* *(1.1)*
16.34 *user_buffers* *(1.2)*

3.10 ОБМЕННЫЕ ВИДЫ

3.10.1 Общие положения

Синтаксис:

< обменный вид > :: = *(1)*
< ассоциативный вид > *(1.1)*
| *< доступный вид >* *(1.2)*
| *< текстовый вид >* *(1.3)*

Семантика: Обменный вид обеспечивает средства для операций ввода—вывода, как описано в главе 7. В языке CHILL нет выражений, обозначающих значения, определяемые обменным видом. Следовательно, над этими значениями не определяются никакие операции.

Примеры:

20.17 *ASSOCIATION* *(1.1)*

3.10.2 Ассоциативные виды

Синтаксис:

< ассоциативный вид > :: = *(1)*
< имя ассоциативного вида > *(1.1)*

Предопределенные имена: Имя *ASSOCIATION* предопределено в качестве имени **ассоциативного вида.**

Семантика: Ячейка ассоциативного вида обеспечивает средства представления отношения с объектом внешней среды. Такое отношение в языке CHILL называется ассоциацией; ассоциации могут создаваться встроенной программой *ASSOCIATE* и прекращаться встроенной программой *DISSOCIATE*.

3.10.3 Доступные виды

Синтаксис:

< доступный вид > :: = *(1)*
ACCESS [(*< индексный вид >*)] [*< вид записи >* [**DYNAMIC**]] *(1.1)*
| *< имя доступного вида >* *(1.2)*

< вид записи > :: = *(2)*
< вид > *(2.1)*

< индексный вид > :: = *(3)*
< дискретный вид > *(3.1)*
| *< постоянный диапазон >* *(3.2)*

Производный синтаксис: *Постоянный диапазон* в представлении индексного вида является производным от дискретного вида **RANGE** (*постоянный диапазон*).

Семантика: Ячейка доступного вида обеспечивает средства расположения файла и переноса значений из CHILL-программы в файл во внешней среде и наоборот.

Доступный вид может определять *вид записи*; этот вид записи определяет **корневой** вид класса значений, которые могут переноситься через ячейку данного доступного вида в файл или из файла. Вид переносимого значения может быть динамическим, то есть **размер** записи может изменяться, если в представлении доступного вида описан атрибут **DYNAMIC** или если *вид записи* — это **изменяющийся** строковый вид. В последнем случае описывать атрибут **DYNAMIC** не требуется.

С помощью доступного вида может также быть определен *индексный вид*; такой индексный вид определяет размер "окна" для (части) файла, из которого произвольным образом можно читать записи или записывать их. Такое окно может быть размещено в (индексируемом) файле с помощью операции соединения. При отсутствии спецификации *индексного вида* можно переносить записи только последовательно.

Статические свойства: Доступный вид обладает следующими наследственными свойствами и имеет:

- необязательный вид **записи**, являющийся данным *видом записи*, если таковая существует. Это **динамический** вид **записи**,если описан атрибут **DYNAMIC** или если *вид записи* — это **изменяющийся** строковый вид; в противном случае это **статический** вид **записи**;
- необязательный **индексный** вид, являющийся данным *индексным видом*.

Статические условия: Необязательный *вид записи* не должен обладать **свойством беззначимости**.

Если описан атрибут **DYNAMIC**, то вид **записи** должен быть **параметризованным** и не должен быть **беспризнаковым** структурным видом.

Индексный вид не должен быть ни **нумеруемым** перечислимым видом, ни **нумеруемым** ограниченным видом.

Примеры:

20.18	**ACCESS** *(index_set) record_type*	*(1.1)*
22.20	**ACCESS** *string* **DYNAMIC**	*(1.1)*
20.18	*record_type*	*(2.1)*
20.18	*index_set*	*(3.1)*

3.10.4 Текстовые виды

Синтаксис:

< *текстовый вид* > :: = *(1)*
TEXT (< *длина текста* >) [< *индексный вид* >] [**DYNAMIC**] *(1.1)*

< *длина текста* > :: = *(2)*
< *целое постоянное выражение* > *(2.1)*

Семантика: Ячейка текстового вида обеспечивает средства переноса значений, представленных в удобочитаемом виде, из CHILL-программы в файл во внешней среде и наоборот. Ячейка текстового вида имеет **запись текста** и субъячейки **доступа**. Субъячейка **записи текста** инициализируется пустой строкой.

Текстовый вид имеет **длину текста**, определяющую максимальную длину записей, которые можно перенести, и, возможно, **индексный** вид, имеющий тот же смысл, что и доступные виды.

Статические свойства: Текстовый вид обладает следующими наследственными свойствами и имеет:

- **длину текста**, являющуюся значением, выработанным *длиной текста*;
- вид **записи текста**, который есть **CHARS** (< *длина текста* >) **VARYING**;
- **доступный** вид, который есть **ACCESS** [(< *индексный вид* >)] **CHARS** (< *длина текста* >) [**DYNAMIC**] (< *индексный вид* > и **DYNAMIC** являются частью вида, только в том случае, если они описаны).

Примеры:

26.8 **TEXT** *(80)* **DYNAMIC** *(1.1)*

3.11 ВРЕМЕННЫЕ ВИДЫ

3.11.1 Общие положения

Синтаксис:

< *временной вид* > :: = *(1)*
< *продолжительный вид* > *(1.1)*
| < *абсолютно-временной вид* > *(1.2)*

Семантика: Временной вид обеспечивает средства контроля времени процессов, как описано в главе 9. Временные значения создаются набором встроенных программ. Над временными значениями определены операции сравнения.

3.11.2 Продолжительные виды

Синтаксис:

< *продолжительный вид* > :: = *(1)*
< *имя продолжительного вида* > *(1.1)*

Предопределенные имена: Имя *DURATION* предопределено в качестве имени **продолжительного вида**.

Семантика: Продолжительный вид определяет значения, представляющие периоды времени. Набор значений, определяемых продолжительным видом, зависит от реализации. Значения продолжительности могут быть представлены в реализации как пары: точность и значение. Значения продолжительности упорядочены интуитивно.

3.11.3 Абсолютно-временные виды

Синтаксис:

< *абсолютно-временной вид* > :: = *(1)*
< *имя абсолютно-временного вида* > *(1.1)*

Предопределенные имена: Имя *TIME* предопределено как имя **абсолютно-временного вида**.

Семантика: Абсолютно-временной вид определяет значения, представляющие моменты времени. Набор значений, определяемых абсолютно-временным видом, зависит от реализации. Абсолютно-временные значения упорядочены интуитивно.

3.12 СОСТАВНЫЕ ВИДЫ

3.12.1 Общие положения

Синтаксис:

< составной вид > :: = *(1)*
< строковый вид > *(1.1)*
| *< массивный вид >* *(1.2)*
| *< структурный вид >* *(1.3)*

Семантика: Составной вид определяет составные значения, то есть значения, состоящие из субкомпонент, которые могут быть получены или могут быть доступны (см. разделы 4.2.6—4.2.10 и 5.2.6—5.2.10).

3.12.2 Строковые виды

Синтаксис:

< строковый вид > :: = *(1)*
< тип строки > (< длина строки >) **[VARYING]** *(1.1)*
| *< параметризованный строковый вид >* *(1.2)*
| *< имя строкового вида >* *(1.3)*

< параметризованный строковый вид > :: = *(2)*
< имя начального строкового вида > (< длина строки >) *(2.1)*
| *< имя параметризованного строкового вида >* *(2.2)*

< имя начального строкового вида > :: = *(3)*
< имя строкового вида > *(3.1)*

< тип строки > :: = *(4)*
BOOLS *(4.1)*
| **CHARS** *(4.2)*

< длина строки > :: = *(5)*
< целое постоянное выражение > *(5.1)*

Семантика: **Фиксированный** строковый вид определяет значения длины битовой или символьной строки, указанных или подразумеваемых строковым видом. **Изменяющийся** строковый вид определяет значения длины битовой или символьной строки, **фактическая длина** которых может меняться динамически от нуля до **длины строки**. Длина известна только во время выполнения программы из значения атрибута **фактическая длина**. Для **фиксированного** строкового вида **фактическая длина** всегда равна **длине строки**. Символьные строки — это последовательности значений символов; битовые строки — это последовательности булевских значений.

Значения строки всегда либо пусты, либо имеют элементы строки, нумеруемые от 0 и дальше.

Значения строк данного строкового вида хорошо упорядочены в соответствии с порядком значений компонент и согласно следующему определению.

Две строки s и t равны тогда и только тогда, когда они пусты или имеют одну и ту же длину l и $s(i) = t(i)$ для всех $0 < i < l$. Строка s предшествует строке t, когда:

- существует такой индекс j, что $s(j) < t(j)$ и $s(0 : j - 1) = t(0 : j - 1)$, либо
- *LENGTH(s) < LENGTH (t)* и *s = t (0* **UP** *LENGTH(s))*.

Операция конкатенации определена над значениями строки. Обычные логические операции определены над значениями битовых строк и осуществляются между их соответствующими элементами (см. раздел 5.3).

Статические свойства: Строковый вид обладает следующими наследственными свойствами:

- Он имеет **длину строки**, равную значению, вырабатываемому *длиной строки*.
- Он имеет **верхнюю границу** и **нижнюю границу**, являющиеся значениями, вырабатываемыми **длиной строки**, минус 1 и 0, соответственно.
- Он является **битовым** строковым видом или **символьным** строковым видом в зависимости от того, описывает ли *тип строки* **BOOLS** или **CHARS** или, является ли *имя начального строкового вида* **битовым** строковым видом или **символьным** строковым видом.
- Он является **изменяющимся** строковым видом, если описан атрибут **VARYING** или если *имя начального строкового вида* является **изменяющимся** строковым видом; в противном случае это **фиксированный** строковый вид.

Строковый вид является **параметризованным** тогда и только тогда, когда он является *параметризованным строковым видом*.

Параметризованный строковый вид имеет **начальный** строковый вид, являющийся видом, обозначаемым *именем начального строкового вида*.

Изменяющийся строковый вид обладает следующим ненаследственным свойством: он имеет **компонентный** вид, определяемый следующим образом:

- Если **изменяющийся** строковый вид имеет форму:

  < тип строки > (< длина строки >) **VARYING**,

  тогда это *< тип строки > (< длина строки >)*;
- Если **изменяющийся** строковый вид имеет форму:

  < имя начального строкового вида > (< длина строки >),

  тогда **компонентный** вид есть *&имя* (*длина строки*), где *&имя* — это виртуально введенное имя **синонимического вида**, **синонимического** с **компонентным** видом *имени начального строкового вида*.
- Если **изменяющийся** строковый вид есть *имя строкового вида*, являющееся именем **синонимического вида**, тогда его **компонентный** вид является компонентным видом определяющего вида имени **синонимического вида**; в противном случае он является именем **нового вида** и тогда его **компонентный** вид есть виртуально введенный **компонентный** вид (см. раздел 3.2.3).

Статические условия: *Длина строки* должна вырабатывать неотрицательное значение.

Значение, вырабатываемое *длиной строки*, непосредственно содержащейся в *параметризованном строковом виде*, должно быть меньше или равно **длине строки** *имени начального строкового вида*. Это условие применимо только к **параметризованным** строковым видам, не введенным виртуально.

Примеры:

7.51 **CHARS** *(20)* *(1.1)*
22.22 **CHARS** *(20)* **VARYING** *(1.1)*

3.12.3 Массивные виды

Синтаксис:

< массивный вид > :: = *(1)*
ARRAY *(< индексный вид > { , < индексный вид > }*)*
*< вид элемента > { < формат элемента > }** *(1.1)*
| *< параметризованный массивный вид >* *(1.2)*
| *< имя массивного вида >* *(1.3)*

< параметризованный массивный вид > :: = *(2)*
< имя начального массивного вида > (< верхний индекс >) *(2.1)*
| *< имя параметризованного массивного вида >* *(2.2)*

< имя начального массивного вида > :: = *(3)*
< имя массивного вида > *(3.1)*

< верхний индекс > :: = *(4)*
< дискретное постоянное выражение > *(4.1)*

< вид элемента > :: = *(5)*
< вид > *(5.1)*

Производный синтаксис: *Массивный вид* с несколькими индексными видами (обозначающими многомерный массив) является производным синтаксисом *массивного вида* с *видом элемента*, являющимся *массивным видом*. Например:

ARRAY *(1:20,1:10) INT*

есть производное от

ARRAY *(***RANGE** *(1:20)) ***ARRAY** *(***RANGE** *(1:10)) INT.*

Только при использовании данного производного синтаксиса допустимо несколько вхождений *формата элемента*. Количество вхождений *формата элемента* должно быть меньше или равно количеству вхождений *индексного вида*. В этом случае самый левый *формат элемента* связан с самым дальним внутренним *видом элемента* и т. д.

Семантика: Массивный вид определяет составные значения, являющиеся списками значений, определяемых его видом элементов. Физический формат ячейки массива или значения массива может контролироваться спецификацией *формата элемента* (см. раздел 3.12.5). Два значения массива равны тогда и только тогда, когда все соответствующие значения элементов равны.

Статические свойства: Массивный вид обладает следующими наследственными свойствами и имеет:

- **индексный** вид, являющийся *индексным видом*, если это не *параметризованный массивный вид*; в противном случае **индексный** вид — это ограниченный вид, образованный следующим образом:

  &имя (*нижняя граница* : *верхняя граница*),

  где *&имя* — имя виртуального **синонимического** вида, **синонимического** с **индексным** видом *имени начального массивного вида*; *нижняя граница* — это нижняя граница **индексного** вида *имени начального массивного вида*, а *верхняя граница* — это *верхний индекс*;

- **верхнюю границу** и **нижнюю границу**, которые являются, соответственно, **верхней границей** и **нижней границей** его **индексного** вида;

- вид **элемента**, являющийся либо *M*, либо **READ** *M*, где *M* — *вид элемента* или вид **элемента** *имени начального массивного вида* соответственно. Вид **элемента** — **READ** *M* тогда и только тогда, когда *M* не является **неизменяемым** видом, а *массивный вид* — **неизменяемый** вид. Вид **элемента** является неявным **неизменяемым** видом, если он есть **READ** *M*;

- **формат элемента**, который есть **формат элемента** его имени *начального массивного вида*, если массивный вид — это *параметризованный массивный вид*; в противном случае это определенный *формат элемента* или же задается по умолчанию реализацией, то есть **PACK** или **NOPACK**;

- **количество элементов**, равное значению, определяемому в виде:

  NUM (*верхняя граница*) − *NUM* (*нижняя граница*) + 1,

  где *верхняя граница* и *нижняя граница* — соответственно **верхняя граница** и **нижняя граница** его **индексного** вида.

- **отображенный** вид, если *формат элемента* описан и является *шагом*.

Массивный вид является **параметризованным** тогда и только тогда, когда он является *параметризованным массивным видом*.

Параметризованный массивный вид имеет **начальный** массивный вид, являющийся видом, обозначаемым *именем начального массивного вида*.

Статические условия: *Класс верхнего индекса* должен быть **совместимым** с **индексным** видом *имени начального массивного вида*, а вырабатываемое им значение должно лежать в диапазоне, определяемым этим **индексным** видом.

Примеры:

5.29	**ARRAY** *(1:16)* **STRUCT** *(c4, c2, c1 BOOL)*	*(1.1)*
11.12	**ARRAY** *(line)* **ARRAY** *(column) square*	*(1.1)*
11.17	*board*	*(1.3)*

3.12.4 Структурные виды

Синтаксис:

< структурный вид > :: = (1)
STRUCT *(< поле > { , < поле >}*)* (1.1)
| *< параметризованный структурный вид >* (1.2)
| *< имя структурного вида >* (1.3)

< поле > :: = (2)
< фиксированное поле > (2.1)
| *< альтернативное поле >* (2.2)

< фиксированное поле > :: = (3)
< список определяющих вхождений имен полей > < вид > [< формат поля >] (3.1)

< альтернативное поле > :: = (4)
CASE [*< список признаков >*] **OF**
< вариантная альтернатива > { , *< вариантная альтернатива >*}*
[ELSE [*< вариантное поле >* { , *< вариантное поле >* }*]] **ESAC** (4.1)

< вариантная альтернатива > :: = (5)
[*< спецификация метки выбора >*] : [*< вариантное поле >*
{ ,*< вариантное поле >* }*] (5.1)

< список признаков > :: = (6)
< имя поля признака > { , *< имя поля признака >* }* (6.1)

< вариантное поле > :: = (7)
< список определяющих вхождений имен поля >
< вид > [*< формат поля >*] (7.1)

< параметризованный структурный вид > :: = (8)
< имя начального вариантного структурного вида >
(< список постоянных выражений >) (8.1)
| *< имя параметризованного структурного вида >* (8.2)

< имя начального вариантного структурного вида > :: = (9)
< имя вариантного структурного вида > (9.1)

< список постоянных выражений > :: = (10)
< дискретное постоянное выражение > { , *< дискретное постоянное выражение >*}* (10.1)

Производный синтаксис: Вхождение *фиксированного поля* или *вариантного поля*, где список *определяющих вхождений имен полей* состоит из нескольких *определяющих вхождений имен полей*, является производным синтаксисом для нескольких вхождений *фиксированного поля* или *вариантного поля* с одним *определяющим вхождением имени поля* соответственно, с определенным *видом* и необязательным *форматом поля* для каждого. В случае *формата поля* этот *формат поля* не должен быть *позицией*. Например:

STRUCT (*I,J BOOL* **PACK**)

является производным от

STRUCT (*I BOOL* **PACK**, *J BOOL* **PACK**).

Семантика: Структурные виды определяют составные значения, состоящие из списка значений, выбираемых по имени компоненты. Каждое значение определяется видом, приписанным имени компоненты. Значения структуры могут располагаться в (составных) ячейках структуры, где имя компоненты служит доступом к субъячейке. Компоненты значения структуры или ячейки называются полями, а их имена — именами **полей**.

Имеются **фиксированные, вариантные** и **параметризованные** структуры.

Фиксированные структуры состоят только из фиксированных полей, то есть полей, которые всегда представлены и которые могут быть доступными без каких-либо динамических проверок.

Вариантные структуры имеют вариантные поля, то есть поля, которые не всегда представлены. Для **теговых вариантных** структур наличие этих полей известно только во время выполнения программы из значения(ий) определенных связанных фиксированных полей, называемых полями **признаков**. **Беспризнаковые вариантные** структуры не имеют полей **признаков**. Поскольку состав **вариантной** структуры может меняться во время выполнения программы, **размер** ячейки вариантной структуры основан на большом выборе (наихудший случай) вариантных альтернатив.

Выбранная в *альтернативном поле вариантная альтернатива* — это та альтернатива, для которой совпадают значения в спецификации метки выбора; если значения не совпадают, то выбирается *вариантная альтернатива*, следующая за ELSE (которая имеется).

Параметризованная структура определяется из **вариантного** структурного вида, для которого выбор вариантных альтернатив статически определен с помощью постоянных выражений. Состав структуры фиксируется с момента создания параметризованной структуры и может не меняться в течение времени выполнения программы. Поля **признаков**, если имеются, являются **неизменяемыми** и автоматически инициализируются описанными значениями. Для ячейки параметризованной структуры может быть выделен точный объем памяти в месте описания или генерации. Заметим, что динамические **параметризованные** структурные виды также существуют; их семантика определена в разделе 3.13.4.

Формат ячейки структуры или значения структуры контролируется спецификацией формата поля (см. раздел 3.12.5).

Два значения структуры равны тогда и только тогда, когда соответствующие значения компонент равны. Однако, если значения структуры — это значения **беспризнаковой вариантной** структуры, то результат сравнения определяется реализацией.

Статические свойства:

общие положения

Структурный вид имеет следующие наследственные свойства:

- Это **фиксированный** структурный вид, если его *структурный вид* непосредственно не содержит вхождение *альтернативного поля*.
- Это **вариантный** структурный вид, если его *структурный вид* содержит по крайней мере одно вхождение *альтернативного поля*.
- Это **параметризованный** структурный вид, если он — *параметризованный структурный вид*.

  Он имеет набор имен **полей**. Этот набор определен для нескольких случаев ниже. Считается, что имя — это имя **поля** тогда и только тогда, когда оно определено в *списке определяющих вхождений имен полей* в *фиксированных полях* или *вариантных полях структурного вида*.

  Каждое *фиксированное поле*, *вариантное поле* и, следовательно, каждое имя **поля** структурного вида имеют вид **поля**, приписанный им, которое либо *M*, либо **READ** *M*, где M — *вид* в *фиксированном поле* или *вариантном поле*. Вид **поля** есть **READ** *M*, если *M* не является **неизменяемым** видом, и либо структурный вид — **неизменяемый вид**, либо поле — это поле **признака парамеризованного** структурного вида. Вид **поля** — неявный **неизменяемый** вид, если он — **READ** *M*.

  Фиксированное поле, *вариантное поле* и, следовательно, имя **поля** данного структурного вида имеют приписанный им **формат поля**, являющийся *форматом поля* в *фиксированном поле* или *вариантном поле*, если имеются; в противном случае — это формат поля, принимаемый по умолчанию, являющийся либо **PACK**, либо **NOPACK**.

- Это **отображенный** вид, если его имена **полей** имеют *формат поля*, являющийся *позицией*.

фиксированные структуры

Фиксированный структурный вид обладает следующим наследственным свойством:

- Он имеет набор имен **полей**, являющийся набором имен, определяемых любым *списком определяющих вхождений имен полей* в *фиксированных полях*. Эти имена **полей** являются именами **фиксированных полей**.

вариантные структуры

Вариантный структурный вид обладает следующими наследственными свойствами:

- Он имеет набор имен **полей**, являющийся объединением набора имен, определяемых любым *списком определяющих вхождений имен полей* в *фиксированных полях*, и набора имен, определяемых любым *списком определяющих вхождений имен полей* в *альтернативных полях*. Имена **полей**, определяемые списком *определяющих вхождений имен полей* в *фиксированных полях* — это имена **фиксированных полей вариантного** структурного вида; его другие имена **полей** — это имена **вариантных полей**.

  Имя **поля вариантного** структурного вида — это имя **поля признака** тогда и только тогда, когда оно встречается в любом *списке признаков альтернативного поля*. *Альтернативные поля*, в которых не описан *список признаков*, являются **беспризнаковыми** *альтернативными полями*.

- **Вариантный** структурный вид является **беспризнаковым вариантным** структурным видом если все его вхождения *альтернативных полей* являются **беспризнаковыми**; в противном случае это **теговый вариантный** структурный вид.
- **Вариантный** структурный вид — это **параметризованный вариантный структурный** вид, если он — либо **теговый вариантный** структурный вид, либо **беспризнаковый вариантный** структурный вид, где для каждого из вхождений *альтернативных полей* приведена *спецификация метки выбора* для всех вхождений в него *вариантной альтернативы*.
- **Параметризованный вариантный** структурный вид имеет список приписанных ему классов, определяемых следующим образом:
  - если это **теговый вариантный** структурный вид, то этот список классов — список классов M_i-значений, где M_i — виды имен **полей признаков** в том порядке, как они определены в *фиксированных полях*;
  - если это **беспризнаковый вариантный** структурный вид, то список образуется из отдельных **результирующих списков классов** каждого *альтернативного поля* путем конкатенации их в том порядке, как они входят в *альтернативные поля*. **Результирующий список классов** вхождения *альтернативного поля* является **результирующим списком классов** из списка вхождений в него *спецификаций меток выбора* (см. раздел 12.3).

параметризованные структуры

Параметризованный структурный вид имеет следующие наследственные свойства:

- Он имеет **начальный вариантный** структурный вид, являющийся видом, обозначаемым *именем начального вариантного структурного вида*.
- Он имеет набор имен **полей**, являющийся объединением набора имен **фиксированных полей** его **начального вариантного** структурного вида и набора тех имен **вариантных полей** его **начального вариантного** структурного вида, которые определены во вхождениях *вариантных альтернатив*, выбираемых по списку значений, определяемых *списком постоянных выражений*.

  Набор имен **полей признаков** *параметризованного структурного вида* — это набор имен **полей признаков** его **начального вариантного** структурного вида.
- Он имеет список приписанных значений, определяемых *списком постоянных выражений*.
- Это **теговый параметризованный** структурный вид, если его **начальный вариантный** структурный вид является **теговым вариантным** структурным видом; в противном случае **параметризованный** структурный вид является **беспризнаковым**.

Информация о динамических **параметризованных** структурных видах содержится в разделе 3.13.4.

Статические условия:

общие положения

Все имена **полей** структурного вида должны быть различными.

Если какое-либо поле имеет формат, являющийся *позицией*, то все поля должны иметь формат поля, который должен быть *позицией*.

вариантные структуры

Имя **поля признака** должно быть именем **фиксированного поля** и должно текстуально определяться перед всеми вхождениями *альтернативного поля*, в *списки признаков* которых оно входит. (Следовательно, поле **признака** предшествует всем **вариантным** полям, зависящим от него.) Вид имени **поля признака** должен быть дискретным видом.

Вид вариантного поля не может обладать ни **свойством неизменяемости**, ни **свойством теговой параметризации**.

Вхождения *альтернативного поля* в **вариантном** структурном виде должны быть все **теговыми** или все **беспризнаковыми**. Для **беспризнаковых** *альтернативных полей спецификация метки выбора* может быть опущена во всех вхождениях *вариантной альтернативы* либо должна присутствовать в каждом вхождении *вариантной альтернативы*.

Если для **беспризнакового вариантного** структурного вида какое-либо из его *альтернативных полей* снабжено *спецификацией метки выбора*, то все его альтернативные поля должны иметь *спецификацию метки выбора*.

Для *альтернативных полей* должны выполняться условия выбора варианта (см. раздел 12.3) и должны иметь силу те же условия полноты, непротиворечивости и совместимости, что и для оператора выбора (см. раздел 6.4). Каждое из имен **полей признаков** *списка признаков* (если имеется) служит как селектор выбора с классом М-значения, где М — вид имени **поля признака**. В случае **беспризнаковых** альтернативных полей проверки с использованием селектора выбора не имеют места.

Для **параметризованного вариантного** структурного вида ни один из классов приписанного ему списка классов не может быть **полным** классом. (Это условие автоматически выполняется в **теговом вариантном** структурном виде.)

параметризованные структуры

Имя начального вариантного структурного вида должно допускать **параметризацию.**

В *списке постоянных выражений* должно быть столько **постоянных** *выражений*, сколько имеется классов в списке классов *имени начального вариантного структурного вида*. Класс каждого **постоянного** выражения должен быть **совместимым** с соответствующим (по позиции) классом из списка классов. Если данный класс — это класс M-значения, то значение, вырабатываемое **постоянным** выражением, должно быть одним из значений, вырабатываемым M.

Примеры:

```
3.3     STRUCT (re, im INT)                                          (1.1)
11.7    STRUCT (status SET (occupied, free),
                CASE status OF
                  (occupied): p piece,
                  (free):
                ESAC)                                                (1.1)
2.6     fraction                                                     (1.3)
11.7    status SET (occupied, free)                                  (3.1)
11.8    status                                                       (6.1)
11.9    p piece                                                      (7.1)
```

3.12.5 Описание формата для массивных и структурных видов

Синтаксис:

```
< формат элемента > :: =                                             (1)
        PACK | NOPACK | < шаг >                                      (1.1)

< формат поля > :: =                                                 (2)
        PACK | NOPACK | < позиция >                                  (2.1)

< шаг > :: =                                                         (3)
        STEP ( < позиция > [ , < величина шага > ] )                 (3.1)

< позиция > :: =                                                     (4)
        POS (< слово > , < начальный бит > , < длина > )             (4.1)
      | POS (< слово > [ , < начальный бит > [ : < конечный бит > ] ] )  (4.2)

< слово > :: =                                                       (5)
        < целое постоянное выражение >                               (5.1)

< величина шага > :: =                                               (6)
        < целое постоянное выражение >                               (6.1)

< начальный бит > :: =                                               (7)
        < целое постоянное выражение >                               (7.1)

< конечный бит > :: =                                                (8)
        < целое постоянное выражение >                               (8.1)

< длина > :: =                                                       (9)
        < целое постоянное выражение >                               (9.1)
```

Семантика: Можно управлять форматом массива или структуры, задавая в соответствующем виде информацию об упаковке или отображении. Это **PACK** или **NOPACK** для информации об упаковке, а для информации об отображении — это либо *шаг* в случае массивных видов, либо *позиция* в случае структурных видов. Отсутствие *формата элемента* или *формата поля* в массивном или структурном видах всегда интерпретируется как информация об упаковке, то есть либо как **PACK**, либо как **NOPACK**.

Если для элементов массива или полей структуры описан атрибут **PACK**, то это означает, что использование поля памяти оптимизируется для элементов массива или полей структуры, в то время как атрибут **NOPACK** означает, что оптимизируется время доступа для элементов массива или полей структуры. **NOPACK** также подразумевает возможность **делать ссылки.**

PACK, NOPACK применяются только на одном уровне, то есть применяются к элементам массива или полям структуры, а не к возможным компонентам элемента массива или поля структуры. Информация о формате всегда приписывается ближайшему виду, в отношении которого она может быть использована и который ее не имеет. Например, если по умолчанию информацию об упаковке представляет **NOPACK**, то:

STRUCT *(f* **ARRAY** *(0:1) m* **PACK***)*

эквивалентно

STRUCT *(f* **ARRAY** *(0:1) m* **PACK NOPACK***)*.

Можно управлять точным форматом массива или структуры, описывая информацию о размещении их компонент в виде. Эта информация задается следующим образом:

- Для массивных видов информация о размещении задается сразу для всех элементов в форме *шага* вслед за массивным видом.
- Для структурных видов информация о размещении задается для каждого поля отдельно в форме *позиции* вслед за видом поля.

Информация об отображении с *позицией* задается в терминах слова и смещений битов. *Позиция* в форме:

POS *(< слово > , < начальный бит >, < длина >)*

определяет смещение битов

*NUM (слово) * WIDTH + NUM (начальный бит)*

и длину в *NUM* (*длина*) битов, где *WIDTH* — количество (определяемое реализацией) битов в слове, а слово — это *целое постоянное выражение*.

Когда в *формате поля* описана *позиция*, она указывает на то, что соответствующее поле начинается при данном смещении битов от начала каждой ячейки данного вида и занимает данную длину.

Шаг в форме:

STEP (< позиция > , < величина шага >)

определяет ряд смещений битов b_i для i от 0 до $n - 1$, где n — **количество элементов** массива, а

$b_i = i$*NUM (*величина шага*);

j-ый элемент массива начинается при смещении битов на $p + b_j$ от начала каждой ячейки массивного вида, где p — смещение битов, заданное в *позиции*. Каждый элемент занимает длину, данную в *позиции*.

По умолчанию

Запись:

POS *(< количество слов >, < начальный бит > : < конечный бит >)*

семантически эквивалентна записи:

POS *(< количество слов >, < начальный бит >, NUM (< конечный бит >) – NUM (< начальный бит >) + 1)*.

Запись:

POS *(< количество слов >, < начальный бит >)*

семантически эквивалентна записи:

POS *(< количество слов >, < начальный бит >, BSIZE),*

где *BSIZE* — минимальное количество битов, занимаемых компонентой, для которой описана *позиция*.

Запись:

POS *(< количество слов >)*

семантически эквивалентна записи:

POS *(< количество слов >, 0, BSIZE)*.

Запись:

STEP *(< позиция >)*

семантически эквивалентна записи:

STEP *(< позиция >, SSIZE)*,

где *SSIZE* есть *< длина >*, описанная в *позиции* или получаемая из *позиции* по правилам, приведенным выше.

Статические свойства: Для каждой ячейки массивного вида формат элемента вида определяет ссылаемость ее субъячеек (включая подмассивы, части массивов) следующим образом:

- либо все подъячейки **ссылочные**, либо ни одна из них таковой не является;
- если формат элемента есть **NOPACK**, то все субъячейки **ссылочные**.

Для любой ячейки структурного вида ссылаемость поля структуры, выбираемая именем **поля**, определяется форматом поля имени **поля** следующим образом:

- имя **поля** является **ссылочным**, если формат поля есть **NOPACK**.

Статические условия: Если вид **элемента** данного массивного вида или вида **поля** имени **поля** данного структурного вида сам является массивным или структурным видом, тогда он должен быть **отображенным** видом, если данный массивный или структурный вид является **отображенным**.

Слово, *начальный бит*, *конечный бит*, *длина* и *величина шага* должны, если описаны, вырабатывать неотрицательное значение; значения, вырабатываемые *начальным битом* и *конечным битом*, должны быть меньше, чем *WIDTH*, количество битов в слове реализации, а значение, вырабатываемое *начальным битом*, должно быть меньше или равно значению, вырабатываемому *конечным битом*.

Каждая реализация определяет для каждого вида минимальное количество битов, необходимое для представления его значений; назовем это количество битов минимальным двоичным заполнением. Для дискретных видов — это некоторое количество битов, равное или больше двоичного логарифма **количества значений** вида. Для массивных видов — это смещение элемента с наибольшим индексом плюс занимаемые элементом биты. Для структурных видов это смещение старшего занимаемого бита.

Для каждой *позиции* описанная *длина* не должна быть меньше минимального двоичного заполнения вида соответствующего поля или компонент массива.

Для каждого **отображенного** массивного вида *величина шага* не должна быть меньше *длины*, заданной неявно или явно в *позиции*.

Непротиворечивость и выполнимость

Непротиворечивость

Ни одна из компонент структуры не может быть описана так, что она занимает некоторые биты, уже занятые другой компонентой одного и того же объекта, за исключением случая двух имен **вариантного поля**, в одном и том же вхождении *альтернативного поля*; однако в последнем случае имена **вариантного поля** не могут быть оба определены в одной и той же *вариантной альтернативе*, а также в определении вслед за **ELSE**.

Выполнимость

Не существует определяемых языком условий выполнимости, за исключением одного, выводимого из правила, гласящего, что ссылаемость субъячейки любой (**ссылочной** и **нессылочной**) ячейки определяется только форматом (элемента или поля), что является свойством вида ячейки. Это накладывает ряд ограничений на отображение компонент, которые сами имеют **ссылочные** компоненты.

Примеры:

17.5 **PACK** *(1.1)*
19.14 **POS** *(1,0:15)* *(4.2)*

3.13 ДИНАМИЧЕСКИЕ ВИДЫ

3.13.1 Общие положения

Динамический вид — это вид, ряд свойств которого известны только во время выполнения программы. Динамические виды — это всегда параметризованные виды с одним или несколькими динамическими параметрами. В целях описания в данном документе введены виртуальные обозначения. Эти виртуальные обозначения предваряются амперсандом (символом &), для того чтобы их можно было отличить от фактических обозначений, встречающихся в тексте CHILL-программы.

3.13.2 Динамические строковые виды

Виртуальное обозначение: &< *имя начального строкового вида* > (< *целое выражение* >)

Семантика: Динамический строковый вид — это параметризованный строковый вид со статически неизвестной длиной.

Статические свойства: Динамические строковые виды обладают теми же свойствами, что и строковые виды, за исключением описанных ниже свойств.

Динамические свойства:

- Динамический строковый вид имеет динамическую **длину строки**, представляющую собой вырабатываемое *целым выражением* значение.
- Динамический строковый вид имеет **верхнюю границу** и **нижнюю границу**, являющиеся значениями, вырабатываемыми **длиной строки**, минус 1 и 0, соответственно.

3.13.3 Динамические массивные виды

Виртуальное обозначение: &< *имя начального массивного вида* > (< *дискретное выражение* >)

Семантика: Динамический массивный вид — это параметризованный массивный вид со статически неизвестной **верхней границей**.

Статические свойства: Динамические массивные виды обладают теми же свойствами, что и массивные виды, за исключением описанных ниже свойств.

Динамические свойства:

- Динамический массивный вид имеет **верхнюю границу**, представляющую собой значение, вырабатываемое *дискретным выражением*, и динамическое *количество элементов*, являющееся значением, вырабатываемым с помощью

  NUM (*дискретное выражение*) – *NUM* (*нижняя граница*) + 1,

  где *нижняя граница* — это **нижняя граница** *имени начального массивного вида*.

3.13.4 Динамические параметризованные структурные виды

Виртуальное обозначение: &< *имя начального вариантного структурного вида* > (< *список выражений* >)

Семантика: Динамический **параметризованный** структурный вид — это **параметризованный** структурный вид со статически неизвестными параметрами.

Статические свойства: Статическими свойствами динамического **параметризованного** структурного вида являются статические свойства статического **параметризованного** структурного вида, за исключением следующего:

- Набор имен **полей** динамического **параметризованного** структурного вида — это набор имен **полей** его **начального вариантного** структурного вида.

Динамические свойства:

- Динамический **параметризованный** структурный вид имеет список приписанных значений, являющийся списком значений, вырабатываемых выражениями из *списка выражений*.

4 ЯЧЕЙКИ И ДОСТУП К НИМ

4.1 ОПИСАНИЯ

4.1.1 Общие положения

Синтаксис:

< оператор описания > :: = *(1)*
DCL *< описание >* { , *< описание >* }*; *(1.1)*

< описание > :: = *(2)*
< описание ячейки > *(2.1)*
| *< описание опознавателя-ячейки >* *(2.2)*

Семантика: Оператор описания объявляет одно или несколько имен для доступа к ячейке.

Примеры:

6.9 **DCL** *j INT := julian_day_number,*
d, m, y INT; *(1.1)*
11.36 *starting_square* **LOC** *:= b(m.lin_1)(m.col_1)* *(2.2)*

4.1.2 Описания ячеек

Синтаксис:

< описание ячейки > :: = *(1)*
< список определяющих вхождений > < вид >[**STATIC**] [*< инициализация >*] *(1.1)*

< инициализация > :: = *(2)*
< инициализация границей области > *(2.1)*
| *< инициализация границей времени жизни >* *(2.2)*

< инициализация границей области > :: = *(3)*
< символ присваивания > < значение > [*< программа обработки исключений >*] *(3.1)*

< инициализация границей времени жизни > :: = *(4)*
INIT *< символ присваивания > < постоянное значение >* *(4.1)*

Семантика: Описание ячейки создает столько ячеек, сколько имеется *определяющих вхождений*, специфицированных в *списке определяющих вхождений*.

При *инициализации границей области значение* вычисляется всякий раз при входе в область, в которой находится описание (см. раздел 10.2), и вырабатываемое значение присваивается ячейке(ам). Перед вычислением *значения* ячейка(ки) содержит(ат) **неопределенное** значение.

При *инициализации границей времени жизни* значение, выдаваемое *постоянным значением*, присваивается ячейке(ам) только однажды в начале времени жизни ячейки(ек) (см. разделы 10.2 и 10.9).

Описание без *инициализации* семантически эквивалентно спецификации *инициализации границей времени жизни* с **неопределенным** значением (см. раздел 5.3.1).

Смысл **неопределенного** значения как инициализации ячейки с приписанным видом, обладающим **свойством теговой параметризации** или **свойством беззначимости**, состоит в следующем:

- **свойство теговой параметризации:** созданная(ые) субъячейка(и) поля **признака** инициализируется(ются) соответствующим значением параметра;
- **свойство беззначимости:**
  - — созданная(ые) (субъ)ячейка(и) события и/или буферная(ые) (субъ)ячейка(и) инициализируется(ются) значением "пусто", то есть за событием или буфером не закреплено никаких задержанных процессов и в буфере нет никаких сообщений;
  - — созданная(ые) ассоциативная(ые) (субъ)ячейка(и) инициализируется(ются) значением "пусто", то есть они не содержат ассоциацию;

— созданная(ые) (субъ)ячейка(и) доступа инициализируется(ются) значением "пусто", то есть они не соединяются с ассоциацией;

— созданная(ые) (субъ)ячейка(и) текста имеет(ют) субъячейку **записи текста**, которая инициализируется пустой строкой, и субъячейку **доступа**, которая инициализируется значением "пусто", то есть она не соединена с ассоциацией.

Семантика атрибута **STATIC** и *программа обработки исключений* описаны в разделе 10.9 и главе 8 соответственно.

Статические свойства: *Определяющее вхождение* в *описании ячейки* определяет имя **ячейки**. Вид, приписанный имени **ячейки**, является *видом*, описанным в *описании ячейки*. Имя **ячейки** является **ссылочным**.

Статистические условия: Классы *значения* или *постоянного значения* должны быть **совместимыми** с *видом*, а полученное значение должно быть одним из значений, определяемых *видом*, или **неопределенным** значением.

Если *вид* обладает **свойством неизменяемости**, то должна быть описана *инициализация*. Если *вид* обладает **свойством беззначимости**, то *инициализация границей области* описываться не должна.

Если *инициализация* описывается, то *значение* должно быть **регионально надежным** для ячейки (см. раздел 11.2.2).

Динамические условия: В случае *инициализации границей области*, применяются условия присваивания *значения* по отношению к *виду* (см. раздел 6.2).

Примеры:

5.7	*k2, x, w, t, s, r BOOL*	*(1.1)*
6.9	*:= julian_day_number*	*(3.1)*
8.4	**INIT** *:=['A':'Z']*	*(4.1)*

4.1.3 Описания опознавателей-ячеек

Синтаксис:

< описание опознавателя-ячейки > :: = *(1)*

< список определяющих вхождений > < вид > **LOC [DYNAMIC]**

< символ присваивания > < ячейка > [*< программа обработки исключений >*] *(1.1)*

Семантика: Описание опознавателя-ячейки создает столько имен доступа к описанной ячейке, сколько имеется *определяющих вхождений*, описанных в *списке определяющих вхождений*. Вид ячейки может быть динамическим только в том случае, если описан атрибут **DYNAMIC**.

Если *ячейка* вычисляется динамически, это вычисление производится всякий раз при входе в область, в которой находится описание опознавателя-ячейки. В этом случае описанное имя обозначает **неопределенную** ячейку до первого вычисления в течение времени жизни доступа, обозначаемого описанным именем (см. разделы 10.2 и 10.9).

Статические свойства: *Определяющее вхождение* в *описании опознавателя-ячейки* определяет имя **опознавателя-ячейки**. Вид, приписанный имени **опознавателя-ячейки**, является, в случае отсутствия описания атрибута **DYNAMIC**, *видом*, описанным в *описании опознавателя-ячейки*; в противном случае это его динамически параметризованная версия, имеющая те же параметры, что и вид *ячейки*.

Имя **опознавателя-ячейки** является **ссылочным** тогда и только тогда, когда описанная *ячейка* является **ссылочной**.

Статические условия: Если в *описании опознавателя-ячейки* описан атрибут **DYNAMIC**, то вид должен быть **параметризованным**. Описанный *вид* должен быть **динамическим совместимым по чтению** с видом *ячейки*, если описан атрибут **DYNAMIC**, в противном случае он **совместим по чтению** с видом *ячейки*.

Ячейка не должна быть *элементом строки* или *подстрокой*, у которых *вид ячейки строки* является **изменяющимся** строковым видом.

Динамические условия: Если описан атрибут **DYNAMIC** и неуспешны проверки **динамических совместимых по чтению условий**, приведенных выше, то имеют место исключения *RANGEFAIL* или *TAGFAIL*.

Примеры:

11.36 *starting square* **LOC** := *b(m.lin_1)(m.col_1)* *(1.1)*

4.2 ЯЧЕЙКИ

4.2.1 Общие положения

Синтаксис:

< ячейка > :: = *(1)*
- *< имя доступа >* *(1.1)*
- *разыменованная закрепленная ссылка >* *(1.2)*
- *разыменованная свободная ссылка >* *(1.3)*
- *< разыменованный ряд >* *(1.4)*
- *< элемент строки >* *(1.5)*
- *< подстрока >* *(1.6)*
- *< элемент массива >* *(1.7)*
- *< подмассив >* *(1.8)*
- *< поле структуры >* *(1.9)*
- *< вызов процедуры возврата ячейки >* *(1.10)*
- *< вызов встроенной программы выдачи ячейки >* *(1.11)*
- *< преобразование ячейки >* *(1.12)*

Семантика: Ячейка — это объект, содержащий значение. Должен иметься доступ к ячейкам для хранения или получения значения.

Статические свойства: *Ячейка* обладает следующими свойствами:

- Она имеет вид, определяемый в соответствующем разделе. Этот вид — либо статический, либо динамический.
- Ячейка является **статической** или не является таковой (см. раздел 10.9).
- Ячейка является **внутрирегиональной** либо **внерегиональной** (см. раздел 11.2.2).
- Она является **ссылочной** либо не является таковой. Согласно положениям соответствующих разделов, определение в языке требует, чтобы одни ячейки были **ссылочными**, а другие — нет. Понятие ссылаемости может в реализации распространяться на другие ячейки, за исключением случая, когда это явно недопустимо.

4.2.2 Имена доступа

Синтаксис:

< имя доступа > :: = *(1)*
< имя ячейки > *(1.1)*
| *< имя опознавателя-ячейки >* *(1.2)*
| *< имя перечисления ячеек >* *(1.3)*
| *< имя ячейки с присоединением >* *(1.4)*

Семантика: Имя доступа вырабатывает ячейку. Имя доступа является одним из следующих:

- имя **ячейки**, то есть имя, явно описанное в *описании ячейки* или неявно описанное в *формальном параметре* без атрибута **LOC**;
- имя **опознавателя-ячейки**, то есть имя, явно описанное в *описании опознавателя-ячейки* или неявно описанное в *формальном параметре* с атрибутом **LOC**;
- имя **перечисления ячеек**, то есть *счетчик цикла* в *перечислении ячеек*;
- имя **ячейки с присоединением**, то есть имя **поля**, используемое в качестве прямого доступа в *операторе цикла с присоединением*.

Если *ячейка*, обозначенная *именем ячейки с присоединением*, является вариантным полем или ячейкой беспризнаковой вариантной структуры, то семантика определяется реализацией.

Статические свойства: Вид (возможно, динамический), приписанный к *имени доступа*, является видом *имени ячейки*, *имени опознавателя-ячейки*, *имени перечисления ячеек* или *имени ячейки с присоединением* соответственно.

Имя *доступа* является **ссылочным** тогда и только тогда, когда это *имя ячейки*, **ссылочное** *имя опознавателя-ячейки*, **ссылочное** *имя перечисления ячеек* или **ссылочное** имя *ячейки с присоединением*.

Динамические условия: При доступе посредством *имени опознавателя-ячейки* это имя не должно обозначать **неопределенную** ячейку.

При осуществлении доступа посредством *имени опознавателя-ячейки* к ячейке, являющейся **вариантным** полем, для этой ячейки должны удовлетворяться условия доступа к вариантному полю (см. раздел 4.2.10). Доступ посредством *имени ячейки с присоединением* может быть причиной исключения *TAGFAIL*, если обозначенная ячейка является **вариантным** полем и не удовлетворяются условия доступа к вариантному полю для ячейки.

Примеры:

4.12 *a* *(1.1)*
11.39 *starting* *(1.2)*
15.35 *each* *(1.3)*
5.10 *c1* *(1.4)*

4.2.3 Разыменованные закрепленные ссылки

Синтаксис:

< разыменованная закрепленная ссылка > :: = *(1)*
< примитивное значение закрепленной ссылки > –> [*< имя вида >*] *(1.1)*

Семантика: Разыменованная закрепленная ссылка вырабатывает ячейку, ссылаемую значением закрепленной ссылки.

Статические свойства: Вид, приписанный *разыменованной закрепленной ссылке*, является *именем вида*, если он описан, в противном случае это **ссылаемый** вид вида *примитивного значения закрепленной ссылки*. Разыменованная *закрепленная ссылка* является **ссылочной**.

Статические условия: *Примитивное значение закрепленной ссылки* должно быть **строгим**. Если описано необязательное *имя вида*, оно должно быть **совместимым по чтению** с **ссылаемым** видом вида *примитивного значения закрепленной ссылки*.

Динамические условия: Время жизни ссылаемой ячейки не должно заканчиваться.

Если *примитивное значение закрепленной ссылки* вырабатывает значение *NULL*, то имеет место исключение *EMPTY*.

Если ссылаемая ячейка является **вариантным** полем, то для ячейки должны удовлетворяться условия доступа к вариантному полю (см. раздел 4.2.10).

Примеры:

10.54 p -> *(1.1)*

4.2.4 Разыменованные свободные ссылки

Синтаксис:

< разыменованная свободная ссылка > :: = *(1)*
< примитивное значение свободной ссылки > -> < имя вида > *(1.1)*

Семантика: Разыменованная свободная ссылка вырабатывает ячейку, ссылаемую значением свободной ссылки.

Статические свойства: Вид, приписанный *разыменованной свободной ссылке*, является *именем вида*. *Разыменованная свободная ссылка* является **ссылочной**.

Статические условия: *Примитивное значение свободной ссылки* должно быть **строгим**.

Динамические условия: Время жизни ссылаемой ячейки не должно заканчиваться.

Если *примитивное значение свободной ссылки* вырабатывает значение *NULL*, то имеет место исключение *EMPTY*.

Имя вида должно быть **совместимым по чтению** с видом ссылаемой ячейки.

Если ссылаемая ячейка является **вариантным** полем, то условия доступа к вариантному полю для данной ячейки должны быть удовлетворены (см. раздел 4.2.10).

4.2.5 Разыменованные ряды

Синтаксис:

< разыменованный ряд > :: = *(1)*
< примитивное значение ряда > -> *(1.1)*

Семантика: Разыменованный ряд вырабатывает ячейку, ссылаемую значением ряда.

Статические условия: Динамический вид, приписанный *разыменованному ряду*, образован следующим образом:

&имя начального вида (< параметр > { , < параметр > }),*

где *имя начального вида* является именем виртуального **синонимического** вида, **синонимическим** со **ссылаемым начальным** видом вида *примитивного значения ряда* и где параметрами, зависящими от **ссылаемого начального** вида, являются:

- динамическая **длина строки** в случае строкового вида;
- динамическая **верхняя граница** в случае массивного вида;
- список значений, связанных с видом ячейки параметризованной структуры, в случае **вариантного** структурного вида.

Разыменованный ряд является **ссылочным**.

Статические условия: *Примитивное значение ряда* должно быть **строгим**.

Динамические условия: Время жизни ссылаемой ячейки не должно заканчиваться.

Если *примитивное значение ряда* вырабатывает *NULL*, то имеет место исключение *EMPTY*.

Если ссылаемая ячейка является **вариантным** полем, то для ячейки должны удовлетворяться условия доступа к вариантному полю (см. раздел 4.2.10).

Примеры:

8.11 input -> *(1.1)*

4.2.6 Элементы строк

Синтаксис:

< элемент строки > :: = *(1)*
< ячейка строки > (< начальный элемент >) *(1.1)*

< начальный элемент > :: = *(2)*
< целое выражение > *(2.1)*

Семантика: Элемент строки вырабатывает (субъ)ячейку, являющуюся элементом описанной ячейки строки, указываемой *начальным элементом*.

Статические свойства: Вид, приписанный *элементу строки*, является *BOOL* или *CHAR* в зависимости от того, является ли вид *ячейки строки* **битовым** строковым видом или **символьным** строковым видом.

Если вид *ячейки строки* является **изменяющимся** строковым видом, тогда *элемент строки* не является **ссылочным**.

Динамические условия: Имеет место исключение *RANGEFAIL*, если не выполняется следующее соотношение:

$$0 \leq NUM\ (начальный\ элемент) \leq L-1,$$

где L — **фактическая длина** *ячейки строки*.

Примеры:

18.16 string ->(i) *(1.1)*

4.2.7 Подстроки

Синтаксис:

< подстрока > :: = *(1)*

< ячейка строки > (< левый элемент > : < правый элемент >) *(1.1)*

| *< ячейка строки > (< начальный элемент >* **UP** *< размер подстроки >)* *(1.2)*

< левый элемент > :: = *(2)*

< целое выражение > *(2.1)*

< правый элемент > :: = *(3)*

< целое выражение > *(3.1)*

< размер подстроки > :: = *(4)*

< целое выражение > *(4.1)*

Семантика: Подстрока вырабатывает ячейку (возможно, динамической) строки, являющейся частью описанной ячейки строки, определяемой *левым элементом* и *правым элементом* или *начальным элементом* и *размером подстроки*. Длина (возможно, динамическая) подстроки определяется из описанных выражений.

Подстрока, в которой *правый элемент* вырабатывает значение, которое меньше значения, вырабатываемого *левым элементом* или у которой *размер подстроки* вырабатывает неположительное значение, обозначает пустую строку.

Статические свойства: Вид (возможно, динамический), приписанный *подстроке*, является **параметризованным** строковым видом, образованным в форме

&имя (размер строки),

где *&имя* — имя виртуального **синонимического** вида, **синонимического** с видом (возможно, динамическим) *ячейки строки*, если это **фиксированный** строковый вид; в противном случае **синонимического** с **компонентным** видом,а *размер строки* есть либо

NUM (правый элемент) — NUM (левый элемент) + 1,

либо

NUM (размер подстроки).

Однако *размер строки* есть 0, если обозначена пустая строка. Вид, приписанный *подстроке*, является статическим, если *размер строки* — **константа**, то есть *правый элемент* и *левый элемент* являются **константой** или *размер подстроки* — **константа**; в противном случае вид является динамическим.

Если вид *ячейки строки* — **изменяющийся** строковый вид, тогда *подстрока* не является **ссылочной**.

Статические условия: Должны выполняться следующие соотношения:

$0 \leq NUM\ (левый\ элемент) \leq L-1$

$0 \leq NUM\ (правый\ элемент) \leq L-1$

$0 \leq NUM\ (начальный\ элемент) \leq L-1$

$NUM\ (начальный\ элемент) + NUM\ (размер\ подстроки) \leq L,$

где L — **фактическая длина** *ячейки строки*. Если L и значение всех *целых выражений* известны статически, то соотношения можно проверить статически.

Динамические условия: Если динамическая часть проверки приведенных выше соотношений неуспешна, то имеет место исключение *RANGEFAIL*.

Примеры:

18.26 blanks (count : 9) *(1.1)*

18.23 string ->(scanstart **UP** *10)* *(1.2)*

4.2.8 Элементы массивов

Синтаксис:

< элемент массива > :: = (1)
< ячейка массива > (< список выражений >) (1.1)

< список выражений > :: = (2)
*< выражение > { , < выражение > }** (2.1)

Производный синтаксис: Обозначение: *(< список выражений >)* является производным синтаксисом для

$$(< \textit{выражение} >)\ \{\ (< \textit{выражение} >)\ \}^*,$$

где количество выражений в скобках равно количеству выражений в *списке выражений*. Таким образом, *элемент массива* в строгом синтаксисе имеет только одно (индексное) выражение.

Семантика: Элемент массива вырабатывает (субъ)ячейку, являющуюся элементом ячейки описанного массива, определяемой *выражением*.

Статические свойства: Вид, приписанный *элементу массива*, является видом **элемента** вида *ячейки массива*.

Элемент массива является **ссылочным**, если **формат элемента** вида *ячейки массива* есть **NOPACK**.

Статические условия: Класс выражения должен быть **совместимым** с **индексным** видом вида *ячейки массива*.

Динамические условия: Имеет место исключение *RANGEFAIL*, если не выполняется следующее соотношение:

$$L \le \textit{выражение} \le U,$$

где L и U — это **нижняя граница** и **верхняя граница** (возможно, динамическая) вида *ячейки массива* соответственно.

Примеры:

11.36 b(m.lin_1)(m.col_1) (1.1)

4.2.9 Подмассивы

Синтаксис:

< подмассив > :: = (1)
< ячейка массива > (< нижний элемент > : < верхний элемент >) (1.1)
| *< ячейка массива > (< первый элемент >* **UP** *< размер подмассива >)* (1.2)

< нижний элемент > :: = (2)
< выражение > (2.1)

< верхний элемент > :: = (3)
< выражение > (3.1)

< первый элемент > :: = (4)
< выражение > (4.1)

Семантика: Подмассив вырабатывает ячейку массива (возможно, динамического), являющуюся частью описанной ячейки массива, указанного *нижним элементом* и *верхним элементом* или *первым элементом* и *размером подмассива*. **Нижняя граница** подмассива равна нижней границе описанного массива; **верхняя граница** (возможно, динамическая) определяется из заданных выражений.

Статические свойства: Вид (возможно, динамический), приписанный *подмассиву*, является **параметризованным** массивным видом, который образован следующим образом:

&имя (верхний индекс),

где *&имя* — имя виртуального **синонимического** вида, **синонимического** с видом (возможно, динамическим) *ячейки массива*, а *верхний индекс* — выражение, класс которого является **совместимым** с классами *нижнего элемента* и *верхнего элемента* и которое вырабатывает такое значение, что:

NUM (верхний индекс) = NUM (L) + NUM (верхний элемент) — NUM (нижний элемент),

либо выражение, класс которого является **совместимым** с классом *первого элемента* и которое вырабатывает такое значение, что:

NUM (верхний индекс) = NUM (L) + NUM (размер подмассива) — 1,

где *L* — **нижняя граница** вида *ячейки массива*.

Вид, приписанный *подмассиву*, является статическим, если верхний индекс — **константа**, то есть если *нижний элемент* и *верхний элемент* являются оба **константой** либо *размер подмассива* — **константа**; в противном случае вид является динамическим.

Подмассив является **ссылочным**, если **формат элемента** вида *ячейки массива* есть **NOPACK**.

Статические условия: Классы *нижнего элемента* и *верхнего элемента* или класс *первого элемента* должны быть **совместимыми** с **индексным** видом *ячейки массива*.

Должны выполняться следующие соотношения:

$L \leq$ *нижний элемент* $\leq$ *верхний элемент* $\leq U$

$1 \leq$ *NUM (размер подмассива)* $\leq$ *NUM (U) — NUM (L) + 1*

NUM (L) $\leq$ *NUM (первый элемент)* $\leq$ *NUM (первый элемент) + NUM (размер подмассива) — 1* $\leq$ *NUM (U),*

где *L* и *U* — соответственно **нижняя граница** и **верхняя граница** вида *ячейки массива*. Если *U* и значение всех *выражений* известны статически, то соотношения можно проверить статически.

Динамические условия: Если динамическая часть проверки соотношений, приведенных выше, была неуспешной, то имеет место исключение *RANGEFAIL*.

Примеры:

17.27 res (0 : count—1) *(1.1)*

4.2.10 Поля структур

Синтаксис:

< поле структуры > :: = *(1)*
< ячейка структуры > . < имя поля > *(1.1)*

Семантика: Поле структуры вырабатывает (субъ)ячейку, являющуюся полем ячейки описанной структуры, указанной *именем поля*. Если *ячейка структуры* имеет **беспризнаковый вариантный** структурный вид, а *имя поля* — это имя **вариантного поля**, то семантика определяется реализацией.

Статические свойства: Вид *поля структуры* — это вид *имени поля*.

Поле структуры является **ссылочным**, если **формат поля** *имени поля* есть **NOPACK**.

Статические условия: *Имя поля* должно быть именем из набора имен **полей** вида *ячейки структуры*.

Динамические условия: *Ячейка* не должна обозначать:

- ячейку **тегового вариантного** структурного вида, в которой соответствующее(ие) значение(я) поля **признака** указывает(ют), что поле не существует;
- ячейку динамического **параметризованного** структурного вида, в которой соответствующий список значений указывает на отсутствие поля.

Вышеупомянутые условия называются условиями доступа к вариантному полю для ячейки (заметим, что условие не содержит исключения). Если условия не удовлетворяются для *ячейки структуры*, то имеет место исключение *TAGFAIL*.

Примеры:

10.57 last ->.info *(1.1)*

4.2.11 Вызовы процедур возврата ячейки

Синтаксис:

< вызов процедуры возврата ячейки > :: = *(1)*
< вызов процедуры возврата ячейки > *(1.1)*

Семантика: Вызов процедуры возврата ячейки вырабатывает ячейку, возвращаемую из процедуры.

Статические свойства: Вид, приписанный *вызову процедуры возврата ячейки*, является видом **спецификации результата** *вызова процедуры возврата ячейки*, если в ней не описан атрибут **DYNAMIC**; в противном случае это динамически параметризованная версия вида, имеющая те же параметры, что и вид выработанной ячейки.

Вызов процедуры возврата ячейки является **ссылочным**, если в **спецификации результата** *вызова процедуры возврата ячейки* не описан атрибут **NONREF**.

Динамические условия: *Вызов процедуры возврата ячейки* не должен вырабатывать **неопределенную** ячейку, а время жизни выработанной ячейки не должно заканчиваться.

4.2.12 Вызовы встроенных программ выдачи ячейки

Синтаксис:

< вызов встроенной программы выдачи ячейки > :: = *(1)*
< вызов встроенной программы выдачи ячейки > *(1.1)*

Семантика: Вызов встроенной программы выдачи ячейки вырабатывает ячейку, возвращаемую из вызова встроенной программы.

Статические свойства: Вид, приписанный *вызову встроенной программы выдачи ячейки*, является видом **спецификации результата** *вызова встроенной программы выдачи ячейки*.

Динамические условия: *Вызов встроенной программы выдачи ячейки* не должен вырабатывать **неопределенную** ячейку, а время жизни вырабатываемого значения не должно заканчиваться.

4.2.13 Преобразования ячеек

Синтаксис:

< преобразование ячейки > :: = *(1)*
< имя вида > < ячейка статического вида >. *(1.1)*

Семантика: Преобразование ячейки вырабатывает ячейку, обозначаемую *ячейкой статического вида*. Однако при этом не принимаются во внимание проверка видов, предусмотренная в языке CHILL, и правила совместимости, а вид явно приписывается ячейке.

Точная динамическая семантика преобразования ячейки зависит от реализации.

Статические свойства: Вид *преобразования ячейки* является *именем вида*.

Преобразование ячейки **ссылочное**.

Статические условия: *Ячейка статического вида* должна быть **ссылочной**.

Должно соблюдаться следующее отношение:

SIZE (имя вида) = SIZE (ячейка статического вида).

5 ЗНАЧЕНИЯ И ОПЕРАЦИИ НАД НИМИ

5.1 ОПРЕДЕЛЕНИЯ СИНОНИМОВ

Синтаксис:

< оператор определения синонима > :: = *(1)*
SYN *< определение синонима >* { , *< определение синонима >* }*; *(1.1)*

< определение синонима > :: = *(2)*
< список определяющих вхождений > [*< вид >*] = *< постоянное значение >* *(2.1)*

Производный синтаксис: *Определение синонима*, *список определяющих вхождений* в котором состоит из нескольких *определяющих вхождений*, получено из нескольких *определяющих* вхождений *синонимов*, по одному на каждое *определяющее вхождение* с одними и теми же *постоянным значением* и *видом*, если имеется. Например, **SYN** *i*, *j* = 3 является производным от **SYN** *i* = 3, *j* = 3.

Семантика: *Определение синонима* определяет имя, обозначающее описанное **постоянное** значение.

Статические свойства: *Определяющее вхождение* в *определении синонима* определяет имя **синонима**.

Класс имени **синонима** в случае, если описан *вид*, есть класс М-значения, где М — *вид*; в противном случае это класс *постоянного значения*.

Имя **синонима** является **неопределенным** тогда и только тогда, когда *постоянное значение* — это **неопределенное** значение (см. раздел 5.3.1).

Имя **синонима** является **константой** тогда и только тогда, когда *постоянное значение* есть **константа**.

Статические условия: Если *вид* описан, то он должен быть **совместимым** с классом *постоянного значения*, а значение, вырабатываемое *постоянным значением*, должно быть одним из значений, определяемых *видом*.

Определения синонимов не должны быть ни рекурсивными, ни взаимно рекурсивными через другие определения синонимов или видов, то есть никакой набор рекурсивных определений не может содержать определений синонимов (см. раздел 3.2.1).

Примеры:

1.17 **SYN** *neutral_for_add=0,*
neutral_for_mult=1 *(1.1)*
2.18 neutral_for_add fraction=[*0,1*] *(2.1)*

5.2 ПРИМИТИВНОЕ ЗНАЧЕНИЕ

5.2.1 Общие положения

Синтаксис:

< примитивное значение > :: = *(1)*
< содержимое ячейки > *(1.1)*
| *< имя значения >* *(1.2)*
| *< константа >* *(1.3)*
| *< кортеж >* *(1.4)*
| *< элемент строки значений >* *(1.5)*
| *< подстрока значений >* *(1.6)*
| *< элемент массива значений >* *(1.7)*
| *< подмассив значений >* *(1.8)*
| *< поле структуры значений >* *(1.9)*
| *< преобразование выражения >* *(1.10)*

| < *вызов процедуры возврата значения* > (1.11)
| < *вызов встроенной программы выдачи значения* > (1.12)
| < *выражение запуска* > (1.13)
| < *операция индикации* > (1.14)
| < *выражение в круглых скобках* > (1.15)

Семантика: Примитивное значение является основной составляющей выражения. Ряд примитивных значений имеет динамический класс, то есть класс, базирующийся на динамическом виде. Для этих примитивных значений проверки совместимости могут быть выполнены только во время выполнения программы. Результатом неуспешной проверки являются исключения *TAGFAIL* или *RANGEFAIL*.

Статические свойства: Класс *примитивного значения* является классом *содержимого ячейки*, *имени значения* и т.д. соответственно.

Примитивное значение является **постоянным** тогда и только тогда, когда это *имя* **постоянного** *значения*, *константа*, **постоянный** *кортеж*, *преобразование* **постоянного** *выражения*, *вызов встроенной программы выдачи* **постоянного** *значения* или **постоянное** *выражение в круглых скобках*.

Примитивное значение является **константой** тогда и только тогда, когда оно имеет *имя*, являющееся **константой**, **дискретной** *константой* либо *вызов встроенной программы выдачи значения* является **константой**.

5.2.2 Содержимое ячейки

Синтаксис:

< *содержимое ячейки* > :: = (1)
< *ячейка* > (1.1)

Семантика: Содержимое ячейки вырабатывает значение, содержащееся в определенной ячейке. Для получения хранимого в ячейке значения к ячейке должен иметься доступ.

Статические свойства: Класс *содержимого ячейки* является классом М-значения, где М — вид (возможно, динамический) *ячейки*.

Статические условия: Вид *ячейки* не должен обладать **свойством беззначимости**.

Динамические условия: Вырабатываемое значение не должно быть **неопределенным**.

Примеры:

3.7 c2.im (1.1)

5.2.3 Имена значений

Синтаксис:

< *имя значения* > :: = (1)
< *имя синонима* > (1.1)
| < *имя перечисления значений* > (1.2)
| < *имя значения с присоединением* > (1.3)
| < *имя принятого значения* > (1.4)
| < *имя общей процедуры* > (1.5)

Семантика: Имя значения передает значение. Имя значения является одним из следующих:

- имя **синонима**, то есть имя, определяемое *оператором определения синонима*;
- имя **перечисления значений**, то есть имя, определяемое *счетчиком цикла* в *перечислении значений*;
- имя **значения с присоединением**, то есть имя **поля**, введенное как имя значения в *операторе цикла с присоединением*;
- имя **принятого значения**, то есть имя, введенное в *операторе варианта получения*;
- имя **общей процедуры** (см. раздел 10.4).

Если значение, обозначенное *именем значения с присоединением*, является вариантным полем значения беспризнаковой вариантной структуры, то семантика определяется реализацией.

Статические свойства: Класс *имени значения* — это класс *имени синонима*, *имени перечисления значений*, *имени значения с присоединением*, *имени принятого значения* или M-производный класс, где M — вид *имени общей процедуры* соответственно.

Имя значения — **константа** тогда и только тогда, когда, если это *имя синонима*, являющееся **константой**.

Имя значения — **постоянная**, если это *имя синонима* или *имя общей процедуры*, обозначающее имя **процедуры**, приписанное *определению процедуры*, не окруженному блоком.

Статические условия: *Имя синонима* не должно быть **неопределенным**.

Динамические условия: Вычисление *имени значения с присоединением* является причиной исключения *TAGFAIL*, если обозначенное значение является **вариантным** полем и для значения не удовлетворяются условия доступа к вариантному полю.

Примеры:

10.12	*max*	*(1.1)*
8.8	*i*	*(1.2)*
15.54	*this_counter*	*(1.4)*

5.2.4 Константы

5.2.4.1 Общие положения

Синтаксис:

< константа > :: =	*(1)*
< целая константа >	*(1.1)*
| *< булевская константа >*	*(1.2)*
| *< символьная константа >*	*(1.3)*
| *< перечислимая константа >*	*(1.4)*
| *< пустая константа >*	*(1.5)*
| *< символьная строковая константа >*	*(1.6)*
| *< битовая строковая константа >*	*(1.7)*

Семантика: Константа передает **постоянное** значение.

Статические свойства: Класс *константы* есть класс *целой константы*, *булевской константы* и т.д. соответственно. *Константа* является **дискретной**, если она либо *целая константа*, *булевская константа*, *символьная константа*, либо *перечислимая константа*.

Буква вместе с последующим апострофом, с которого начинается *целая константа*, *булевская константа* и *битовая строковая константа* (то есть *B'*, *D'*, *H'*, *O'*, *b'*, *d'*, *h'*, *o'*) есть классификатор константы.

5.2.4.2 Целые константы

Синтаксис:

< целая константа > :: = (1)
| *< десятичная целая константа >* (1.1)
| *< двоичная целая константа >* (1.2)
| *< восмеричная целая константа >* (1.3)
| *< шестнадцатеричная целая константа >* (1.4)

< десятичная целая константа > :: = (2)
[{ *D* | *d* }'] { *< цифра >* | _ }$^{+}$ (2.1)

< двоичная целая константа > :: = (3)
{ *B* | *b* }' { *0* | *1* | _ }$^{+}$ (3.1)

< восьмеричная целая константа > :: = (4)
{ *O* | *o* }' { *< восьмеричная цифра >* | _ }$^{+}$ (4.1)

< шестнадцатеричная целая константа > :: = (5)
{ *H* | *h* }' { *< шестнадцатеричная цифра >* | _ }$^{+}$ (5.1)

< шестнадцатеричная цифра > :: = (6)
< цифра > | *A* | *B* | *C* | *D* | *E* | *F* | *a* | *b* | *c* | *d* | *e* | *f* (6.1)

< восьмеричная цифра > :: = (7)
0 | *1* | *2* | *3* | *4* | *5* | *6* | *7* (7.1)

Семантика: Целая константа передает неотрицательное целое значение. Используется обычная десятичная запись (основание системы счисления 10), а также двоичная (основание системы счисления 2), восьмеричная (основание системы счисления 8) и шестнадцатеричная (основание системы счисления 16). Символ подчеркивания (_) является несущественным, то есть служит целям удобочитаемости и не влияет на указанное значение.

Статические свойства: Класс *целого литерала* является *INT*-производным классом. *Целый литерал* является **постоянной** и **константой**.

Статические условия: Строка, следующая за апострофом (') и вся *целая константа* не должны состоять из одних символов подчеркивания.

Примеры:

6.11 *1_721_119* (1.1)
D'1_721_119 (1.1)
B'101011_110100 (1.2)
O'53_64 (1.3)
H'AF4 (1.4)

5.2.4.3 Булевские константы

Синтаксис:

< булевская константа > :: = (1)
< имя булевской константы > (1.1)

Предопределенные имена: Имена *FALSE* и *TRUE* являются предопределенными именами **булевских констант.**

Семантика: Булевская константа передает булевское значение.

Статические свойства: Класс *булевской константы* является *BOOL*-производным классом. *Булевская константа* является **постоянной и константой.**

Примеры:

5.46 *FALSE* *(1.1)*

5.2.4.4 Символьные константы

Синтаксис:

< символьная константа > :: = *(1)*
' *< символ >* | *< последовательность символов >*' *(1.1)*

Семантика: Символьная константа передает символьное значение. Кроме печатного представления, может использоваться *управляющая последовательность*.

Статические свойства: Класс *символьной константы* есть *CHAR*-производный класс. *Символьная константа* является **постоянной и константой.**

Статические условия: *Управляющая последовательность* в *символьной константе* должна обозначать только один символ.

Примеры:

7.9 *'M'* *(1.1)*

5.2.4.5 Перечислимые константы

Синтаксис:

< перечислимая константа > :: = *(1)*
< имя элемента перечисления > *(1.1)*

Семантика: Перечислимая константа передает перечислимое значение. Перечислимая константа является именем, определенным в перечислимом виде.

Статические свойства: Класс *перечислимой константы* есть M-производный класс, где M — **перечислимый** вид, приписанный *имени элемента перечисления*. *Перечислимая константа* является **постоянной и константой.**

Примеры:

6.51 *dec* *(1.1)*
11.78 *king* *(1.1)*

5.2.4.6 Пустая константа

Синтаксис:

< пустая константа > :: = *(1)*
< имя пустой константы > *(1.1)*

Предопределенные имена: Имя *NULL* является **предопределенным** как имя **пустой константы.**

Семантика: Пустая константа передает либо пустое ссылочное значение, то есть значение, которое не отсылает к ячейке, значение пустой процедуры, то есть значение, не указывающее на процедуру, или значение пустого экземпляра, то есть значение, не идентифицирующее процесс.

Статические свойства: Класс *пустой константы* является **нулевым** классом. *Пустой литерал* является **постоянной.**

Примеры:

10.43 NULL *(1.1)*

5.2.4.7 Символьные строковые константы

Синтаксис:

< символьная строковая константа > :: = *(1)*
" { *< нерезервируемый символ >* | *< кавычки >* | *< управляющая последовательность >* }* " *(1.1)*

< кавычки > :: = *(2)*
"" *(2.1)*

< управляющая последовательность > :: = *(3)*
^(*< целое постоянное выражение >* { , *< целое постоянное выражение >* }*) *(3.1)*
| ^< *неспециальный символ >* *(3.2)*
| ^^ *(3.3)*

Семантика: Символьная строковая константа передает значение символьной строки, длина которой может быть равной нулю. Это список значений элементов строки; значения даны для элементов в возрастающем порядке по их индексу слева направо. Для представления кавычек (") в символьной строковой константе они записываются дважды ("").

Кроме печатного представления, может использоваться *управляющая последовательность*. *Управляющая последовательность*, в которой за диакритическим символом (^) следует открывающая скобка, обозначает последовательность символов, которые представлены в ней *целым постоянным выражением*; в противном случае, если за одним диакритическим символом следует другой диакритический символ, он обозначает сам себя, в противном случае он обозначает символ, представление которого получено логическим отрицанием седьмого бита (b7) внутреннего представления *неспециального символа* в ней (см. добавление A).

Статические свойства: **Длина строки** *символьного строкового литерала* — это количество *нерезервируемых символов*, *кавычек* и символов, обозначенных вхождениями *управляющей последовательности.*

Класс *символьной строковой константы* есть **CHARS** (*n*)-производный класс, где *n* — **длина строки** *символьной строковой константы. Символьная строковая константа* есть **постоянная.**

Статические условия: Значение, вырабатываемое *целым постоянным выражением* в *управляющей последовательности* должно принадлежать диапазону значений, определяемых представлениями символов в наборе символов языка CHILL (см. добавление A).

Примеры:

8.20 "A—B< ZAA9K' " *(1.1)*

5.2.4.8 Битовые строковые константы

Синтаксис:

< *битовая строковая константа* > :: = (1)
< *двоичная битовая строковая константа* > (1.1)
| < *восьмеричная битовая строковая константа* > (1.2)
| < *шестнадцатеричная битовая строковая константа* > (1.3)

< *двоичная битовая строковая константа* > :: = (2)
{ *B* | *b* } ' { *0* | *1* | _ }*' (2.1)

< *восьмеричная битовая строковая константа* > :: = (3)
{ *O* | *o* } ' { < *восьмеричная цифра* > | _ }*' (3.1)

< *шестнадцатеричная битовая строковая константа* > :: = (4)
{ *H* | *h* } ' { < *шестнадцатеричная цифра* > _ }*' (4.1)

Семантика: Битовая строковая константа представляет значения битовой строки, длина которой может быть нулевой. Могут использоваться двоичное, восьмеричное и шестнадцатеричное представления. Символ подчеркивания (_) является несущественным, то есть он служит только целям удобочитаемости и не влияет на указанное значение.

Битовая строковая константа — это список значений элементов строки: значения даны для элементов в возрастающем порядке их индекса слева направо.

Статические свойства: **Длина строки** *битовой строковой константы* является количеством вхождений нуля и единицы после *B'*, утроенным количеством вхождений *восьмеричных цифр* после *O'* или учетверенным количеством вхождений *шестнадцатеричных цифр* после *H'*.

Класс *битовой строковой константы* есть **BOOLS** (*n*)-производный класс, где *n* — **длина строки** *битовой строковой константы*. *Битовая строковая константа* является **постоянной**.

Примеры:

B'101011_110100' (1.1)
O'53_64' (1.2)
H'AF4' (1.3)

5.2.5 Кортежи

Синтаксис:

< *кортеж* > :: = (1)
[< *имя вида* >] (: { < *кортеж множества* > |
< *кортеж массива* > | < *кортеж структуры* > } :) (1.1)

< *кортеж множества* > :: = (2)
[{< *выражение* > | < *диапазон* > } { , { < *выражение* > | < *диапазон* > } }*] (2.1)

< *диапазон* > :: = (3)
< *выражение* > : < *выражение* > (3.1)

< *кортеж массива* > :: = (4)
< *кортеж непомеченного массива* > (4.1)
| < *кортеж помеченного массива* > (4.2)

< *кортеж непомеченного массива* > :: = (5)
< *значение* > { , < *значение* >}* (5.1)

< *кортеж помеченного массива* > :: = (6)
< *список меток выбора* > : < *значение* >
{ ,< *список меток выбора* > : < *значение* > }* (6.1)

< *кортеж структуры* > :: = (7)
< *кортеж непомеченной структуры* > (7.1)
| < *кортеж помеченной структуры* > (7.2)

< *кортеж непомеченной структуры* > :: = (8)
< *значение* > { , < *значение* > }* (8.1)

< кортеж помеченной структуры > :: = *(9)*

< список имен полей > : < значение >

{,*< список имен полей > : < значение >*}* *(9.1)*

< список имен полей > :: = *(10)*

< имя поля > { , . *< имя поля >*}* *(10.1)*

Производный синтаксис: Открывающая и закрывающая скобки кортежа, [и], являются производным синтаксисом для (:и:), соответственно. Это не обозначается в синтаксисе, чтобы избежать путаницы с использованием квадратных скобок как метасимволов.

Семантика: Кортеж передает значение множества, значение массива или значение структуры.

Если это значение множества, оно состоит из списка выражений и/или диапазонов, обозначающих те значения элементов, которые входят в значение множества. Диапазон обозначает те значения, которые лежат между значениями, вырабатываемыми выражениями в диапазоне, или же являющимися одними из этих значений. Если второе выражение вырабатывает значение, которое меньше значения, получаемого из первого выражения, то диапазон пуст, то есть он не обозначает никаких значений. Кортеж множества может обозначать значение пустого множества.

Если это значение массива, то это (возможно, помеченный список) значений элементов массива; в кортеже непомеченного массива значения для элементов приводятся в возрастающем порядке по их индексу; в кортеже помеченного массива даны значения элементов по их индексам, описанным в списке меток выбора для значений. Этот механизм можно использовать для уплотнения записи для кортежей больших массивов, когда многие значения одни и те же. Метка **ELSE** обозначает значения индексов, не выраженных явно. Метка * обозначает все значения индексов (подробные сведения см. в разделе 12.3).

Если это значение структуры, то это набор (возможно, помеченный) значений полей структуры. В кортеже непомеченной структуры значения для полей приведены в том же порядке, как они описаны в приписанном структурном виде. В кортеже помеченной структуры значения даны для тех полей, имена которых описаны в списке имен полей для значения.

Порядок вычисления выражений и значений в кортеже не определен и можно считать, что выражения и значения вычисляются в смешанном порядке.

Статические условия: Класс *кортежа* есть класс М-значения, где М — *имя вида*, если описано. В противном случае М зависит от контекста, где встречается *кортеж*. Эта зависимость выражается следующим образом:

- если *кортеж* есть *значение* или *постоянное значение* в *инициализации описания ячейки*, тогда М — *вид* в *описании ячейки*;
- если *кортеж* — *значение* в правой части в *операторе однократного присваивания*, тогда М — вид (возможно, динамический) *ячейки* в левой части;
- если *кортеж* есть *постоянное значение* в определении синонима с описанным видом, то М — это *вид*;
- если *кортеж* есть *фактический параметр* в *вызове процедуры* или в *выражении запуска*, где атрибут **DYNAMIC** не описан в соответствующей *спецификации параметра*, тогда М — вид в соответствующей *спецификации параметра*;
- если *кортеж* есть *значение* в *операторе возврата* или *операторе результата*, то М — вид **спецификации результата** имени **процедуры** *оператора результата* или *оператора возврата* (см. раздел 6.8);
- если *кортеж* есть *значение* в *операторе посылки*, тогда это соответствующий вид, описанный в определении сигнала *имени сигнала*, или **буферный элементный** вид вида *буферной ячейки*;
- если *кортеж* есть *выражение* в *кортеже массива*, тогда М — вид **элемента** вида *кортежа массива*;
- если *кортеж* есть *выражение* в *кортеже непомеченной структуры* или *кортеже помеченной структуры*, где соответствующий *список имен полей* состоит только из одного *имени поля*, тогда М — вид поля в *кортеже структуры*, для которой описан кортеж;
- если *кортеж* есть *значение* в вызове встроенных программ *GETSTACK* или *ALLOCATE*, тогда М — вид, обозначенный *аргументом вида*.

Кортеж есть **постоянная** тогда и только тогда, когда каждое *значение* или *выражение*, встречающееся в кортеже, есть **постоянная**.

Статические условия: Необязательное *имя вида* может быть исключено только в контекстах, описанных выше. Должны выполняться следующие правила совместимости в зависимости от того, описан ли *кортеж множества*, *кортеж массива* или *кортеж структуры*:

a. *кортеж множества*

1. Вид *кортежа* должен быть множественным видом.
2. Класс каждого *выражения* должен быть **совместимым** с **элементным** видом вида *кортежа*.
3. Для **постоянного** кортежа множества значение, вырабатываемое каждым *выражением*, должно быть одним из значений, определяемых **элементным** видом.

b. *кортеж массива*

1. Вид *кортежа* должен быть массивным видом.
2. Класс каждого *значения* должен быть **совместимым** с видом **элемента** для вида *кортежа*.
3. В случае *кортежа непомеченного массива* должно быть столько вхождений *значения*, сколько имеется **элементов** массивного вида *кортежа*.
4. В случае *кортежа помеченного массива* должны выполняться условия выбора варианта для списка вхождений *списка меток выбора* (см. раздел 12.3). **Результирующий класс** списка должен быть **совместимым** с **индексным** видом вида *кортежа*. Список спецификаций меток выбора должен быть **полным**.
5. В случае *кортежа помеченного массива* значения, явно указанные каждой меткой выбора из *списка меток выбора*, должны быть значениями, определяемыми **индексным** видом *кортежа*.
6. В *кортеже непомеченного массива* по крайней мере одно вхождение значения должно быть *выражением*.
7. Для **постоянного** *кортежа массива*, где вид **элемента** вида *кортежа* является дискретным видом, каждое описанное *значение* должно представлять значение, определяемое этим видом **элемента** до тех пор, пока оно не является **неопределенным** значением.

c. *кортеж структуры*

1. Вид кортежа должен быть структурным видом.
2. Этот вид не должен быть структурным видом, имеющим **невидимые** имена **полей** (см. раздел 12.2.5).

В случае *кортежа непомеченной структуры*:

- Если вид *кортежа* не является ни **вариантным** структурным видом, ни **параметризованным** структурным видом, тогда:
  3. Должно быть столько вхождений *значения*, сколько имеется имен **полей** в списке имен **полей** вида *кортежа*.
  4. Класс каждого *значения* должен быть **совместимым** с видом соответствующего (по позиции) именем **поля** вида *кортежа*.
- Если вид *кортежа* — **теговый вариантный** структурный вид или **теговый параметризованный** структурный вид, тогда:
  5. Каждое *значение*, описанное для поля **признака**, должно быть *дискретным постоянным выражением*.
  6. Должно быть столько вхождений значения, сколько имеется имен **полей**, определяемых как существующие значением(ями), вырабатываемыми вхождениями *дискретного постоянного выражения*, описанными для полей **признаков**.
  7. Класс каждого *значения* должен быть **совместимым** с видом соответствующего имени **поля**.
- Если вид *кортежа* — **беспризнаковый вариантный** структурный вид или **беспризнаковый параметризованный** структурный вид, тогда:
  8. *Кортеж непомеченной структуры* не допускается.

В случае *кортежа помеченной структуры*:

- Если вид *кортежа* не является ни **вариантным** структурным видом, ни **параметризованным** структурным видом, тогда:

  9. Каждое имя **поля** списка имен **полей** вида *кортежа* должно упоминаться один и только один раз в *списке имен полей* и в том же порядке, как и в виде *кортежа*.

  10. Класс каждого *значения* должен быть **совместимым** с видом каждого описанного имени **поля** из *списка имен полей*, помечающего это *значение*.

- Если вид *кортежа* — **теговый вариантный** структурный вид или **теговый параметризованный** структурный вид, тогда:

  11. Каждое *значение*, описанное для поля **признака**, должно быть *дискретным постоянным выражением*.

  12. Могут быть описаны только имена **полей**, соответствующие полям, определяемым как существующие значением(ями), вырабатываемыми вхождениями *дискретного постоянного выражения*, описанными для полей **признаков**, и все они должны быть описаны и располагаться в том же порядке, как в виде *кортежа*.

  13. Класс каждого *значения* должен быть **совместимым** с видом любого описанного имени **поля** из *списка имен полей*, помечающего это *значение*.

- Если вид *кортежа* — **беспризнаковый вариантный** структурный вид или **беспризнаковый параметризованный** структурный вид, тогда:

  14. Имена **полей**, упомянутые в *списке имен полей*, которые определены в одном и том же *альтернативном поле*, должны быть определены в одной и той же *вариантной альтернативе* или после **ELSE**. Все имена **полей** выбранной вариантной альтернативы или определенные после **ELSE** должны упоминаться один и только один раз в том же порядке, как и в виде *кортежа*.

  15. Класс каждого *значения* должен быть **совместимым** с видом любого описанного имени **поля** из *списка имен полей*, помечающего это *значение*.

16. Если вид *кортежа* — **теговый параметризованный** структурный вид, то список значений, вырабатываемых вхождениями *дискретного постоянного выражения*, описанных для полей **признаков**, должен быть тем же, что и список значений вида *кортежа*.

17. Для **постоянного** *кортежа структуры* каждое описанное *значение* для поля с дискретным видом должно давать значение, определяемое именем **поля** до тех пор, пока оно не является **неопределенным** значением.

18. По крайней мере одно вхождение *значения* должно быть *выражением*.

Ни один *кортеж* не может иметь двух вхождений *значения* в *кортеж*, таких что одно из них является **внерегиональным**, а другое — **внутрирегиональным** (см. раздел 11.2.2).

Динамические условия: Условия присваивания выполняются для любого значения относительно **элементного** вида, вида **элемента** или соответствующего вида **поля** в случае *кортежа множества*, *кортежа массива* или *кортежа структуры* соответственно (см. раздел 6.2) (см. условия a2, b2, c4, c7, c10, c13 и c15).

Если *кортеж* имеет динамический массивный вид и если не выполняется одно из условий (b3 или b5), то имеет место исключение *RANGEFAIL*.

Если *кортеж* имеет динамический **параметризованный** структурный вид и если не выполняется любое из условий (c14 или c16), то имеет место исключение *TAGFAIL*.

Значение, вырабатываемое *кортежем*, не должно быть **неопределенным**.

Примеры:

9.6	*number_list*[ ]	*(1.1)*
9.7	[2:max]	*(2.1)*
8.26	[*('A'):3,('B','K','Z'):1,(ELSE):0*]	*(6.1)*
17.5	[*(*):'* ']	*(6.1)*
12.35	*(:NULL,NULL,536:)*	*(7.1)*
11.18	[*.status:occupied.,p:[white,rook*]]	*(9.1)*

5.2.6 Элементы строки значений

Синтаксис:

< элемент строки значений > :: = *(1)*

< примитивное значение строки > (< начальный элемент >) *(1.1)*

Примечание. — Если *примитивное значение строки* есть *ячейка строки*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *элемент строки* (см. раздел 4.2.6).

Семантика: Элемент строки значений передает значение, являющееся элементом описанного значения строки, обозначенного *начальным элементом*.

Статические свойства: Класс *элемента строки значений* есть класс *BOOL*-значения или класс *CHAR*-значения в зависимости от того, является ли вид *примитивного значения строки* **битовым** строковым видом или **символьным** строковым видом.

Динамические условия: Значение, передаваемое *элементом строки значений*, не должно быть **неопределенным**.

Имеет место исключение *RANGEFAIL*, если не выполняются следующие соотношения:

$$0 \leq NUM\ (\text{начальный элемент}) \leq L-1,$$

где L — **фактическая длина** *примитивного значения строки*.

5.2.7 Подстроки значений

Синтаксис:

< подстрока значений > :: = *(1)*

< примитивное значение строки > (< левый элемент > : < правый элемент >) *(1.1)*

| *< примитивное значение строки > (< начальный элемент >* **UP** *< размер подстроки >)* *(1.2)*

Примечание. — Если *примитивное значение строки* есть *ячейка строки*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *подстрока* (см. раздел 4.2.7).

Семантика: Подстрока значений выдает значение строки (возможно, динамическое), являющееся частью описанного значения строки, указанного *левым элементом* и *правым элементом* или *начальным элементом* и *размером подстроки*. Длина (возможно, динамическая) подстроки определяется из описанных выражений.

Подстрока, *правый элемент* которой вырабатывает значение, которое меньше значения вырабатываемого *левым элементом* или в которой *размер подстроки* дает неположительное значение, обозначает пустую строку.

Статические свойства: Класс (возможно, динамический) *подстроки значений* есть класс М-значения, если *примитивное значение строки* является **строгим**; в противном случае — это М-производный класс, где М — **параметризованный** строковый вид, составленный в форме:

&имя (размер строки),

где *&имя* — имя виртуального **синонимического** вида, **синонимического** с **корневым** (возможно, динамическим) видом *примитивного значения строки*, если это **фиксированный** строковый вид; в противном случае синонимического с **компонентным** видом, а *размер строки* есть либо

NUM (правый элемент) — NUM (левый элемент) + 1,

либо

NUM (размер подстроки).

Однако если обозначена пустая строка, то размер строки равен нулю. Класс *подстроки значений* является статическим, если *размер строки* есть **константа**, то есть *левый элемент* и *правый элемент* являются **константой** или же *размер подстроки* является **константой**; в противном случае класс является динамическим.

Статические условия: Должны выполняться следующие соотношения:

$$0 \leq NUM\ (левый\ элемент) \leq L-1$$

$$0 \leq NUM\ (правый\ элемент) \leq L-1$$

$$0 \leq NUM\ (начальный\ элемент) \leq L-1$$

$$NUM\ (\ начальный\ элемент\ ) + NUM\ (\ размер\ подстроки\ ) \leq L,$$

где L — **фактическая длина** *примитивного значения строки*. Если L и значение всех *целых выражений* известны статически, то соотношения можно проверить статически.

Динамические условия: Значение, вырабатываемое *подстрокой значений*, не должно быть **неопределенным**.

Если динамическая часть проверки вышеприведенных соотношений неуспешна, то имеет место исключение *RANGEFAIL*.

5.2.8 Элементы массива значений

Синтаксис:

< элемент массива значений > :: = *(1)*

< примитивное значение массива > (< список выражений >) *(1.1)*

Примечание. — Если *примитивное значение массива* есть *ячейка массива*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *элемент массива* (см. раздел 4.2.8).

Производный синтаксис: См. раздел 4.2.8.

Семантика: Элемент массива значений дает значение, являющееся элементом описанного значения массива, указанного *выражением*.

Статические свойства: Класс *элемента массива значений* есть класс М-значения, где М — вид **элемента** вида *примитивного значения массива*.

Статические условия: Класс выражения должен быть **совместимым** с **индексным** видом вида *примитивного значения массива*.

Динамические условия: Значение, вырабатываемое *элементом массива значений*, не должно быть **неопределенным**.

Имеет место исключение *RANGEFAIL*, если не выполняется следующее соотношение:

$$L \leq выражение \leq U,$$

где L и U — **нижняя граница** и (возможно, динамическая) **верхняя граница** вида *примитивного значения массива* соответственно.

5.2.9. Подмассивы значений

Синтаксис:

< подмассив значений > :: = *(1)*

< примитивное значение массива >
(< нижний элемент > : < верхний элемент >) *(1.1)*

| *< примитивное значение массива > (< первый элемент >* **UP** *< размер подмассива >)* *(1.2)*

Примечание. — Если *примитивное значение массива* есть *ячейка массива*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *подмассив* (см. раздел 4.2.9).

Семантика: Подмассив значений вырабатывает значение (возможно, динамическое) массива, которое является частью описанного значения массива, указываемого *нижним элементом* и *верхним элементом* или *первым элементом* и *размером подмассива*. **Нижняя граница** подмассива значений равна **нижней границе** описанного значения массива; **верхняя** (возможно, динамическая) **граница** определяется из описанных выражений.

Статические свойства: Класс (возможно, динамический) *подмассива значений* является классом M-значения, где M — **параметризованный** массивный вид, образованный в форме:

& имя (верхний индекс),

где *&имя* — имя виртуального **синонимического** вида, **синонимического** с (возможно, динамическим) видом *примитивного значения массива*, а *верхний индекс* есть выражение, класс которого **совместим** с классами *нижнего элемента* и *верхнего элемента* и которое вырабатывает такое значение, что:

NUM (верхний индекс) = NUM (L) + NUM (верхний элемент) — NUM (нижний элемент),

или выражение, класс которого **совместим** с классом *первого элемента* и которое вырабатывает такое значение, что:

NUM (верхний индекс) = NUM (L) + NUM (размер подмассива) — 1,

где *L* — **нижняя граница** вида *примитивного значения массива*.

Класс *подмассива значений* является статическим, если *верхний индекс* есть **константа**, то есть и *нижний элемент*, и *верхний элемент* являются **константой** или *размер подмассива* является **константой**; в противном случае класс будет динамическим.

Статические условия: Классы *нижнего элемента* и *верхнего элемента* или класс *первого элемента* должны быть **совместимыми** с **индексным** видом *примитивного значения массива*.

Должны выполняться следующие соотношения:

$L \leq$ *нижний элемент* $\leq$ *верхний элемент* $\leq U$

$1 \leq$ *NUM (размер подмассива)* $\leq$ *NUM (U) — NUM (L) + 1*

NUM (L) $\leq$ *NUM (первый элемент)* $\leq$ *NUM (первый элемент) + NUM (размер подмассива) — 1* $\leq$ *NUM (U),*

где *L* и *U* — соответственно, **нижняя граница** и **верхняя граница** вида *примитивного значения массива*. Если *U* и значение всех *выражений* известны статически, то соотношения могут быть проверены статически.

Динамические условия: Значение, вырабатываемое *подмассивом значений*, не должно быть **неопределенным**.

Если динамическая часть проверки соотношений неуспешна, то имеет место исключение *RANGEFAIL*.

5.2.10 Поля структуры значений

Синтаксис:

< поле структуры значений > :: = (1)
< примитивное значение структуры > . < имя поля > (1.1)

Примечание. — Если *примитивное значение структуры* есть *ячейка структуры*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *поле структуры* (см. раздел 4.2.10).

Семантика: Поле структуры значений вырабатывает значение, являющееся полем описанного значения структуры, указанного *именем поля*. Если *примитивное значение структуры* имеет **беспризнаковый вариантный** структурный вид, а *имя поля* есть имя **вариантного поля**, то семантика определяется реализацией.

Статические свойства: Класс *поля структуры значений* есть класс М—значения, где М — вид *имени поля*.

Статические условия: *Имя поля* должно быть именем из набора имен **полей** вида *примитивного значения структуры*.

Динамические условия: Значение, вырабатываемое *полем структуры значений*, не должно быть **неопределенным**.

Значение не должно обозначать:

- значение **тегового вариантного** структурного вида, в котором соответствующее(ие) значение(ия) поля **признака** указывает(ют) на то, что обозначенное поле не существует;
- значение динамического **параметризованного** структурного вида, соответствующий список значений которого указывает на то, что поле не существует.

Упомянутые выше условия называются условиями доступа к вариантному полю для значения (заметим, что в условиях не предусмотрено исключение). Исключение *TAGFAIL* имеет место, если условия для *примитивного значения структуры* не выполняются.

Примеры:

16.51 (**RECEIVE** *user_buffer*).*allocator* (1.1)

5.2.11 Преобразования выражений

Синтаксис:

< преобразование выражения > :: = (1)
< имя вида > (< выражение >) (1.1)

Примечание. — Если *выражение* есть *ячейка статического вида*, то синтаксическая конструкция является неопределенной и будет интерпретироваться как *преобразование ячейки* (см. раздел 4.2.13).

Семантика: Преобразование выражений не принимает во внимание проверку вида и правила совместимости в языке CHILL. Оно явным образом приписывает вид выражению. Если вид *имени вида* есть дискретный вид, а класс значения, вырабатываемого выражением, является дискретным, то значение, вырабатываемое преобразованием выражения, таково что:

NUM (имя вида (выражение)) = NUM (выражение).

В противном случае значение, вырабатываемое преобразованием выражения, определяется реализацией и зависит от внутреннего представления значений.

Статические свойства: Класс *преобразования выражения* есть класс М—значения, где М — *имя вида*. *Преобразование выражения* есть **постоянная** тогда и только тогда, когда *выражение* есть **постоянная**.

Статические условия: *Имя вида* не должно обладать **свойством беззначимости.** Реализация может налагать дополнительные статические условия.

Динамические условия: Если класс значения, вырабатываемого *выражением*, дискретный и если вид *имени вида* есть дискретный вид, не определяющий значение с внутренним представлением, равным *NUM (выражение)*, тогда имеет место исключение *OVERFLOW*. Реализация может налагать дополнительные динамические условия, нарушение которых приводит к исключению, определяемому реализацией.

5.2.12 Вызовы процедур возврата значения

Синтаксис:

< вызов процедуры возврата значения > :: = *(1)*
< вызов процедуры возврата значения > *(1.1)*

Семантика: Вызов процедуры возврата значения выдает значение, возвращаемое из процедуры.

Статические свойства: Класс *вызова процедуры возврата значения* есть класс M-значения, где M — вид **спецификации результата** *вызова процедуры возврата значения.*

Динамические условия: *Вызов процедуры возврата значения* не должен выдавать **неопределенное** значение (см. разделы 5.3.1 и 6.8).

Примеры:

6.50 julian_day_number([10,dec,1979]) *(1.1)*
11.63 ok_bishop(b,m) *(1.1)*

5.2.13 Вызовы встроенных программ выдачи значения

Синтаксис:

< вызов встроенной программы выдачи значения > :: = *(1)*
< вызов встроенной программы выдачи значения > *(1.1)*

Семантика: Вызов встроенной программы выдачи значения выдает значение, возвращаемое встроенной программой.

Статические свойства: Класс, приписанный *вызову встроенной программы выдачи значения* является классом *встроенной программы выдачи значения.*

Динамические условия: *Вызов встроенной программы выдачи значения* не должен выдавать **неопределенное** значение (см. разделы 5.3.1 и 6.8).

5.2.14 Выражения запуска

Синтаксис:

< выражение запуска > :: = (1)
START *< имя процесса >* ([*< список фактических параметров >*]) (1.1)

Семантика: Вычисление выражения запуска создает и активирует новый процесс, определение которого указывается *именем процесса* (см. главу 11). Выражение запуска вырабатывает значение экземпляра, идентифицирующее созданный процесс. Передача параметров аналогична передаче параметров процедуры; однако могут быть даны дополнительные фактические параметры, смысл которых определяется реализацией.

Статические свойства: Класс *выражения запуска* есть *INSTANCE*-производный класс.

Статические условия: Количество вхождений *фактического параметра* в *списке фактических параметров* не должно быть меньше количества вхождений *формального параметра* в *списке формальных параметров* определения процесса *имени процесса*. Если количество фактических параметров равно m, а количество формальных параметров — n ($m \geq n$), то требования по совместимости и **региональности** для первых n фактических параметров являются теми же, что и для передачи параметров процедуры (см. раздел 6.7). Статические условия для остальных фактических параметров определяются реализацией.

Динамические условия: При передаче параметров используются условия присваивания любого фактического значения по отношению к виду связанного с ним формального параметра (см. раздел 6.7).

Если требования памяти не могут быть удовлетворены, то *выражение запуска* является причиной исключения *SPACEFAIL*.

Примеры:

15.35 **START** *counter( )* (1.1)

5.2.15 Операция индикации

Синтаксис:

< операция индикации > :: = (1)
THIS (1.1)

Семантика: Операция индикации создает единственное значение экземпляра, индентифицирующее процесс, выполнивший данную операцию.

Статические свойства: Класс *операции индикации* есть *INSTANCE*-производный класс.

5.2.16 Выражение в круглых скобках

Синтаксис:

< выражение в круглых скобках > :: = (1)
(*< выражение >*) (1.1)

Семантика: Выражение в круглых скобках вырабатывает значение, полученное в результате вычисления выражения.

Статические свойства: Класс *выражения в круглых скобках* есть класс *выражения*.

Выражение в круглых скобках есть **постоянная (константа)** тогда и только тогда, когда *выражение* есть **постоянная (константа)**.

Примеры:

5.10 *(a1* **OR** *b1)* *(1.1)*

5.3 ЗНАЧЕНИЯ И ВЫРАЖЕНИЯ

5.3.1 Общие положения

Синтаксис:

< значение > :: = *(1)*
< выражение > *(1.1)*
| *< неопределенное значение >* *(1.2)*

< неопределенное значение > :: = *(2)*
* *(2.1)*
| *< имя неопределенного синонима >* *(2.2)*

Семантика: Значение — это либо **неопределенное** значение, либо значение (определяемое в языке CHILL), полученное в результате вычисления выражения.

Если явно не указано иное, то порядок вычисления составляющих выражения и их субсоставляющих и т.д. является неопределенным, и можно считать, что они вычисляются в смешанном порядке. Их необходимо только вычислить к моменту, когда выдаваемое значение определяется однозначно. Если в контексте требуется, чтобы выражение было **постоянным** или **константой**, то предполагается, что вычисление производится до момента начала выполнения программы и не может быть причиной исключения. В реализации задаются диапазоны допустимых значений для выражений, являющихся **постоянными** или **константами**, а программа может быть отключена, если в результате таких вычислений до момента начала выполнения программы получено значение, выходящее за границы, определяемые реализацией.

Статические свойства: Класс *значения* есть класс *выражения* или *неопределенного значения* соответственно.

Класс *неопределенного значения* есть **полный** класс, если *неопределенное значение* есть *; в противном случае класс есть класс *имени неопределенного синонима*.

Значение есть **постоянная** тогда и только тогда, когда это — *неопределенное значение* или *выражение*, являющееся **постоянной**. *Значение* является **константой** тогда и только тогда, когда это — *выражение*, являющееся **константой**.

Динамические свойства: Считается, что значение является **неопределенным**, если оно обозначено *неопределенным значением* или явно указано в данном документе. Составное значение является **неопределенным** тогда и только тогда, когда все его субкомпоненты (то есть значения подстрок, значения элементов, значения полей) являются **неопределенными**.

Примеры:

6.40 *(146_097*c)/4 +(1_461*y)/4*
+(153+m+c)/5+day+1_721_119 *(1.1)*

5.3.2 Выражения

Синтаксис:

< *выражение* > :: = (1)
< *операнд-0* > (1.1)
| < *условное выражение* > (1.2)

< *условное выражение* > :: = (2)
| **IF** < *булевское выражение* > < *then-альтернатива* >
< *else-альтернатива* > **FI** (2.1)

| **CASE** < *список селекторов выбора* > **OF** { < *альтернатива выбора значения* > } +
[**ELSE** < *субвыражение* >] **ESAC** (2.2)

< *then-альтернатива* > :: = (3)
THEN < *субвыражение* > (3.1)

< *else-альтернатива* > :: = (4)
ELSE < *субвыражение* > (4.1)
| **ELSIF** < *булевское выражение* >
< *then-альтернатива* > < *else-альтернатива* > (4.2)

< *субвыражение* > :: = (5)
< *выражение* > (5.1)

< *альтернатива выбора значения* > :: = (6)
< *спецификация метки выбора* > : < *субвыражение* > ; (6.1)

Семантика: Если описан атрибут **IF**, то вычисляется *булевское выражение*; если в итоге получается *TRUE*, то результатом является значение, вырабатываемое *субвыражением* в *then-альтернативе*, в противном случае результат — это значение, вырабатываемое *else-альтернативой*.

Значение, вырабатываемое *else-альтернативой* является значением *субвыражения*, если описан атрибут **ELSE**, в противном случае вычисляется *булевское выражение* и если в результате получается *TRUE*, то это значение является значением, вырабатываемым *субвыражением* в *then-альтернативе*, в противном случае результатом является значение, вырабатываемое *else-альтернативой*.

Если описан атрибут **CASE**, то вычисляются *субвыражения* в *списке селекторов выбора*; если *спецификация метки выбора* совпадает, то результатом является значение, вырабатываемое соответствующим *субвыражением*, в противном случае результатом является значение, вырабатываемое *субвыражением*, следующим за **ELSE** (который имеется).

Неиспользованные *субвыражения* в *условном выражении* не вычисляются.

Статические свойства: Если выражение есть *операнд-0*, то класс *выражения* есть класс *операнда-0*. *Если это условное выражение*, то класс *выражения* есть класс М-значения, где М—вид, зависящий от контекста, в который входит *условное выражение* согласно тем правилам, которые определяют вид класса кортежа без *имени вида* (см. раздел 5.2.5).

Выражение есть **постоянная (константа)** тогда и только тогда, когда оно есть либо *операнд-0*, который является **постоянной (константой)**, либо *условное выражение*, в котором все *булевские выражения* или *список селекторов выбора* являются **постоянной (константой)** и в котором все *субвыражения* являются **постоянной (константой)**.

Статические условия: Если *выражение* — это *условное выражение*, то применяются следующие положения:

- *условное выражение* может входить только в контексты, в которых перед ним входит кортеж без *имени вида*;
- каждое *субвыражение* должно быть **совместимым** с видом, полученным из контекста с теми же правилами, что и для кортежей. Однако динамическая часть отношения совместимости применяется только к выбранному *субвыражению*;

- если описан атрибут **CASE**, то должны выполняться условия выбора варианта (см. раздел 12.3) и те же требования по совместимости, непротиворечивости и полноте, что и для оператора выбора (см. раздел 6.4);

- ни одно из *условных выражений* не должно содержать таких двух вхождений *субвыражений*, чтобы одно из них было **внерегиональным**, а другое — **внутрирегиональным** (см. раздел 11.2.2).

Динамические условия: В случае *условного выражения* применяются условия присваивания значения, выработанного выбранным *субвыражением*, по отношению к виду M, полученному из контекста.

5.3.3 Операнд-0

Синтаксис:

< операнд -0 > :: = *(1)*
< операнд-1 > *(1.1)*
| *< субоперанд-0 >* { **OR** | **ORIF** | **XOR** } *< операнд-1 >* *(1.2)*

< субоперанд-0 > :: = *(2)*
< операнд-0 > *(2.1)*

Семантика: Если описаны **OR**, **ORIF** или **XOR**, то *субоперанд-0* и *операнд-1* вырабатывают:

- булевские значения, в случае которых **OR** и **XOR** обозначают соответственно операции "включающая дизъюнкция" и "исключающая дизъюнкция", вырабатывающие булевское значение. Если описан **ORIF**, а *операнд-0* вырабатывает булевское значение *TRUE*, то он будет результатом; в противном случае результатом является *операнд-1*;

- битовые строковые значения, в случае которых **OR** или **XOR** обозначают логические операции над каждым элементом битовых строк, вырабатывающие битовые строковые значения;

- значения множеств, в случае которых **OR** обозначает объединение обоих значений множеств, а **XOR** обозначает значение множества, состоящее из таких значений элементов, которые находятся только в одном из описанных значений множеств (например, A **XOR** B = A—B **OR** B—A).

Статические свойства: Если *операнд-0* есть *операнд-1*, то класс *операнда-0* есть класс *операнда-1*. Если описаны **OR**, **ORIF** или **XOR**, то класс *операнда-0* есть **результирующий** класс классов *субоперанда-0* и *операнда-1*.

Операнд-0 есть **постоянная (константа)** тогда и только тогда, когда он является либо *операндом-1*, который есть **постоянная (константа)**, либо состоит из *операнда-0* и *операнда-1*, которые оба являются **постоянной (константой)**.

Статические условия: Если описаны **OR**, **ORIF** или **XOR**, то класс *субоперанда-0* должен быть **совместимым** с классом *операнда-1*. Если описан **ORIF**, то оба класса должны иметь булевский **корневой** вид, в противном случае оба класса должны иметь булевский, множественный или битовый строковый **корневой** вид, в случае которых **фактическая длина** *субоперанда-0* и *операнда-1* должна быть одной и той же. Эта проверка является динамической, если один или оба вида являются динамическими или **изменяющимися** строковыми видами.

Динамические условия: В случае **OR** или **XOR** имеет место исключение *RANGEFAIL*, если один или оба операнда имеют динамический класс и динамическая часть упомянутой выше проверки на совместимость неуспешна.

Примеры:

10.31 *i< min* *(1.1)*
10.31 *i< min* **OR** *i >max* *(1.2)*

5.3.4 Операнд-1

Синтаксис:

< операнд-1 > :: = *(1)*
< операнд-2 > *(1.1)*
| *< субоперанд-1 >* { **AND** | **ANDIF** } *< операнд-2 >* *(1.2)*

< субоперанд-1 > :: = *(2)*
< операнд-1 > *(2.1)*

Семантика: Если описаны **AND** и **ANDIF**, то субоперанд-1 и операнд-2 вырабатывают:

- булевские значения, в случае которых **AND** обозначает логическую операцию "конъюнкции", вырабатывающую булевское значение. Если описан **ANDIF**, а *субоперанд-1* вырабатывает булевское значение *FALSE*, тогда он является результатом; в противном случае результатом является *операнд-2*;
- битовые строковые значения, в случае которых **AND** обозначает логическую операцию над каждым элементом битовых строк, вырабатывающую битовое строковое значение;
- значение множеств, в случае которых **AND** обозначает операцию пересечения значений множеств, вырабатывающую в качестве результата значение множества.

Статические свойства: Если *операнд-1* есть *операнд-2*, то класс *операнда-1* есть класс *операнда-2*.

Если описаны **AND** или **ANDIF**, класс *операнда-1* является **результирующим классом** классов *субоперанда-1* и *операнда-2*.

Операнд-1 является **постоянной (константой)** тогда и только тогда, когда он является либо *операндом-2*, который есть **постоянная (константа)**, либо составлен из *операнда-1* и *операнда-2*, которые оба являются **постоянной (константой)**.

Статические условия: Если описаны **AND** или **ANDIF**, то класс *субоперанда-1* должен быть **совместимым** с классом *операнда-2*. Если описан **ANDIF**, то оба класса должны иметь булевский **корневой** вид; в противном случае оба класса должны иметь булевский, множественный или **битовый** строковый **корневой** вид, в случае которых **фактическая длина** *субоперанда-1* и *операнда-2* должна быть одной и той же. Эта проверка является динамической, если один или оба вида являются динамическими или **изменяющимися** строковыми видами.

Динамические условия: В случае **AND** имеет место исключение *RANGEFAIL*, если один или оба операнда имеют динамический класс и динамическая часть вышеупомянутой проверки на совместимость неуспешна.

Примеры:

5.10 *(a1* **OR** *b1)* *(1.1)*
5.10 **NOT** *k2* **AND** *(a1* **OR** *b1)* *(1.2)*

5.3.5 Операнд-2

Синтаксис:

< операнд-2 > :: = *(1)*
< операнд-3 > *(1.1)*
| *< субоперанд-2 > < операция-3 > < операнд-3 >* *(1.2)*

< субоперанд-2 > :: = *(2)*
< операнд-2 > *(2.1)*

< операция-3 > :: = *(3)*
< операция сравнения > *(3.1)*
| *< операция принадлежности >* *(3.2)*
| *< операция включения множества >* *(3.3)*

< операция сравнения > :: = *(4)*

= | / = | > | > = | < | < = *(4.1)*

< операция принадлежности > :: = *(5)*

IN *(5.1)*

< операция включения множества > :: = *(6)*

< = | >= | < | > *(6.1)*

Семантика: Операции равенства (=) и неравенства (/=) определены между значениями данного вида. Другие операции сравнения (меньше чем: < , меньше чем или равно: < =, больше чем: >, больше чем или равно: >=) определены между значениями данного дискретного, временного или строкового вида. Все операции сравнения вырабатывают в качестве результата булевское значение.

Операция принадлежности определена между значением элемента и значением множества. Операция вырабатывает *TRUE*, если значение элемента находится в значении описанного множества, в противном случае она вырабатывает *FALSE*.

Операции включения множества определены между значениями множеств; они проверяют, содержится ли значение множества (< =), строго говоря, содержится ли значение множества (<) в другом значении множества, а также содержит ли значение множества (>=), строго говоря, содержит ли значение множества (>) другое значение множества. Операция включения множества вырабатывает в качестве результата булевское значение.

Статические свойства: Если *операнд-2* есть *операнд-3*, то класс *операнда-2* есть класс *операнда-3*. Если описана *операция-3*, то класс *операнда-2* есть *BOOL*-производный класс.

Операнд-2 есть **постоянная (константа)** тогда и только тогда, когда он либо *операнд-3*, являющийся **постоянной (константой)**, либо образован из *субоперанда-2* и *операнда-3*, которые оба являются **постоянной (константой)**.

Статические условия: Если описана *операция-3*, то должны выполняться следующие требования по совместимости между классом *субоперанда-2* и классом *операнда-3*, а именно:

- если *операция-3* есть = или /=, то оба класса должны быть **совместимыми**;
- если *операция-3* есть *операция сравнения*, отличная от = или /=, то оба класса должны быть **совместимыми** и должны иметь дискретный, временной или строковый **корневой** вид;
- если *операция-3* есть *операция принадлежности*, то класс *операнда-3* должен иметь множественный **корневой** вид, а класс *субоперанда-2* должен быть **совместимым** с **элементным** видом этого **корневого** вида;
- если *операция-3* есть *операция включения множества*, то оба класса должны быть **совместимыми** и должны иметь множественный **корневой** вид.

Динамические условия: В случае *операции сравнения* имеют место исключения *RANGEFAIL* или *TAGFAIL*, если один или оба операнда имеют динамический класс и динамическая часть упомянутой выше проверки на совместимость неуспешна. Исключение *TAGFAIL* имеет место тогда и только тогда, когда динамический класс базируется на динамическом **параметризованном** структурном виде.

Примеры:

10.50 NULL *(1.1)*

10.50 last=NULL *(1.2)*

5.3.6 Операнд-3

Синтаксис:

< операнд-3 > :: = (1)
< операнд-4 > (1.1)
| *< субоперанд-3 > < операция-4 > < операнд-4).* (1.2)

< субоперанд-3 > :: = (2)
< операнд-3 > (2.1)

< операция-4 > :: = (3)
< арифметическая аддитивная операция > (3.1)
| *< операция конкатенации строк >* (3.2)
| *< операция разности множеств >* (3.3)

< арифметическая аддитивная операция > :: = (4)
+ | − (4.1)

< операция конкатенации строк > :: = (5)
// (5.1)

< операция разности множеств > :: = (6)
− (6.1)

Семантика: Если *операция-4* есть арифметическая аддитивная операция, то оба операнда вырабатывают целые значения и результирующее значение является суммой (+) или разностью (−) двух значений.

Если *операция-4* есть операция конкатенации строк, то оба операнда вырабатывают либо значения битовых строк, либо значения символьных строк; результирующее значение состоит из конкатенации этих значений. Булевские (символьные) значения также допустимы; они рассматриваются как значения битовых (символьных) строк длины 1.

Если *операция-4* есть операция разности множеств, то оба операнда вырабатывают значения множеств, и результирующим значением является значение множества, состоящее из тех значений элементов, которые содержатся в значении, вырабатываемом *субоперандом-3*, и не содержатся в значении, вырабатываемом *операндом-4*.

Статические свойства: Если *операнд-3* есть *операнд-4*, то класс *операнда-3* есть класс *операнда-4*. Если описана *операция-4*, то класс *операнда-3* определяется *операцией-4* следующим образом:

- если *операция-4* есть *операция конкатенации строк*, то класс *операнда-3* зависит от классов *операнда-4* и *субоперанда-3*, в которых операнд, являющийся булевским или символьным значением, рассматривается как значение, класс которого есть **BOOLS** (1)-производный класс или **CHARS** (1)-производный класс соответственно:

  — если ни один из них не является **строгим**, то класс есть **BOOLS** (*n*)-производный класс или **CHARS** (*n*)-производный класс в зависимости от того, являются ли оба операнда битовыми или символьными строками, где *n* — сумма **длин строк корневых** видов обоих классов;

  — в противном случае класс есть класс *&имя* (*n*)-значения, где *&имя* — имя виртуального **синонимического вида**, **синонимического** с **корневым** видом **результирующего класса** классов операндов, а *n* — сумма **длин строк корневых** видов обоих классов

  (этот класс является динамическим, если один или оба операнда имеют динамический класс).

- если *операция-4* есть *арифметическая аддитивная операция* или *операция разности множеств*, то класс *операнда-3* есть **результирующий класс** классов *операнда-4* и *субоперанда-3*.

Операнд-3 есть **постоянная (константа)** тогда и только тогда, когда он есть либо *операнд-4*, являющийся **постоянной (константой)**, либо образован из *операнда-3* и *операнда-4*, которые оба являются **постоянной (константой)**, а *операция-4* есть либо *арифметическая аддитивная операция*, либо *операция разности множеств*.

Если *операция-4* есть *операция конкатенации строк*, то *операнд-3* есть **постоянная**, если он образован из *операнда-3* и *операнда-4*, которые оба являются **постоянной**.

Статические условия: Если описана *операция-4*, то должны выполняться следующие правила совместимости:

- если *операция-4* есть *арифметическая аддитивная операция*, то классы обоих операндов должны быть **совместимыми** и оба должны иметь целый **корневой** вид;
- если *операция-4* есть *операция конкатенации строк*, тогда:
  - — классы обоих операндов должны быть **совместимыми** и оба должны иметь **битовый** строковый **корневой** вид или **символьный** строковый **корневой** вид, или
  - — классы обоих операндов должны быть **совместимыми** с видом *BOOL*, и оба должны быть **совместимыми** с видом *CHAR*;
  - — класс одного операнда должен иметь **битовый (символьный)** строковой **корневой** вид, а класс другого операнда должен быть **совместимым** с видом *BOOL* (*CHAR*);
- если *операция-4* есть *операция разности множеств*, то классы обоих операндов должны быть **совместимыми**, и оба операнда должны иметь множественный **корневой** вид.

Динамические условия: В случае *операнда-3*, не являющегося **постоянной**, имеет место исключение *OVERFLOW*, если сложение (+) или вычитание (–) приводят к значению, которое не является одним из значений, определяемых **корневым** видом класса *операнда-3*.

Примеры:

1.6 *j* *(1.1)*
1.6 *i+j* *(1.2)*

5.3.7 Операнд-4

Синтаксис:

< *операнд-4* > :: = *(1)*
< *операнд-5* > *(1.1)*
| < *субоперанд-4* > < *арифметическая мультипликативная операция* >)
< *операнд-5* > *(1.2)*

< *субоперанд-4* > :: = *(2)*
< *операнд-4* > *(2.1)*

< *арифметическая мультипликативная операция* > :: = *(3)*
* | / | **MOD** | **REM** *(3.1)*

Семантика: Если описана арифметическая мультипликативная операция, то *субоперанд-4* и *операнд-5* вырабатывают целые значения, а результирующим целым значением будут произведение (*), частное (/), взятие по модулю (**MOD**) или остаток от деления (**REM**) обоих значений.

Операция взятия по модулю определяется таким образом, что *i* **MOD** *j* вырабатывает единственное целое значение *k*, $0 \leq k < j$, такое что существует целое значение *n*, для которого $i = n * j + k$; *j* должно быть больше нуля.

Операция получения частного определяется таким образом, что все соотношения

$ABS(x/y) = ABS(x)/ABS(y)$ *и*
$sign(x/y) = sign(x)/sign(y)$ *и*
$ABS(x) - (ABS(x)/ABS(y)*ABS(y) = ABS(x)$**MOD** $ABS(y)$

выдают значения *TRUE* для всех целых значений *x* и *y*, где sign(*x*) = —1, если $x < 0$, в остальных случаях sign(*x*) = 1.

Операция получения остатка от деления определяется таким образом, что *x* **REM** $y = x - (x/y)^{*}y$ выдает значение *TRUE* для всех целых значений *x* и *y*.

Статические свойства: Если *операнд-4* есть *операнд-5*, то класс *операнда-4* есть класс *операнда-5*; в противном случае класс *операнда-4* есть **результирующий класс** классов *субоперанда-4* и *операнда-5*.

Операнд-4 есть **постоянная (константа)** тогда и только тогда, когда он есть либо *операнд-5*, являющийся **постоянной (константой)**, либо образован из *операнда-4* и *операнда-5*, которые оба являются **постоянными (константами)**.

Статические условия: Если описана *арифметическая мультипликативная операция*, то классы *операнда-5* и *субоперанда-4* должны быть **совместимыми**, и оба класса должны иметь целый **корневой** вид.

Динамические условия: В случае *операнда-4*, не являющегося **постоянной**, имеет место исключение *OVERFLOW*, если операции умножения (*), деления (/), взятия по модулю (**MOD**) или получения остатка от деления (**REM**) приводят к значению, не являющемуся ни одним из значений, определяемых **корневым** видом класса *операнда-4*, либо эти операции выполняются над значениями операнда, для которых операция математически не определена, то есть при делении или получении остатка с *операндом-5* получается нуль или операция взятия по модулю с *операндом-5* приводит к неположительному целому значению.

Примеры:

6.15 *1_461* *(1.1)*
6.15 *(4 * d+3)/1_461* *(1.2)*

5.3.8 Операнд-5

Синтаксис:

< *операнд-5* > :: = *(1)*
[< *унарная операция* >] < *операнд-6* > *(1.1)*

< *унарная операция* > :: = *(2)*
— | **NOT** *(2.1)*
| < *операция повторения строки* > *(2.2)*

< *операция повторения строки* > :: = *(3)*
(< *целое постоянное выражение* >) *(3.1)*

Семантика: Если унарная операция есть операция изменения знака (—), то *операнд-6* представляет целое значение, а результирующее целое значение — это предыдущее целое значение с измененным знаком.

Если унарная операция есть **NOT**, то *операнд-6* вырабатывает булевское значение, битовое строковое значение или значение множества. В первых двух случаях происходит логическое отрицание булевского значения или элементов битового строкового значения. В последнем случае вырабатывается значение дополнения множества, то есть набор таких значений элементов, которые не входят в значение множества операнда.

Если унарная операция есть операция повторения строки, то *операнд-6* есть *символьная строковая константа* или *битовая строковая константа*. Если *целое постоянное выражение* вырабатывает 0, то результатом является значение пустой строки; в противном случае результатом является значение строки, образованной в результате конкатенации одной и той же строки с самой собой, причем конкатенация выполняется столько раз, сколько задается значением, вырабатываемым постоянным выражением, минус 1.

Статические свойства: Если *операнд-5* есть *операнд-6*, то класс *операнда-5* есть класс *операнда-6*.

Если описана *унарная операция*, то класс *операнда-5* есть:

- **результирующий класс** *операнда-6*, если *унарная операция* есть — или **NOT**;
- **CHARS** (*n*)-производный или **BOOLS** (*n*)-производный класс (в зависимости от того, была ли константа *символьной строковой константой* или *битовой строковой константой*), где $n = r * l$, и r — значение, вырабатываемое *целым постоянным выражением*, а l — **длина строки** строковой константы, если *унарная операция* есть *операция повторения строки*.

Операнд-5 есть **постоянная** тогда и только тогда, когда *операнд-6* есть **постоянная**. *Операнд-5* есть **константа** тогда и только тогда, когда *операнд-6* есть **константа**, а *унарная операция* есть — или **NOT**.

Статические условия: Если *унарная операция* есть "–", то класс *операнда-6* должен иметь целый **корневой** вид.

Если *унарная операция* есть **NOT**, то класс *операнда-6* должен иметь булевский, **битовый** строковый или множественный **корневой** вид.

Если *унарная операция* есть *операция повторения строки*, то *операнд-6* должен быть *символьной строковой константой* или *битовой строковой константой*. *Целое постоянное выражение* должно вырабатывать неотрицательное целое значение.

Динамические условия: Если *операнд-5* не есть **постоянная**, то имеет место исключение *OVERFLOW*, если операция изменения знака (—) приводит к значению, которое не является ни одним из значений, определяемых **корневым** видом класса *операнда-5*.

Примеры:

5.10 **NOT** *k2* *(1.1)*
7.54 *(6)* " *(1.1)*
7.54 *(6)* *(2.2)*

5.3.9 Операнд-6

Синтаксис:

< операнд-6 > :: = *(1)*
< ссылаемая ячейка > *(1.1)*
| *< выражение получения >* *(1.2)*
| *< примитивное значение >* *(1.3)*

< ссылаемая ячейка > :: = *(2)*
—> *< ячейка >* *(2.1)*

< выражение получения > :: = *(3)*
RECEIVE *< буферная ячейка >* *(3.1)*

Семантика: Ссылаемая ячейка выдает ссылку на описанную ячейку.

Выражение получения извлекает значение из *буферной ячейки*. Процесс, выполняющий действие, может стать задержанным и может реактивировать другой процесс, задержанный по посылке в описанную буферную ячейку (подробные сведения содержатся в разделе 6.19.3).

Статические свойства: Класс *операнда-6* есть класс *ссылаемой ячейки*, *выражения получения* или *примитивного значения* соответственно. Класс *ссылаемой ячейки* есть M-ссылочный класс, где M — вид *ячейки*. Класс *выражения получения* есть класс M-значения, где M — вид **буферного элемента** вида *буферной ячейки*.

Операнд-6 есть **постоянная** тогда и только тогда, когда *примитивное значение* есть **постоянная** либо *ссылаемая ячейка* есть **постоянная**. *Ссылаемая ячейка* есть **постоянная** тогда и только тогда, когда *ячейка* является **статической**. *Операнд-6* есть **константа** тогда и только тогда, когда *примитивное значение* есть **константа**.

Статические условия: *Ячейка* должна быть **ссылочной**.

Динамические условия: Время жизни *буферной ячейки* не должно заканчиваться, пока действующий процесс задерживается по ней.

Примеры:

8.25 —> *c* *(2.1)*
16.51 **RECEIVE** *user_buffer* *(3.1)*

6 ДЕЙСТВИЯ

6.1 ОБЩИЕ ПОЛОЖЕНИЯ

Синтаксис:

< оператор действия > :: = (1)

[*< определяющее вхождение >:*] *< действие >* [*< программа обработки исключений >*] [*< строка простого имени >*] ; (1.1)

| *< модуль >* (1.2)

| *< модуль спецификации >* (1.3)

| *< контекстовый модуль >* (1.4)

< действие > :: = (2)

< сложный оператор > (2.1)

| *< оператор присваивания >* (2.2)

| *< оператор вызова >* (2.3)

| *< оператор выхода >* (2.4)

| *< оператор возврата >* (2.5)

| *< оператор результата >* (2.6)

| *< оператор перехода >* (2.7)

| *< оператор контроля >* (2.8)

| *< пустой оператор >* (2.9)

| *< оператор запуска >* (2.10)

| *< оператор останова >* (2.11)

| *< оператор задержки >* (2.12)

| *< оператор продолжения >* (2.13)

| *< оператор посылки >* (2.14)

| *< оператор причины >* (2.15)

< сложный оператор > :: = (3)

< условный оператор > (3.1)

| *< оператор выбора >* (3.2)

| *< оператор цикла >* (3.3)

| *< блок begin-end >* (3.4)

| *< оператор выбора задержки >* (3.5)

| *< оператор варианта получения >* (3.6)

| *< оператор временных соотношений >* (3.7)

Семантика: Операторы действия содержат алгоритмическую часть CHILL-программы. Любой оператор действия может быть помечен. Те действия, которые никогда не приведут к исключениям, никогда не будут иметь прилагаемой программы обработки исключений.

Статические свойства: *Определяющее вхождение* в *операторе действия* определяет имя **метки**.

Статические условия: *Строка простого имени* может выдаваться только после *действия*, являющегося *сложным оператором* или если специфицирована программа *обработки исключений* и только если специфицировано *определяющее вхождение*. *Строка простого имени* должна быть той же строкой имени, что и *определяющее вхождение*.

6.2 ОПЕРАТОР ПРИСВАИВАНИЯ

Синтаксис:

< оператор присваивания > :: = (1)

< оператор однократного присваивания > (1.1)

| *< оператор многократного присваивания >* (1.2)

< оператор однократного присваивания > (2)

< ячейка > < символ присваивания > < значение > (2.1)

| *< ячейка > < операция присваивания > < выражение >* (2.2)

< оператор многократного присваивания > :: = (3)

< ячейка > { , *< ячейка >* }$^{+}$ *< символ присваивания > < значение >* (3.1)

< операция присваивания > :: = *(4)*

< замкнутая двоичная операция > < символ присваивания > *(4.1)*

< замкнутая двоичная операция > :: = *(5)*

OR | XOR | AND *(5.1)*

| *< операция разности множеств >* *(5.2)*

| *< арифметическая аддитивная операция >* *(5.3)*

| *< арифметическая мультипликативная операция >* *(5.4)*

| *< операция конкатенации строк >* *(5.5)*

< символ присваивания > :: = *(6)*

: = *(6.1)*

Семантика: Оператор присваивания заносит значение в одну или несколько ячеек.

При использовании символа присваивания значение справа записывается в ячейку(и), специфицированную(ые) с правой стороны.

При использовании операции присваивания значение, содержащееся в ячейке, соединяется со значением с правой стороны (в этом порядке) в соответствии с семантикой специфицированной замкнутой двоичной операцией, и результат снова заносится в ту же ячейку.

Вычисления ячейки(ек) слева, значения с правой стороны и самих присваиваний выполняются неопределенным и даже смешанным образом. Любое присваивание может быть выполнено, как только будут вычислены значение и ячейка.

Если ячейка (или какие-либо ячейки) является полем **признаков** вариантной структуры, то семантика полей признаков, зависящая от вариантной структуры, определяется реализацией.

Статические условия: Виды всех вхождений *ячеек* должны быть **эквивалентными** и не должны обладать ни **свойством неизменяемости**, ни **свойством безначимости**. Каждый вид должен быть **совместимым** с классом *значения*. Проверки являются динамическими в случае, когда имеются ячейки динамического вида и/или значение с динамическим классом.

Значение должно быть **регионально надежным** для каждой *ячейки* (см. раздел 11.2.2).

Если какая-либо *ячейка* имеет **фиксированный** строковый вид, то **длина строки** вида и **фактическая длина** значения должна быть одинаковой; в противном случае, если ячейка имеет **изменяющийся** строковый вид, тогда **длина строки** вида не должна быть меньше **фактической длины** значения. Эта проверка является динамической, если один или оба вида являются динамическими видами или **изменяющимися** строковыми видами. Это условие называется условием присваивания строки.

Динамические условия: Исключения *RANGEFAIL* или *TAGFAIL* имеют место, если вид ячейки и/или вид значения являются динамическими видами или динамической частью упомянутых выше проверок на совместимость исключений.

Исключение *RANGEFAIL* имеет место, если вид ячейки и/или вид значения являются **изменяющимися** строковыми видами и динамической частью упомянутых выше исключений проверок на совместимость.

Исключение *RANGEFAIL* имеет место, если любая *ячейка* имеет ограниченный вид и значение, получаемое в результате вычисления *значения*, не является ни одним из значений, определяемых ограниченным видом, ни **неопределенным** значением.

Упомянутые выше динамические условия вместе с условием присваивания строки называются условиями присваивания значения по отношению к виду.

В случае *операции присваивания* происходят те же исключения, как если бы выражение:

< ячейка > < замкнутая двоичная операция > (< выражение >)

было вычислено, а полученное значение хранилось бы в конкретной ячейке (заметим, что ячейка вычисляется только один раз).

Примеры:

4.12 *a := b + c* *(1.1)*

10.25 *stackindex− := 1* *(2.1)*

19.19 *x− >.prex, x− >.next := NULL* *(3.1)*

10.25 *− :=* *(4.1)*

6.3 УСЛОВНЫЙ ОПЕРАТОР

Синтаксис:

< условный оператор > :: = *(1)*

IF *< булевское выражение > < then-часть >* [*< else-часть >*] **FI** *(1.1)*

< then-часть > :: = *(2)*

THEN *< список операторов действия >* *(2.1)*

< else-часть > :: = *(3)*

ELSE *< список операторов действия >* *(3.1)*

| **ELSIF** *< булевское выражение > < then-часть >* [*< else-часть >*] *(3.2)*

Производный синтаксис: Запись

ELSIF *< булевское выражение > < then-часть >* [*< else-часть >*]

является производным синтаксисом для

ELSE IF *< булевское выражение > < then-часть >* [*< else-часть >*] **FI;**

Семантика: Условный оператор — это условное двойное ветвление. Если *булевское выражение* дает *TRUE*, то вводится список операторов действия за ключевым словом **THEN**; в противном случае вводится список операторов действия, если таковой имеется, следующий за **ELSE**.

Динамические условия: Имеет место исключение *SPACEFAIL*, если не могут быть удовлетворены требования по сохранению.

Примеры:

7.22
```
IF n > = 50 THEN rn(r) := 'L';
        n− := 50;
        r+ := 1;
FI
```
(1.1)

10.50
```
IF last = NULL
        THEN first,last := p;
        ELSE last− >.succ :=p ;
        p− >.pred := last;
        last := p;
FI
```
(1.1)

6.4 ОПЕРАТОР ВЫБОРА

Синтаксис:

< оператор выбора > :: = *(1)*

CASE *< список селекторов выбора >* **OF** [*< список диапазонов >* ;]
{ *< альтернатива выбора >* }$^{+}$
[**ELSE** *< список операторов действия >*] **ESAC** *(1.1)*

< список селекторов выбора > :: = *(2)*

< дискретное выражение > { , *< дискретное выражение >* }* *(2.1)*

< список диапазонов > :: = *(3)*

< имя дискретного вида > { , *< имя дискретного вида >* }* *(3.1)*

< альтернатива выбора > :: = *(4)*

< спецификация метки выбора > : < список операторов действия > *(4.1)*

Семантика: Оператор выбора — это множественное ветвление. Он состоит из спецификации одного или нескольких дискретных выражений (список селекторов выбора) и ряда помеченных списков операторов действия (альтернативы выбора). Каждый список операторов действия помечается спецификацией метки выбора, состоящей из списка спецификаций списков меток выбора (по одной на каждый селектор выбора). Каждый список меток выбора определяет набор значений. Использование списка дискретных выражений в списке селекторов выбора позволяет произвести выбор альтернативы на основе множества условий.

Оператор выбора вводит тот список операторов действия, для которого значения, данные в спецификации метки выбора, соответствуют значениям в списке селекторов выбора; если такого соответствия нет, то вводится *список операторов действия*, следующий за **ELSE**.

Выражения в списке селекторов выбора вычисляются в неопределенном и даже смешанном порядке. Их необходимо вычислить только к тому моменту, когда альтернатива выбора определяется однозначно.

Статические условия: Для *списка спецификаций меток выбора* имеются условия варианта выбора (см. раздел 12.3).

Количество *дискретных выражений* в *списке селекторов выбора* должно быть равно количеству классов в **результирующем списке классов** списка экземпляров *списков меток выбора* и, если имеется, количеству *имен дискретного вида* в *списке диапазонов*.

Класс любого *дискретного выражения* в *списке селекторов выбора* должен быть **совместимым** с соответствующим (по позиции) классом **результирующего списка классов** экземпляров *списков меток выбора* и, если имеется, быть **совместимым** с соответствующим (по позиции) *именем дискретного вида* в *списке диапазонов*. Последний вид должен быть также **совместимым** с соответствующим классом **результирующего списка классов**.

Любое значение, вырабатываемое *выражением дискретной константы*, определяемое *диапазоном констант* или *именем дискретного вида* в *метке выбора* (см. раздел 12.3), должно находиться в диапазоне соответствующего *имени дискретного вида списка диапазонов* (если имеется), а также в области, определяемой видом соответствующего *дискретного выражения* в *списке селекторов выбора*, если это **строгое** *дискретное выражение*. В последнем случае значения, определяемые соответствующим *именем дискретного вида списка диапазонов* (если имеется) должно находиться в этом диапазоне.

Необязательная часть **ELSE**, соответствующая синтаксису, может быть опущена только в том случае, если список экземпляров *списков меток выбора* является **полным** (см. раздел 12.3).

Динамические условия: Возникает исключение *RANGEFAIL*, если *список диапазонов* описан и значение, вырабатываемое *дискретным выражением* в *списке селекторов выбора*, не находится внутри границ, указанных соответствующим *именем дискретного вида* в *списке диапазонов*.

Имеет место исключение *SPACEFAIL*, если не могут быть удовлетворены требования по хранению.

Примеры:

```
4.11  CASE order OF
          (1) :  a := b + c;
                 RETURN;
          (2) : d := 0;
                 (ELSE) : d := 1;
      ESAC                                                    (1.1)

11.43 starting.p.kind,starting.p.color                        (2.1)

11.58 (rook), (*) :
          IF NOT ok_rook(b,m)
             THEN
                CAUSE illegal;
          FI;                                                 (4.1)
```

6.5 ОПЕРАТОР ЦИКЛА

6.5.1 Общие положения

Синтаксис:

< *оператор цикла* > :: = (1)

DO [< *управляющая часть* > ;] < *список операторов действия* > **OD** (1.1)

< *управляющая часть* > :: = (2)

< *управление по счетчику* > [< *управление по условию* >] (2.1)

| < *управление по условию* > (2.2)

| < *присоединение* > (2.3)

Семантика: Оператор цикла имеет один из трех различных видов: вариант с управлением по условию и вариант с управлением по счетчику, при этом оба эти варианта предназначены для выполнения цикла. Третий вариант — вариант присоединения — это удобная сокращенная запись для эффективного доступа к полям структур. Если управляющая часть не указана, то список операторов действия выполняется один раз, когда выполняется оператор цикла.

Когда объединены вариант управления по счетчику и вариант управления по условию, то управление по условию вычисляется после управления по счетчику и только если действие оператора цикла не завершается управлением по счетчику.

Если указанная управляющая часть является управлением по счетчику и/или управлением по условию, тогда до тех пор, пока управление по счетчику остается внутри области оператора цикла, список операторов действия выполняется согласно управляющей части, но область цикла повторно не выполняется для каждого исполнения списка операторов действия.

Динамические условия: Если требования памяти не могут быть удовлетворены, возникает исключительная ситуация *SPACEFAIL*.

Примеры:

```
4.17  DO FOR i := 1 TO c;
          op (a, b, d, order-1);
          d := a;
      OD                                                      (1.1)

15.58 DO WITH each;
          IF this_counter = counter
             THEN
                status := idle;
                EXIT find_counter;
          FI;
       OD                                                     (1.1)
```

6.5.2 Управление по счетчику

Синтаксис:

< управление по счетчику > :: = (1)
FOR { *< итерация >* { , *< итерация >* } *| **EVER** } (1.1)

< итерация > :: = (2)
< перечисление значений > (2.1)
| *< перечисление ячеек >* (2.2)

< перечисление значений > :: = (3)
< перечисление шагов > (3.1)
| *< перечисление диапазонов >* (3.2)
| *< перечисление множеств >* (3.3)

< перечисление шагов > :: = (4)
< счетчик цикла > < символ присваивания >
< начальное значение > [*< значение шага >*] [**DOWN**] *< конечное значение >* (4.1)

< счетчик цикла > :: = (5)
< определяющее вхождение > (5.1)

< начальное значение > :: = (6)
< дискретное выражение > (6.1)

< значение шага > :: = (7)
BY *< целое выражение >* (7.1)

< конечное значение > :: = (8)
TO *< дискретное выражение >* (8.1)

< перечисление диапазонов > :: = (9)
< счетчик цикла > [**DOWN**] **IN** *< имя дискретного вида >* (9.1)

< перечисление множеств > :: = (10)
< счетчик цикла > [**DOWN**] **IN** *< множественное выражение >* (10.1)

< перечисление ячеек > :: = (11)
< счетчик цикла > [**DOWN**] **IN** *< составной объект >* (11.1)

< составной объект > :: = (12)
< ячейка массива > (12.1)
| *< выражение массива >* (12.2)
| *< ячейка строки >* (12.3)
| *< выражение строки >* (12.4)

Примечание.— Если *составной объект* — это *ячейка (строки, массива)*, то синтаксическая неопределенность разрешается путем интерпретации *составного объекта* как *ячейки*, а не *выражения*.

Семантика: Управление по счетчику может задаваться по нескольким счетчикам цикла. Значения счетчиков цикла вычисляются всякий раз в произвольном порядке перед выполнением списка операторов действия, и они вычисляются только до момента принятия решения об окончании цикла. Если хотя бы один счетчик цикла указывает на окончание цикла, то действие оператора цикла заканчивается.

1. **Цикл с бесконечным повторением**

   Список действий повторяется бесконечно. Действие оператора цикла заканчивается только в результате передачи управления во вне.

2. **Перечисление значений**

   Список операторов действия выполняется повторно для набора определенных значений счетчиков цикла. Набор значений задается либо *именем дискретного вида* (перечисление диапазонов), либо множественным значением (перечисление множеств), либо начальным значением, значением шага и конечным значением (перечисление шагов).

   Счетчик цикла неявно определяет имя, обозначающее его значение или ячейку в списке операторов действия.

Перечисление диапазонов

В случае перечисления диапазонов без спецификации или со спецификацией **DOWN** начальное значение счетчика цикла является наименьшим (наибольшим) значением из набора значений, определяемых *именем дискретного вида*. Для последующих выполнений списка операторов действия *следующее значение* будет вычисляться таким образом:

SUCC (предыдущее значение) (PRED (предыдущее значение)).

Цикл завершается, если список операторов действия выполнен для наибольшего (наименьшего) значения, определяемого *именем дискретного вида*.

Перечисление множеств

В случае перечисления множеств без спецификации или со спецификацией **DOWN** начальное значение счетчика цикла является наименьшим (наибольшим) значением элемента отмеченного множественного значения. Если множественное значение пусто, то список операторов действия не выполняется. Для последующих выполнений списка операторов действия следующее значение будет являться следующим наибольшим (наименьшим) значением элемента в множественном значении. Цикл завершается выполнением списка операторов действия для наибольшего (наименьшего) значения. При выполнении оператора цикла **множественное** выражение вычисляется только один раз.

Перечисление шагов

В случае перечисления шагов без спецификации или со спецификацией **DOWN** набор значений счетчика цикла определяется начальным значением, конечным значением и, возможно, значением шага. При выполнении оператора цикла эти выражения вычисляются только один раз в неопределенном и, возможно, смешанном порядке. Значение шага всегда положительно. Перед каждым выполнением списка операторов действия происходит проверка окончания цикла. Первоначально проверка выполняется для определения, является ли начальное значение счетчика цикла большим (меньшим) по сравнению с конечным значением. Для последующих выполнений *следующее значение* вычисляется так:

предыдущее значение + значение шага (предыдущее значение — значение шага)

в случае спецификации *значения шага*; в противном случае оно вычисляется так:

SUCC (предыдущее значение) (PRED (предыдущее значение)).

Цикл завершается при вычислении значения, которое больше (меньше) конечного значения, либо завершение цикла вызывается исключительной ситуацией *OVERFLOW*.

3. **Перечисление ячеек**

В случае перечисления ячеек без спецификации или со спецификацией **DOWN** список операторов действия вводится повторно для набора ячеек, являющихся элементами ячейки массива, обозначенной как *ячейка массива*, или компонентами ячейки строки, обозначенной как *ячейка строки*. Если указано, что *выражение массива* или *выражение строки* не является ячейкой, то ячейка, содержащая специфицируемое значение, создается неявно. Время жизни созданной ячейки является оператором цикла. Тип созданной ячейки является динамическим, если значение относится к динамическому классу. Семантика такова, как будто перед каждым выполнением списка операторов действия имеется описание опознавателя-ячейки:

DCL < *счетчик цикла* > < *вид* > **LOC** := < *составной объект* > (< *индекс* >) ,

где *вид* — это вид элемента ячейки массива или *&имя* (1), такой что *&имя* — это виртуальное имя **синонимического вида**, **синонимическое** по отношению к виду ячейки строки, если это **фиксированный** строковый вид, в противном случае оно является синонимическим по отношению к **компонентному** виду, и где *индекс* первоначально устанавливается на **нижнюю границу (верхнюю границу)** вида ячейки, а *индекс* перед каждым последующим выполнением списка операторов действия устанавливается на *SUCC (индекс) (PRED (индекс))*. Список операторов действия выполняться не будет, если **фактическая длина** *ячейки строки* равна нулю. Оператор цикла завершается, если *индекс* сразу после выполнения списка операторов действия равен **верхней границе (нижней границе)** вида ячейки. При выполнении оператора цикла *составной объект* вычисляется только один раз.

Статические свойства: Счетчик цикла имеет строку имен, являющуюся строкой имен его *определяющего вхождения*.

Перечисление значений

Имя, определяемое *счетчиком цикла*, является именем **перечисления значений**.

Перечисление шагов

Класс имени, определяемый *счетчиком цикла*, является **результирующим классом** из классов *начального значения*, *значения шага* (если имеется) и *конечного значения*.

Перечисление диапазонов

Класс имени, определяемый счетчиком цикла, является классом М-значения, где М — *имя дискретного вида*.

Перечисление множеств

Класс имени, определяемый *счетчиком цикла*, является классом М-значения, где М-**элементный** вид вида (**строгого**) *множественного выражения*.

Перечисление ячеек

Имя, определяемое *счетчиком цикла*, является именем **перечисления ячеек**. Его вид является видом **элемента** для вида *ячейки массива* или *выражения массива* или *&имени*(1) строкового вида, где *&имя* — это виртуальное имя **синонимического вида**, являющееся **синонимическим** по отношению к виду *ячейки строки* или **корневому** виду *выражения строки*.

Имя **перечисления ячеек** является **ссылочным**, если формат элемента вида *ячейки массива* является **NOPACK**.

Статические условия: Классы *начального значения*, *конечного значения*, *значения шага* (если имеется) должны быть попарно **совместимыми**.

Корневой вид класса *счетчика цикла* в *перечислении значений* не должен быть **нумеруемым** перечислимым видом.

Динамические условия: Если значение, вырабатываемое *значением шага*, не больше нуля, то имеет место исключение *RANGEFAIL*. Это исключение возникает вне блока оператора цикла.

Примеры:

4.17	**FOR** *i* := *1* TO *c*	*(1.1)*
15.37	**FOR EVER**	*(1.1)*
4.17	*i* := *1* TO *c*	*(3.1)*
9.12	*j* := *MIN (sieve)* **BY** *MIN (sieve)* **TO** *max*	*(3.1)*
14.28	*i* **IN** *INT (1 : 100)*	*(3.2)*

6.5.3 Управление по условию

Синтаксис:

< управление по условию > :: = *(1)*
WHILE *< булевское выражение >* *(1.1)*

Семантика: Булевское выражение вычисляется сразу же перед вводом списка операторов действия (после вычисления управления по счетчику, если оно имеется). Если в результате вычисления его значение *TRUE*, то выполняется список операторов действия; в противном случае действие оператора цикла завершается.

Примеры:

7.35 **WHILE** *n* > = *1* *(1.1)*

6.5.4 Присоединение

Синтаксис:

< *присоединение* > :: = (1)
WITH < *управление присоединением* > { , < *управление присоединением* > }* (1.1)

< *управление присоединением* > :: = (2)
< *ячейка структуры* > (2.1)
| < *примитивное значение структуры* > (2.2)

Примечание.— Если *управление присоединением* является *ячейкой структуры*, то синтаксическая неопределенность разрешается путем интерпретации *управления присоединением* как *ячейки*, а не *примитивного значения*.

Семантика: **(Видимые)** имена полей вида ячеек структуры или описанного значения структуры в каждом *управлении присоединением* становятся доступными в результате непосредственного доступа к полям.

Правила видимости таковы, как будто определяющее вхождение имени поля было введено для каждого имени **поля**, сопровождающего вид ячейки или примитивное значение и с той же строкой имен, что и имя поля.

Если описана *ячейка структуры*, то имена доступа с той же строкой имен, что и имена полей вида *ячейки структуры*, описываются неявно путем обозначения подъячеек ячейки структуры.

Если описано *примитивное значение структуры*, то имена значений с той же строкой имен, что и имена полей вида **(строгого)** *примитивного значения структуры*, определяются неявно путем обозначения подзначений значения структуры.

При выполнении оператора цикла описанные ячейки структуры и/или значения структуры вычисляются один раз перед выполнением оператора цикла. Осуществляется это в неопределенном и, возможно, смешанном порядке.

Статические свойства: (Виртуальное) определяющее вхождение, введенное для имени **поля**, имеет ту же строку имен, что и *определяющее вхождение имени поля* для данного имени **поля**.

Если описано *примитивное значение структуры*, то (виртуальное) определяющее вхождение в *присоединение* определяет имя **значения с присоединением**. Его класс — это класс М-значения, где М — вид того имени **поля** структурного вида *примитивного значения структуры*, которое становится доступным как имя **значения с присоединением**.

Если описана *ячейка структуры*, то (виртуальное) определяющее вхождение в *присоединение* определяет имя **ячейки с присоединением**. Его вид — это вид того имени **поля** вида *ячейки структуры*, которая становится доступной как имя **ячейки с присоединением**. Имя **ячейки с присоединением** является **ссылочным**, если формат **поля** с соответствующим именем поля является **NOPACK**.

Примеры:

15.58 **WITH** *each* *(1.1)*

6.6 ОПЕРАТОР ВЫХОДА

Синтаксис:

< *оператор выхода* > :: = (1)
EXIT < *имя метки* > (1.1)

Семантика: Оператор выхода используется для выхода из сложного оператора действия или модуля. Выполнение выхода осуществляется сразу же после ближайшего охватывающего сложного оператора действия или модуля, помеченного *именем метки*.

Статические условия: *Оператор выхода* может находиться внутри сложного оператора действия или модуля, у которого расположенное впереди *определяющее вхождение* имеет ту же строку имен, что и *имя метки*.

Если *оператор выхода* находится внутри определения процедуры или процесса, то сложный оператор действия или модуль, из которых осуществляется выход, должны также находиться внутри определения той же процедуры или процесса (то есть оператор выхода не может использоваться для выхода из процедур или процессов).

Оператор выхода не снабжен программой *обработки исключений*.

Примеры:

15.62 **EXIT** *find_counter* *(1.1)*

6.7 ОПЕРАТОР ВЫЗОВА

Синтаксис:

< оператор вызова > :: = *(1)*
< вызов процедуры > *(1.1)*
| *< вызов встроенной программы >* *(1.2)*

< вызов процедуры > :: = *(2)*
{ *< имя процедуры >* | *< примитивное значение процедуры >* }
([*< список фактических параметров >*]) *(2.1)*

< список фактических параметров > :: = *(3)*
< фактический параметр > { , *< фактический параметр >* }* *(3.1)*

< фактический параметр > :: = *(4)*
< значение > *(4.1)*
| *< ячейка >* *(4.2)*

< вызов встроенной программы > :: = *(5)*
< имя встроенной программы > ([*< список параметров встроенной программы >*]) *(5.1)*

< список параметров встроенной программы > :: = *(6)*
< параметр встроенной программы > { , *< параметр встроенной программы >* }* *(6.1)*

< параметр встроенной программы > :: = *(7)*
< значение > *(7.1)*
| *< ячейка >* *(7.2)*
| *< нерезервируемое имя >* [(*< список параметров встроенной программы >*)] *(7.3)*

Примечание.— Если *фактический параметр* или *параметр встроенной программы* являются *ячейкой*, то синтаксическая неопределенность разрешается путем интерпретации ее как *ячейки*, а не *значения*.

Семантика: Оператор вызова приводит к вызову либо процедуры, либо встроенной программы. Вызов процедуры приводит к вызову **общей** процедуры, указанной значением, получаемым из *примитивного значения процедуры* или вырабатываемым процедурой, указанной *именем процедуры*. Фактические значения и ячейки, описанные в списке фактических параметров, передаютя в процедуру.

Вызов встроенной программы — это либо *вызов встроенной CHILL-программы*, либо *вызов встроенной программы* реализации (см. разделы 6.20 и 13.1 соответственно).

Значение, ячейка или любое имя, определяемое программой, не являющееся строкой **резервируемого** простого имени, могут передаваться как *параметр встроенной программы*. Вызов встроенной программы может возвращать значение или ячейку.

Встроенная программа может быть родовой, то есть ее класс (если это вызов встроенной программы, возвращающей **значение**) или ее вид (если это вызов встроенной программы, возвращающей **ячейку**) может зависеть не только от *имени встроенной программы*, но также и от статических свойств передаваемых фактических параметров и статического контекста вызова.

Статические свойства: *Вызов процедуры* имеет следующие присущие ему свойства: список **спецификаций параметров**, возможно, **спецификацию результата**, возможно, пустой набор имен исключений, **универсальность**, **рекурсивность** и, возможно, вызов процедуры может быть **внутрирегиональным** (последнее возможно только относительно *имени процедуры*; см. раздел 11.2.2). Эти свойства наследуются из *имени процедуры* или любого вида, **совместимого** с классом *примитивного значения процедуры* (в последнем случае универсальность всегда является **общей**).

Вызов процедуры со **спецификацией результата** является *вызовом процедуры возврата ячейки* тогда и только тогда, когда в **спецификации результата** описан атрибут **LOC**; в противном случае это *вызов процедуры возврата значения*.

Имя встроенной программы является именем, определяемым в языке CHILL или определяемым реализацией, которое при рассмотрении определяется в области определения воображаемого внешнего процесса или в любом контексте (см. раздел 10.8).

Вызов встроенной программы является *вызовом встроенной программы*, возвращающей **ячейку**, если в результате вызова выдается ячейка; если же в результате вызова выдается значение, то это *вызов встроенной программы*, выдающей **значение**.

Статические условия: Число вхождений *фактических параметров* в *вызове процедуры* должно быть таким же, что и число ее спецификаций параметров. Требования по совместимости для *фактического параметра* и соответствующей (по позиции) спецификации параметра *вызова процедуры* таковы:

- Если спецификация параметра имеет атрибут **IN** (по умолчанию), то *фактический параметр* должен быть *значением*, класс которого **совместим** с видом в соответствующей спецификации параметра. Вид последнего не должен иметь **свойства беззначимости**. *Фактический параметр* — это *значение*, которое должно быть **регионально надежным** для *вызова процедуры*.

- Если спецификация параметра имеет атрибуты **INOUT** или **OUT**, то *фактический параметр* должен быть *ячейкой*, вид которой должен быть **совместимым** с классом M-значения, где M — это вид в соответствующей спецификации параметра. Вид (фактической) *ячейки* должен быть статическим и не должен обладать ни **свойством неизменяемости**, ни **свойством беззначимости**. *Фактический параметр* — это *ячейка*. Его можно рассматривать как *значение*, которое должно быть **регионально надежным** при *вызове процедуры*.

- Если спецификация параметра имеет атрибут **INOUT**, то вид в спецификации параметра должен быть **совместимым** с классом M-значения, где M — вид *ячейки*.

- Если спецификация параметра имеет атрибут **LOC**, описанный без атрибута **DYNAMIC**, то *фактический параметр* должен быть *ячейкой*, являющейся и **ссылочной**, и такой, что вид в спецификации параметра является **совместимым по чтению** с видом (фактической) *ячейки*, либо *фактический параметр* должен быть *значением*, не являющимся *ячейкой*, но класс которой **совместим** с видом в спецификации параметра.

- Если спецификация параметра имеет атрибут **LOC**, описанный с атрибутом **DYNAMIC**, то *фактический параметр* должен быть *ячейкой*, являющейся и **ссылочной**, и такой, что вид в спецификации параметра является **динамическим, совместимым по чтению** с видом (фактической) *ячейки*, либо *фактический параметр* должен быть *значением*, не являющимся *ячейкой*, но класс которого **совместим** с параметризованной версией данного вида.

- Если спецификация параметра имеет атрибут **LOC**, тогда

  — если *фактический параметр* является *ячейкой*, он должен иметь ту же **региональность**, что и *вызов процедуры*;

  — если *фактический параметр* является *значением*, то он должен быть **регионально надежным** для *вызова процедуры*.

Динамические условия: *Вызов процедуры* или *вызов встроенной программы* может быть причиной любого исключения, указанного в списке имен исключений. *Вызов процедуры* может быть причиной исключения *EMPTY*, если *примитивное значение процедуры* дает *NULL*; вызов процедуры может быть причиной исключения *SPACEFAIL*, если не могут быть удовлетворены требования по памяти. Если выполняется **рекурсивный** вызов **нерекурсивной** процедуры, то процедура не должна вызывать себя ни прямо, ни косвенно.

Передача параметров может вызывать следующие исключения:

- Если спецификация параметра имеет атрибуты **IN** или **INOUT**, то условия присваивания (фактического) значения относительно вида спецификации параметра применяются в момент вызова (см. раздел 6.2), а возможные исключения могут иметь место до вызова процедуры.

- Если спецификация параметра имеет атрибуты **INOUT** или **OUT**, то условия присваивания локального значения формального параметра относительно вида (фактической) ячейки применяются в момент возврата (см. раздел 6.2), а возможные исключения могут иметь место после возврата процедуры.

- Если спецификация параметра имеет атрибут **LOC** и *фактический параметр* является *значением*, не являющимся *ячейкой*, то условия присваивания (фактического) *значения* относительно вида спецификации параметра в момент вызова и возможные исключения могут иметь место до вызова процедуры (см. раздел 6.2).

Примитивное значение процедуры не должно выдавать процедуру, определяемую в определении процесса, активация которого отличается от активации процесса, выполняющего вызов процедуры (иная, чем воображаемый внешний процесс), а время жизни означенной процедуры не должно заканчиваться.

Примеры:

4.18 *op (a, b, d, order-1)* *(1.1)*

6.8 ОПЕРАТОР РЕЗУЛЬТАТА И ВОЗВРАТА

Синтаксис:

< *оператор возврата* > :: = *(1)*
RETURN [< *результат* >] *(1.1)*

< *оператор результата* > :: = *(2)*
RESULT < *результат* > *(2.1)*

< *результат* > :: = *(3)*
< *значение* > *(3.1)*
| < *ячейка* > *(3.2)*

Производный синтаксис: *Оператор возврата* с *результатом* является производным от **DO RESULT** < *результат* > ; **RETURN**; **OD**.

Семантика: Оператор результата используется для определения результата, вырабатываемого вызовом процедуры. Этот результат может быть ячейкой или значением. Оператор возврата вызывает возврат из вызова процедуры, в определении которой он находится. Если процедура возвращает результат, то этот результат определяется оператором результата, который выполняется последним. Если не выполняется оператор результата, то вызов процедуры вырабатывает соответственно **неопределенную** ячейку и **неопределенное** значение.

Статические свойства: *Оператор результата* и *оператор возврата* имеют приписанное имя **процедуры**, которое является именем ближайшего окружающего определения процедуры.

Статические условия: *Оператор результата* и *оператор возврата* должны быть текстуально окружены определением процедуры. *Оператор результата* может быть описан только в том случае, если его имя **процедуры** имеет **спецификацию результата**.

Программа *обработки исключения* не может быть приписана к *оператору возврата* (без *результата*).

Если атрибут **LOC (LOC DYNAMIC)** описан в **спецификации результата** имени **процедуры** *оператора результата*, то *результат* должен быть *ячейкой*, такой что вид в **спецификации результата совместим по чтению (динамическая совместимость по чтению)** с видом *ячейки*. *Ячейка* должна быть **ссылочной**, если атрибут **NONREF** не описан в **спецификации результата**. *Результат* является *ячейкой*, которая должна иметь ту же **региональность**, что и имя **процедуры**, приписанной к *оператору результата*.

Если атрибут **LOC** не описан в **спецификации результата** имени **процедуры** *оператора результата*, то *результат* должен быть *значением*, класс которого **совместим** с видом в **спецификации результата**. *Результат* является *значением*, которое должно быть **регионально надежным** для имени **процедуры**, приписанной к *оператору результата*.

Динамические условия: Если атрибут **LOC** не описан в **спецификации результата** имени **процедуры**, то применяются условия присваивания *значения* в *операторе результата* по отношению к виду в **спецификации результата** имени **процедуры** возврата значения.

Примеры:

4.21	**RETURN**	*(1.1)*
1.6	**RESULT** $i + j$	*(2.1)*
5.19	*c*	*(3.1)*

6.9 ОПЕРАТОР ПЕРЕХОДА

Синтаксис:

< *оператор перехода* > :: = *(1)*

GOTO < *имя метки* > *(1.1)*

Семантика: Оператор перехода вызывает передачу управления. Выполнение возобновляется оператором действия, помеченного *именем метки*.

Статические условия: Если *оператор перехода* размещается внутри определения процедуры или процесса, то метка, указанная *именем метки*, также должна быть определена внутри определения (то есть нельзя выйти за пределы активации процедуры или процесса).

К *оператору перехода* не должна быть приписана программа *обработки исключений*.

6.10 ОПЕРАТОР КОНТРОЛЯ

Синтаксис:

< *оператор контроля* > :: = *(1)*

ASSERT < *булевское выражение* > *(1.1)*

Семантика: Оператор контроля предоставляет средства проверки некоторого условия.

Динамические условия: Если *булевское выражение* вырабатывает *FALSE*, то имеет место исключение *ASSERTFAIL*.

Примеры:

4.7	**ASSERT** $b > 0$ **AND** $c > 0$ **AND** *order* > 0	*(1.1)*

6.11 ПУСТОЙ ОПЕРАТОР

Синтаксис:

< *пустой оператор* > :: = *(1)*

< *пусто* > *(1.1)*

< *пусто* > :: = *(2)*

Семантика: Пустой оператор не вызывает никакого действия.

Статические условия: *Программа обработки исключений* не может быть придана *пустому оператору*.

6.12 ОПЕРАТОР ПРИЧИНЫ

Синтаксис:

< *оператор причины* > :: = (1)
CAUSE < *имя исключения* > (1.1)

Семантика: Оператор причины возбуждает исключение, имя которого указывается *именем исключения*.

Статические условия: *Программа обработки исключений* не может быть приписана *оператору причины*.

Примеры:

4.9 **CAUSE** *wrong_input* (1.1)

6.13 ОПЕРАТОР ЗАПУСКА

Синтаксис:

< *оператор запуска* > :: = (1)
< *выражение запуска* > (1.1)

Семантика: Оператор запуска вычисляет выражение запуска (см. раздел 5.2.14) без использования результирующего экземплярного значения.

Примеры:

14.45 **START** *call_distributor ( )* (1.1)

6.14 ОПЕРАТОР ОСТАНОВА

Синтаксис:

< *оператор останова* > :: = (1)
STOP (1.1)

Семантика: Оператор останова прекращает процесс, который выполняет данный оператор (см. раздел 11.1).

Статические условия: *Программа обработки исключений* не может быть приписана к *оператору останова*.

6.15 ОПЕРАТОР ПРОДОЛЖЕНИЯ

Синтаксис:

< *оператор продолжения* > :: = (1)
CONTINUE < *ячейка события* > (1.1)

Семантика: Оператор продолжения вычисляет *ячейку события*.

Если ячейка события имеет непустой набор приданных задержанных процессов, один из них, обладающий наивысшим приоритетом, будет реактивирован. Если имеется несколько таких процессов, один из них будет выбран способом, определенным реализацией. Если таких процессов нет, то оператор продолжения больше не действует.

Если процесс реактивируется, он устраняется из всех наборов задержанных процессов, в которые он входил.

Примеры:

13.25 **CONTINUE** *resource_freed* *(1.1)*

6.16 ОПЕРАТОР ЗАДЕРЖКИ

Синтаксис:

< *оператор задержки* > :: = *(1)*
DELAY < *ячейка события* > [< *приоритет* >] *(1.1)*

< *приоритет* > :: = *(2)*
PRIORITY < *целое постоянное выражение* > *(2.1)*

Семантика: Оператор задержки вычисляет *ячейку события*.

Затем имеет место исключение *DELAYFAIL* (см. ниже) или задержка выполняющего процесса.

Если выполняющий процесс становится задержанным, он добавляется к набору задержанных процессов с приоритетом набора, приданных описанной ячейке события. Приоритет равен описанному (если имеется), в противном случае он равен нулю (низший приоритет).

Динамические свойства: Процесс, выполняющий оператор задержки, становится **прерываемым**, когда он достигает момента выполнения, при котором он может стать задержанным. Он перестает быть **прерываемым**, когда проходит этот момент.

Статические условия: *Целое постоянное выражение* не должно вырабатывать отрицательное значение.

Динамические условия: Имеет место исключение *DELAYFAIL*, если *ячейка события* имеет вид с приписанной **длиной события**, равной количеству уже задержанных процессов по ячейке события.

Время жизни *ячейки события* не должно заканчиваться, пока выполняющий процесс задерживается по этим ячейкам.

Примеры:

13.18 **DELAY** *resource_freed* *(1.1)*

6.17 ОПЕРАТОР ВЫБОРА ЗАДЕРЖКИ

Синтаксис:

< оператор выбора задержки > :: = (1)

DELAY CASE [SET *< ячейка экземпляра >* **[** *< приоритет >* **] ;**
| *< приоритет >* **;] {** *< вариант задержки >* **}**$^{+}$
ESAC (1.1)

< вариант задержки > :: = (2)

(< список событий >) : < список операторов действия > (2.1)

< список событий > :: = (3)

< ячейка события > **{ ,** *< ячейка события >* **}*** (3.1)

Семантика: Оператор выбора задержки вычисляет в произвольном и, возможно, в смешанном порядке *ячейку экземпляра* (если имеется) и все *ячейки событий*, описанные в *варианте задержки*.

Имеет место исключение *DELAYFAIL* (см. ниже) или выполняющий процесс становится задержанным.

Если выполняющий оператор процесс становится задержанным, он добавляется к набору задержанных процессов с приоритетом набора, приданных к каждой из описанных ячеек событий. Приоритет равен описанному (если имеется), в противном случае он равен нулю (низший приоритет).

Если задержанный процесс реактивируется другим процессом, выполняющим оператор продолжения по ячейке события, то вводится соответствующий *список операторов действия*. Если некоторые *варианты задержки* описывают ту же ячейку события, то выбор между ними не описывается. Если описана *ячейка экземпляра*, то перед вводом в нее заносится значение экземпляра, определяющее процесс, выполнивший оператор продолжения.

Динамические свойства: Процесс, выполняющий оператор выбора задержки, становится **прерываемым**, когда он достигает момента выполнения, при котором он может стать задержанным. Он перестает быть **прерываемым**, когда проходит этот момент.

Статические условия: Вид *ячейки экземпляра* не должен обладать **свойством неизменяемости**. *Целое постоянное выражение* в *приоритете* не должно вырабатывать отрицательное значение.

Динамические условия: Имеет место исключение *DELAYFAIL*, если некоторая *ячейка события* имеет вид с приписанной **длиной события**, равной количеству уже задержанных процессов по этой ячейке события.

Время жизни ни одной из *ячеек события* не должно заканчиваться, пока выполняющий процесс задерживается по этим ячейкам.

Имеет место исключение *SPACEFAIL*, если не могут быть удовлетворены требования по памяти.

Примеры:

14.26
```
DELAY CASE
    (operator_is_ready) : / * некоторые действия * /
    (switch_is_closed) : DO FOR i IN INT (1 : 100);
                         CONTINUE operator_is_ready;
                         / * очистить очередь * /
                         OD

ESAC
```
(1.1)

6.18 ОПЕРАТОР ПОСЫЛКИ

6.18.1 Общие положения

Синтаксис:

< оператор посылки > :: = *(1)*

< оператор посылки сигнала > *(1.1)*

| *< оператор посылки в буфер >* *(1.2)*

Семантика: Оператор посылки инициирует перенос информации синхронизации от посылающего процесса. Детализированная семантика зависит от того, является ли объект синхронизации сигналом или буфером.

6.18.2 Оператор посылки сигнала

Синтаксис:

< оператор посылки сигнала > :: = *(1)*

SEND *< имя сигнала >* [(*< значение >* { ,*< значение >* }*)]
[TO *< примитивное значение экземпляра >*] [*< приоритет >*] *(1.1)*

Семантика: Оператор посылки сигнала вычисляет, в произвольном и, возможно, в смешанном порядке список *значений*, если они имеются, и *примитивное значение экземпляра*, если оно имеется.

Сигнал, описанный *именем сигнала*, составляется для передачи из описанных сигналов и приоритета. Приоритет равен заданному, если существует, в противном случае он равен нулю (низший приоритет).

Если имя **сигнала** имеет приданное ему имя **процесса**, то только процессы с этим именем могут принимать сигнал; если описано *примитивное значение экземпляра*, то только этот процесс может принимать сигнал. В противном случае сигнал может быть принят любым процессом.

Если сигнал имеет непустой набор приданных ему задержанных процессов, среди которых один или несколько процессов могут принимать сигнал, то один из этих процессов будет реактивирован. Если имеется несколько таких процессов, то один из них определяется способом, зависящим от реализации. Если таких процессов нет, то сигнал находится в состоянии ожидания.

Если процесс реактивируется, он исключается из всех наборов задержанных процессов, в которые он входил.

Статические условия: Количество вхождений *значений* должно быть равно количеству видов *имени сигнала*. Класс каждого значения должен быть **совместимым** с соответствующим видом *имени сигнала*. Вхождение без *значения* может быть **внутрирегиональным** (см. раздел 11.2.2). *Целое постоянное выражение* в *приоритете* не должно вырабатывать отрицательное значение.

Динамические условия: Используются условия присваивания каждого *значения* в отношении соответствующего ему вида *имени сигнала*.

Если *примитивное значение экземпляра* дает *NULL*, то имеет место исключение *EMPTY*.

Время жизни процесса, указанное значением, выдаваемым *примитивным значением экземпляра*, не должно заканчиваться в момент выполнения оператора посылки сигнала.

Если *имя сигнала* имеет приписанное имя **процесса**, не являющееся именем процесса, указанного значением, выдаваемым *примитивным значением экземпляра*, то имеет место исключение *SEND-FAIL*.

Примеры:

15.78 SEND *ready* TO *received_user* *(1.1)*

15.86 SEND *readout(count)* TO *user* *(1.1)*

6.18.3 Оператор посылки в буфер

Синтаксис:

< оператор посылки в буфер > :: = *(1)*

SEND *< буферная ячейка > (< значение >)* [*< приоритет >*] *(1.1)*

Семантика: Оператор посылки в буфер вычисляет *буферную ячейку* и *значение* в произвольном порядке.

Если буферная ячейка имеет непустой набор приписанных задержанных процессов, то один из них будет реактивирован. Если имеется несколько таких процессов, один из них будет выбран способом, зависящим от реализации. Если таких процессов нет и емкость буферной ячейки исчерпана, то процесс, выполняющий оператор посылки в буфер, задерживается с некоторым приоритетом. В противном случае происходит запись в память значения с некоторым приоритетом. Приоритет равен заданному (если имеется), в противном случае он равен нулю (низший приоритет). Емкость буфера исчерпана, если *буферная ячейка* имеет вид с приписанной **длиной буфера**, равной количеству значений, уже хранящихся в буферной ячейке.

Если процесс, выполняющий оператор посылки в буфер, становится задержанным, он добавляется к набору задержанных посылающих процессов, приписанных к буферной ячейке. Если процесс реактивируется, он исключается из всех наборов задержанных процессов, в которые он входил.

Динамические свойства: Процесс, выполняющий оператор посылки в буфер, становится **прерываемым**, когда он достигает момента выполнения, при котором он может стать задержанным. Он прекращает быть **прерываемым**, когда проходит данный момент.

Статические условия: Класс *значения* должен быть **совместимым** с видом **элементов буфера** вида *буферной ячейки*. *Значение* не должно быть **внутрирегиональным** (см. раздел 11.2.2). *Целое постоянное выражение* в *приоритете* не должно вырабатывать отрицательное значение.

Динамические условия: Используются условия присваивания *значения* в отношении вида **элементов буфера** вида *буферной ячейки*; могут иметь место некоторые исключения перед тем, как процесс может стать задержанным.

Время жизни *буферной ячейки* не должно заканчиваться, пока процесс, выполняющий оператор посылки в буфер, является задержанным по ней.

Примеры:

16.119 **SEND** user–> *([ready, –> counter_buffer]) k* *(1.1)*

6.19 ОПЕРАТОР ВАРИАНТА ПОЛУЧЕНИЯ

6.19.1 Общие положения

Синтаксис:

< оператор варианта получения > :: = *(1)*

< оператор варианта получения сигнала > *(1.1)*

| *< оператор варианта получения из буфера >* *(1.2)*

Семантика: Оператор варианта получения принимает информацию синхронизации, передаваемую оператором посылки. Детализированная семантика зависит от используемого объекта синхронизации, являющегося либо сигналом, либо буфером. Ввод оператора варианта получения не обязательно приводит к задержке процесса, выполняющего оператор (подробности см. в главе 11).

6.19.2 Оператор варианта получения сигнала

Синтаксис:

< оператор варианта получения сигнала > :: = *(1)*

RECEIVE CASE [SET *< ячейка экземпляра >* **;]**
{ *< альтернатива получения сигнала >* }$^{+}$
[ELSE *< список операторов действия >* **] ESAC** *(1.1)*

< альтернатива получения сигнала > :: = *(2)*

(< имя сигнала > **[IN** *< список определяющих вхождений >* **]) :**
< список операторов действия > *(2.1)*

Семантика: Оператор варианта получения сигнала вычисляет *ячейку экземпляра*, если имеется.

Затем процесс выполнения оператора варианта получения сигнала (сразу же) принимает сигнал или, если описан атрибут **ELSE**, вводит соответствующий *список операторов действия*, в противном случае процесс становится задержанным. Процесс, выполняющий оператор варианта получения сигнала, сразу же получает сигнал, если одно из *имен сигнала*, описанное в *альтернативе получения сигнала*, находится в состоянии ожидания и может быть получено процессом. Если может быть получено несколько сигналов, то способом, зависящим от реализации, выбирается один с наивысшим приоритетом.

Если процесс, выполняющий оператор варианта получения сигнала, становится задержанным, то он добавляется к набору задержанных процессов, приписанных к каждому из описанных сигналов. Если задержанный процесс реактивируется другим процессом, выполняющим оператор посылки сигнала, то он получает сигнал.

Если процесс, выполняющий оператор варианта получения сигнала, принимает сигнал, то вводится соответствующий *список операторов действия*. Если *ячейка экземпляра* описана, то до ввода в нее заносится для хранения значение экземпляра, идентифицирующее процесс, который послал полученный сигнал. Если имя **сигнала** для принятого сигнала имеет приписанный список видов, то описывается список имен **принятых значений**; сигнал переносит список значений, а имена **принятых значений** обозначают соответствующее им значение в введенном *списке операторов действия*.

Статические свойства: *Определяющее вхождение* в *списке определяющих вхождений альтернативы получения сигнала* определяет имя **принятого значения**. Его класс — это класс M-значения, где M — соответствующий вид в списке видов, приписанном *имени сигнала* и стоящим перед ним.

Динамические свойства: Процесс, выполняющий оператор варианта получения сигнала, становится **прерываемым**, когда он достигает момента выполнения оператора, при котором он может стать задержанным. **Прерывание** прекращается, когда он проходит этот момент.

Статические условия: Вид *ячейки экземпляра* не должен иметь **свойства неизменяемости**.

Все вхождения *имени сигнала* должны быть различными.

Необязательный атрибут **IN** и *список определяющих вхождений* в *альтернативе получения сигнала* должны быть описаны тогда и только тогда, когда *имя сигнала* имеет непустой набор видов. Количество имен в *списке определяющих вхождений* должно быть равно количеству видов *имени сигнала*.

Динамические условия: Если требования по памяти не могут быть удовлетворены, то имеет место исключение *SPACEFAIL*.

Примеры:

15.83
```
RECEIVE CASE
    (advance): count + := 1;
    (terminate):
        SEND readout(count) TO user;
        EXIT work_loop;
ESAC                                                    (1.1)
```

6.19.3 Оператор варианта получения из буфера

Синтаксис:

< оператор варианта получения из буфера > :: - (1)

RECEIVE CASE [SET *< ячейка экземпляра >* **;]**
{ *< вариант получения из буфера >* }$^{+}$
[ELSE *< список операторов действия >* **]**
ESAC (1.1)

< вариант получения из буфера > :: = (2)

(*< буферная ячейка >* **IN** *< определяющее вхождение >*) :
< список операторов действия > (2.1)

Семантика: Оператор варианта получения из буфера вычисляет в произвольном и, возможно, в смешанном порядке *ячейку экземпляра* (если имеется) и все *буферные ячейки*, описанные в *варианте получения из буфера*.

Затем процесс, выполняющий оператор варианта получения из буфера (сразу же) получает значение или, если описан атрибут **ELSE**, вводит соответствующий *список операторов действия*, в противном случае процесс становится задержанным. Процесс, выполняющий оператор варианта получения из буфера, сразу же получает значение, если оно хранится в одной из описанных буферных ячеек, или если посылающий процесс задержан по одной из описанных буферных ячеек. Если может быть получено несколько значений, то способом, зависящим от реализации, выбирается одно значение с наивысшим приоритетом.

Если процесс, выполняющий оператор варианта получения из буфера, становится задержанным, то он добавляется к набору задержанных процессов, приписанных к каждой из описанных буферных ячеек. Если задержанный процесс реактивируется другим процессом, выполняющим оператор посылки в буфер, то он получает это значение.

Если процесс, выполняющий оператор варианта получения из буфера, получает значение, то вводится соответствующий *список операторов действия*. Если несколько *вариантов получения из буфера* описывают одну и ту же буферную ячейку, то выбор между ними не определен. Если описана *ячейка экземпляра*, то перед вводом в нее для хранения заносится значение экземпляра, идентифицирующее процесс, который послал принятое значение. Описанное имя **принятого значения** обозначает полученное значение во введенном *списке операторов действия*.

Другой процесс реактивируется, если процесс, выполняющий оператор варианта получения из буфера, получает значение из буферной ячейки, приписанный набор задержанных посылающих процессов которой не является пустым. Реактивированный процесс — это один из приписанных процессов с наивысшим приоритетом, если принятое значение хранилось в буферной ячейке, в противном случае это процесс, посылающий принятое значение. В первом случае значение, посылаемое реактивированным процессом, хранится в буферной ячейке (емкость которой остается исчерпанной), и если может быть реактивировано несколько процессов, то один из них выбирается способом, определяемым реализацией. Реактивированный процесс исключается из набора задержанных передающих процессов, приписанных к буферной ячейке.

Статические свойства: *Определяющее вхождение* в *варианте получения из буфера* определяет имя **принятого значения**. Его класс — это класс M-значения, где M — вид **элементов буфера** вида *буферной ячейки*, помечающего *вариант получения из буфера*.

Динамические свойства: Процесс, выполняющий оператор варианта получения из буфера, становится **прерываемым**, когда он достигает момента выполнения, при котором он может стать задержанным. Он перестает быть **прерываемым**, когда проходит этот момент.

Статические условия: Вид *ячейки экземпляра* не должен обладать **свойством неизменяемости**.

Динамические условия: Если не могут быть удовлетворены требования памяти, то имеет место исключение *SPACEFAIL*.

Время жизни *буферных ячеек* не должно заканчиваться, пока процесс, выполняющий оператор варианта получения из буфера, является задержанным по ним.

6.20 ВЫЗОВЫ ВСТРОЕННЫХ CHILL-ПРОГРАММ

Синтаксис:

< вызов встроенной CHILL-программы > :: = *(1)*
< вызов простой встроенной CHILL-программы > *(1.1)*
| *< вызов встроенной CHILL-программы выдачи ячейки >* *(1.2)*
| *< вызов встроенной CHILL-программы выдачи значения >* *(1.3)*

Предопределенные имена: Имена встроенных CHILL-программ являются предопределенными, как имена **встроенных программ** (см. раздел 6.7).

Семантика: *Вызов встроенной CHILL-программы* является либо *вызовом простой встроенной CHILL-программы*, не вырабатывающей результатов (см. раздел 6.20.1), либо *вызовом встроенной CHILL-программы выдачи ячейки*, вырабатывающей ячейку (см. раздел 6.20.2), либо *вызовом встроенной CHILL-программы выдачи значения*, вырабатывающей значение (см. раздел 6.20.3).

Статические свойства: *Вызов встроенной CHILL-программы — это вызов встроенной программы, выдачи* **ячейки**, если это *вызов встроенной CHILL-программы выдачи ячейки*; это *вызов встроенной программы выдачи* **значения**, если это *встроенная CHILL-программа выдачи значения*.

6.20.1 Вызовы простых встроенных CHILL-программ

Синтаксис:

< вызов простой встроенной CHILL-программы > :: = *(1)*
< вызов встроенной программы окончания > *(1.1)*
| *< вызов простой встроенной программы ввода—вывода >* *(1.2)*
| *< вызов простой встроенной программы временных соотношений >* *(1.3)*

Семантика: *Вызов простой встроенной CHILL-программы* — это *вызов встроенной программы*, не вырабатывающей значения или ячейку. Простые встроенные программы ввода—вывода определены в главе 7. Простые встроенные программы временных соотношений определены в главе 9.

6.20.2 Вызовы встроенных CHILL-программ выдачи ячейки

Синтаксис:

< вызов встроенной CHILL-программы выдачи ячейки > :: = *(1)*
< вызов встроенной программы выдачи ячейки ввода—вывода > *(1.1)*

Семантика: *Вызов встроенной CHILL-программы выдачи ячейки* — это *вызов встроенной программы*, выдающей ячейку. Встроенные программы выдачи ячейки ввода—вывода определены в главе 7.

6.20.3 Вызовы встроенных CHILL-программ выдачи значения

Синтаксис:

< вызов встроенной CHILL-программы выдачи значения > :: = (1)
NUM (< дискретное выражение >) (1.1)
| *PRED (< дискретное выражение >)* (1.2)
| *SUCC (< дискретное выражение >)* (1.3)
| *ABS (< целое выражение >)* (1.4)
| *CARD (< множественное выражение >)* (1.5)
| *MAX (< множественное выражение >)* (1.6)
| *MIN (< множественное выражение >)* (1.7)
| *SIZE ({ < ячейка > | < аргумент вида > })* (1.8)
| *UPPER (< верхний—нижний аргумент >)* (1.9)
| *LOWER (< верхний—нижний аргумент >)* (1.10)
| *LENGTH (< аргумент длины >)* (1.11)
| *< вызов встроенной программы размещения >* (1.12)
| *< вызов встроенной программы выдачи значения ввода—вывода >* (1.13)
| *< вызов встроенной программы выдачи значения времени >* (1.14)

< аргумент вида > :: = (2)
< имя вида > (2.1)
| *< имя массивного вида > (< выражение >)* (2.2)
| *< имя строкового вида > (< целое выражение >)* (2.3)
| *< имя вариантного структурного вида > (< список выражений >)* (2.4)

< верхний—нижний аргумент > :: = (3)
< ячейка массива > (3.1)
| *< выражение массива >* (3.2)
| *< имя массивного вида >* (3.3)
| *< ячейка строки >* (3.4)
| *< выражение строки >* (3.5)
| *< имя строкового вида >* (3.6)
| *< дискретная ячейка >* (3.7)
| *< дискретное выражение >* (3.8)
| *< имя дискретного вида >* (3.9)

< аргумент длины > :: = (4)
< ячейка строки > (4.1)
| *< выражение строки >* (4.2)

Примечание.— Если *верхний—нижний аргумент* является *ячейкой (массива, строки)* или *(дискретной)* ячейкой, то синтаксическая неопределенность разрешается путем интерпретации *верхнего—нижнего аргумента* скорее как *ячейки*, чем *выражения* или *примитивного значения*. Если *аргумент длины* — это *ячейка строки*, то синтаксическая неопределенность разрешается путем интерпретации *аргумента длины* как *ячейки*, а не *выражения*.

Семантика: *Вызов встроенной CHILL-программы выдачи значения* — это *вызов встроенной программы*, выдающей значение.

NUM вырабатывает целое значение с тем же внутренним представлением, что и значение ее аргумента.

PRED и *SUCC* выдают соответственно следующее верхнее и нижнее дискретное значение их аргумента.

ABS выдает абсолютное значение ее аргумента.

CARD, *MAX* и *MIN* определены на множественных значениях. *CARD* выдает количество значений элементов в ее аргументе.

MAX и *MIN* выдают соответственно наибольшее и наименьшее значение элемента в их аргументе.

SIZE определена на **ссылочных** ячейках и (возможно, динамических) видах. В первом случае она выдает количество адресуемых блоков памяти, занятых этой ячейкой; во втором случае — количество адресуемых блоков памяти, занимаемых **ссылочной** ячейкой данного вида. Вид является статическим, если *аргумент вида* является *именем вида*, в противном случае это динамически параметризуемая версия вида с параметрами, описанными в *аргументе вида*. В первом случае ячейка не будет вычисляться во время прогона программы.

UPPER и *LOWER* определены на (возможно, динамических):

- ячейках массива, строки и дискретных ячейках, выдавая **верхнюю** и **нижнюю границы** вида ячейки;
- выражениях строки и массива, выдавая **верхнюю** и **нижнюю границы** вида класса значения;
- **строгих** дискретных выражениях, выдавая **верхнюю** и **нижнюю границы** вида класса значения;
- именах массивного, строкового и дискретного **видов**, выдавая **верхнюю** и **нижнюю границы** вида.

LENGTH выдает **фактическую длину** ее аргумента.

Статические свойства: Класс вызова встроенной программы *NUM* — это *INT*-производный класс. Вызов встроенной программы является **постоянным** тогда и только тогда, когда аргумент является либо **постоянной**, либо **(литеральной) константой**.

Класс вызова встроенных программ *PRED* или *SUCC* является **результирующим классом** аргумента. Вызов встроенной программы является **постоянной (константой)** тогда и только тогда, когда аргумент есть **постоянная (константа)**.

Класс вызова встроенной программы *ABS* является **результирующим классом** аргумента. Вызов встроенной программы является **постоянной (константой)** тогда и только тогда, когда аргумент есть **постоянная (константа)**.

Класс вызова встроенной программы *CARD* является *INT*-производным классом. Вызов встроенной программы является **постоянной** тогда и только тогда, когда аргумент есть **постоянная**.

Класс вызова встроенных программ *MAX* и *MIN* есть класс М-значения, где М — **элементный** вид для вида *множественного выражения*. Вызов встроенной программы является **постоянной** тогда и только тогда, когда аргумент есть **постоянная**.

Класс вызова встроенной программы *SIZE* является *INT*-производным классом. Вызов встроенной программы есть **постоянная** тогда и только тогда, когда вид аргумента является статическим.

Класс вызова встроенных программ *UPPER* и *LOWER* — это:

- класс М-значения, если *верхний—нижний аргумент* является *ячейкой массива*, *выражением массива* или *именем массивного вида*, где М — **индексный** вид *ячейки массива*, *выражения массива* или *имени массивного вида* соответственно;
- *INT*-производный класс, если *верхний—нижний аргумент* — это *ячейка строки*, *выражение строки* или *имя строкового вида*;
- класс М-значения, если *верхний—нижний аргумент* — это *дискретная ячейка*, *дискретное выражение* или *имя дискретного вида*, где М — это вид *дискретной ячейки*, *дискретного выражения* или *имени дискретного вида* соответственно.

Вызов встроенных программ *UPPER* и *LOWER* является **постоянной**, если *верхний—нижний аргумент* является *именем (массивного, строкового* или *дискретного) вида*, если вид *ячейки массива* или *строки* является статическим, если *выражение строки* или *массива* имеет статический класс или если *верхний—нижний аргумент* есть *дискретное выражение* или *дискретная ячейка*.

Класс вызова встроенной программы *LENGTH* — это *INT*-производный класс.

Статические условия: Если аргумент вызова встроенных программ *PRED* и *SUCC* является **постоянной**, он, соответственно, не должен быть наименьшим или наибольшим дискретным значением, определяемым **корневым** видом класса аргумента. **Корневой** вид аргумента *дискретного выражения* программ *PRED* и *SUCC* не должен быть **ненумерованным** перечислимым видом.

Если аргумент вызова встроенных программ *MAX* или *MIN* является **постоянной**, то он не должен быть пустым множественным значением.

Аргумент *ячейки* программы *SIZE* должен быть **ссылочным**.

Дискретное выражение как аргумент для *LOWER* и *UPPER* должен быть **строгим**.

Для *аргумента вида*, не являющегося *именем единственного вида*, должны соблюдаться следующие условия совместимости:

- класс *выражения* должен быть **совместимым** с **индексным** видом *имени массивного вида*;
- *имя вариантного структурного вида* должно быть **параметризованным** и должно иметься столько выражений в *списке выражений*, сколько имеется классов в его списке классов, а класс каждого выражения должен быть **совместимым** с соответствующим классом в списке классов.

Динамические условия: *PRED* и *SUCC* являются причиной исключительной ситуации *OVERFLOW*, если они применяются к наибольшему и наименьшему дискретному значению, определяемому **корневым** видом класса аргумента.

NUM и *CARD* могут быть причиной исключительной ситуации *OVERFLOW*, если результирующее значение не входит в набор значений, определяемых с помощью *INT*.

MAX и *MIN* могут быть причиной исключительной ситуации *EMPTY*, если они применяются по отношению к пустым множественным значениям.

ABS может быть причиной исключительной ситуации *OVERFLOW*, если результирующее значение находится вне границ, определяемых **корневым** видом класса аргумента.

Имеет место исключение *RANGEFAIL*, если в *аргументе вида*:

- *выражение* вырабатывает значение, не входящее в набор значений, определяемых **индексным** видом *имени массивного вида*;
- *целое выражение* вырабатывает отрицательное значение или значение, которое больше чем **длина строки** *имени строкового вида*;
- любое выражение в *списке выражений*, для которого соответствующий класс в списке классов *имени вариантного структурного вида* является классом М-значения (то есть является **строгим**), вырабатывает значение, не входящее в набор значений, определяемых М.

Примеры:

9.12	*MIN (sieve)*	*(1.7)*
11.47	*PRED (col_1)*	*(1.2)*
11.47	*SUCC (col_1)*	*(1.3)*

6.20.4 Встроенные программы оперирования динамической памятью

Синтаксис:

< вызов встроенной программы размещения > :: = *(1)*
GETSTACK (< аргумент вида > [, < значение >]) *(1.1)*
| *ALLOCATE (< аргумент вида > [, < значение >])* *(1.2)*

< вызов встроенной программы окончания > :: = *(2)*
TERMINATE (< ссылочное примитивное значение >) *(2.1)*

Семантика: *GETSTACK* и *ALLOCATE* создают ячейку определенного вида и вырабатывают ссылочное значение для созданной ячейки. *GETSTACK* создает эту ячейку в стеке (см. раздел 10.9). Создается ячейка, вид которой тот же, что и у *аргумента вида*, и вырабатывается относящееся к нему значение. Созданная ячейка инициализируется *значением*, если имеется в синтаксическом описании; в противном случае она инициализируется **неопределенным** значением (см. раздел 4.1.2).

TERMINATE заканчивает время жизни ячейки, на которую ссылаются по значению, получаемому из *ссылочного примитивного значения*. При реализации это выглядело бы как последующее освобождение памяти, ранее занятой данной ячейкой.

Статические свойства: Класс вызова встроенных программ *GETSTACK* и *ALLOCATE* является М-ссылочным классом, где М — вид *аргумента вида*. М — это либо *имя вида*, либо **параметризованный** вид, организованный следующим образом:

&< имя массивного вида > (< выражение >) или

&< имя строкового вида > (< целое выражение >) или

&< имя вариантного структурного вида > (< список выражений >)

соответственно.

Вызов встроенных программ *GETSTACK* и *ALLOCATE* является **внутрирегиональным**, если он окружен регионом, в противном случае он является **внерегиональным**.

Статические условия: Класс *значения* (если оно имеется) в вызове встроенных программ *GETSTACK* и *ALLOCATE*, должен быть **совместимым** с видом *аргумента вида*; эта проверка является динамической в случае, если вид *аргумента вида* является динамическим.

Если первый аргумент в *GETSTACK* или *ALLOCATE* имеет **свойство неизменяемости**, то должен иметься второй аргумент.

Значение (если таковое имеется) в вызове встроенных программ *GETSTACK* и *ALLOCATE* должно быть **регионально надежным** для созданной ячейки.

Динамические свойства: Ссылочное значение является **размещенным** ссылочным значением тогда и только тогда, если оно возвращается в результате вызова встроенной программы *ALLOCATE*.

Динамические условия: Если требования памяти не могут быть удовлетворены, *GETSTACK* может стать причиной исключительной ситуации *SPACEFAIL*.

Если требования памяти не могут быть удовлетворены, *ALLOCATE* может стать причиной исключительной ситуации *ALLOCATEFAIL*.

Для *GETSTACK* и *ALLOCATE* применяются условия присваивания значения, получаемого из синтаксического определения *значения*, по отношению к виду *аргумента вида*.

Если *ссылочное примитивное значение* дает значение *NULL*, то *TERMINATE* может вызывать исключительную ситуацию *EMPTY*.

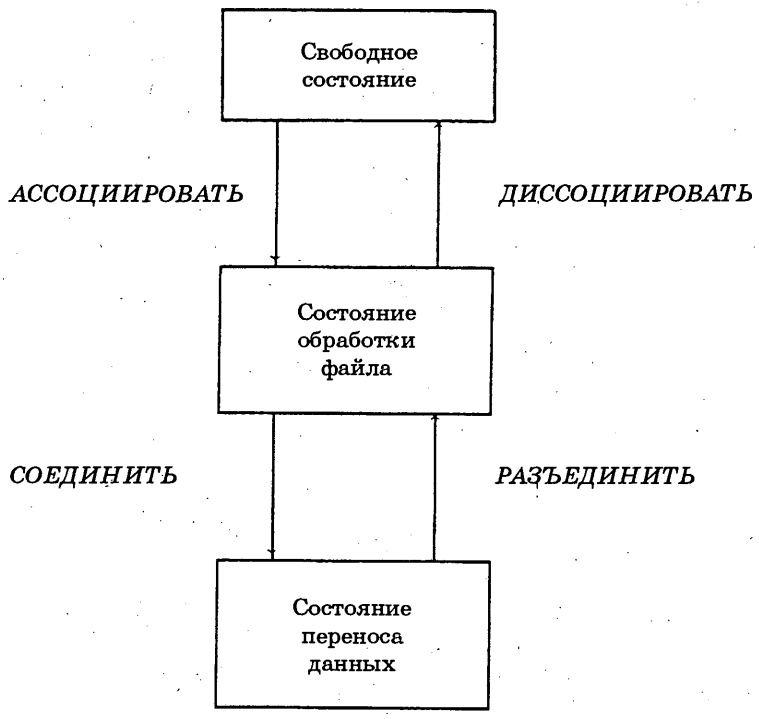
Ссылочное примитивное значение должно давать **размещенное** ссылочное значение. Время жизни ссылочной ячейки не должно заканчиваться.

7 ВВОД И ВЫВОД

7.1 ЭТАЛОННАЯ МОДЕЛЬ ВВОДА/ВЫВОДА

Модель используется для описания средств ввода/вывода в зависимости от реализации; в ней различаются три состояния данной ассоциативной ячейки: свободное состояние, состояние обработки файла и состояние переноса данных.

На приведенной ниже схеме указаны три состояния и возможные переходы от состояния к состоянию.



Ассоциативная ячейка не содержит значения. Нет связи с объектом внешней среды.

Ассоциативная ячейка содержит ассоциацию. Операции, подобные операциям создать и удалить файл или изменить его свойства.

Ячейка доступа связана с ассоциативной ячейкой. Перенос данных в файл или из файла: операции чтения и записи.

В модели предполагается, что объекты, на которые при реализации часто ссылаются как на наборы данных, файлы или устройства, существуют во внешней среде, то есть во внешней среде CHILL-программы. Такой объект внешней среды в модели называется файлом. Файл может быть физическим устройством, линией связи и даже файлом в системе управления файлами; вообще говоря, файл — это объект, вырабатывающий и/или использующий данные.

Операции с файлами на языке CHILL требуют наличия ассоциации; ассоциация создается ассоциативной операцией и она идентифицирует файл. Ассоциация имеет атрибуты; эти атрибуты описывают свойства файла, который закреплен или мог быть закреплен за ассоциацией.

В свободном состоянии нет связи или взаимодействия между CHILL-программой и объектами внешней среды. Ассоциативная операция изменяет состояние модели от свободного состояния до состояния обработки файла. В качестве одного аргумента эта операция берет ассоциативную ячейку и определяемое реализацией обозначение объекта внешней среды, для которого должна быть создана ассоциация; могут использоваться дополнительные аргументы для указания вида ассоциации для объекта и начальных значений атрибутов ассоциации. Отдельная ассоциация также подразумевает (зависящий от реализации) набор операций, которые могут применяться к файлу, закрепленному за этой ассоциацией.

В состоянии обработки файла можно оперировать файлом и его свойством через ассоциацию при условии, что ассоциация включает отдельную операцию; для операций, изменяющих свойства файла, для файла вообще потребуется исключающая ассоциация.

Вообще, в модели используются исключающие ассоциации, то есть в одно и то же время для данного объекта внешней среды имеется только одна ассоциация. Однако при реализациях допускается создание нескольких ассоциаций для одного и того же объекта при условии, что объект используется разными пользователями (программами) и/или разными ассоциациями в одной и той же программе. Все операции в состоянии обработки файла используют ассоциацию в качестве аргумента.

Для окончания ассоциации для объекта внешней среды используется диссоциативная операция; эта операция вызывает переход от состояния обработки файла снова к свободному состоянию.

Перенос данных в файл или из файла возможен только в состоянии переноса данных; операции переноса требуют, чтобы ячейка доступа была соединена с ассоциацией для данного файла. Операция соединения соединяет ячейку доступа с ассоциацией и изменяет состояние модели на состояние переноса данных. Операция использует в качестве аргументов ассоциативную ячейку и ячейку доступа; ассоциативная ячейка содержит ассоциацию для файла, в который или из которого данные могут быть перенесены посредством ячейки доступа. Дополнительные аргументы операции соединения обозначают, для какого типа операций переноса должна быть соединена ячейка доступа, и на какую запись должен быть настроен файл. В одно и то же время с ассоциативной ячейкой может быть соединена только одна ячейка доступа.

Операция разъединения использует ячейку доступа в качестве аргумента и разъединяет ее с ассоциацией, с которой она была соединена; она изменяет состояние модели снова на состояние обработки файла.

В состоянии переноса данных ячейка доступа должна использоваться как аргумент операции переноса; имеются две операции переноса, а именно, операция чтения для переноса данных из файла в программу и операция записи для переноса данных из программы в файл. Операции переноса используют вид ячейки доступа для преобразования CHILL-значений в записи файла и наоборот.

Файл рассматривается в модели как массив значений; каждый элемент этого массива относится к записи файла. Вид элемента этого массива определяется операцией соединения, являющейся видом записи соединяемой ячейки доступа. Значение индекса присваивается каждой записи файла; это значение однозначно идентифицирует каждую запись файла. В описании операций соединения и переноса будут использоваться три специальных значения индекса, а именно, **базовый** индекс, **текущий** индекс и индекс **переноса**. **Базовый** индекс устанавливается операцией соединения и остается неизменным до следующей операции соединения; он используется для вычисления индекса **переноса** в операциях переноса и **текущего** индекса в операции соединения. Индекс **переноса** обозначает позицию в файле, где будет иметь место перенос; **текущий** индекс обозначает запись, на которую в настоящий момент настроен файл.

7.2 АССОЦИАТИВНЫЕ ЗНАЧЕНИЯ

7.2.1 Общие положения

Ассоциативное значение отражает свойства файла, который закреплен или может быть закреплен за ним. Значение отдельной ассоциации также подразумевает (зависящий от применения) набор операций с файлом, возможно, закрепленных за ним.

Ассоциативные значения не имеют обозначения, но они содержатся в ячейках ассоциативного вида; не существует выражений, обозначающих значение ассоциативного вида. Ассоциативными значениями могут только оперировать встроенные программы, использующие ассоциативную ячейку в качестве параметра.

7.2.2 Атрибуты ассоциативных значений

Ассоциативное значение имеет атрибуты; атрибуты описывают свойства ассоциации и файла, которые закреплены или могут быть закреплены за ним.

В языке определены следующие атрибуты:

- **существующий**: указывает, что (возможно, пустой) файл закреплен за ассоциацией;
- **читаемый**: указывает, что операции чтения возможны для файла, когда он закреплен за ассоциацией;
- **возможный для записи**: указывает, что операции записи возможны для файла, когда он закреплен за ассоциацией;
- **индексируемый**: указывает, что файл, будучи закрепленным за ассоциацией, позволяет осуществлять произвольный доступ к своим записям;
- **последовательный**: указывает, что файл, будучи закрепленным за ассоциацией, позволяет осуществлять последовательный доступ к своим записям;
- **переменный**: указывает, что **размер** записей файла, когда он закреплен за ассоциацией, может меняться в пределах файла.

Эти атрибуты имеют булевское значение; атрибуты инициализируются при создании ассоциации и могут изменяться в результате отдельных операций над ассоциацией. Этот список содержит атрибуты, определяемые только в языке; при реализациях могут быть добавлены новые атрибуты в соответствии с потребностями реализаций.

7.3 ЗНАЧЕНИЯ ДОСТУПА

7.3.1 Общие положения

Значения доступа содержатся в ячейках доступного вида. Ячейка доступа необходима для переноса данных из файла во внешнюю среду и наоборот.

Значения доступа не имеют обозначения, но содержатся в ячейках доступного вида; не существует выражения, обозначающего значение доступного вида. Значениями доступа могут только оперировать встроенные программы, использующие ячейку доступа в качестве параметра.

7.3.2 Атрибуты значений доступа

Значения доступа имеют атрибуты, описывающие динамические свойства значений, семантику операций переноса и условия возникновения исключений.

В языке CHILL определены следующие атрибуты:

- **использование**: указывает, для какой(их) операции(ий) переноса ячейка доступа соединена с ассоциацией; атрибут устанавливается операцией соединения;
- **вне файла**: указывает, находится или не находится индекс **переноса**, вычисленный последней операцией чтения, в файле; атрибут инициализируется операцией соединения значением *FALSE* и устанавливается каждой операцией чтения.

7.4 ВСТРОЕННЫЕ ПРОГРАММЫ ВВОДА—ВЫВОДА

7.4.1 Общие положения

Встроенные программы, определяемые в языке, определены для операций на ассоциативных ячейках и ячейках доступа и для проверки и изменения атрибутов их значений.

Встроенные программы будут описаны в последующих разделах.

Синтаксис:

< вызов встроенной программы выдачи значения ввода—вывода > :: = (1)
< вызов встроенной программы выдачи атрибутов ассоциации > (1.1)
| *< вызов встроенной программы индикации ассоциации >* (1.2)
| *< вызов встроенной программы выдачи атрибутов доступа >* (1.3)
| *< вызов встроенной программы переноса данных из файла >* (1.4)
| *< вызов встроенной программы получения текста >* (1.5)

< вызов простой встроенной программы ввода—вывода > :: = (2)
< вызов встроенной программы диссоциации > (2.1)
| *< вызов встроенной программы модификации >* (2.2)
| *< вызов встроенной программы соединения >* (2.3)
| *< вызов встроенной программы разъединения >* (2.4)
| *< вызов встроенной программы переноса данных в файл >* (2.5)
| *< вызов встроенной программы переноса текста >* (2.6)
| *< вызов встроенной программы хранения текста >* (2.7)

< вызов встроенной программы выдачи ячейки ввода—вывода > :: = (3)
< вызов встроенной программы ассоциации > (3.1)

Статические условия: *Параметр встроенной программы* во встроенной программе ввода—вывода, то есть *ассоциативная ячейка*, *ячейка доступа* или *текста*, должен быть **ссылочным**.

7.4.2 Ассоциация с объектом внешней среды

Синтаксис:

< вызов встроенной программы ассоциации > :: = (1)
ASSOCIATE (< ассоциативная ячейка > [*, < список ассоциативных параметров >*] *)* (1.1)

< вызов встроенной программы индикации ассоциации > :: = (2)
ISASSOCIATED (< ассоциативная ячейка >) (2.1)

< список ассоциативных параметров > :: = (3)
< ассоциативный параметр > { *, < ассоциативный параметр >* }* (3.1)

< ассоциативный параметр > :: = (4)
< ячейка > (4.1)
| *< значение >* (4.2)

Семантика: *ASSOCIATE* создает ассоциацию с объектом внешней среды. Она инициализирует *ассоциативную ячейку* с помощью созданной ассоциации, а также атрибуты созданной ассоциации. Ассоциативная ячейка также возвращается в результате вызова встроенной программы. Отдельная созданная ассоциация определяется ячейками и/или значениями, входящими в *список ассоциативных параметров*; виды (классы) и семантика этих ячеек (значений) зависят от реализации.

ISASSOCIATED возвращает значение *TRUE*, если *ассоциативная ячейка* содержит ассоциацию, и значение *FALSE* в противном случае.

Статические свойства: Класс вызова встроенной программы *ISASSOCIATED* является *BOOL*-производным классом. Вид вызова встроенной программы *ASSOCIATE* является видом *ассоциативной ячейки*.

Региональность вызова встроенной программы *ASSOCIATION* та же, что и у *ассоциативной ячейки*.

Статические условия: Вид и класс каждого *ассоциативного параметра* определяется реализацией.

Динамические условия: Если *ассоциативная ячейка* уже содержит ассоциацию или если ассоциация не может быть создана по причинам, зависящим от реализации, то *ASSOCIATE* может стать причиной появления исключительной ситуации *ASSOCIATEFAIL*.

Примеры:

20.21 ASSOCIATE (file_association, "DSK:RECORDS.DAT"); (1.1)

7.4.3 Диссоциация с объектом внешней среды

Синтаксис:

< вызов встроенной программы диссоциации > :: = (1)
DISSOCIATE (< ассоциативная ячейка >) (1.1)

Семантика: *DISSOCIATE* прекращает ассоциацию с объектом внешней среды. Ячейка доступа, все еще соединенная с ассоциацией, содержащейся в ассоциативной ячейке, разъединяется до прекращения ассоциации.

Динамические условия: Если *ассоциативная ячейка* не содержит ассоциацию, то *DISSOCIATE* вызывает исключение *NOTASSOCIATED*.

Примеры:

22.38 DISSOCIATE (association); (1.1)

7.4.4 Доступ к ассоциативным атрибутам

Синтаксис:

< вызов встроенной программы выдачи атрибутов ассоциации > :: = (1)
EXISTING (< ассоциативная ячейка >) (1.1)
| *READABLE (< ассоциативная ячейка >)* (1.2)
| *WRITEABLE (< ассоциативная ячейка >)* (1.3)
| *INDEXABLE (< ассоциативная ячейка >)* (1.4)
| *SEQUENCIBLE (< ассоциативная ячейка >)* (1.5)
| *VARIABLE (< ассоциативная ячейка >)* (1.6)

Семантика: *EXISTING*, *READABLE*, *WRITEABLE*, *INDEXABLE*, *SEQUENCIBLE* и *VARIABLE* возвращают соответственно значение атрибута **(существующий, читаемый, возможный для записи, индексируемый, последовательный, переменный)** ассоциации, содержащегося в *ассоциативной ячейке*.

Статические свойства: Класс *вызова встроенной программы выдачи атрибутов ассоциации* является *BOOL*-производным классом.

Динамические условия: Вызов *встроенной программы выдачи атрибутов ассоциации* приводит к исключительной ситуации *NOTASSOCIATED*, если *ассоциативная ячейка* не содержит ассоциацию.

7.4.5 Модификация ассоциативных атрибутов

Синтаксис:

< вызов встроенной программы модификации > :: = (1)
CREATE (< ассоциативная ячейка >) (1.1)
| *DELETE (< ассоциативная ячейка >)* (1.2)
| *MODIFY (< ассоциативная ячейка > [, < список параметров модификации >])* (1.3)

< список параметров модификации > :: = (2)
*< параметр модификации > { , < параметр модификации > }** (2.1)

< параметр модификации > :: = (3)
< значение > (3.1)
| *< ячейка >* (3.2)

Семантика: *CREATE* создает пустой файл и закрепляет его за ассоциацией, обозначенной *ассоциативной ячейкой*. Атрибут **существующий** указанной ассоциации устанавливается в *TRUE*, если операция достигает цели.

DELETE отделяет файл от ассоциации, обозначенной *ассоциативной ячейкой* и удаляет файл. Атрибут **существующий** указанной ассоциации устанавливается в *FALSE*, если операция достигает цели.

MODIFY обеспечивает средства изменения приоритетов объекта внешней среды, для которого существует ассоциация и который обозначается *ассоциативной ячейкой*; ячейки и/или значения, имеющиеся в *списке параметров модификации*, описывают, каким образом должны модифицироваться приоритеты. Виды (классы) и семантика этих ячеек (значений) определяются реализацией.

Динамические условия: Если *ассоциативная ячейка* не содержит ассоциацию, то *CREATE*, *DELETE* и *MODIFY* вызывают исключительную ситуацию *NOTASSOCIATED*.

CREATE вызывает исключительную ситуацию *CREATEFAIL*, если имеет место одно из следующих условий:

- атрибут **существующий** ассоциации имеет значение *TRUE*;
- создание файла не удается (определяется реализацией).

DELETE вызывает исключительную ситуацию *DELETEFAIL*, если имеет место одно из следующих условий:

- атрибут **существующий** ассоциации имеет значение *FALSE*;
- удаление файла не происходит (зависит от реализации).

MODIFY вызывает исключительную ситуацию *MODIFYFAIL*, если свойства, определяемые *списком параметров модификации*, не могут быть модифицированы или же это не разрешается; условия возникновения этой исключительной ситуации определяются реализацией.

Примеры:

21.39 *CREATE (outassoc);* *(1.1)*

21.69 *DELETE (curassoc);* *(1.2)*

7.4.6 Соединение с ячейкой доступа

Синтаксис:

< вызов встроенной программы соединения > :: = *(1)*

CONNECT (< ячейка переноса, ассоциативная ячейка >,
< выражение использования > [, < где-выражение >
[, < индексное выражение >]]) *(1.1)*

< ячейка переноса > :: = *(2)*
< ячейка доступа > *(2.1)*
| *< ячейка текста >* *(2.2)*

< выражение использования > :: = *(3)*
< выражение > *(3.1)*

< где-выражение > :: = *(4)*
< выражение > *(4.1)*

< индексное выражение > :: = *(5)*
< выражение > *(5.1)*

Предопределенные имена: Для управления операцией соединения, выполняемой встроенной программой *CONNECT*, в языке предопределены два имени **синонимического вида**, а именно, *USAGE* (*ИСПОЛЬЗОВАНИЕ*) и *WHERE* (*ГДЕ*); определяющими их видами являются **SET** (*READONLY, WRITEONLY, READWRITE*) и **SET** (*FIRST, SAME, LAST*) соответственно.

Значения вида *USAGE* указывают, для какого типа операций переноса ячейка доступа должна быть соединена с ассоциацией, в то время как значения вида *WHERE* указывают, как файл, закрепленный за ассоциацией, должен быть размещен операцией соединения.

Семантика: *CONNECT* соединяет ячейку доступа, обозначенную *ячейкой переноса*, с ассоциацией, содержащейся в *ассоциативной ячейке*; должен иметься файл, закрепленный за обозначенной ассоциацией, то есть атрибут **существующий** ассоциации должен иметь значение *TRUE*.

Ячейка доступа, обозначенная *ячейкой переноса*, сама является ячейкой, если это *ячейка доступа*; в противном случае это подъячейка доступа *ячейки текста*.

Значение, вырабатываемое *выражением использования*, указывает, для какого типа операций переноса ячейка доступа должна быть соединена с файлом. Если выражение вырабатывает *READONLY*, то соединение подготовлено для операций только чтения; если оно вырабатывает *WRITEONLY*, соединение устанавливается для операций только записи; если оно вырабатывает *READWRITE*, то соединение устанавливается для операций чтения и записи.

Атрибут **индексируемый** обозначенной ассоциации должен иметь значение *TRUE*, если ячейка доступа имеет **индексный** вид, в то время как атрибут **последовательный** должен иметь значение *TRUE*, если ячейка не имеет **индексного** вида.

CONNECT располагает (повторно) файл, закрепленный за обозначенной ассоциацией; то есть она устанавливает (новый) **базовый** индекс и **текущий** индекс в файле. (Новый) **базовый** индекс зависит от значения, вырабатываемого *где-выражением*:

- если *где-выражение* вырабатывает *FIRST* или оно не специфицировано, то **базовый** индекс устанавливается на 0; то есть файл располагается перед первой записью;
- если *где-выражение* вырабатывает *SAME*, то **базовый** индекс устанавливается на **текущий** индекс в файле; то есть позиция файла не изменяется;
- если *где-выражение* вырабатывает *LAST*, то **базовый** индекс устанавливается на N, где N обозначает количество записей в файле; то есть файл располагается после последней записи.

После установки **базового** индекса **текущий** индекс будет устанавливаться программой *CONNECT*. Этот **текущий** индекс зависит от дополнительной спецификации *индексного выражения*:

- если никакое индексное выражение не специфицируется, **текущий** индекс устанавливается на (новый) **базовый** индекс;
- если *индексное выражение* специфицируется, то **текущий** индекс устанавливается на

  базовый индекс + *NUM*(*v*) – *NUM*(*l*),

  где *l* обозначает **нижнюю границу индексного** вида ячейки доступа, а *v* — значение, вырабатываемое *индексным выражением*.

Если ячейка доступа соединяется для последующих операций записи (то есть ячейка доступа не имеет **индексного** вида, а *выражение использования* вырабатывает *WRITEONLY*), тогда эти записи в файле, имеющем индекс больше (нового) **текущего** индекса, будут исключены из файла; то есть файл может быть усечен или очищен программой *CONNECT*.

Ячейка доступа, не имеющая индексного вида, не может быть соединена с ассоциацией для одновременного выполнения операций чтения и записи.

Любая ячейка доступа, с которой может быть соединена обозначенная ассоциация, будет неявно разъединена перед тем, как ассоциация будет соединена с ячейкой, обозначенной *ячейкой переноса*.

CONNECT инициализирует атрибут **вне файла** ячейки доступа значением *FALSE* и устанавливает атрибут **использование** в соответствии со значением, вырабатываемым *выражением использования*.

Статические свойства: Вид, закрепленный за *ячейкой переноса*, является видом *ячейки доступа* или **доступным** видом *ячейки текста* соответственно.

Статические условия: Вид *ячейки переноса* должен быть **индексным** видом, если описано *индексное выражение*; класс значения, вырабатываемого *индексным выражением*, должен быть **совместимым** с этим **индексным** видом. *Ячейка переноса* должна иметь ту же **региональность**, что и *ассоциативная ячейка*.

Класс значения, вырабатываемого *выражением использования*, должен быть **совместимым** с *USAGE*-производным классом.

Класс значения, вырабатываемого *где-выражением*, должен быть **совместимым** с *WHERE*-производным классом.

Динамические условия: Если *ассоциативная ячейка* не содержит ассоциацию, то *CONNECT* вызывает исключительную ситуацию *NOTASSOCIATED*.

CONNECT является причиной исключительной ситуации *CONNECTFAIL*, если имеет место одно из следующих условий:

- атрибут **существующий** ассоциации есть *FALSE*;
- атрибут **читаемый** ассоциации есть *FALSE* и *выражение использования* вырабатывает *READONLY* или *READWRITE*;
- атрибут **возможный для записи** ассоциации есть *FALSE* и *выражение использования* вырабатывает *WRITEONLY* или *READWRITE;*
- атрибут **индексируемый** ассоциации есть *FALSE* и ячейка доступа имеет **индексный** вид;
- атрибут **последовательный** ассоциации есть *FALSE* и ячейка доступа не имеет **индексного** вида;
- *где-выражение* вырабатывает *SAME*, в то время как ассоциация, содержащаяся в *ассоциативной ячейке*, не соединена с ячейкой доступа;
- атрибут **переменный** ассоциации есть *FALSE* и ячейка доступа имеет **динамический** вид **записи**, в то время как *выражение использования* вырабатывает *WRITEONLY* или *READWRITE*;

- атрибут **переменный** ассоциации есть *TRUE* и ячейка доступа имеет **статический** вид **записи**, в то время как *выражение использования* вырабатывает *READONLY* или *READWRITE*;

- ячейка доступа имеет **индексный** вид, в то время как *выражение использования* вырабатывает *READWRITE*;

- ассоциация, содержащаяся в *ассоциативной ячейке*, не может быть соединена с ячейкой доступа из-за условий, определяемых реализацией.

CONNECT может быть причиной исключительной ситуации *RANGEFAIL*, если **индексный** вид ячейки доступа является ограниченным видом и *индексное выражение* вырабатывает значение, находящееся за границами этого ограниченного вида.

Имеет место исключение *EMPTY*, если **ссылка доступа** *ячейки текста* выдает значение *NULL*.

Примеры:

20.22 CONNECT (record_file, file_association, READWRITE); *(1.1)*

20.22 READWRITE *(3.1)*

7.4.7 Разъединение с ячейкой доступа

Синтаксис:

< вызов встроенной программы разъединения > :: = *(1)*
DISCONNECT (< ячейка переноса >) *(1.1)*

Семантика: *DISCONNECT* разъединяет *ячейку доступа*, обозначенную как *ячейка переноса*, с ассоциацией, с которой она была связана.

Динамические условия: *DISCONNECT* является причиной исключения *NOTCONNECTED*, если ячейка доступа, обозначенная как *ячейка переноса*, не соединена с ассоциацией.

7.4.8 Доступ к атрибутам ячеек доступа

Синтаксис:

<вызов встроенной программы выдачи атрибутов доступа>::= *(1)*
GETASSOCIATION (< ячейка переноса >) *(1.1)*
| *GETUSAGE (< ячейка переноса >)* *(1.2)*
| *OUTOFFILE (< ячейка переноса >)* *(1.3)*

Семантика: *GETASSOCIATION* возвращает ссылочное значение в ассоциативную ячейку, с которой соединена ячейка доступа, обозначенная *ячейкой переноса*; она возвращает *NULL*, если ячейка доступа не соединена с ассоциацией.

GETUSAGE возвращает значение атрибута **использование**; то есть *READONLY* (*WRITEONLY*), если ячейка доступа соединена только для операций чтения (записи), или *READWRITE*, если ячейка доступа соединена как для операций чтения, так и записи.

OUTOFFILE возвращает значение атрибута **вне файла** ячейки доступа; то есть *TRUE*, если последняя операция чтения вычисляла индекс **переноса**, который был вне файла; в противном случае *FALSE*.

Статические свойства: Класс вызова встроенной программы *GETASSOCIATION* есть *ASSOCIATION*-ссылочный класс. **Региональность** вызова встроенной программы *GETASSOCIATION* та же, что и у *ячейки переноса*.

Класс вызова встроенной программы *OUTOFFILE* — это *BOOL*-производный класс.

Класс вызова встроенной программы *GETUSAGE* — это *USAGE*-производный класс.

Динамические условия: *GETUSAGE* и *OUTOFFILE* могут быть причиной исключения *NOTCONNECTED*, если ячейка доступа не соединена с ассоциацией.

Примеры:

21.47 OUTOFFILE (infiles (FALSE)) *(1.3)*

7.4.9 Операции переноса данных

Синтаксис:

< вызов встроенной программы переноса данных из файла > :: = *(1)*
READRECORD (< ячейка доступа > [, < индексное выражение >]
[, < ячейка памяти >]) *(1.1)*

< вызов встроенной программы переноса данных в файл > : : = *(2)*
WRITERECORD (< ячейка доступа > [, < индексное выражение >] ,
< выражение записи > *(2.1)*

< ячейка памяти > :: = *(3)*
< ячейка статического вида > *(3.1)*

< выражение записи > :: = *(4)*
< выражение > *(4.1)*

Примечание.— Если *ячейка доступа* имеет **индексный** вид, то синтаксическая неопределенность разрешается путем интерпретации второго аргумента как *индексного выражения*, а не *ячейки памяти*.

Семантика: Для переноса данных из файла или в файл определены встроенные программы *WRITERECORD* и *READRECORD*. *Ячейка доступа* должна иметь вид **записи** и соединяться с ассоциацией для переноса данных из файла или в файл, закрепленный за этой ассоциацией. Направление переноса не должно противоречить значению атрибута **использование** *ячейки доступа*.

Перед переносом вычисляется индекс **переноса**, то есть позиция переносимой записи в файле. Если *ячейка доступа* не имеет **индексного** вида, тогда индекс **переноса** равен **текущему** индексу, увеличенному на 1; если *ячейка доступа* имеет **индексный** вид, то индекс **переноса** вычисляется следующим образом:

индекс **переноса** : = **базовый** индекс + $NUM(v) - NUM(l) + 1$,

где l — это **нижняя граница** вида для **индексного** вида *ячейки доступа*, а v — значение, вырабатываемое *индексным выражением*. Если перенос записи с вычисленным индексом **переноса** выполнен успешно, то **текущий** индекс становится индексом **переноса**.

Операция чтения

READRECORD переносит данные из файла во внешней среде в CHILL-программу.

Если вычисленный индекс **переноса** не находится в файле, то атрибут **вне файла** устанавливается по значению *TRUE*; в противном случае файл размещается, запись считывается, а атрибут **вне файла** устанавливается по значению *FALSE*.

Считанная запись не должна иметь **неопределенное** значение; результат операции чтения определяется реализацией, если считываемая из файла запись не является допустимым значением согласно виду **записи** *ячейки доступа*.

Если *ячейка памяти* описана, тогда значение считанной записи присваивается этой ячейке. Если *ячейка памяти* не описана, то значение будет присвоено неявно созданной ячейке; время жизни этой ячейки заканчивается, когда *ячейка доступа* разъединяется или соединяется повторно. Не определяется, создается ли ссылаемая ячейка операцией соединения только один раз или же всякий раз, когда выполняется операция чтения.

READRECORD в обоих случаях возвращает ссылочное значение, относящееся к (возможно, динамического вида) ячейке, которой было присвоено значение.

Если атрибут **вне файла** установлен по значению *TRUE* в результате вызова встроенной программы, тогда значение *NULL* возвращается в результате этого вызова.

Операция записи

WRITERECORD переносит данные из CHILL-программы в файл во внешней среде. Файл настраивается на запись по вычисленному индексу и запись осуществляется.

После успешного проведения записи количество записей заносится в индекс **переноса**, если последний больше фактического количества записей.

Запись, осуществленная программой *WRITERECORD*, является значением, вырабатываемым *выражением записи*.

Статические свойства: Класс значения, которое было считано программой *READRECORD*, является классом M-значения, где M — вид **записи** *ячейки доступа*, если она имеет **статический** вид **записи**, или динамически параметризованной версией *ячейки доступа*, если ячейка имеет **динамический** вид **записи**; параметрами такого динамически параметризованного вида записи являются:

- динамическая **длина строки** значения строки, которое было считано в случае строкового вида;
- динамическая **верхняя граница** значения массива, которое было считано в случае массивного вида;
- список (теговых) значений, связанных с видом значения структуры, которое было считано в случае **вариантной** структуры.

Класс вызова встроенной программы *READRECORD* является M-ссылочным классом, если *ячейка памяти* не описана; в противном случае это S-ссылочный класс, где S — вид *ячейки памяти*.

Региональность вызова встроенной программы *READRECORD* та же, что и у *ячейки памяти*, если она описана; в противном случае это класс *ячейки доступа*.

Статические условия: *Ячейка доступа* должна иметь вид **записи**.

Индексное выражение может быть не описано, если *ячейка доступа* не имеет **индексного** вида, и должно быть описано, если *ячейка доступа* имеет **индексный** вид; класс значения, вырабатываемого *индексным выражением*, должен быть **совместимым** с **индексным** видом.

Ячейка памяти должна быть **ссылочной**.

Вид *ячейки памяти* не должен иметь **свойства неизменяемости**.

Если *ячейка памяти* описана, тогда вид *ячейки памяти* должен быть **эквивалентным** виду **записи** *ячейки доступа*, если она имеет **статический** вид **записи** или **изменяющийся** строковый вид **записи**; в противном случае это динамически параметризованная версия ее; параметры такого динамически параметризованного вида являются параметрами считанного значения.

Класс значения, вырабатываемого *выражением* записи должен быть **совместимым** с видом **записи** *ячейки доступа*, если она имеет *статический* вид **записи** или **изменяющийся** строковый вид **записи**, в противном случае должна существовать динамически параметризованная версия вида **записи**, которая **совместима** с классом *выражения записи*. Применяются условия присваивания значения *выражения записи* по отношению к упомянутому выше виду.

Динамические условия: Если не выполняется динамическая часть упомянутой выше проверки на совместимость, то имеют место исключения *RANGEFAIL* или *TAGFAIL*.

Вызов встроенных программ *READRECORD* и *WRITERECORD* может быть причиной исключения *NOTCONNECTED*, если *ячейка доступа* не соединена с ассоциацией.

Вызов встроенных программ *READRECORD* и *WRITERECORD* может быть причиной исключения *RANGEFAIL*, если **индексный** вид *ячейки доступа* является ограниченным видом, а *индексное выражение* вырабатывает значение, находящееся вне границ данного ограниченного вида.

Вызов встроенной программы *READRECORD* приводит к исключению *READFAIL*, если имеет место одно из следующих условий:

- значение атрибута **использование** есть *WRITEONLY*;
- значение атрибута **вне файла** есть *TRUE*, а *ячейка доступа* соединена с последующими операциями чтения;
- из-за условий внешней среды считывание записи с вычисленным индексом не удается.

Вызов встроенной программы *WRITERECORD* приводит к исключению *WRITEFAIL*, если имеет место одно из следующих условий:

- значение атрибута **использование** есть *READONLY*;
- из-за условий внешней среды осуществление записи с вычисленным индексом не удается.

Если имеет место исключение *RANGEFAIL* или исключение *NOTCONNECTED*, то эти исключения имеют место перед тем, как изменяется значение любого атрибута и размещается файл.

Примеры:

20.24	*READRECORD (record_file, curindex, record_buffer);*	*(1.1)*
22.25	*READRECORD (fileaccess);*	*(1.1)*
20.32	*WRITERECORD (record_file, curindex, record_buffer);*	*(2.1)*
21.61	*WRITERECORD (outfile, buffers(flag));*	*(2.1)*
20.24	*record_buffer*	*(3.1)*
21.61	*buffers(flag)*	*(4.1)*

7.5 ВВОД—ВЫВОД ТЕКСТА

7.5.1 Общие положения

Операции вывода текста позволяют представить CHILL-значения в удобной для чтения форме; операции ввода текста осуществляют обратное преобразование.

Операции переноса текста определяются в основной модели ввода/вывода в языке CHILL и оперируют с файлами, доступ к которым может быть последовательным или произвольным, а записи файлов могут иметь фиксированную или переменную длину.

В модели предполагается, что каждая запись имеет приписанную информацию по расположению (возможно отсутствие такой информации), которая используется при реализациях в виде символов управления кареткой или просто управляющих символов.

Оперирование с текстовыми файлами в языке CHILL требует наличия ассоциации; перенос данных в текстовый файл или из текстового файла требует наличия ячейки **текста**, связанной с ассоциацией для данного файла.

Операции переноса текста могут применяться к значениям в языке CHILL, которые могут быть записями некоторого текстового файла, и к ячейкам в языке CHILL, которые не обязательно относятся к работе программы по вводу—выводу.

Возможность восстановления из части текста исходных значений, записанных на языке CHILL, вообще говоря, не гарантируется, но скорее всего, это зависит от специфики используемого представления значений.

Текстовые значения содержатся в ячейках текстового вида. Необходимо, чтобы ячейка текста переносила данные в удобной для чтения форме.

Текстовые значения не имеют обозначения, но они содержатся в ячейках текстового вида; не существует выражения, обозначающего значение текстового вида. Текстовые значения используются только встроенными программами, использующими ячейки текста в качестве параметров.

7.5.2 Атрибуты текстовых значений

Текстовые значения обладают атрибутами, описывающими их динамические свойства. Определены следующие атрибуты:

- **фактический индекс**: указывает позицию следующего символа **записи текста**, который должен быть считан или записан. Атрибут имеет вид, являющийся *INT (0:L)*, где *L* — **длина текста** вида значения. Он инициализируется значением 0 при создании ячейки текста;
- **ссылка на запись текста**: указывает ссылочное значение для подъячейки **записи текста** ячейки текста. Атрибут имеет вид, являющийся **REF** *M*, где *M* — это вид **записи текста** для вида значения.
- **ссылка доступа**: указывает ссылочное значение для подъячейки **доступа** ячейки текста. Атрибут имеет вид, являющийся **REF** *M*, где *M* — вид **доступа** для вида значения.

7.5.3 Операции переноса текста

Синтаксис:

< вызов встроенной программы переноса текста > :: = (1)
READTEXT (< список аргументов ввода—вывода текста >) (1.1)
| *WRITETEXT (< список аргументов ввода—вывода текста >)* (1.2)

<список аргументов ввода—вывода текста> :: = (2)
< аргумент текста > [, *< индексное выражение >*] ,
< аргумент формата > [*< список ввода—вывода >*] (2.1)

< аргумент текста > :: = (3)
< ячейка текста > (3.1)
| *< ячейка символьной строки >* (3.2)
| *< выражение символьной строки >* (3.3)

< аргумент формата > :: = (4)
< выражение символьной строки > (4.1)

< список ввода—вывода > :: = (5)
< элемент списка ввода—вывода > { , *< элемент списка ввода—вывода >* }* (5.1)

< элемент списка ввода—вывода > :: = (6)
< аргумент значения > (6.1)
| *< аргумент ячейки >* (6.2)

< аргумент ячейки > :: = (7)
< дискретная ячейка > (7.1)
| *< ячейка строки >* (7.2)

< аргумент значения > :: = (8)
< дискретное выражение > (8.1)
| *<выражение строки>* (8.2)

Примечание.— Если *элемент списка ввода—вывода* — это ячейка, то синтаксическая неопределенность разрешается путем интерпретации *элемента списка ввода—вывода* как *аргумента ячейки*, а не *аргумента значения*.

Семантика: *READTEXT* использует функции преобразования, редактирования и управления вводом—выводом, содержащиеся в *аргументе формата* **записи текста**, обозначенном *аргументом текста*; она (возможно) вырабатывает список значений, присваиваемых элементам *списка ввода—вывода* в той последовательности, как они описаны. Программа *WRITETEXT* выполняет обратную операцию. Неявные операции ввода/вывода не выполняются.

Если аргумент текста — это *ячейка символьной строки* или *выражение символьной строки*, тогда применяются функции преобразования и редактирования без какой-либо связи с внешней средой. В этом случае **фактический индекс** обозначает ячейку, созданную неявно в начале вызова встроенной программы и инициализированную нулем (0). **Запись текста** — это символьная строка, обозначенная *ячейкой символьной строки* или *выражением символьной строки*, а **длина текста** — это **длина строки**.

Элементами *списка ввода—вывода* могут быть:

- *аргументы значений* и *аргументы ячеек* или
- **переменные** *размеры предложения*, как описано ниже.

Связь между аргументом формата и списком ввода—вывода

Значение, выдаваемое *аргументом формата*, должно иметь форму *строки управления форматом* (см. раздел 7.5.4).

Во время вызова встроенной программы ввода—вывода текста *строка управления форматом* (см. раздел 7.5.4), обозначенная *аргументом формата* и *списком ввода—вывода*, сканируется слева направо. Каждое появление *текста формата* и *спецификации формата* интерпретируется и предпринимаются следующие соответствующие действия:

a. *текст формата*

В *READTEXT* **запись текста** должна содержать на месте **фактического индекса** подстроку, равную строке из *текста формата*. В *WRITETEXT* строка из *текста формата* переносится в **запись текста**. Семантика та же, как если бы встречались *спецификация формата* %*C* и *элемент списка ввода—вывода*, дающий то же значение строки, что и получаемые из *текста формата*.

b. *спецификация формата*

Если *спецификация формата* содержит *коэффициент повторения*, тогда она эквивалентна такому количеству *элементов формата*, какое указано *коэффициентом повторения*.

Если *спецификация формата* — это *предложение формата*, тогда она содержит *управляющий код*. Если *управляющий код* — это *предложение преобразования*, тогда *элемент списка ввода—вывода* берется из *списка ввода—вывода* и к нему применяется функция преобразования, выбираемая *кодом преобразования*, *спецификаторами преобразования* и *размером предложения* (см. раздел 7.5.5). Если *управляющий код* — это *предложение редактирования* или предложение ввода—вывода, тогда функция редактирования или ввода—вывода, выбираемая по *коду редактирования* или *коду ввода—вывода* и *размеру предложения*, применяется к *аргументу текста* без ссылки на *список ввода—вывода* (см. разделы 7.5.6 и 7.5.7).

Если *размер предложения* — **переменный**, то значение берется из списка, обозначающего параметр **размера** преобразования или функцию управления редактированием.

Если *спецификация формата* — это *параметризованное предложение*, тогда сканируется *строка управления форматом*, содержащаяся в нем.

Интерпретация *строки управления форматом* заканчивается, когда будет достигнут конец строки, информация о котором вырабатывается *строкой управления форматом*.

Элементы списка ввода—вывода из *списка ввода—вывода* сканируются в том порядке, как они описаны.

Статические условия: Если *аргумент текста* — это *ячейка строки*, ее вид должен быть **изменяющимся** строковым видом.

Индексное выражение может не описываться, если *аргумент текста* не является *ячейкой текста* или же он является *ячейкой текста* и его **доступный** вид — это не индексный вид, или же индексное выражение описывается, если **доступный** вид — это **индексный** вид; класс значения, вырабатываемого *индексным выражением*, должен быть **совместимым** с классом значения **индексного** вида.

Аргумент текста в вызове встроенной программы *WRITETEXT* должен быть ячейкой.

Ячейка строки в *аргументе текста* должна быть **ссылочной**.

Динамические условия: Имеет место исключение *TEXTFAIL*, если:

- значение строки, получаемое из *аргумента формата*, не может быть произведено как конечное порождение *строки управления форматом*;
- производится попытка присвоить значение **фактическому индексу**, которое меньше нуля или больше **длины текста**;
- во время интерпретации достигнут конец *строки управления форматом* и *список ввода—вывода* просканирован не полностью или нет элементов в *списке ввода—вывода*, а *строка управления форматом* содержит дополнительные *коды преобразования* или **переменные** *размеры предложения*;
- встречается *предложение ввода—вывода*, а *аргумент текста* не является *ячейкой текста*, или
- *текст формата* встречается в *READTEXT*, а **запись текста** не содержит на позиции **фактического индекса** строку, равную строке, получаемой из *текста формата*.

Может иметь место исключение при вызове встроенной программы *READRECORD* или *WRITERECORD*, если выполняется функция управления вводом—выводом и нарушается любое из определенных динамических условий.

Примеры:

26.18 WRITETEXT (output, "%B%/", 10) *(1.2)*

7.5.4 Строка управления форматом

Синтаксис:

< строка управления форматом > :: = *(1)*
[*< текст формата >*] { *< спецификация формата >* [*< текст формата >*] }* *(1.1)*

< текст формата > :: = *(2)*
{ *< символ, не обозначающий проценты >* | *< процент >* } *(2.1)*

<процент>::= *(3)*
%% *(3.1)*

< спецификация формата > ::= *(4)*
% [*< коэффициент повторения >*] *< элемент формата >* *(4.1)*

< коэффициент повторения > :: = *(5)*
{ *< цифра >* }* *(5.1)*

< элемент формата > :: = *(6)*
< предложение формата > *(6.1)*
| *< предложение в круглых скобках >* *(6.2)*

< предложение формата > :: = *(7)*
< управляющий код > [%.] *(7.1)*

< управляющий код > :: = *(8)*
< предложение преобразования > *(8.1)*
| *< предложение редактирования >* *(8.2)*
| *< предложение ввода—вывода >* *(8.3)*

< предложение в круглых скобках > :: = *(9)*
(< строка управления форматом > %) *(9.1)*

Примечание. — *Спецификация формата* заканчивается первым символом, не являющимся частью *элемента формата*. Внутри *элементов формата* пробелы и спецификаторы формата могут не использоваться. Точка (.) может использоваться для завершения *предложения формата*. Она входит в *предложение формата* и служит только целям разграничения. При представлении символ обозначения процентов (%) внутри *текста формата* должен записываться дважды (%%).

Семантика: *Строка управления форматом* описывает внешнюю форму передаваемых значений и расположение данных в записях. *Строка управления форматом* состоит из экземпляров *текста формата*, обозначающих фиксированные части записей, и из экземпляров *спецификаций формата*, обозначающих внешнее представление CHILL-значений, что позволяет редактировать **запись текста** или управлять фактическими операциями ввода—вывода.

Спецификация формата, содержащая *коэффициент повторения* и *предложение формата*, эквивалентна такому количеству идентичных экземпляров *спецификации формата предложения формата*, какое задано *коэффициентом повторения*. *Коэффициент повторения* может быть равен 0; в этом случае *спецификация формата* не рассматривается. Например, "%3D4" эквивалентно "%D4%D4%D4".

Для *цифр коэффициента повторения* приняты десятичные цифры.

Строка управления форматом в *предложении в круглых скобках* сканируется много раз в соответствии с *коэффициентом повторения*. Если описания нет, то по умолчанию предполагается, что коэффициент повторения = *1*.

Примеры:

26.20 size = %C%/ *(1.1)*

7.5.5 Преобразование

Синтаксис:

< предложение преобразования > :: = *(1)*
*< код преобразования > { < спецификатор преобразования > }** *(1.1)*
[< размер предложения >]

< код преобразования > :: = *(2)*
B | O | H | C *(2.1)*

< спецификатор преобразования > :: = *(3)*
L | E | P < символ > *(3.1)*

< размер предложения > :: = *(4)*
{ < цифра > }+ | V *(4.1)*

Производный синтаксис: *Предложение преобразования*, в котором отсутствует *размер предложения*, является производным синтаксисом для *предложения преобразования*, у которого описан *размер предложения*, равный нулю (0).

Семантика: Преобразование в вызове встроенной программы *READTEXT* преобразует строку, являющуюся внешним представлением, в CHLL-значение. Преобразование в вызове встроенной программы *WRITETEXT* выполняет обратное преобразование. *Код преобразования* вместе со *спецификатором преобразования* описывают тип преобразования и детали запрашиваемой операции, такие как выравнивание, обработка переполнений и заполнение.

Внешнее представление — это строка, длина которой обычно зависит от преобразуемого значения. Эта строка может содержать минимальное количество символов, необходимых для представления CHILL-значения (свободный формат), или иметь заданную длину (фиксированный формат).

В фиксированном формате **размер** вырезки, начиная с позиции **фактического параметра**, считывается из **записи текста** или записывается в нее в соответствии с выравниваем и заполнением, выбираемыми по *спецификаторам формата*, а именно:

- в *READTEXT*: все символы заполнения (слева или справа согласно правилам выравнивания), если имеются, исключаются. Однако, когда считываются символы или *фиксированные* символьные строки, максимальное количество исключаемых символов заполнения N равно **размеру** минус L, где L равно единице или **длине строки** соответственно. Если $N < 0$, никакие символы не исключаются. Остальные символы используются для внешнего представления.
- в *WRITETEXT*: если длина внешнего представления меньше или равна **размеру**, то в вырезке символы выравниваются налево или направо (согласно правилам выравнивания). Неиспользуемые элементы (если имеются) заполняются символом заполнения. В противном случае строка обрезается (слева, если выравнивание идет справа, либо справа в противном случае) или же передаются символы индикатора "переполнения" **размера** (*), если имеется спецификатор E. Усечение применяется к внешнему представлению, включая знак минус, если имеется.

Для свободного формата выполняется следующее:

- в *READTEXT*: символы заполнения, если имеются, пропускаются, за исключением, когда символ или символьная строка считываются, а *спецификатор преобразования* P не описывается. Тогда берется внешнее представление в качестве самой длинной вырезки символов, которая начинается с **фактического индекса** и включает последующие символы, которые лексически могут принадлежать ей, как определено ниже;
- в *WRITETEXT*: строка, полученная в результате преобразования, вводится, начиная с позиции **фактического индекса**.

В *WRITETEXT* строка, являющаяся внешним представлением, переносится в следующую **запись текста**, несмотря на ее **фактическую длину**. После переноса **фактический индекс** автоматически продвигается на следующую доступную позицию символа, а **фактическая длина** устанавливается на максимальное значение между **фактическим индексом** и (старой) **фактической длиной**.

Размер предложения — **постоянная**, если состоит из *цифр*. Предполагается использовать десятичную запись числа. В противном случае это **переменная**.

Если **размер** равен нулю, тогда выбирается свободный формат, в противном случае **размер** — это длина фиксированного формата.

Если **размер** мал для того, чтобы разместить строку, то предпринимаются соответствующие действия в зависимости от *спецификатора преобразования*.

В *READTEXT* используемое внешнее представление описано ниже для вида *аргумента ячейки*.

В *WRITETEXT* используемое внешнее представление описано ниже для вида M класса M-значения и M-производного класса значения из *аргумента значения*.

Коды преобразования

Коды преобразования представлены отдельными буквами. Определены следующие *коды преобразования*:

B: двоичное представление;

O: восьмеричное представление;

H: шестнадцатеричное представление;

C: преобразование, которое указывает на внешнее представление CHILL-значений по умолчанию и зависит от вида преобразуемого значения (см. ниже).

Внешнее представление зависит от *кода преобразования* и вида преобразуемого значения.

Спецификаторы преобразования

Спецификаторы преобразования представлены отдельными буквами. Определены следующие *спецификаторы преобразования*:

L: выравнивание по левому знаку. Если такое выравнивание отсутствует, то выравнивание происходит по правому знаку. В свободном формате спецификатор не имеет значения.

E: переполнение. В *WRITETEXT* выбрана индикация переполнения; если спецификатор отсутствует, тогда выполняется усечение. В *READTEXT* или в свободном формате этот спецификатор не имеет значения.

P: заполнение. Символ, следующий за спецификатором, описывает символ заполнения. Если *P* отсутствует, то по умолчанию заполняющим символом является пробел. В *READTEXT* в случае выбора свободного формата пробелы и HT (горизонтальное табулирование) рассматриваются как один и тот же символ для пропусков и в случае спецификации после спецификатора, и при использовании по умолчанию.

Внешнее представление

Внешнее представление CHILL-значений определено следующим образом:

a. *целые*

Целые значения лексически представлены в виде одной или нескольких цифр по десятичному основанию (по умолчанию) без начальных нулей и с начальным знаком, если отрицательные. Начальный знак плюс и начальные нули опущены в атрибуте *READTEXT*. Используются следующие *коды преобразования*: *B*, *O*, *C* и *H*. *Код преобразования C* служит для десятичного представления. Цифры представления — это цифры, выбираемые по коду преобразования.

b. *булевские*

Булевские значения лексически представлены в виде *строки простого имени* и есть *TRUE* и *FALSE* (в верхнем регистре (например, *TRUE*) или в нижнем регистре (например, *true*) в зависимости от представления, выбранного при реализации для строк **специальных** простых имен). Используется *код преобразования C*.

c. *символы*

Символьные значения лексически представлены как строки длины *1*. Используется *код преобразования C*.

d. *наборы*

Значения перечислимого вида представлены в виде строк простых имен, являющихся (литеральными) константами наборов. Используется *код преобразования C*.

e. *диапазоны*

Значения диапазонов имеют то же представление, что и значения их **корневого** вида. Однако только представления значений, определяемых ограниченным видом, принадлежат к набору внешних представлений, связанных с ограниченным видом.

f. *символьные строки*

Значения символьной строки лексически представлены в виде строк символов длины *L*. В *WRITETEXT L* — это **фактическая длина**. В *READTEXT L* — это **длина строки**, если строка — **фиксированная**, в противном случае это **изменяющаяся** строка, а *L* — **длина строки**, если не будет меньшего количества символов, доступных в (под)**записи текста** при позиции **фактического индекса**, в случае которого *L* — это количество допустимых символов. Используется *код преобразования C*.

g. *битовые строки*

Значения битовой строки лексически представлены в виде строк двоичных цифр. Для определения количества цифр используют те же правила, что и в символьных строках. Используется *код преобразования C*.

Динамические свойства: *Размер предложения* — это **размер**, являющийся значением, образуемым *цифрами*, или значением из *списка ввода—вывода*, если *размер предложения* — **переменная**.

Динамические условия: Имеет место исключение *TEXTFAIL*, если:

- в *READTEXT* **запись текста** не содержит подстроки, начинающейся с **фактического индекса** (после исключения или пропуска символов заполнения; см. ниже), что может интерпретироваться как внешнее представление одного из значений вида текущего *аргумента ячейки* (включая попытку считать непустое внешнее представление из **записи текста**, когда **фактическая длина = фактическому индексу**), или

- в *WRITETEXT* подстрока, являющаяся внешним представлением текущего *аргумента значения*, не может быть перенесена в следующую **запись текста**, начиная с **фактического индекса**, или

- в *READTEXT* встречается *код преобразования, а* текущий элемент в *списке ввода—вывода* не является ячейкой либо вид ячейки обладает **свойством неизменяемости**, или

- встречается **переменный** *размер предложения*, а соответствующий элемент *списка ввода—вывода* в *списке ввода—вывода* не имеет целого класса или же он меньше нуля (0).

Примеры:

26.21 CL6 *(1.1)*

7.5.6 Редактирование

Синтаксис:

< предложение редактирования > :: = *(1)*
< код редактирования > [*< размер предложения >*] *(1.1)*

< код редактирования > :: = *(2)*
X | < | > | *T* *(2.1)*

Производный синтаксис: *Предложение редактирования*, в котором нет *размера предложения*, является производным синтаксисом для *предложения редактирования*, в котором описан *размер предложения*, равный *1*, если *код редактирования* не *T*, в противном случае размер предложения соответственно равен нулю.

Семантика: Определены следующие функции редактирования:

X: пробел — символы пробела в **размере** вводятся или пропускаются;

>: пропуск справа — **фактический индекс** передвигается вправо к позициям **размера**.

<: пропуск слева — **фактический индекс** передвигается влево к позициям **размера**;

T: табулирование — **фактический индекс** передвигается на позицию **размера**.

В *WRITETEXT*, если **фактический индекс** передвигается к позиции, которая больше **фактической длины**, то строка из *N* пробелов, где *N* — разность между **фактическим индексом** и (старой) **фактической длиной**, добавляется к **записи текста**. **Фактическая длина** устанавливается на максимальное значение между **фактическим индексом** и (старой) **фактической длиной**.

Динамические условия: Имеет место исключение *TEXTFAIL*, если:

- **фактический индекс** подвинут на позицию, которая меньше нуля (0) или больше **длины текста**, или
- в *READTEXT* **фактический индекс** подвинут на позицию, которая больше **фактической длины**, или
- в *READTEXT* описан *код редактирования X*, а в **записи текста** на позиции **фактического индекса** отсутствует строка пробелов **размера** или символы HT (горизонтальное табулирование).

Примеры:

26.22 X *(1.1)*

7.5.7 Управление вводом—выводом

Синтаксис:

< предложение ввода—вывода > :: = *(1)*
< код ввода—вывода > *(1.1)*

< код ввода—вывода > :: = *(2)*
/ | — | + | ? | ! = *(2.1)*

Семантика: Функции управления вводом—выводом (за исключением %=) выполняют операцию ввода—вывода. Они осуществляют точное управление переносом **записи текста**. В *READTEXT* все функции имеют один и тот же результат, выражающийся в чтении следующей записи из файла. В *WRITETEXT* переносятся **запись текста** и соответствующее представление информации управления кареткой. Начальная позиция каретки в то время, когда *ячейка текста* соединена так, что первый символ первой **записи текста** печатается в начале первой незанятой строки (независимо от информации по размещению, содержащейся в **записи текста**).

Место каретки описано с помощью следующих абстрактных операций с текущими столбцом, строкой и страницей (*x*, *y*, *z*), причем столбцы нумеруются с нуля, начиная с левого края, а строки — с нуля, начиная с верхнего края.

nl(*w*): каретка передвигается на *w* строк вниз с начала строки (новая позиция: (0, (*y* + *w*) mod *p*,*z* + (*y* + *w*) / *p*, где *p* — количество строк на странице));

np(*w*): каретка передвигается на *w* страниц вниз с начала строки (новая позиция: (0, 0, *z* + *w*)).

Имеются следующие функции управления:

/: следующая запись — запись распечатывается на следующей строке (nl(1), печать записи, nl(0));

+: следующая страница — запись печатается в верхней части следующей страницы (np(1), печать записи, nl(0));

—: текущая строка — запись печатается на текущей строке (печать записи, nl(0));

?: подсказка — запись печатается на следующей строке. Каретка переведена в конец строки (nl(1), печать записи);

!: эмитировать — управление кареткой не происходит (печать записи);

=: конец страницы определяет расположение следующей записи (если имеется) вверху следующей страницы (отменяется расположение, выполненное до печати записи). Не вызывает никаких операций по вводу—выводу.

Перенос при вводе—выводе выполняется следующим образом:

- в *READTEXT* семантика та же, что при выполнении программы *READRECORD* (*A,I,R*), где *A* — подъячейка **доступа** *ячейки текста*, *I* — *индексное выражение* (если имеется), а *R* обозначает **запись текста**. После переноса при вводе—выводе **фактический индекс** устанавливается в ноль (0), а **фактическая длина** — на **длину строки** значения строки, которое было считано;

- в *WRITETEXT* семантика та же, что при выполнении программы *WRITERECORD* (*A,I,R*), где *A* — подъячейка **доступа** *ячейки текста*, *I* — *индексное выражение* (если имеется), а *R* обозначает **запись текста**. Переносится также соответствующая информация по размещению. Если вид **записи** доступа не является **динамическим**, то **запись текста** заполняется в конце пробелами, а ее **фактическая длина** устанавливается на **длину текста** до осуществления переноса. После переноса при вводе—выводе **фактический индекс** и **фактическая длина** устанавливаются в ноль (0).

Примеры:

26.21 / *(1.1)*

7.5.8 Доступ к атрибутам ячейки текста

Синтаксис:

< вызов встроенной программы получения текста > :: = *(1)*
GETTEXTRECORD (< ячейка текста >) *(1.1)*
| *GETTEXTINDEX (< ячейка текста >)* *(1.2)*
| *GETTEXTACCESS (< ячейка текста >)* *(1.3)*
| *EOLN (< ячейка текста >)* *(1.4)*

< вызов встроенной программы хранения текста > :: = *(2)*
SETTEXTRECORD (< ячейка текста > , < ячейка символьной строки >) *(2.1)*
| *SETTEXTINDEX (< ячейка текста > , < целое выражение >)* *(2.2)*
| *SETTEXTACCESS (< ячейка текста > , < ячейка доступа >)* *(2.3)*

Семантика: *GETTEXTRECORD* возвращает **ссылку на запись текста** *ячейки текста*.

GETTEXTINDEX возвращает **фактический индекс** *ячейки текста*.

GETTEXTACCESS возвращает **ссылку на доступ** *ячейки текста*.

EOLN вырабатывает *TRUE*, если в **записи текста** нет больше доступных символов (то есть если **фактический индекс** равен **фактической длине**).

SETTEXTRECORD хранит ссылку на ячейку, выдаваемую *ячейкой символьной строки* в **ссылку на запись текста** *ячейки текста*.

SETTEXTINDEX обладает той же семантикой, что и *предложение редактирования* в *WRITETEXT*, в которой *код редактирования* — это *T*, а *размер предложения* дает то же значение, что и *целое выражение*, применяемое к **записи текста**, обозначенного *ячейкой текста*.

SETTEXTACCESS хранит ссылку на ячейку, выдаваемую *ячейкой доступа* в **ссылку на доступ** *ячейки текста*.

Статические свойства: Класс вызова встроенной программы *GETTEXTRECORD* является M-ссылочным классом, где M — вид **записи текста** *ячейки текста*.

Класс вызова встроенной программы *GETTEXTINDEX* есть *INT*-производный класс.

Класс вызова встроенной программы *GETTEXTACCESS* является M-ссылочным классом, где M — **доступный** вид *ячейки текста*.

Класс вызова встроенной программы *EOLN* является *BOOL*-производным классом.

Вызов встроенных программ *GETTEXTRECORD* или *GETTEXTACCESS* имеет ту же **региональность**, что и *ячейка текста*.

Статические условия: Вид аргумента *ячейки символьной строки* программы *SETTEXTRECORD* должен быть **совместимым по чтению** с видом **записи текста** *ячейки текста*.

Вид аргумента *ячейки доступа* программы *SETTEXTACCESS* должен быть **совместимым по чтению** с **доступным** видом *ячейки текста*.

Аргумент *ячейки* в программах *SETTEXTRECORD* и *SETTEXTACCESS* должен иметь ту же **региональность**, что и *ячейка текста*.

Динамические условия: Имеет место исключение *TEXTFAIL*, если аргумент *целого выражения* программы *SETTEXTINDEX* выдает значение, которое меньше нуля (0) или больше **длины текста** *ячейки текста*.

Примеры:

26.23 GETTEXTINDEX (output) *(1.2)*

8 ОБРАБОТКА ИСКЛЮЧЕНИЙ

8.1 ОБЩИЕ ПОЛОЖЕНИЯ

Исключение является либо исключением, определяемым в языке, когда исключение снабжается именем, определенным в языке, исключением, определяемым пользователем, либо исключением, определяемым реализацией. Исключение, определяемое в языке, вызывается динамическим нарушением динамических условий. Любое исключение может вызываться выполнением оператора причины.

Когда исключение возникает, оно может быть обработано, то есть будет выполняться список операторов действия соответствующей программы обработки исключений.

Обработка исключений определяется таким образом, что при любом операторе статически известно, какое исключение может иметь место (то есть статически известно, каких исключений быть не может), а для каких исключений может быть найдена соответствующая программа обработки или какие исключения могут быть переданы в точку вызова процедуры. Если имеет место исключение и для него не может быть найдена программа обработки, то считается, что в программе есть ошибка.

Когда возникает исключение при операторе действия или операторе описания, то выполнение оператора происходит неопределенным образом, если в соответствующем разделе ничего не говорится по этому поводу.

8.2 ПРОГРАММЫ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Синтаксис:

< программа обработки исключений > :: = *(1)*

ON { *< выбор исключения >* } * [**ELSE** *< список операторов действия >*] **END** *(1.1)*

< выбор исключения > :: = *(2)*

(< список исключений >) : < список операторов действия > *(2.1)*

Семантика: Если программа обработки исключений соответствует исключению E (согласно положениям раздела 8.3), то она вводится. Если E содержится в *списке исключений выбора исключений* в *программе обработки исключений*, то вводится соответствующий *список операторов действия*; в противном случае описывается **ELSE** и вводится соответствующий *список операторов действия*.

При достижении конца выбранного *списка операторов действия* заканчивается работа программы *обработки исключений* с закрепленной за ней логической конструкцией.

Статические условия: Все *имена исключений* во всех вхождениях *списков исключений* должны быть разными.

Динамические условия: Имеет место исключение *SPACEFAIL*, если введен список операторов действия и требования памяти не могут быть выполнены.

Примеры:

10.47 **ON**
(ALLOCATEFAIL): **CAUSE** *overflow;*
END *(1.1)*

8.3 ИДЕНТИФИКАЦИЯ ПРОГРАММ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Когда имеет место исключение E при выполнении операции или в модуле A, операторе обработки данных или регионе D, то исключение может быть обработано соответствующей программой обработки исключений; то есть в программе обработки исключений будет выполняться список операторов действия или же исключение может быть передано в точку вызова процедуры; или же при невозможности и того, и другого считается, что в программе есть ошибка.

Для любой операции или модуля A, оператора обработки данных и региона D можно определить статически возможность нахождения данного исключения E в A или D соответствующей программы обработки исключений или же исключение можно отправить в точку вызова процедуры.

Определение соответствующей программы обработки исключений для A и D по отношению к исключению с именем E таково:

1. Если программа обработки исключений связана с именем исключения E в *списке исключений* или с описанием ELSE и дополняет A и D или включена в A и D, а E находится в области, непосредственно включающей в себя программу обработки исключений, то такая программа обработки исключений соответствует имени исключения E.

2. В противном случае, если A и D непосредственно включены в сложный оператор, модуль или регион, то соответствующая программа обработки исключений (если имеется) по отношению к исключению с именем E пригодна для сложного оператора, модуля или региона.

3. В противном случае, если A или D содержатся в области определения процедуры, тогда:

   - если программа обработки исключений, связанная с именем E в списке исключений или описанием ELSE, добавляется к определению процедуры, тогда эта программа обработки соответствует обрабатываемому исключению;

   - в противном случае, если E содержится в списке исключений определения процедуры, тогда исключение с именем E передается в точку вызова процедуры;

   - в противном случае программы обработки исключений не существует.

4. В противном случае, если A или D содержатся в области определения процедуры, тогда:

   - если программа обработки исключений, связанная с именем E в списке исключений или описанием ELSE, добавляется к определению процесса, тогда эта программа обработки исключений соответствует обрабатываемому исключению;

   - в противном случае программы обработки исключений не существует; однако в этом случае может иметься программа обработки исключений, определяемая реализацией (см. раздел 13.4).

5. В противном случае, если A есть действие из списка операторов действия в программе обработки исключений, тогда соответствующая исключению программа будет соответствующей программой обработки исключений для действия A' или оператора обработки данных или региона D', связанной с именем исключения E, программа обработки которого добавлена или включена в A' и D', но рассматривается так, как будто она не была специфицирована.

Если есть причины для исключения и передача управления к соответствующей программе обработки исключений подразумевает выход из блоков, то при выходе из блока очищается локальная память.

9 КОНТРОЛЬ ВРЕМЕНИ

9.1 ОБЩИЕ ПОЛОЖЕНИЯ

Предполагается, что понятие времени является внешним по отношению к CHILL-программе (системе). В CHILL не описываются определенные временные свойства, но обеспечены механизмы использования программы для взаимодействия с программой наблюдения за временем во внешней среде.

9.2 ПРЕРЫВАЕМЫЕ ПРОЦЕССЫ

Существует понятие **прерываемого** процесса для идентификации определенных моментов времени при выполнении программы, когда может иметь место прерывание времени, то есть, когда система контроля времени вмешивается в нормальное выполнение процесса.

Процесс становится **прерываемым**, когда он достигает корректного заданного момента при выполнении определенных операторов. В языке CHILL дано определение **прерываемого** процесса при выполнении конкретных операторов; в реализации может быть определен **прерываемый** процесс при выполнении дополнительных операторов.

9.3 ОПЕРАТОРЫ ВРЕМЕННЫХ СООТНОШЕНИЙ

Синтаксис:

< оператор временных соотношений > :: = *(1)*
< оператор относительных временных соотношений > *(1.1)*
| *< оператор абсолютных временных соотношений >* *(1.2)*
| *< оператор циклических временных соотношений >* *(1.3)*

Семантика: Оператор временных соотношений описывает процессы контроля времени для процесса, выполняющего оператор временных соотношений. Контроль времени может быть инициирован, а также он может истекать и прекращать существование. Некоторые виды контроля времени могут быть связаны с одним и тем же процессом из-за оператора циклических временных соотношений и вложенности операторов временных соотношений.

Прерывание времени существует, когда процесс является **прерываемым** и по крайней мере один из его видов контроля времени прекращает существовать. Наличие прерывания времени подразумевает прекращение существования начального контроля времени; кроме того, прерывание приводит к передаче управления, связанного с этим контролем времени, в контролируемый процесс. Если контролируемый процесс был задержан, он становится реактивированным.

Контроль времени прекращает свое существование, когда управление передается из оператора временных соотношений, инициировавшего контроль времени.

9.3.1 Оператор относительных временных соотношений

Синтаксис:

< оператор относительных временных соотношений > :: = *(1)*

AFTER *< примитивное значение продолжительности >* **[DELAY] IN**
< список операторов действия > < программа обработки исключений временных соотношений > **END** *(1.1)*

< программа обработки исключений временных соотношений > :: = *(2)*
TIMEOUT *< список операторов действия >* *(2.1)*

Семантика: Вычисляется *примитивное значение продолжительности*, инициируется контроль времени, затем вводится *список операторов действия.*

Если **DELAY** не описана, то контроль времени инициируется до ввода *списка операторов действия*, в противном случае он инициируется тогда, когда процесс выполнения оператора относительных временных соотношений становится **прерываемым** в момент выполнения, описываемый *оператором действия списка операторов действия.*

Если **DELAY** описана, то контроль времени прекращает существование, если он был инициирован, а процесс выполнения оператора относительных временных соотношений перестает быть **прерываемым.**

Контроль времени истекает, если он не прекратил существование в течение определенного после инициации периода времени.

Передача управления, связанного с контролем времени, осуществляется к *списку операторов действия программы обработки исключений временных соотношений*.

Статические условия: Если **DELAY** описана, то *список операторов действия* должен состоять точно из *одного оператора действия*, который сам может быть причиной того, что процесс выполнения оператора относительных временных соотношений становится **прерываемым**.

Динамические условия: Если инициация контроля времени не удается по причинам, зависящим от реализации, то имеет место исключение *TIMERFAIL*.

9.3.2 Оператор абсолютных временных соотношений

Синтаксис:

< *оператор абсолютных временных соотношений* > :: = (1)

AT < *примитивное значение абсолютного времени* > **IN**
< *список операторов действия* > < *программа обработки исключений временных соотношений* > **END** (1.1)

Семантика: Вычисляется *примитивное значение абсолютного времени*, инициируется контроль времени, затем вводится *список операторов действия*.

Контроль времени истекает, если он не прекратил существования в заданный момент времени или после этого момента.

Передача управления, связанного с контролем времени, осуществляется к *списку операторов действия программы обработки исключений временных соотношений*.

Динамические условия: Если инициация контроля времени не удается по причинам, зависящим от реализации, то имеет место исключение *TIMERFAIL*.

9.3.3 Оператор циклических временных соотношений

Синтаксис:

< *оператор циклических временных соотношений* > :: = (1)

CYCLE < *примитивное значение продолжительности* > **IN**
< *список операторов действия* > **END** (1.1)

Семантика: Оператор циклических временных соотношений предназначен для обеспечения ввода списка операторов действия процессом выполнения оператора циклических временных соотношений в определенные интервалы времени без накопления погрешностей (это означает, что время выполнения для *списка операторов действия* в среднем должно быть меньше заданного значения продолжительности). Вычисляется *примитивное значение продолжительности*, инициируется контроль относительного времени, затем вводится *список операторов действия*.

Контроль времени истекает, если он не прекратил существования через определенный после инициации период времени. Независимо от истечения времени инициируется новый контроль времени с тем же значением продолжительности.

Передача управления, связанного с контролем времени, осуществляется в начало *списка операторов действия*.

Заметим, что выполнение оператора циклических временных соотношений может заканчиваться только в результате передачи управления из этого оператора.

Динамические свойства: Процесс, выполняющий оператор циклических временных соотношений, становится **прерываемым** тогда и только тогда, когда управление передается в конец *списка операторов действия*.

Динамические условия: Если инициация контроля времени не удается по причинам, зависящим от реализации, то имеет место исключение *TIMERFAIL*.

9.4 ВСТРОЕННЫЕ ПРОГРАММЫ ВРЕМЕНИ

Синтаксис:

< вызов встроенной программы выдачи значения времени > :: = (1)
< вызов встроенной программы продолжительности > (1.1)
| *< вызов встроенной программы абсолютного времени >* (1.2)

Семантика: Разные реализации отличаются друг от друга различными требованиями и возможностями, относящимися к точности и диапазону значений времени. Описанные ниже встроенные программы учитывают эти различия с точки зрения возможности переноса этих программ из одной реализации в другую.

9.4.1 Встроенные программы продолжительности

Синтаксис:

< вызов встроенной программы продолжительности > :: = (1)
MILLISECS (< целое выражение >) (1.1)
| *SECS < целое выражение >)* (1.2)
| *MINUTES < целое выражение >)* (1.3)
| *HOURS < целое выражение >)* (1.4)
| *DAYS < целое выражение >)* (1.5)

Семантика: Вызов встроенной программы продолжительности приводит к выработке значения продолжительности, определяемого реализацией и, возможно, с изменяющейся точностью (то есть программы *MILLISECS* (*1000*) и *SECS* (*1*) могут вырабатывать различные значения продолжительности); это значение является наилучшим приближением для указанного периода времени в пределах выбранной точности.

Статические свойства: Класс вызова встроенной программы продолжительности является *DURATION*-производным классом.

Динамические условия: Если при реализации нельзя получить значение времени, обозначающее указанный период времени, то имеет место исключение *RANGEFAIL*.

9.4.2 Встроенная программа абсолютного времени

Синтаксис:

< вызов встроенной программы абсолютного времени > :: = (1)
ABSTIME ([[[[[[< выражение года > ,] < выражение месяца > ,]
< выражение дня > ,] < выражение часа > ,]
< выражение минут > ,] < выражение секунд >]) (1.1)

< выражение года > :: = (2)
< целое выражение > (2.1)

< выражение месяца > :: = (3)
< целое выражение > (3.1)

< выражение дня > :: = (4)
< целое выражение > (4.1)

< *выражение часа* > :: = (5)
< *целое выражение* > (5.1)

< *выражение минут* > :: = (6)
< *целое выражение* > (6.1)

< *выражение секунд* > :: = (7)
< *целое выражение* > (7.1)

Семантика: В результате вызова встроенной программы *ABSTIME* вырабатывается значение абсолютного времени, обозначающее момент времени по грегорианскому календарю, указанный в списке параметров. Если опущены параметры высшего порядка, то указанный момент времени является следующим моментом времени, соответствующим имеющимся параметрам низкого порядка (например, *ABSTIME* (*15,12,00,00*) обозначает полдень 15-го числа данного или следующего месяца).

Если не описаны никакие параметры, то значение абсолютного времени обозначает момент его выработки программой.

Статические свойства: Класс вызова встроенной программы абсолютного времени является *TIME*-производным классом.

Динамические условия: Если при реализации нельзя получить значения абсолютного времени для указанного момента времени, то имеет место исключение *RANGEFAIL*.

9.4.3 Вызов встроенной программы временных соотношений

Синтаксис:

< *вызов простой встроенной программы временных соотношений* > :: = (1)
WAIT ( ) (1.1)
| *EXPIRED* ( ) (1.2)
| INTTIME (< *примитивное значение абсолютного времени* > , [[[[< *ячейка года* >
< *ячейка месяца* > ,] < *ячейка дня* > ,]
< *ячейка часа* > ,] < *ячейка минут* > ,]
< *ячейка секунд* >) (1.3)

< *ячейка года* > :: = (2)
< *целое выражение* > (2.1)

< *ячейка месяца* > :: = (3)
< *целое выражение* > (3.1)

< *ячейка дня* > :: = (4)
< *целое выражение* > (4.1)

< *ячейка часа* > :: = (5)
< *целое выражение* > (5.1)

< *ячейка минут* > :: = (6)
< *целое выражение* > (6.1)

< *ячейка секунд* > :: = (7)
< *целое выражение* > (7.1)

Семантика: Программа *WAIT* безусловным образом снабжает процесс выполнения свойством **прерываемости**: выполнение может быть закончено только в результате прерывания времени.

Программа *EXPIRED* снабжает процесс выполнения свойством **прерываемости**, если истек один из видов контроля времени, связанный с процессом выполнения; в противном случае данная программа ни на что не влияет.

Программа *INTTIME* присваивает описанным целым ячейкам целое представление момента времени по грегорианскому календарю описанного *примитивным значением абсолютного времени*.

Статические условия: Все описанные целые ячейки должны быть **ссылочными**, а их виды могут не иметь **свойства неизменяемости**.

Динамические свойства: Программа *WAIT* снабжает процесс выполнения свойством **прерываемости**.

Программа *EXPIRED* снабжает процесс выполнения свойством **прерываемости**, если имеется связанный с ним истекший контроль времени.

10 СТРУКТУРА ПРОГРАММ

10.1 ОБЩИЕ ПОЛОЖЕНИЯ

Условный оператор, оператор выбора, оператор цикла, оператор выбора задержки, блок, модуль, регион, модуль спецификации, регион спецификации, контекст, оператор варианта получения, определение процедуры и *определение процесса* определяют структуру программ; то есть они определяют область действия имен и время жизни ячеек, созданных в них.

- Словом "блок" обозначаются:
  - *список операторов действия* в *операторе цикла*, включая любой *счетчик цикла* и *управление по условию*;
  - *список операторов действия* в *then-части условного оператора*;
  - *список операторов действия* в *альтернативе выбора оператора выбора*;
  - *список операторов действия* в *варианте задержки оператора выбора задержки*;
  - *блок begin-end*;
  - *определение процедуры*, включая *спецификацию результата* и *спецификацию параметров* для всех *формальных параметров списка формальных параметров*;
  - *определение процесса*, исключая *спецификацию параметров* для всех *формальных параметров списка формальных параметров*;
  - *список операторов действия* в *варианте получения из буфера* или в *альтернативе получения сигнала*, включая любые *определяющие вхождения* из *списка определяющих вхождений*, следующего за **IN**;
  - *список операторов действия* после **ELSE** в *условном операторе*, *операторе выбора*, *операторе варианта получения* или *программе обработки исключений*;
  - *выбор исключения* в *программе обработки исключений*;
  - *список операторов действия* в *операторе относительных временных соотношений*, *операторе абсолютных временных соотношений*, *операторе циклических временных соотношений* или в *программе обработки исключений временных соотношений*.
- Словом "модульон" обозначаются:
  - *модуль* или *регион*, исключая *контекстовый список* и *определяющее вхождение*, если имеются;
  - *модуль спецификации* или *регион спецификации*, исключая *контекстовый список*, если имеется;
  - *контекст*.
- Словом "группа " обозначается либо блок, либо модульон.
- Словом "область" или "область группы " обозначается та часть группы, которая не окружена (см. раздел 10.2) внутренней группой.

Группа влияет на область действия каждого имени, созданного в своей области. Имена создают следующие *определяющие вхождения*:

- *Определяющее вхождение* в *списке определяющих вхождений описания*, *определения вида* или *определения синонима* или в *определении сигнала* при создании *имени* в области, где находятся *описание*, *определение вида*, *определение синонима* или *определение сигнала* соответственно.
- *Определяющее вхождение* в *перечислимом виде* создает имя в области, непосредственно объемлющей *перечислимый вид*.
- *Определяющее вхождение* из *списка определяющих вхождений* в *списке формальных параметров* создает имя в области связанного с ними *определения процедуры* или *определения процесса*.
- *Определяющее вхождение* перед двоеточием, за которым следуют *действие*, *регион*, *определение процедуры* или *определение процесса*, создает имя в области, где находятся *действие*, *регион*, *определение процедуры* и *определение процесса* соответственно.
- (Виртуальное) *определяющее вхождение*, вводимое *присоединением* или *счетчиком цикла*, создает имя в области блока связанного с ним *оператора цикла*.
- *Определяющее вхождение* из *списка определяющих вхождений варианта получения* из *буфера* или *альтернативы получения сигнала* создает имя в области блока связанных с ним *варианта получения* из *буфера* или *альтернативы получения сигнала* соответственно.
- (Виртуальное) *определяющее вхождение* для имени, предопределенного в языке или определяемого реализацией, создает имя в области воображаемого внешнего процесса (см. раздел 10.8).

Те места, где используется имя, называются вхождениями с использованием имени. Правила связывания имен связывают *определяющее вхождение* с каждым вхождением с использованием имени (см. раздел 12.2.2).

Имя имеет определенную область действия, то есть ту часть программы, где могут быть видны его определение или описания и, как следствие, где его можно использовать без ограничений. Считается, что имя **видимо** в этой части. Ячейки и процедуры имеют определенное время жизни, то есть это та часть программы, где они существуют. Блоки определяют видимость имен и время жизни ячеек, созданных в них. Модульоны определяют только видимость; время жизни ячеек, созданных в области модульона, будет тем же, как если бы они были созданы в области первого окружающего блока. Модульоны позволяют ограничивать видимость имен. Например, имя, созданное в области модуля, не будет автоматически **видимым** во внешних или внутренних модулях, хотя время жизни позволяло бы это.

10.2 ОБЛАСТИ И ВЛОЖЕННОСТЬ

Синтаксис:

< тело блока > :: = (1)
< список операторов данных > < список операторов действия > (1.1)

< тело процедуры > :: = (2)
< список операторов данных > < список операторов действия > (2.1)

< тело процесса > :: = (3)
< список операторов данных > < список операторов действия > (3.1)

< тело модуля > :: = (4)
{ < оператор данных > | < оператор видимости > | < регион > |
< регион спецификации > } < список операторов действия >* (4.1)

< тело региона > :: = (5)
*{ < оператор данных > | < оператор видимости >}** (5.1)

< тело модуля спецификации > :: = (6)
{< оператор квазиданных > | < оператор видимости > |
*< модуль спецификации > | < регион спецификации > }** (6.1)

< тело региона спецификации > :: = (7)
*{< оператор квазиданных > | < оператор видимости > }** (7.1)

< тело модуля контекста > :: = (8)
{ < оператор квазиданных >} | < оператор видимости > |
*< модуль спецификации > | < регион спецификации > }** (8.1)

< список операторов действия > :: = (9)
*{ < оператор действия > }** (9.1)

< список операторов данных > :: = (10)
{ < оператор данных > } (10.1)

< оператор данных > :: = (11)
< оператор описания > (11.1)
| < оператор определения > (11.2)

< оператор описания > :: = (12)
< оператор определения синонимического вида > (12.1)
| < оператор определения нового вида > (12.2)
| < оператор определения синонима > (12.3)
| < оператор определения процедуры > (12.4)
| < оператор определения процесса > (12.5)
| < оператор определения сигнала > (12.6)
| < пусто >; (12.7)

Семантика: Когда вводится область блока, выполняются все инициализации созданных ячеек границами времени жизни. Кроме того, выполняются инициализации границами области в области блока, а также, возможно, динамические расчеты в описаниях опознавателей-ячеек, инициализации границами областей в регионах и действия в том порядке, как они описаны в тексте.

Когда вводится область модульона, то выполняются инициализации границами области, а также, возможно, динамические расчеты в описаниях опознавателей-ячеек, инициализации границами областей в регионах и действия (если модульон является модулем), находящиеся в области модульона, в том порядке, как они описаны в тексте.

Оператор данных, действие, модуль или регион заканчиваются либо своим завершением, либо окончанием работы программы обработки исключений.

Когда оканчивается инициализация границами области, описание опознавателя-ячейки, действие, модуль, регион, процедура или процесс, то в зависимости от оператора или вида окончания выполнение операций возобновляется в следующем порядке:

- если оператор заканчивается завершением выполнения программы обработки исключений, то выполнение операций возобновляется со следующим оператором;
- в противном случае, если это действие, подразумевающее передачу управления, то выполнение операций возобновляется с оператором, определяемым для данного действия (см. разделы 6.5, 6.6, 6.8, 6.9);
- в противном случае, если это процедура, то управление возвращается в точку вызова (см. раздел 10.4);
- в противном случае, если это процесс, то выполнение процесса (или программы, если это внешний процесс) заканчивается (см. раздел 11.1) и возобновляется (возможно) с другим процессом;
- в противном случае управление будет передано к последующему оператору.

Статические свойства: Любая область непосредственно заключается в несколько групп либо ни в одну группу, следующим образом:

- Если область — это область *оператора цикла, блок bigin-end, определение процедуры, определение процесса*, тогда она непосредственно заключена в группу, в области которой находятся *оператор цикла, блок begin-end, определение процедуры* или *определение процесса* соответственно, и только в эту группу.
- Если область — это *список операторов действия оператора временных соотношений* или *программы обработки исключений временных соотношений*, либо один из *списков операторов действия условного оператора, оператора выбора, оператора выбора задержки*, тогда она непосредственно заключается в группу, в области которой находятся *оператор временных соотношений, программа обработки исключений временных соотношений, условный оператор, оператор выбора* или *оператор* выбора задержки, и только в эту группу.
- Если область — это *список операторов действия, вариант получения* из *буфера, альтернатива получения сигнала* или *список операторов действия*, следующий за **ELSE** в *операторе варианта получения* из *буфера* или в *операторе получения сигнала*, тогда она непосредственно заключается в группу, в области которой находятся *оператор варианта получения* из *буфера* или *оператор варианта получения сигнала*, и только в эту группу.
- Если область — это *список операторов действия* в *выборе исключения* или *список операторов действия*, следующий за **ELSE** в *программе обработки исключений*, которая не добавлена к группе, тогда она непосредственно заключается в группу, в области которой находится оператор, к которому добавляется *программа обработки исключений*, и только в эту группу.
- Если область — это *выбор исключения* или *список операторов действия* после **ELSE** в *программе обработки исключений*, добавляемой к группе, тогда она непосредственно заключена в группу, к которой добавлена *программа обработки исключений*, и только в эту группу.
- Если область — это *модуль, регион, модуль спецификации* или *регион спецификации*, тогда она непосредственно заключается в группу, в области которой она находится, а также непосредственно заключается в *контекст* прямо перед *модулем, регионом, модулем спецификации* или *регионом спецификации*, если имеется. Это единственный случай, где область имеет несколько непосредственно объемлющих групп.
- Если область — это *контекст*, тогда она непосредственно заключается в *контекст* прямо перед ним. Если такого *контекста* нет, то у области нет непосредственно объемлющей группы.

Область имеет непосредственно объемлющие области, являющиеся областями непосредственно объемлющих групп. Оператор имеет единственную непосредственно объемлющую группу, а именно группу, в которой находится оператор. Считается, что область непосредственно объемлет группу (область) тогда и только тогда, когда область является непосредственно объемлющей областью группы (области).

Считается, что оператор (область) окружен группой тогда и только тогда, когда либо группа является непосредственно объемлющей группой оператора (области) либо непосредственно объемлющая область окружена группой.

Считается, что область введена, если имеется:

- Область модуля: модуль выполняется как действие (например, не считается, что модуль введен, когда оператор перехода передает управление по имени **метки**, определенной внутри модуля).
- Область блока begin-end: блок begin-end выполняется как действие.
- Область региона: встречается регион (например, не считается, что регион введен, когда вызывается одна из его **критических** процедур).
- Область процедуры: процедура вводится посредством вызова процедуры.
- Область процесса: процесс активируется посредством вычисления выражения запуска.
- Область цикла: оператор цикла выполняется как действие после вычисления выражений или ячеек в управляющей части.
- Область варианта получения из буфера, альтернативы получения сигнала: выполняется альтернатива по получению значения из буфера или сигнала.
- Область выбора исключения: выбор исключения выполняется в случае исключения.
- Прочие области блока: введен список операторов действия.

Считается, что список операторов действия введен тогда и только тогда, когда его первый оператор, если имеется, принимает управление от внешнего списка операторов действия.

Область является **квазиобластью**, если это *область модуля спецификации*, *региона спецификации* или *контекста*, в противном случае это **действительная** область.

Определяющее вхождение является **квазиопределяющим** *вхождением*, если указанное выражение:

- окружено *контекстом*, а не модулем или регионом, или
- окружено *простым модулем спецификации* или *простым регионом спецификации*, или
- не окружено ни одной из упомянутых выше групп, а окружено *спецификацией модуля* или *спецификацией региона* и содержится в *квазиописании*, *операторе определения квазипроцедуры* или *операторе определения квазипроцесса* и не является *определяющим вхождением* имени **элемента перечисления**,

в противном случае это **действительное** *определяющее вхождение*.

10.3 БЛОКИ BEGIN-END

Синтаксис:

< *блок begin-end* > :: = *(1)*
BEGIN < *тело блока begin-end* > **END** *(1.1)*

Семантика: Блок begin-end является действием, содержащим, возможно, локальные описания и определения. Он задает как видимость локально созданных имен, так и определяет время жизни локально созданных ячеек (см. разделы 10.9 и 12.2).

Динамические условия: Если не могут быть выполнены условия памяти, то имеет место исключение *SPACEFAIL*.

Примеры: см. 15.73—15.90.

10.4 ОПРЕДЕЛЕНИЯ ПРОЦЕДУР

Синтаксис:

< оператор определения процедуры > :: = *(1)*
< определяющее вхождение > : < определение процедуры >
[*< программа обработки исключений >* **]** **[**< *строка простого имени* > **]**; *(1.1)*

< определение процедуры > :: = *(2)*
PROC ([*< список формальных параметров >* **])** **[**< *спецификация результата* > **]**
[EXCEPTIONS *(< список исключений >)* **]** *< список атрибутов процедуры >*
< тело процедуры > **END** *(2.1)*

< список формальных параметров > :: = *(3)*
< формальный параметр > { ,*< формальный параметр >* }* *(3.1)*

< формальный параметр > :: = *(4)*
< список определяющих вхождений > < спецификация параметра > *(4.1)*

< список атрибутов процедуры > :: = *(5)*
[*< универсальность >* **] [RECURSIVE]** *(5.1)*

< универсальность > :: = *(6)*
GENERALL *(6.1)*
| **SIMPLE** *(6.2)*
| **INLINE** *(6.3)*

Производный синтаксис: *Формальный параметр*, у которого в *списке определяющих вхождений* содержится несколько *определяющих вхождений*, получается из нескольких вхождений *формального параметра*, отделенных запятыми, по одному на каждое *определяющее вхождение* и с одной и той же *спецификацией параметра* для каждого вхождения. Например, *i, j, INT* **LOC** получено из *i INT* **LOC**, *j INT* **LOC**.

Семантика: Оператор определения процедуры устанавливает (возможно) параметризованную последовательность действий (операторов), которые могут быть вызваны из различных мест программы. Процедура заканчивается, и управление возвращается в точку вызова либо путем выполнения действия возврата, либо в результате достижения конца *тела процедуры*, либо в результате окончания работы программы обработки исключений, добавленной к определению процедуры (отказы). Могут быть описаны следующие процедуры различной степени сложности:

a. **простые** процедуры (**SIMPLE**) — это процедуры, с которыми нельзя оперировать динамически. С ними нельзя обращаться как со значениями, то есть они не могут храниться в ячейке процедуры и их нельзя передать в качестве параметров при вызове процедуры или возвратить как результат вызова процедуры;

b. **общие** процедуры (**GENERAL**) не имеют ограничений **простых** процедур и с ними можно обращаться как со значениями процедуры;

c. **внутренние** процедуры (**INLINE**) имеют те же ограничения, что и **простые** процедуры. Они не могут быть **рекурсивными**. Они имеют ту же семантику, что и обычные процедуры, но компилятор будет вводить генерируемый объектный код в точке вызова в первую очередь по отношению к вводу исходного кода с целью фактического вызова процедуры.

Только **простые** и **общие** процедуры можно описать как (взаимно)**рекурсивные**. Когда атрибуты процедуры не описаны, то по умолчанию принимаются атрибуты реализации.

Процедура может возвращать значение или ячейку (указанную атрибутом **LOC** в спецификации результата).

Определяющее вхождение перед определением процедуры устанавливает имя процедуры.

передача параметров

Имеются два основных механизма передачи параметров: "передача значением" (**IN**, **OUT** и **INOUT**) и "передача ячейкой" (**LOC**).

передача значением

При передаче параметра значением значение передается в процедуру как параметр и хранится в локальной ячейке определенного параметрического вида. Результат таков, как если бы в начале вызова процедуры для определяющих *вхождений формального параметра* встретилось бы описание ячейки:

DCL < *определяющее вхождение* > < *вид* > :: = < *фактический параметр* >.

Однако процедура вводится после вычисления фактического параметра. Дополнительно для явного указания на передачу значением может быть описано ключевое слово **IN**.

Если описан атрибут **INOUT**, то значение фактического параметра можно получить из ячейки и даже перед возвратом текущее значение формального параметра хранится в фактической ячейке.

Действие атрибута **OUT** такое же, что и атрибута **INOUT**, за тем исключением, что начальное значение ячейки фактического параметра не переписывается в ячейку формального параметра при вводе процедуры; следовательно, формальный параметр имеет **неопределенное** начальное значение. Если процедура является причиной исключения в точке вызова, то не должна выполняться операция восстановления хранимой информации.

передача ячейкой

При передаче параметра ячейкой ячейка (возможно, динамического вида) передается в качестве параметра в тело процедуры. Таким образом могут передаваться только **ссылочные** ячейки. Результат таков, как если бы в точке ввода процедуры для *определяющих вхождений формального параметра* встретился бы оператор описания опознавателя-ячейки:

DCL < *определяющее вхождение* > < *вид* >
LOC [DYNAMIC] : = < *фактический параметр* >;

Однако процедура вводится после вычисления фактического параметра.

Если описанное *значение* не является *ячейкой*, то ячейка, содержащая описанное значение, будет создана неявно и передана в точке вызова. Время жизни созданной ячейки является вызовом процедуры. Вид созданной ячейки — динамический, если значение имеет динамический класс.

передача результата

И значение, и ячейка могут быть возвращены из процедуры. В первом случае *значение* описывается в любом *операторе результата*, во втором случае — *ячейка* (см. раздел 6.8). Если в *спецификации результата* не дан атрибут **NONREF**, то *ячейка* должна быть **ссылочной**. Возвращаемое значение или ячейка определяются наиболее ранним выполненным оператором результата до возврата. Если процедура со спецификацией результата осуществляет возврат при невыполненном операторе результата, то процедура возвращает **неопределенное** значение или **неопределенную** ячейку. В этом случае вызов процедуры не может использоваться ни в качестве вызова процедуры возврата ячейки (см. раздел 4.2.11), ни в качестве вызова процедуры возврата значения (см. раздел 5.2.12), а только как вызов оператора (см. раздел 6.7).

Статические свойства: *Определяющее вхождение* в *операторе определения процедуры* определяет имя **процедуры**.

Имя **процедуры** имеет приписанное ему *определение процедуры*, то есть *определение процедуры* в операторе, в котором установлено имя **процедуры**.

Имя **процедуры** имеет следующие приписанные свойства, как они определены в его *определении процедуры*, и:

- содержит список **спецификаций параметров**, определяемых вхождениями *спецификаций параметров* в *списке формальных параметров*, при этом каждый параметр содержит вид и, возможно, атрибут параметра;
- содержит, возможно, **спецификацию результата** состоящую из вида и дополнительного атрибута результата;
- содержит, возможно, пустой список имен исключений, являющихся именами из *списка исключений*;
- имеет **универсальность**, то есть, если *универсальность* описана: **общее**, **простое** или **внутреннее** в зависимости от того, описаны ли **GENERAL**, **SIMPLE** или **INLINE**; в противном случае по умолчанию в реализациях описаны **общее** и **простое**. Если имя **процедуры** определено внутри региона, то его **универсальность** — **простое**;
- обладает **рекурсивностью** и является **рекурсивным**, если описан атрибут **RECURSIVE**; в противном случае по умолчанию в реализации описывается либо **рекурсивное**, либо **нерекурсивное**. Однако если **универсальность** является **внутренней** или если имя **процедуры** **критическое** (см. раздел 11.2.1), то **рекурсивность** определяется как **нерекурсивно**.

Имя **процедуры**, являющееся **общим**, есть имя **общей процедуры**. Имя **общей процедуры** имеет следующий закрепленный за ним процедурный вид:

PROC *(* [< *список параметров* >] *)* [< *спецификация результата* >]
EXCEPTIONS *(* < *список исключений* > *)*] [**RECURSIVE**],

где < *спецификация результата* >, *если имеется, и* < *список исключений* > *являются теми же, что и в их определении процедуры, а* < *список параметров* > — *это последовательность вхождений* < *спецификации параметра* > *в списке формальных параметров*, разделенных запятыми.

Имя, определяемое в *списке определяющих вхождений формального параметра*, является именем **ячейки** тогда и только тогда, когда *спецификация параметра* в *формальном параметре* не содержит атрибута **LOC**. Если же она содержит атрибут **LOC**, то это имя **опознавателя-ячейки**. Любое такое имя ячейки или имя **опознавателя-ячейки** является **ссылочным**.

Статические условия: Если имя **процедуры** является **внутрирегиональным** (см. раздел 11.2.2), то в определении процедуры не должно быть спецификации атрибута **GENERAL**.

Если имя **процедуры** является **критическим** (см. раздел 11.2.1), то ее определение не должно содержать спецификации атрибута **GENERAL** или **RECURSIVE**.

В определении процедуры не могут специфицироваться ни **INLINE**, ни **RECURSIVE**.

При описании *строка простого имени* должна быть равна строке имени *определяющего вхождения* перед *определением процедуры*.

Если **LOC** описан только в *спецификации параметра* или *спецификации результата*, то вид в нем может обладать **свойством беззначимости**.

Все имена исключений из *списка исключений* должны быть различными.

Примеры:

1.4 *add:*

PROC *(i,j INT)* **RETURNS** *(INT)* **EXCEPTIONS** *(OVERFLOW);*
RESULT *i + j;*
END *add;* *(1.1)*

10.5 ОПРЕДЕЛЕНИЯ ПРОЦЕССОВ

Синтаксис:

< *оператор определения процесса* > :: = *(1)*
< *определяющее вхождение* > : < *определение процесса* >
[< *программа обработки исключений* >] [< *строка простого имени* >]; *(1.1)*

< *определение процесса* > :: = *(2)*

PROCESS *(* [< *список формальных параметров* >] *)* < *тело процесса* > **END** *(2.1)*

Семантика: Оператор определения процесса устанавливает возможно параметризованную последовательность действий, которые могут быть запущены для параллельного выполнения из различных мест программы (см. главу 11).

Статические свойства: *Определяющее вхождение* в *операторе определения процесса* устанавливает имя **процесса**.

Имя **процесса** имеет следующее присущее ему свойство, как это определено в его *определении процесса*, и содержит:

- список **спецификаций параметров**, определяемых вхождениями *спецификаций параметров* в *списке формальных параметров*, причем каждый параметр содержит вид и, возможно, атрибут параметра.

Статические свойства: При спецификации *строка простого имени* должна быть равна строке имени *определяющего вхождения* перед *определением процесса*.

Оператор определения процесса не должен быть окружен регионом или блоком, отличающимися от определения воображаемого внешнего процесса (см. раздел 10.8).

Атрибуты параметров из *списка формальных параметров* не должны быть **INOUT** или **OUT**.

Если атрибут **LOC** описан только в *спецификации параметра* в *формальном параметре* из *списка формальных параметров*, то вид в нем может обладать **свойством беззначимости**.

Примеры:

14.13

```
PROCESS ( );
    wait
    PROC (x INT)
        /*некоторое действие ожидания*/
    END wait;
    DO FOR EVER;
        wait(10/*секунды*/);
        CONTINUE operator_is_ready;
    OD;
END                                                    (2.1)
```

10.6 МОДУЛИ

Синтаксис:

```
< модуль > :: =                                                          (1)
        [ < контекстовый список > ] [ < определяющее вхождение >: ]
        MODULE [ BODY ] < тело модуля > END
        [ < программа обработки исключений > ] [ < строка простого имени > ];   (1.1)
      | < удаленный модульон >                                           (1.2)
```

Семантика: Модуль — это оператор действия содержащий, возможно, локальные описания и определения. Модуль — это средство ограничения видимости строк имен; он не влияет на время жизни локально описанных ячеек.

Подробные правила видимости модулей приведены в разделе 12.2.

Статические свойства: *Определяющее вхождение* в *модуле* определяет имя **модуля**, а также имя **метки**. Имя имеет приписанный ему *модуль* (видимый как модульон, то есть исключая *контекстовый список* и *определяющее вхождение*, если имеются).

Модуль организуется по модульному принципу тогда и только тогда, когда описан *контекстовый список*.

Модуль представляет собой **тело модуля** тогда и только тогда, когда описан атрибут **BODY**.

Статические условия: При спецификации *строка простого имени* должна быть равной строке имени *определяющего вхождения*.

Удаленный модульон в *модуле* должен относиться к *модулю*.

Примеры:

7.48

```
MODULE
    SEIZE convert;
    DCL n INT INIT :=1979;
    DCL rn CHARS (20) INIT :=(20)' ';
    GRANT n,rn;
    convert( );
    ASSERT rn="MDCCCCLXXVIIII"//(6)' ';
END                                                    (1.1)
```

10.7 РЕГИОНЫ

Синтаксис:

< *регион* > :: = (1)
[< *контекстовый список* >] [< *определяющее вхождение* >:]
REGION [**BODY**] < *тело региона* > **END**
[< *программа обработки исключений* >]
[< *строка простого имени* >]; (1.1)
| < *удаленный модульон* > (1.2)

Семантика: Регион — это средство обеспечения взаимно исключающего доступа к его локально описанным объектам данных для параллельного выполнения процессов (см. главу 11). Он определяет видимость локально созданных имен таким же образом, как и модуль.

Статические свойства: *Определяющее вхождение* в регионе устанавливает имя **региона**. Оно имеет приписанный ему регион (видимый как модульон, то есть исключая *контекстовый список* и *определяющее вхождение*, если имеются).

Регион организуется по модульному принципу тогда и только тогда, когда описан *контекстовый список*.

Регион представляет собой **тело региона** тогда и только тогда, когда описан атрибут **BODY**.

Статические условия: При спецификации *строка простого имени* должна быть равной строке имени *определяющего вхождения*.

Регион не должен быть окружен блоком, кроме определения воображаемого внешнего процесса.

Удаленный модульон в *регионе* должен относиться к региону.

Примеры: см. 13.1—13.28.

10.8 ПРОГРАММА

Синтаксис:

< *программа* > :: = (1)
{ < *модуль* > | < *модуль спецификации* > | < *регион* >
| < *регион спецификации* > }$^{+}$ (1.1)

Семантика: Программы состоят из списка модулей или регионов, окруженных определением воображаемого внешнего процесса.

Определения имен, предопределенных в языке CHILL (см. добавление C.2), обусловленные реализацией встроенные программы и целые виды рассматриваются с точки зрения времени жизни, обозначенного в области определения воображаемого внешнего процесса. Вопросы видимости имен описаны в разделе 12.2.

10.9 РАСПРЕДЕЛЕНИЕ ПАМЯТИ И ВРЕМЯ ЖИЗНИ

Время, в течение которого ячейка или процедура существуют в программе, называется их временем жизни.

Ячейка создается описанием или выполнением вызова встроенных программ *GETSTACK* или *ALLOCATE*.

Время жизни ячейки, описанной в области блока, является временем, в течение которого управление находится в этом блоке или в процедуре, вызов которых исходит из этого блока в случае его описания с атрибутом **STATIC**. Время жизни ячейки, описанной в области модульона, является тем же, как если бы она была описана в области ближайшего окружающего блока модульона. Время жизни ячейки, описанной с атрибутом **STATIC**, является тем же, как если бы она была описана в области определения воображаемого внешнего процесса. Это подразумевает, что для описания ячейки с атрибутом **STATIC** распределение памяти происходит только один раз, а именно, при запуске воображаемого внешнего процесса. Если такое описание имеется внутри определения процедуры или определения процесса, то для всех вызовов и активаций будет существовать только одна ячейка.

Время жизни ячейки, созданной в результате выполнения вызова встроенной программы *GETSTACK*, заканчивается при завершении непосредственно объемлющего блока.

Время жизни ячейки, созданной в результате выполнения вызова встроенной программы *ALLOCATE*, это время от момента вызова программы *ALLOCATE* до момента, когда доступ к ячейке не может больше осуществляться со стороны любой CHILL-программы. Этот случай имеет место всегда, когда встроенная программа *TERMINATE* используется по отношению к **размещенному** ссылочному значению для обращения к ячейке.

Время жизни доступа, созданное в описании опознавателя-ячейки, является непосредственно объемлющим блоком описания опознавателя-ячейки.

Время жизни процедуры — это непосредственно объемлющий блок определения процедуры.

Статические свойства: Считается, что *ячейка* является **статической** тогда и только тогда, когда это *ячейка статического вида* одной из следующих разновидностей, как, например:

- *Имя ячейки*, описанное с атрибутом **STATIC**, или определение которого не окружено блоком, отличающимся от определения воображаемого внешнего процесса.
- *Элемент строки* или *подстрока*, где *ячейка строки* является **статической** и либо *левый* и *правый элементы*, либо *начальный элемент* и *размер вырезки* (размер подстроки) являются **постоянными**.
- *Элемент массива*, где *ячейка массива* является **статической**, а *выраженние* — **постоянным**.
- *Подмассив*, где *ячейка массива* является **статической** и либо *нижний* и *верхний элементы*, либо *первый элемент* и *размер вырезки* (размер подмассива) являются **постоянными**.
- *Поле структуры*, где *ячейка структуры* является **статической**.
- *Преобразование ячейки*, *ячейка* в котором является **статической**.

10.10 КОНСТРУКЦИИ ДЛЯ МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ

Модули и регионы являются элементарными единицами (частями), на которые может быть разделена CHILL-программа, разработанная по модульному принципу. Текст таких частей (модулей) указывается удаленными конструкциями (см. раздел 10.10.1). В языке CHILL определены синтаксис и семантика завершенных программ, в которых все вхождения удаленных частей виртуально заменены соответствующим текстом.

10.10.1 Удаленные модули

Синтаксис:

< удаленный модульон > :: = *(1)*

[*< строка простого имени >* :] **REMOTE** *< указатель модуля >*; *(1.1)*

< удаленная спецификация > :: = *(2)*

[*< строка простого имени >* :] **SPEC REMOTE** *< указатель модуля >*; *(2.1)*

< *удаленный контекст* > :: = (3)
CONTEXT REMOTE < *указатель модуля* >
[< *тело модуля контекста* >] **FOR** (3.1)

< *контекстовый модуль* > :: = (4)
CONTEXT MODULE REMOTE < *указатель модуля* > ; (4.1)

< *указатель модуля* > :: = (5)
< *символьная строковая константа* > (5.1)
| < *имя ссылки на текст* > (5.2)
| < *пусто* > (5.3)

Производный синтаксис: Обозначение

CONTEXT MODULE REMOTE < *указатель модуля* >

является производным синтаксисом для

CONTEXT REMOTE < *указатель модуля* > **FOR**
MODULE SEIZE ALL; END;

Примечание.— Эта конструкция избыточна, но может быть использована для проверки логической непротиворечивости.

Семантика: *Удаленные модульоны*, *удаленные спецификации*, *удаленные контексты* и *контекстовые модули* являются средством представления исходного текста программы в виде набора (взаимосвязанных) файлов.

Указатель модуля касается описания части (модуля) исходного текста на языке CHILL в зависимости от реализации при следующих условиях:

- Если *указатель модуля* пуст, то исходный текст берется из места, определяемого структурой программы.
- Если *указатель модуля* содержит *символьную строковую константу*, то она используется для поиска текста.
- Если *указатель модуля* содержит *имя ссылки на текст*, то это имя интерпретируется в зависимости от реализации для поиска исходного текста.

Программа с 1) *удаленными модульонами*, 2) *удаленными спецификациями* эквивалентна программе, построенной путем замены 1) и 2) модулем текста на языке CHILL, к которому относится *указатель модуля*.

Программа с *удаленными контекстами* эквивалентна программе, разработанной путем замены каждого *удаленного контекста* на модуль текста на языке CHILL, к которому относится *указатель модуля*, в котором *тело модуля контекста* вводится виртуально сразу после последнего вхождения *тела модуля контекста* в *контекстовом списке*, обращение к которому осуществляется по *указателю модуля*.

Если указанный модуль недоступен в виде текста на языке CHILL, тогда его *указатель модуля* рассматривается при обращении к эквивалентному модулю текста на языке CHILL, введенному виртуально.

Хотя семантика удаленного модуля определяется в терминах замены, в языке CHILL никакие замены текста не допускаются.

Статические условия: *Указатель модуля* в 1) *удаленном модульоне*, 2) *удаленной спецификации*, 3) *удаленном контексте*, 4) *контекстовом модуле* должен относиться к описанию модуля исходного текста, который является конечным результатом 1) *модуля или региона*, не являющихся *удаленным модульоном*, 2) *модуля спецификации* или *региона спецификации*, не являющихся *удаленной спецификацией*, 3), 4) *контекстового списка*, не являющегося *удаленным контекстом*.

Когда исходный текст, к которому обращаются по *указателю модуля* в удаленном модульоне, начинается с *определяющего вхождения*, то *удаленный модульон* должен начинаться со *строки простого имени*, являющейся строкой имени этого *определяющего вхождения*.

Когда исходный текст, к которому обращаются по *указателю модуля* в *удаленной спецификации*, начинается со *строки простого имени*, то *удаленная спецификация* должна начинаться с той же *строки простого имени*.

Примеры:

25.9 *stack:* **REMOTE** *"example 27 or 28";* *(1.1)*
25.9 *"example 27 or 28"* *(5.1)*

10.10.2 Модули спецификаций, регионы спецификаций и контексты

Синтаксис:

< *модуль спецификации* > :: = *(1)*
< *модуль простой спецификации* > *(1.1)*
| < *спецификация модуля* > *(1.2)*
| < *удаленная спецификация* > *(1.3)*

< *модуль простой спецификации* > :: = *(2)*
[< *контекстовый список* >] [< *строка простого имени* >:] **SPEC MODULE**
< *тело модуля спецификации* > **END** [< *строка простого имени* >]; *(2.1)*

< *спецификация модуля* > :: = *(3)*
[< *контекстовый список* >] < *строка простого имени* > : **MODULE SPEC**
< *тело модуля спецификации* > **END** [< *строка простого имени* >] ; *(3.1)*

< *регион спецификации* > :: = *(4)*
< *регион простой спецификации* > *(4.1)*
| < *спецификация региона* > *(4.2)*
| < *удаленная спецификация* > *(4.3)*

< *регион простой спецификации* > :: = *(5)*
[< *контекстовый список* >] [< *строка простого имени* >:] **SPEC REGION**
< *тело региона спецификации* > **END** < *строка простого имени* >]; *(5.1)*

< *спецификация региона* > :: = *(6)*
[< *контекстовый список* >] < *строка простого имени* >: **REGION SPEC**
< *тело региона спецификации* > **END** < *строка простого имени* >]; *(6.1)*

< *контекстовый список* > :: = *(7)*
< *контекст* > {< *контекст* >}* *(7.1)*
| < *удаленный контекст* > *(7.2)*

< *контекст* > :: = *(8)*
CONTEXT < *тело модуля контекста* > **FOR** *(8.1)*

Семантика: *Модули простой спецификации*, *регионы простой спецификации* и *контексты* используются для описания статических свойств имен. Они избыточны, но их можно использовать при модульном программировании.

Строки простых имен в *модулях спецификаций* и *регионах спецификаций* не являются именами, они не **связаны** и не подчиняются правилам видимости.

1) *Модули спецификаций*, 2) *регионы спецификаций* в **действительной** области определяют свойства одного или нескольких 1) *модулей*, 2) *регионов*, скомпилированных по модульному принципу и объемлемых данной областью. Тексты таких 1) *модулей*, 2) *регионов* определяются вхождениями *удаленных модульонов*. *Контекстовый список* указывает на окружающие области (заметим, что сконструированный по модульному принципу модульон всегда впереди себя имеет *контекстовый список*).

Для каждой *строки имени* OP ! NS, **видимого** в области 1) *спецификации модуля*, 2) *спецификации региона* и **связанного** с **квазиопределяющим** *вхождением* и которое разрешено в **действительной** области в виде NP ! NS, считается, что (виртуальный) оператор разрешения с той же *строкой* **старого** *имени* OP ! NS и *строкой* **нового** *имени* NP ! NS введен в область соответствующего 1) **тела модуля** и 2) **тела региона**.

Статические условия: В *модуле спецификации* или *регионе спецификации* дополнительная *строка простого имени*, следующая за **END**, может присутствовать только тогда, когда перед **SPEC** имеется дополнительная *строка простого имени*. Если имеются обе эти строки, у них должны быть одинаковые строки имен.

Контекст, не имеющий непосредственно объемлющей группы, может не содержать операторов видимости.

Действительная область, содержащая 1) *модуль спецификации*, 2) *регион спецификации*, должна также содержать по меньшей мере *удаленный модульон* и наоборот.

Если **действительная** область содержит 1) *модуль*, являющийся **телом модуля**, 2) *регион*, являющийся **телом региона**, тогда она должна также содержать 1) *спецификацию модуля*, 2) *спецификацию региона*, так чтобы *строки простых имен* перед ними имели одинаковые строки имен. Считается, что 1) *спецификация модуля* и 2) *спецификация региона* имеют **соответствующее** 1) **тело модуля** и 2) **тело региона**.

Удаленная спецификация в 1) *модуле спецификации* и 2) *регионе спецификации* должны относиться к 1) *модулю спецификации* и 2) *региону спецификации*.

Примеры:

```
23.2    letter_count:
        SPEC MODULE
            SEIZE max;
            count: PROC (input ROW CHARS (max) IN,
                output ARRAY ('A':'Z') INT OUT) END;
            GRANT count;
        END letter_count;                                   (1.1)

24.1    CONTEXT
            count: PROC (ROW CHARS (max) IN,
                ARRAY ('A':'Z') INT OUT) END;
        FOR                                                 (8.1)
```

10.10.3 Квазиоператоры

Синтаксис:

```
< оператор квазиданных > :: =                               (1)
        < оператор квазиописания >                          (1.1)
      | < оператор квазиопределения >                       (1.2)

< оператор квазиописания > :: =                             (2)
            DCL < квазиописание > { , < квазиописание > }*;  (2.1)

< квазиописание > :: =                                      (3)
        < описание квазиячейки >                            (3.1)
      | < описание опознавателя квазиячейки >               (3.2)

< описание квазиячейки > :: =                               (4)
        < список определяющих вхождений > < вид > [ STATIC ]  (4.1)

< описание опознавателя квазиячейки > :: =                  (5)
        < список определяющих вхождений > < вид >
        LOC [NONREF] [DYNAMIC]                              (5.1)

< оператор квазиопределения > :: =                          (6)
        < оператор определения синонимического вида >       (6.1)
      | < оператор определения нового вида >                (6.2)
      | < оператор определения синонима >                   (6.3)
      | < оператор определения квазисинонима >              (6.4)
      | < оператор определения квазипроцедуры >             (6.5)
      | < оператор определения квазипроцесса >              (6.6)
      | < оператор определения квазисигнала >               (6.7)
      | < пусто > ;                                         (6.8)
```

< *оператор определения квазисинонима* > :: = (7)

SYN < *определение квазисинонима* > { , < *определение квазисинонима* > }*; (7.1)

< *определение квазисинонима* > :: = (8)

< *список определяющих вхождений* > { < *вид* > = [< *постоянное значение* >] |
[< *вид* >] = < *постоянное выражение* > } (8.1)

< *оператор определения квазипроцедуры* > :: = (9)

< *определяющее вхождение* > : **PROC** ([< *список квазиформальных параметров* >])
[< *спецификация результата* >] [**EXCEPTIONS** (< *список исключений* >)]
< *список атрибутов процедуры* > **END** [< *строка простого имени* >]; (9.1)

< *список квазиформальных параметров* > :: = (10)

< *квазиформальный параметр* > { , < *квазиформальный параметр* > }* (10.1)

< *квазиформальный параметр* > :: = (11)

< *строка простого имени* > { ,< *строка простого имени* > }*
< *спецификация параметра* > (11.1)

< *оператор определения квазипроцесса* > :: = (12)

< *определяющее вхождение* > : **PROCESS** ([< *список квазиформальных параметров* >])
END [< *строка простого имени* >]; (12.1)

< *оператор определения квазисигнала* > :: = (13)

SIGNAL < *определение квазисигнала* > { ,< *определение квазисигнала* > }*; (13.1)

< *определение квазисигнала* > :: = (14)

< *определяющее вхождение* > [= (< *вид* > { , < *вид* > }*)] [**TO**] (14.1)

Семантика: Квазиоператоры используются в *модулях спецификаций, регионах спецификаций* и *контекстах* для спецификации статических свойств имен. Эти спецификации избыточны, но квазиоператоры могут быть использованы в модульном программировании.

В реализации, где не гарантируется равенство значений между **квазипостоянными** именами **синонимов** и соответствующими им **действительными** именами, может не разрешаться индикация *постоянного значения*.

Заметим, что для имен **меток** в языке CHILL не существуют **квазиопределяющие** *вхождения*.

Статические свойства: Квазиоператоры — это ограниченное представление соответствующих *операторов* с теми же статическими свойствами.

Имя, определяемое *определяющим вхождением*, в *описании опознавателя квазиячейки* является **ссылочным**, если не описан атрибут **NONREF**.

Статические условия: Квазиоператоры — это ограниченное представление соответствующих операторов со всеми их статическими условиями.

Оператор определения квазисинонима может быть непосредственно объемлемым только в *модуле простой спецификации*, *регионе простой спецификации* или *контексте*. *Оператор определения синонима* в *операторе квазиопределения* может быть непосредственно объемлемым только в *спецификации модуля* или *спецификации региона*.

10.10.4 Сопоставление квазиопределяющих и определяющих вхождений

Считается, что два *определяющих вхождения* **совпадают**, если у них идентичные семантические категории, а также:

- Если это имена **синонимов**, тогда они должны иметь одну и ту же **региональность** и значение, **корневой** вид их классов должен быть **похожим**, они оба должны иметь класс М-значения, М-производный класс, М-ссылочный класс, **нулевой** и **полный** классы, и если одно из них — имя квазисинонима и является **константой**, то и другое должно быть таким же.
- Если это имена **элементов перечисления**, то их **перечислимые** виды должны быть **похожими**.
- Если это имена **нового вида** или **синонимического вида**, то их виды должны быть **похожими**.
- Если это имена **ячеек** или **опознавателей ячеек**, тогда они должны иметь одну и ту же **региональность**, оба должны быть **ссылочными** или не быть таковыми, оба должны быть **статическими** или не быть таковыми, а их виды должны быть **похожими**.

- Если это имена **процедур**, тогда они должны иметь одну и ту же **региональность** и **универсальность**, они оба должны быть **критическими** или не быть таковыми, они должны удовлетворять одним и тем же условиям похожести как виды процедур, а соответствующие (по позиции) *строки простых имен* в *списке формальных параметров* и *списке квазиформальных параметров* должны быть одинаковыми.

- Если это имена **процессов**, тогда параметры их определений процессов должны удовлетворять одним и тем же условиям по совпадению и похожести как параметры имен **процедур**.

- Если это имена **сигналов**, тогда они оба должны описывать или не описывать атрибут **TO**, их списки видов должны содержать одно и то же количество видов, а соответствующие виды должны быть **похожими**.

Если два структурных вида **связаны новизной** в области R, тогда они должны иметь один и тот же набор **видимых** имен полей в R.

Имеют место следующие правила:

- Если *строка имени* в области, не являющейся *областью модуля спецификации*, *региона спецификации* или *контекста* **связана** с **квазиопределяющим** *вхождением*, тогда она должна быть **связана** также с *определяющим вхождением*, не являющимся **квазиопределяющим** *вхождением*, и, кроме того,

  — *строка имени* будет **связана** с **квазиопределяющим** *вхождением* QD, а также с **действительным** *определяющим вхождением* RD в области R; тогда:

    1. QR и RD должны **совпадать**, как определено выше, и

    2. RD и QD должны быть заключены в объемлемую группу R или же не должны быть заключены в нее, или если R — область *модуля* или *региона*, являющихся **телом модуля** или **телом региона**, то QD должно быть заключено в группу **соответствующей** *спецификации модуля* или *региона*, а RD должно быть заключено в группу R.

  — если *строка имени* в **действительной** области R связана с **квазиопределяющим** *вхождением*, которое заключено в группу R (то есть окружено модульоном спецификации), тогда она должна быть также **связана** с **действительным** *определяющим вхождением*, которое окружено группой *модуля* и *региона*, указанных в *удаленном модульоне*, непосредственно включенном в R (неформально, если интерфейс позволяет, то это должно быть реализовано). Если **квазиопределяющее** *вхождение* заключено в группу *спецификации модуля* или *спецификации региона*, тогда **действительное** *определяющее вхождение* должно быть заключено в группу **соответствующего** *модульона*;

  — если *строка имени* в **действительной** области R **связана** с **действительным** *определяющим вхождением*, включенным в группу *модуля* или *региона*, указанных *удаленным модульоном*, непосредственно объемлемым в R, тогда она должна быть **связана** также с **квазиопределяющим** *вхождением*, непосредственно включенным в группу R (то есть окруженным модульоном спецификации. Неформально, если реализация позволяет, то интерфейс осуществляется);

  — для каждой *строки имени* в области Q *модуля спецификации* или *региона спецификации*, непосредственно включенных в **действительную** область R, **связанную** с *определяющим вхождением*, не окруженным Q, должна быть идентичная *строка имени* в области *модуля* или *региона*, которая указана *удаленным модульоном*, непосредственно включенным в R, **связанную** с тем же *определяющим вхождением* (неформально, если интерфейс занимает ресурс, то это должно реализовываться).

- Если две *строки имен* **связаны** с одними и теми же 1) **действительным** *определяющим вхождением*, 2) **квазиопределяющим** *вхождением* в области, то обе *строки имен* должны быть **связаны** с теми же 1) **квазиопределяющим** *вхождением*, 2) **действительным** *определяющим вхождением* или же обе *строки имени* не будут **связаны**.

- **Действительная новизна**, может не быть **новизной**, **связанной** с двумя "**квазиновизнами**" в любой области.

  Пусть **квазиновизна** QN и **действительная новизна** RN — это "**новизны**", **связанные** друг с другом в области R, тогда RN и QN должны быть включены в объемлемую группу R или обе не должны включаться в группу R, или если R — это область *модуля* или *региона*, являющихся **телом модуля** или **телом региона**, тогда RN должна быть включена в группу R, а QN должна быть заключена в группу **соответствующей** *спецификации модуля* или *спецификации региона*.

11 ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ

11.1 ПРОЦЕССЫ И ИХ ОПРЕДЕЛЕНИЯ

Процесс — это последовательное выполнение ряда операторов. Он может выполняться параллельно с другими процессами. Поведение процесса описывается определением процесса (см. раздел 10.5), описывающим объекты, локальные по отношению к процессу, и ряд операторов действия, выполняемых последовательно.

Процесс создается в результате вычисления выражения запуска (см. раздел 5.2.14). Он становится активным (то есть выполняется) и может выполняться параллельно с другими процессами. Созданный процесс — это активация определения, указанного именем **процесса** в определении процесса. Может создаваться и выполняться параллельно произвольное количество процессов с одним и тем же определением. Каждый процесс однозначно идентифицируется значением экземпляра, получаемым как результат выражения запуска или вычисления операции **THIS**. Создание процесса является причиной создания его локально описанных ячеек, за исключением тех, которые описаны с атрибутом **STATIC** (см. раздел 10.9), и локально определяемых значений и процедур. Считается, что локально описанные ячейки, значения и процедуры имеют ту же активацию, что и созданный процесс, к которому они принадлежат. Предполагается, что воображаемый внешний процесс (см. раздел 10.8), являющийся целой выполняемой CHILL-программой и созданный выражением запуска, выполняется системой, управляющей выполнением программы. При создании процесса его формальные параметры, если имеются, обозначают значения и ячейки, представляемые соответствующими фактическими параметрами в выражении запуска.

Процесс прекращается в результате выполнения оператора останова, при достижении конца тела процесса или при прекращении работы программы обработки исключений, описанной в конце определения процесса (полный проход). Если воображаемым внешним процессом выполняется оператор останова или он осуществляет полный проход, то полное прекращение процесса происходит тогда и только тогда, когда все другие процессы в программе закончены.

Процесс на уровне программирования на языке CHILL всегда находится в двух состояниях: он либо активен (то есть выполняется), либо задержан (то есть ждет условий выполнения). Переход процесса из активного состояния в состояние задержки называется задержкой процесса; переход от состояния задержки в активное состояние называется реактивацией процесса.

11.2 ВЗАИМОИСКЛЮЧЕНИЕ И РЕГИОНЫ

11.2.1 Общие положения

Регионы (см. раздел 10.7) — это средство, обеспечивающее процессам взаимоисключающий доступ к ячейкам, описанным в регионах. Статические контекстовые условия (см. раздел 11.2.2) таковы, что доступ процесса (который не является воображаемым внешним процессом) к ячейкам, описанным в регионе, может осуществляться только в результате вызова процедур, определяемых внутри региона и допускаемых регионом.

Считается, что имя **процедуры** обозначает **критическую** процедуру (это имя **критической процедуры**), если она определена внутри региона и допускается регионом.

Считается, что регион свободен тогда и только тогда, когда управление не принадлежит ни одной из его **критических** процедур или же принадлежит региону, самому выполняющему инициализации границей области.

Регион заперт (для предотвращеия параллельного выполнения), если:

- осуществлен вход в регион (заметим, что поскольку регионы не окружены блоком, то не могут предприниматься одновременные попытки войти в регион);
- вызвана **критическая** процедура региона;
- процесс, задержанный в регионе, реактивируется;

Зона освобождается и становится снова свободной, если:

- осуществлен выход из региона;
- осуществляется возврат **критической** процедуры;
- **критическая** процедура выполняет действие, в результате которого процесс становится задержанным (см. раздел 11.3). В случае динамически вложенных вызовов **критической** процедуры освобождается только последний запертый регион;
- процесс выполнения **критической** процедуры заканчивается. В случае динамически вложенных вызовов критической процедуры будут освобождаться все регионы, запертые процессом.

Если в то время, когда регион заперт, процесс пытается вызвать одну из его **критических** процедур либо задержанный в регионе процесс реактивируется, то процесс приостанавливается до освобождения региона. (Заметим, что процесс остается активным в смысле языка CHILL.)

Когда регион освобождается и приостановлено несколько процессов, пытающихся вызвать одну из его **критических** процедур или быть реактивированными в одной из его **критических** процедур, то для запирания региона выбирается только один процесс согласно алгоритму расписания, определяемому реализацией.

11.2.2 Региональность

С целью статической проверки того факта, что доступ к описанной в регионе ячейке может быть осуществлен только в результате вызова **критических** процедур или в результате входа в регион для выполнения инициализаций по границе области, должны быть выполнены следующие статические контекстовые условия:

- требования **региональности**, упомянутые в соответствующих разделах (оператор присваивания, вызов процедуры, оператор посылки, оператор результата и т. д.);
- **внутрирегиональные** процедуры не являются **общими** (см. раздел 10.4);
- **критические** процедуры не являются ни **общими**, ни **рекурсивными** (см. раздел 10.4).

Ячейка и *вызов процедуры* являются **внутрирегиональными** и **внерегиональными**, что касается их **региональности**. *Значение* бывает **внутрирегиональным**, **внерегиональным** или **nil** (пустой указатель), что касается его **региональности**. Эти свойства определяются ниже.

1. **Ячейка**

   Ячейка является **внутрирегиональной** тогда и только тогда, когда имеется:

   - *имя доступа*, которое либо:
     - — *имя ячейки*, описанное текстуально внутри *региона* или *региона спецификации* и не определенное в *формальном параметре* **критической** процедуры;
     - — *имя опознавателя-ячейки*, где *ячейка* в ее описании является **внутрирегиональной** или определена в *формальном параметре* **внутрирегиональной** процедуры;
     - — *имя перечисления ячейки*, где *ячейка массива* или *ячейка строки* соответствующего *оператора цикла* являются **внутрирегиональными**;
     - — *имя ячейки с присоединением*, где *ячейка структуры* в соответствующем *операторе цикла* является **внутрирегиональной**.
   - *разыменованная закрепленная ссылка*, где *примитивное значение закрепленной ссылки* является **внутрирегиональным**;
   - *разыменованная свободная ссылка*, где *примитивное значение свободной ссылки* является **внутрирегиональным**;
   - *разыменованный ряд*, где *примитивное значение ряда* является **внутрирегиональным**;
   - *элемент массива* или *подмассив*, где *ячейка массива* является **внутрирегиональной**;
   - *элемент строки* или *подстрока*, где *ячейка строки* является **внутрирегиональной**;
   - *поле структуры*, где *ячейка структуры* является **внутрирегиональной**;
   - *вызов процедуры возврата ячейки*, где описанное *имя процедуры* является **внутрирегиональным**;
   - *вызов встроенной программы выдачи ячейки*, где в описании на языке CHILL или в реализации указана **внутрирегиональность** вызова;
   - *преобразование ячейки*, где *ячейка статического вида* в нем является **внутрирегиональной**.

   Ячейка, не являющаяся **внутрирегиональной**, является **внерегиональной**.

2. **Значение**

   Региональность *значения* зависит от класса значения. Если значение имеет М-производный класс, **полный** класс или **нулевой** класс, то его **региональностью** является **nil**. В противном случае, если значение имеет класс М-значения или М-ссылочный класс, то его **региональность** зависит от вида М следующим образом:

   Если *значение* имеет класс М-значения и М не имеет **свойства ссылаемости**, тогда его **региональность** **nil**; в противном случае *значение* — это *операнд-6* (и имеет **свойство ссылаемости**) или *условное выражение*:

Если это *примитивное значение*, в таком случае:

- Если это *содержимое ячейки*, являющееся *ячейкой*, тогда значение — это содержимое *ячейки*;
- Если это *имя значения*, в таком случае:
  - если это *имя синонима*, тогда это имя *постоянного значения* в его определении;
  - если это *имя значения с присоединением*, тогда это имя *примитивного значения структуры* в соответствующем операторе цикла;
  - если это *имя принятого значения*, тогда оно **внерегиональное.**
- Если это *кортеж*, тогда если одно из вхождений *значения* в нем имеет **региональность не nil**, в этом случае это региональность данного *значения* (сделанный выбор не имеет значения; см. статические условия раздела 5.2.5); в противном случае его региональность — **nil**;
- Если это *элемент массива значений* или *подмассив значений*, тогда это *примитивное значение массива* в нем.
- Если это *поле структуры значений*, тогда значение — это *примитивное значение структуры* в нем.
- Если это *преобразование выражения*, тогда значение — это значение *выражения преобразования.*
- Если это *вызов процедуры возврата значения*, тогда значение — это значение *вызова* такой *процедуры.*
- Если это *вызов встроенной программы выдачи значения*, то в описании на языке CHILL или в реализации описывается, является ли значение **внутрирегиональным** или **внерегиональным.**

Если значение — *ссылаемая ячейка*, тогда значение — это значение *ячейки.*

Если значение — это *выражение получения*, тогда оно **внерегиональное.**

Если значение — *условное выражение*, тогда если одно из вхождений *субвыражения* в нем имеет **региональность не nil**, в таком случае это региональность данного *субвыражения* (сделанный выбор не имеет значения; см. статические условия раздела 5.3.2); в противном случае его региональность — **nil.**

3. **Имя процедуры**

   Имя процедуры является **внутрирегиональным** тогда и только тогда, если оно определено внутри *региона* или *региона спецификации* и не является **критическим** (то есть не разрешено регионом). В противном случае оно — **внерегиональное.**

4. **Вызов процедуры**

   Вызов процедуры является **внутрирегиональным,** если он содержит *имя процедуры*, являющееся **внутрирегиональным;** в противном случае он — **внерегиональный.**

Значение является **регионально надежным** для нетерминального символа (используемого только для *ячейки*, *вызова процедуры* и *имени процедуры*) тогда и только тогда, когда:

- нетерминальный символ является **внерегиональным,** а *значение* не является **внутрирегиональным;**
- нетерминальный символ является **внутрирегиональным,** а *значение* не является **внерегиональным;**
- **региональность** нетерминального символа **nil.**

11.3 ЗАДЕРЖКА ПРОЦЕССА

Активный процесс может стать задержанным при выполнении (вычислении) одного из следующих действий (выражений):

- оператора задержки (см. раздел 6.16);
- оператора выбора задержки (см. раздел 6.17);
- выражение получения (см. раздел 5.3.9);
- оператора варианта получения сигнала (см. раздел 6.19.2);
- оператора варианта получения из буфера (см. раздел 6.19.3);
- оператора посылки в буфер (см. раздел 6.18.3).

Если процесс становится задержанным, когда управление находится в **критической** процедуре, то освобождается соответствующий регион. Динамический контекст процесса сохраняется до его реактивации. Затем процесс снова пытается запереть регион, что может вызвать приостановку процесса.

11.4 РЕАКТИВАЦИЯ ПРОЦЕССА

Задержанный процесс может стать реактивированным при наличии контроля и прерывания времени (см. главу 9). Он может также стать реактивированным, если другой процесс выполнит (вычислит) одно из следующих действий (выражений):

- оператор продолжения (см. раздел 6.15);
- оператор посылки сигнала (см. раздел 6.18.2);
- оператор посылки в буфер (см. раздел 6.18.3);
- выражение получения (см. раздел 5.3.9);
- оператор варианта получения из буфера (см. раздел 6.19.3).

Когда процесс реактивирует другой процесс при запертом регионе, то он остается активным, то есть он не освобождает регион в данный момент.

11.5 ОПЕРАТОРЫ ОПРЕДЕЛЕНИЯ СИГНАЛОВ

Синтаксис:

< оператор определения сигнала > :: = *(1)*
SIGNAL *< определение сигнала >* { , *< определение сигнала >*}*; *(1.1)*

< определение сигнала > :: = *(2)*
< определяющее вхождение > [= (*< вид >* { , *< вид >* }*)] [**TO** < имя процесса >] *(2.1)*

Семантика: Определение сигнала описывает функцию композиции и декомпозиции значений, передаваемых между процессами. Если сигнал посылается, то передается определенный список значений. Если в операторе варианта получения нет процесса, ожидающего сигнал, тогда значения сохраняются до тех пор, пока процесс не получит эти значения.

Статические свойства: *Определяющее вхождение* в *определении сигнала* определяет имя **сигнала**.

Имя **сигнала** обладает следующими свойствами и содержит:

- необязательный список приписанных видов, являющихся видами, указанными в *определении сигнала*;
- необязательное приписанное имя **процесса**, являющееся *именем процесса*, описанного после **TO**.

Статические условия: Ни один из *видов* в *определении сигнала* не может обладать **свойством беззначимости**.

Примеры:

15.27 **SIGNAL** *initiate = (INSTANCE),*
terminate; *(1.1)*

12 ОБЩИЕ СЕМАНТИЧЕСКИЕ СВОЙСТВА

12.1 ПРАВИЛА ДЛЯ ВИДОВ

12.1.1 Свойства видов и классов

12.1.1.1 Свойство неизменяемости

Неформальное описание

Вид обладает **свойством неизменяемости**, если это **неизменяемый** вид или если он содержит компоненту или субкомпоненту и т. д. **неизменяемого** вида.

Определение

Вид обладает **свойством неизменяемости** тогда и только тогда, когда это:

- массивный вид с видом **элемента**, обладающим **свойством неизменяемости**;
- структурный вид, где по крайней мере один из его видов **полей** обладает **свойством неизменяемости** и где поле не является полем **признака** с неявным **неизменяемым** видом **параметризованного** структурного вида;
- **неизменяемый** вид.

12.1.1.2 Параметризованные виды

Неформальное описание

Вид является **параметризованным**, если его можно параметризовать.

Определение

Вид является **параметризованным** тогда и только тогда, когда это:

- строковый вид;
- массивный вид;
- **параметризованный вариантный** структурный вид.

12.1.1.3 Свойство ссылаемости

Неформальное описание

Вид обладает **свойством ссылаемости**, если это ссылочный вид или содержит компоненту или субкомпоненту и пр. ссылочного вида.

Определение

Вид обладает **свойством ссылаемости** тогда и только тогда, когда это:

- ссылочный вид;
- массивный вид с видом **элемента**, обладающим **свойством ссылаемости**;
- структурный вид, где по крайней мере один из его видов **полей** обладает **свойством ссылаемости**.

12.1.1.4 Свойство теговой параметризации

Неформальное описание

Вид обладает **свойством теговой параметризации**, если это **теговый параметризованный** структурный вид или если он содержит компоненту или субкомпоненту и пр. **тегового параметризованного** структурного вида.

Определение

Вид обладает **свойством теговой параметризации** тогда и только тогда, когда это:

- массивный вид с видом **элемента**, обладающим **свойством теговой параметризации**;
- структурный вид, где по крайней мере один из его видов **поля** обладает **свойством теговой парамеризации**;
- **теговый параметризованный** структурный вид.

12.1.1.5 Свойство беззначимости

Неформальное описание

Вид обладает **свойством беззначимости**, если для вида не существует обозначения выражения или примитивного значения.

Определение

Вид обладает **свойством беззначимости** тогда и только тогда, когда это:

- событийный вид, буферный вид, доступный вид, ассоциативный вид или текстовый вид;
- массивный вид с видом **элемента**, обладающим **свойством беззначимости**;
- структурный вид, где по крайней мере один из его видов **поля** обладает **свойством беззначимости**.

12.1.1.6 Корневой вид

Любой вид M имеет **корневой** вид, определяемый как:

- M, если M — не ограниченный вид;
- **родительский** вид вида M, если M — ограниченный вид.

Любой класс M-значения или M-производный класс имеет **корневой** вид, являющийся **корневым** видом для M.

12.1.1.7 Результирующий класс

Для данных двух **совместимых** классов (см. раздел 12.1.2.16), являющихся **полным** классом, классом M-значения или M-производным классом, где M — дискретный вид, множественный вид или строковый вид, **результирующий класс** определяется в терминах понятия **результирующего** вида R для M и N и **корневого** вида P для M.

Для двух **подобных** видов M и N **результирующий** вид R определяется следующим образом:

- если **корневой** вид для одного из них является **фиксированным** строковым видом, а для другого — **изменяющимся** строковым видом, тогда это **корневой** вид одного из них (M или N), являющийся **изменяющимся** строковым видом;
- в противном случае это P.

Результирующий класс определяется следующим образом:

- **результирующий класс** класса M-значения и класса N-значения есть класс R-значения;
- **результирующий класс** класса M-значения и N-производного класса или **полный** класс есть класс P-значения;
- **результирующий класс** M-производного класса и N-производного класса есть R-производный класс;
- **результирующий класс** M-производного класса и **полный** класс есть P-производный класс;
- **результирующий класс полного** класса и **полный** класс есть **полный** класс.

Пусть дан список C_i попарно **совместимых** классов (i=1,...,n), тогда **результирующий класс** списка классов определяется рекурсивно как **результирующий класс результирующего класса** списка C_i (i=1,...,n—1) и класса C_n, если n > 1; в противном случае он определяется как **результирующий класс** классов C_1 и C_2.

12.1.2 Отношения видов и классов

12.1.2.1 Общие положения

В последующих разделах определяются отношения совместимости между видами, классами, а также между видами и классами. Эти отношения используются во всем документе при определении статических условий.

Отношения совместимости непосредственно определяются в терминах других отношений, главным образом используемых в данной главе для вышеупомянутых целей.

12.1.2.2 Отношения эквивалентности видов

Неформальное описание

При формулировке отношений совместимости учитываются следующие отношения эквивалентности:

- Два вида **подобны**, если они одного и того же рода, то есть если они имеют одни и те же свойства наследования.
- Два вида **v-эквивалентны** (эквивалентны по значению), если они **подобны** и имеют одну и ту же **новизну**.
- Два вида **эквивалентны**, если они **v-эквивалентны** и если также принимаются во внимание возможные различия в представлении значений в памяти или минимальный размер занимаемой памяти.
- Два вида **l-эквивалентны** (эквивалентны по ячейке), если они **эквивалентны**, а также имеют одну и ту же спецификацию **неизменяемости**.
- Два вида **похожи**, если они неразличимы; то есть если все операции, которые можно применить по отношению к объектам одного вида, применимы также к другому виду при условии, что **новизна** не принимается во внимание.
- Два вида **связаны новизной**, если они **похожи** и имеют одинаковую спецификацию **новизны**.

Определение

В следующих разделах отношения эквивалентности видов даны в форме (отдельного) набора отношений. Полные алгоритмы эквивалентности получаются в результате симметричного, рефлексивного и транзитивного замыкания данного набора отношений. Виды, упомянутые в отношениях, могут быть введены виртуально или быть динамическими. В последнем случае полная проверка эквивалентности может быть выполнена только во время прогона. Ошибка при проверке динамической части приводит к исключениям *RANGEFAIL* или *TAGFAIL* (см. соответствующие разделы).

Проверка двух рекурсивных видов на предмет эквивалентности потребует проверки ассоциативных видов в соответствующих частях набора рекурсивных видов, с помощью которых они определяются. Эквивалентность между видами сохраняется при отсутствии противоречий. (Как следствие, ход проверки алгоритма завершается успешно, если два сравнивавшихся ранее вида при сравнении совпали.)

12.1.2.3 Отношение подобия

Два вида **подобны** тогда и только тогда, когда это:

- целые виды;
- булевские виды;
- символьные виды;
- такие перечислимые виды, что:
  1. они определяют одно и то же **количество значений**;
  2. для каждого имени **элемента перечисления**, определяемого одним видом, существует имя **элемента перечисления**, определяемое другим видом, имеющим ту же строку имени и то же значение представления;
  3. оба вида — **нумеруемые** перечислимые виды или **ненумеруемые** перечислимые виды;
- ограниченные виды с **подобными родительскими** видами;
- один из них — ограниченный вид, **родительский** вид которого **подобен** другому виду;
- такие множественные виды, что их элементные виды **эквивалентны**;
- такие закрепленные ссылочные виды, что их **ссылаемые** виды **эквивалентны**;

- свободные ссылочные виды;
- такие рядовые виды, что их **ссылаемые начальные** виды **эквивалентны**;
- такие процедурные виды, что:
  1. они имеют одно и то же количество **спецификаций параметров** и соответствующие (по позиции) **спецификации параметров** имеют **l-эквивалентные** виды и те же атрибуты параметров, если имеются;
  2. оба вида имеют или не имеют **спецификацию результата**. При наличии **спецификации результата** оба вида должны иметь **l-эквивалентные** виды и те же атрибуты, если имеются;
  3. они имеют один и тот же список имен **исключений**;
  4. они обладают одной и той же **рекурсивностью**;
- экземплярные виды;
- такие событийные виды, что оба вида не имеют **длины события** или оба имеют одну и ту же **длину события**;
- такие буферные виды, что:
  1. оба вида не имеют **длины буфера** или оба имеют одну и ту же **длину буфера**;
  2. они имеют **l-эквивалентные буферные элементные** виды;
- ассоциативные виды
- такие доступные виды, что:
  1. оба вида не имеют **индексного** вида или оба имеют **эквивалентные индексные** виды;
  2. по крайней мере один из них имеет вид **записи**, или оба имеют виды **записи**, которые являются **l-эквивалентными**, или имеют **статические** или **динамические** виды **записи**.
- такие текстовые виды, которые имеют:
  1. одну и ту же **длину текста**;
  2. **l-эквивалентные текстовые** виды **записи**;
  3. **l-эквивалентные доступные** виды.
- продолжительные виды;
- абсолютно-временные виды;
- такие строковые виды, что они оба являются **битовыми** строковыми видами или **символьными** строковыми видами;
- такие массивные виды, что:
  1. их **индексные** виды **v-эквивалентны**;
  2. их виды **элементов эквивалентны**;
  3. их **форматы элементов эквивалентны**;
  4. они имеют одно и то же **количество элементов**. Эта проверка является динамической, если один или оба вида является(ются) динамическим(и). Ошибка при проверке приводит к исключению *RANGEFAIL*;
- структурные виды, не являющиеся **параметризованными** структурными видами, и такие виды, что:
  1. в строгом синтаксисе они имеют одно и то же количество *полей* и соответствующие (по позиции) *поля* **эквивалентны**;
  2. если они оба — **параметризованные вариантные** структурные виды, то списки их классов должны быть **совместимыми**;
- такие **параметризованные** структурные виды, что:
  1. их **начальные вариантные** структурные виды **подобны**;
  2. их соответствующие (по позиции) значения одни и те же. Эта проверка является динамической, если один или оба вида является(ются) динамическим(и). Ошибка при проверке приводит к исключению *TAGFAIL*.

12.1.2.4 Отношение v-эквивалентности

Два вида **v-эквивалентны** тогда и только тогда, когда они **подобны** и имеют одну и ту же **новизну**.

12.1.2.5 Отношение эквивалентности

Два вида **эквивалентны** тогда и только тогда, когда они **v-эквивалентны** и:

- если один из них ограниченного вида, то другой должен также быть ограниченного вида, и обе **верхние границы** должны быть равными, также как и **нижние границы**;

- если один из них — это **фиксированный** строковый вид, то другой также должен быть **фиксированного** строкового вида, и они должны иметь одну и ту же **длину строки**. Эта проверка является динамической в случае, когда один вид является динамическим или оба вида являются динамическими. Ошибка при проверке приводит к исключению *RANGEFAIL*;

- если один из видов — **изменяющийся** строковый вид, то другой также должен быть **изменяющимся** строковым видом, и они должны иметь одну и ту же **длину строки**. Эта проверка является динамической, если один или оба вида является(ются) динамическим(и). Ошибка при проверке приводит к исключению *RANGEFAIL*.

12.1.2.6 Отношение l-эквивалентности

Два вида **l-эквивалентны** тогда и только тогда, когда они **эквивалентны** и если один из них **неизменяемого** вида, то и другой должен быть **неизменяемого** вида, и если они:

- закрепленные ссылочные виды, то их **ссылаемые** виды должны быть **l-эквивалентными**;

- рядовые виды, то их **ссылаемые начальные** виды должны быть **l-эквивалентными**;

- массивные виды, то их виды **элементов** должны быть **l-эквивалентными**;

- структурные виды, не являющиеся **параметризованными** структурными видами, то соответствующие (по позиции) *поля* в строгом синтаксисе должны быть **l-эквивалентными**; если они являются **параметризованными** структурными видами, то их **начальные вариантные** структурные виды должны быть **l-эквивалентными**.

12.1.2.7 Отношения эквивалентности и l-эквивалентности для полей

Два *поля* (оба *поля* в контексте двух данных структурных видов) являются 1) **эквивалентными**, 2) **l-эквивалентными** тогда и только тогда, когда оба *поля* — это , являющиеся 1) **эквивалентными**, 2) **l-эквивалентными**, либо оба поля — это *альтернативные поля*, являющиеся 1) **эквивалентными**, 2) **l-эквивалентными**.

Отношения **эквивалентности** и **l-эквивалентности** определяются рекурсивно для соответствующих *фиксированных полей*, *вариантных полей*, *альтернативных полей* и *вариантных альтернатив* соответственно следующим образом:

- *Фиксированные поля* и *вариантные поля*
  1. Оба *фиксированных поля* или оба *вариантных поля* должны иметь **эквивалентный формат поля**.
  2. Оба вида **поля** должны быть 1) **эквивалентными**, 2) **l-эквивалентными**.

- *Альтернативные поля*
  1. Оба *альтернативных поля* имеют или оба не имеют *списки признаков*. В первом случае *списки признаков* должны иметь одно и то же количество имен **полей признаков** и соответствующие (по позиции) имена **полей признаков** должны обозначать соответствующие *фиксированные поля*.
  2. Оба поля должны иметь одно и то же количество *вариантных альтернатив* и соответствующие (по позиции) *вариантные альтернативы* должны быть 1) **эквивалентными**, 2) **l-эквивалентными**. *фиксированные поля*
  3. Оба поля не должны иметь или оба должны иметь описанный атрибут **ELSE**. В последнем случае за ним должно следовать одно и то же количество *вариантных полей* и соответствующие (по позиции) *вариантные поля* должны быть 1) **эквивалентными**, 2) **l-эквивалентными**.

- *Вариантные альтернативы*
  1. Обе *вариантные альтернативы* должны иметь одно и то же количество *списков меток выбора* и соответствующие (по позиции) *списки меток выбора* должны быть либо оба *несоответствующими*, либо должны определять один и тот же набор значений.
  2. Обе *вариантные альтернативы* должны иметь одно и то же количество *вариантных полей* и соответствующие (по позиции) *вариантные поля* должны быть 1) **эквивалентными**, 2) **l-эквивалентными**.

12.1.2.8 Отношение эквивалентности для формата

В конце раздела было предположение, что каждая *позиция* имеет вид:

POS *(< номер > , < начальный бит >,< длина >)*,

а каждый *шаг* имеет вид:

STEP *(< позиция > , < величина шага >)*.

В разделе 3.12.5 даны соответствующие правила приведения позиции или шага к требуемому виду.

- Формат поля

  Два **формата поля эквивалентны**, если они оба **NOPACK**, оба **PACK** или оба *позиция*. В последнем случае один атрибут *позиция* должен быть **эквивалентен** другому (см. ниже).

- Формат элементов

  Два **формата элементов эквивалентны**, если они оба есть **NOPACK**, **PACK** или оба *шаг*. В последнем случае *позиция* одного *шага* должна быть **эквивалентна** *позиции* другого шага, а *величина шага* должна быть одной и той же для двух **форматов элементов**.

- Позиция

  Позиция **эквивалентна** другой *позиции* тогда и только тогда, когда оба *вхождения слова*, оба вхождения *начального бита* и *оба вхождения длины* дают одно и то же значение.

12.1.2.9 Отношение похожести

Два вида **похожи** тогда и только тогда, когда оба вида являются или не являются **неизменяемыми** видами, имеют **новизну nil** или имеют одну и ту же **новизну**, а также если это:

- целые виды;
- булевские виды;
- символьные виды;
- **подобные** перечислимые виды;
- ограниченные виды с одинаковыми **верхними** и **нижними границами**;
- такие множественные виды, что их **элементные** виды **похожи**;
- такие закрепленные ссылочные виды, что их **ссылаемые** виды **похожи**;
- свободные ссылочные виды;
- такие рядовые виды, что их **ссылаемые начальные** виды **похожи**;
- такие процедурные виды, что:
  1. они имеют одно и то же количество **спецификаций параметров** и соответствующие (по позиции) **спецификации параметров** имеют **похожие** виды и одни и те же атрибуты параметров, если они имеются;
  2. оба вида имеют или не имеют **спецификацию результата**. При наличии **спецификации результата** они должны иметь **похожие** виды и одни и те же атрибуты, если они имеются;
  3. они имеют один и тот же список имен **исключений**;
  4. они имеют одну и ту же **рекурсивность**;
- экземплярные виды;
- такие событийные виды, что оба вида не имеют **длины события** или имеют одну и ту же **длину события**;
- такие буферные виды, что:
  1. оба вида не имеют **длины буфера** или имеют одну и ту же **длину буфера**;
  2. оба вида имеют **похожие буферные элементные** виды;
- ассоциативные виды;
- такие доступные виды, что:
  1. оба вида не имеют **индексного** вида или оба имеют **похожие индексные** виды;
  2. по крайней мере один не имеет вида **записи** или оба имеют виды **записи**, которые **похожи**, оба имеют **статические** или **динамические** виды **записи**:
- такие текстовые виды, что:
  1. они имеют одну и ту же **длину текста**;
  2. их виды **записи текста похожи**;
  3. их **доступные** виды **похожи**;
- продолжительные виды;
- абсолютно-временные виды;

- такие строковые виды, что:
  1. оба вида — это **битовые** строковые виды или **символьные** строковые виды;
  2. они имеют одну и ту же **длину строки**;
  3. оба вида — это **фиксированные** строковые виды или **изменяющиеся** строковые виды;
- такие массивные виды, что:
  1. их **индексные** виды **похожи**;
  2. их виды **элементов похожи**;
  3. их **форматы элементов эквивалентны**;
  4. они имеют одно и то же **количество элементов**;
- такие структурные виды, не являющиеся **параметризованными** структурными видами, что:
  1. в строгом синтаксисе они имеют одно и то же количество *полей* и соответствующие (по позиции) *поля* **похожи**;
  2. если оба вида — **параметризованные вариантные** структурные виды, то их списки классов должны быть **совместимыми**;
- такие **параметризованные** структурные виды, что:
  1. их **начальные вариантные** структурные виды **похожи**;
  2. их соответствующие (по позиции) значения являются одинаковыми.

12.1.2.10 Отношение похожести для полей

Два *поля* (оба *поля* в контексте двух данных структурных видов) **похожи** тогда и только тогда, когда оба *поля* являются либо *фиксированными полями*, которые **похожи** , либо *альтернативными полями*, которые **похожи**.

Отношение **похожести** определяется рекурсивно для соответствующих *фиксированных полей*, *вариантных полей*, *альтернативных полей* и *вариантных альтернатив* соответственно следующим образом:

- *Фиксированные поля* и *вариантные поля*
  1. Оба *фиксированных* или *вариантных поля* должны иметь **эквивалентный формат поля**.
  2. Оба вида **поля** должны быть **похожими**.
  3. Оба *фиксированных поля* или оба *вариантных поля* должны иметь одну и ту же приписанную *строку имени*.
- *Альтернативные поля*
  1. Оба *альтернативных поля* имеют или не имеют *списки признаков*. В первом случае *списки признаков* должны иметь одно и то же количество имен **поля признаков**, а соответствующие (по позиции) имена **полей признаков** должны обозначать соответствующие *фиксированные поля*.
  2. Оба поля должны иметь одно и то же количество *вариантных альтернатив*, а соответствующие (по позиции) *вариантные альтернативы* должны быть **похожими**;
  3. Оба поля не должны иметь или должны иметь описанный атрибут ELSE. В последнем случае за этим атрибутом должно следовать одно и то же количество *вариантных полей* и соответствующие (по позиции) *вариантные поля* должны быть **похожими**.
- *Вариантные альтернативы*
  1. Обе *вариантные альтернативы* должны иметь одно и то же количество *списков меток выбора* и соответствующие (по позиции) *списки меток выбора* должны быть либо *несоответствующими*, либо должны определять один и тот же набор значений.
  2. Обе *вариантные альтернативы* должны иметь одно и то же количество *вариантных полей* и соответствующие (по позиции) *вариантные поля* должны быть **похожими**.

12.1.2.11 Отношение связи новизной

Неформальное описание

Каждый **квазиновый** вид в программе должен представлять только один **действительный** новый вид. Это устанавливается следующим образом: когда *строка имени* **связана** как с **действительным** *определяющим вхождением*, так и с **квазиопределяющим** *вхождением*, то все включенные новые виды должны быть представлены попарно. Затем устанавливается **связь новизной** между "**новизнами**".

Определение

Отношение **спаренности по новизне** применяется между двумя видами и областью. Для каждой *строки имени*, **связанной** в области R, как с **действительным** *определяющим вхождением*, так и с **квазиопределяющим** *вхождением*, в случае если последние — это:

- имена **синонимов**, то **корневые** виды их классов **спарены по новизне** в R;
- имена **элементов перечислений**, тогда виды приписанных **перечислимых** видов **спарены по новизне** в R;
- имена **ячейки** или **опознавателя-ячейки**, то виды их ячеек **спарены по новизне** в R;
- имена **процедур**, то виды **спецификаций параметров** и **спецификации результатов**, если имеются, **спарены по новизне** в R;
- имена **процессов**, то виды **спецификаций параметров спарены по новизне** в R;
- имена **сигналов**, то виды в списке видов **спарены по новизне** в R;

Если два вида **спарены по новизне** в области R, тогда если это:

- множественные виды, то их **элементные** виды **спарены по новизне** в R;
- закрепленные ссылочные виды, то их **ссылаемые** виды **спарены по новизне** в R;
- рядовые виды, то их **ссылаемые начальные** виды **спарены по новизне** в R;
- процедурные виды, то их виды **спецификаций параметров** и **спецификации результата**, если имеются, **спарены по новизне** в R;
- буферные виды, то их **буферные элементные** виды **спарены по новизне** в R;
- доступные виды, то их **индексные** виды и виды **записи**, если имеются, **спарены по новизне** в R;
- текстовые виды, то их **индексные** виды, если имеются, **спарены по новизне** в R;
- массивные виды, то их **индексные** виды и виды **элементов спарены по новизне** в R;
- структурные виды, то их виды **поля спарены по новизне** в R.

Если два вида **спарены по новизне** в области R, а их **новизна** не равна одна другой, тогда **действительная новизна** и **квазиновизна** видов **связаны новизной** в R.

Одна **новизна** и другая **новизна** считаются одинаковыми, если:

- каждая из них — одна и та же **действительная** новизна;
- одна из них — **действительная новизна**, другая — **квазиновизна**, а **связаны** они **новизной**.

12.1.2.12 Отношение совместимости по чтению

Неформальное описание

Отношение **совместимости по чтению** уместно для **эквивалентных** видов. Считается, что вид M совместим по чтению с видом N, если он или его возможные (суб)компоненты имеют одинаковые или более точные спецификации **неизменяемости (только-чтения)** и, если это ссылочные виды, то они ссылаются к **l-эквивалентным** ячейкам. Таким образом, это отношение несимметрично.

Примеры:

READ REF READ *CHAR* **совместим по чтению** с REF READ *CHAR*.

Определение

Считается, что вид M **совместим по чтению** с видом N (несимметричное отношение) тогда и только тогда, если M и N **эквивалентны** и, если N — **неизменяемый** вид, тогда M тоже должен быть **неизменяемым** видом, и, кроме того:

- если M и N — закрепленные ссылочные виды, то **ссылаемый** вид для M должен быть **l-эквивалентным** с **ссылаемым** видом для N;
- если M и N — рядовые виды, то **ссылаемый начальный** вид для M должен быть **l-эквивалентным** с **ссылаемым начальным** видом для N;

- если M и N — массивные виды, то вид **элемента** для M должен быть **совместимым по чтению** с видом **элемента** для N;

- если M и N — структурные виды, не являющиеся **параметризованными** структурными видами, то любой вид **поля** для M должен быть **совместимым по чтению** с соответствующим видом **поля** для N. Если M и N — **параметризованные** структурные виды, то **начальный вариантный** структурный вид для M должен быть **совместимым по чтению** с **начальным вариантным** структурным видом для N.

12.1.2.13 Отношения динамической эквивалентности и совместимости по чтению

Неформальное описание

Отношения 1) **динамический эквивалентный**, 2) **динамический, совместимый по чтению** уместны только для видов, которые могут быть динамическими, то есть для строковых, массивных и **вариантных** структурных видов. Считается, что **параметризованный** вид M является 1) **динамическим эквивалентным**, 2) **динамическим, совместимым по чтению** с (возможно, динамическим) видом N, если существует динамически параметризуемая версия для M, которая 1) **эквивалентна**, 2) **совместима по чтению** с N.

Определение

Вид M — 1) **динамический эквивалентный** для вида N, 2) **динамический, совместимый по чтению** с видом N (несимметричное отношение) тогда и только тогда, когда выполняется одно из следующих положений:

- M и N — такие строковые виды, что M(p) — 1) **эквивалентен**, 2) **совместим по чтению** с N, где p — (возможно, динамическая) длина N. Значение p не должно быть больше **длины строки** в M. Данная проверка — динамическая, если N — динамический вид. Ошибка при проверке приводит к исключению *RANGEFAIL*;

- M и N — такие массивные виды, что M(p) 1) **эквивалентен**, 2) **совместим по чтению** с N, где p таково, что *NUM*(p)—*LOWER*(M) + 1 является (возможно, динамическим) **количеством элементов** в N. Значение p не должно быть больше **верхней границы** для M. Эта проверка является динамической, если N — динамический вид. Ошибка при проверке приводит к исключению *RANGEFAIL*;

- M — это **параметризованный вариантный** структурный вид, а N — такой **параметризованный** структурный вид, что M(p_1,...,p_n) 1) **эквивалентен**, 2) **совместим по чтению** с N, где p_1,...,p_n обозначают список значений из N.

12.1.2.14 Отношение ограничения

Неформальное описание

Отношение **ограничения** уместно для **эквивалентных** видов со **свойством ссылаемости**. Считается, что вид M — **ограничиваемый** до вида N, если он или его возможные (суб)компоненты ссылаются на ячейки с одинаковой или более точной спецификацией **неизменяемости**, чем ячейки, ссылаемые видом N. Таким образом, это отношение несимметрично.

Примеры:

REF READ *INT* **ограничиваем** до **REF** *INT*

STRUCT (*P* **REF READ** *BOOL*) **ограничиваем** до **STRUCT** (*Q* **REF** *BOOL*)

Определение

Вид M **ограничиваем** до вида N (несимметричное отношение) тогда и только тогда, когда M и N — **эквивалентны**, и кроме того, если M и N:

- закрепленные ссылочные виды, то **ссылаемый** вид для M должен быть **совместимым по чтению** с **ссылаемым** видом для N;

- рядовые виды, то **ссылаемый начальный** вид для M должен быть **совместимым по чтению** с **ссылаемым начальным** видом для N;

- массивные виды, то вид **элемента** для M должен быть **ограничиваемым** до вида **элемента** для N;

- структурные виды, то каждый вид **поля** для M должен быть **ограничиваемым** до соответствующего вида **поля** для N.

12.1.2.15 Совместимость вида и класса

- Любой вид M **совместим** с **полным** классом.
- Вид M **совместим** с **нулевым** классом тогда и только тогда, когда M — **ссылочный вид**, процедурный вид или экземплярный вид;
- Вид M **совместим** с N-ссылочным классом тогда и только тогда, когда он является ссылочным видом и выполняется одно из следующих условий:
    1. N — статический вид, а M — закрепленный ссылочный вид, **ссылаемый** вид которого **совместим по чтению** с N;
    2. N — статический вид, а M — свободный ссылочный вид;
    3. M — рядовый вид, **ссылаемый начальный** вид которого является **динамическим, совместимым по чтению** с N.
- Вид M **совместим** с N-производным классом тогда и только тогда, когда M и N **подобны**.
- Вид M **совместим** с классом N-значения тогда и только тогда, когда выполняется одно из следующих условий:
    1. если M не обладает **свойством ссылаемости**, то M и N должны быть **v-эквивалентны**;
    2. если M не обладает **свойством ссылаемости**, то M должен быть **ограничиваемым** до N.

12.1.2.16 Совместимость между классами

- Любой класс **совместим** сам с собой.
- **Полный** класс **совместим** с любым другим классом.
- **Нулевой** класс **совместим** с любым M-ссылочным классом.
- **Нулевой** класс **совместим** с M-производным классом или классом M-значения тогда и только тогда, когда M — ссылочный, процедурный или экземплярный вид.
- M-ссылочный класс **совместим** с N-ссылочным классом тогда и только тогда, когда M и N **эквивалентны**. Если M и/или N является(ются) динамическим видом, то динамическая часть прооверки на эквивалентность не принимается во внимание, то есть исключений быть не может.
- M-ссылочный класс **совместим** с классом N-значения тогда и только тогда, когда N — ссылочный вид и выполняется одно из следующих условий:
    1. M — статический вид, а N — закрепленный ссылочный вид, **ссылаемый** вид которого **эквивалентен** M;
    2. M — статический вид, а N — свободный ссылочный вид;
    3. N — рядовый вид, **ссылаемый начальный** вид которого является **динамическим эквивалентным** с M.
- M-производный класс **совместим** с N-производным классом или классом N-значения тогда и только тогда, когда M и N **подобны**.
- Класс M-значения **совместим** с классом N-значения тогда и только тогда, когда M и N — **v-эквивалентны**.

Два списка классов **совместимы** тогда и только тогда, когда оба списка имеют одно и то же количество классов, а соответствующие (по позиции) классы **совместимы**.

12.2 ВИДИМОСТЬ И СВЯЗЫВАНИЕ ИМЕН

Определение видимости и связывание имен основывается на следующей терминологии:

- *строка имени*: обозначает терминальную строку с приписанной строкой **канонического** имени (см. раздел 2.7) и свойствами видимости;
- имя: обозначает *строку простого имени*, связанную с *определяющим вхождением*, которое ее создало (см. раздел 10.1);
- *имя*: обозначает вхождение с использованием имени (возможно, со строкой префиксного имени).

12.2.1 Степени видимости

Правила связывания основаны на видимости *строк имен* в областях программы. Каждая *строка имени* имеет в области видимости одну из следующих четырех степеней:

Видимость	Свойства (неформальные)
явно сильно видима	*Строка имени* **видима** в силу создания, разрешения, занятия или наследования от спецификации к телу
неявно сильно видима	*Строка имени* предопределена или наследуется в силу вложенности блоков
слабо видима	*Строка имени* **подразумевается сильно видимой** *строкой имени*
невидима	*Строка имени* не может использоваться

Таблица 1. Степени видимости

Считается, что *строка имени* **сильно видима** в области, если она **явно сильно видима** или **неявно сильно видима** в этой области. Считается, что *строка имени* **видима**, если она **видима слабо** или **сильно** в этой области. В противном случае считается, что *строка имени* **невидима** в данной области. Операторы структурирования программы и операторы видимости однозначно определяют, к какому классу видимости принадлежит каждая *строка имени*.

Когда *строка имени* **видима** в области, она может быть **непосредственно связана** со *строкой* другого *имени* в другой области, или **непосредственно связана** с *определяющим вхождением* в программе. Правила **непосредственной связи** приведены в разделе 12.2.3. Отметим, что любое использование правила вводит новую **непосредственную связь** для *строки имени*.

Понятие **связи** (не обязательно **непосредственной**) определяется на основе **непосредственной связи** следующим образом:

Считается, что *строка имени* N_1, **видимая** в области R_1, **связана** со *строкой имени* N_2 в области R_2 или с *определяющим вхождением* D тогда и только тогда, когда выполняется одно из следующих условий:

- N_1 в R_1 **непосредственно связана** с N_2 в R_2 или с D. Однако, если N_1 **непосредственно связана** с несколькими *определяющими вхождениями* в R_1, тогда все эти *определяющие вхождения*, кроме одного, являются лишними, а N_1 **связывается** с любым одним из них в R_1.

- N_1 в R_1 **непосредственно связана** с некоторой N в некоторой R, а N в R **связана** с N_2 в R_2 или с D.

12.2.2 Условия видимости и связывание имен

В каждой области программы должны удовлетворяться следующие условия:

- Если *строка имени* **сильно видима** в области и имеет несколько **непосредственных связей**, тогда:

  — она должна быть **непосредственно связана** только с *определяющими вхождениями*, а эти *определяющие вхождения* должны определять одни и те же элементы перечисления перечислимых видов, которые **подобны**, или

  — она должна быть **связана** точно с одним **действительным** *определяющим вхождением* и одним **квазиопределяющим** *вхождением*.

Если *строка имени* **слабо видима** в области и **связана** как **слабо видимая** *строка имени* в этой области с *определяющими вхождениями*, которые не определяют один и тот же элемент перечисления **подобных** перечислимых видов, то считается, что строка имени имеет **слабый конфликт** в этой области.

Считается, что *строка имени* NS, **видимая** в области R, **связана** в R с несколькими *определяющими вхождениями*, если выполняются следующие правила:

- если NS **сильно видима** в R, то NS **связана** с *определяющими вхождениями*, с которыми она **связана** в R (как **сильно видимая** *строка имени*). Если она **связана** и с **квазиопределяющим** *вхождением*, и с **действительным** *определяющим вхождением*, тогда **квазиопределяющее** *вхождение* является лишним и далее не участвует в видимости и связывании имен (то есть оно не занимается, не разрешается, не наследуется и не вводит **неявных** имен);

- иначе, если NS **слабо видима** в R, то она **связана** с *определяющими вхождениями*, с которыми она **связана** в R (как **слабо видимая** строка имени) при условии, что NS не имеет **слабого конфликта** в R (**слабые конфликты** допустимы в области, если в области не существует *имени* со *строкой имени*, имеющей **слабый конфликт**);

- в противном случае NS не **связана** в R.

Статические условия: *Строка имени*, приписанная за каждым *именем*, непосредственно заключенным в область, должна быть **связанной** в этой области.

Связывание имен: *Имя* N с приписанной *строкой имени* NS в области R **связано** с *определяющими вхождениями*, с которыми NS **связана** в R.

12.2.3 Видимость в областях

12.2.3.1 Общие положения

Строка имени является **явно сильно видимой** в области согласно следующим правилам:

- *строка имени* занимается в области (см. раздел 12.2.3.5);

- *строка имени* разрешается в области (см. раздел 12.2.3.4);

- имеется *определяющее вхождение* с этой *строкой имени* в области. В этом случае *строка имени* в области **непосредственно связана** с *определяющим вхождением*. (Заметим, что *строка имени* может быть **непосредственно связана с несколькими** *определяющими вхождениями* в области.)

- область — это 1) *тело модуля*, 2) *тело региона*, а *строка имени* **явно сильно видима** в области **соответствующего** 1) *модуля спецификации*, 2) *региона спецификации*. *Строка имени* **непосредственно связана** со *строкой имени* в соответствующей области.

Строка имени, которая не является **явно сильно видимой** в области, является **неявно сильно видимой** в ней согласно следующим правилам:

- область — это блок, и строка имени **сильно видима** в непосредственно объемлющей области. Считается, что *строка имени* наследуется блоком, и она **непосредственно связана** с той же *строкой имени* в непосредственно объемлющей области;

- область — это неблок, в котором наследуется *строка имени*, и *строка имени* является *строкой имени*, определяемой в языке (см. добавление C.2) или в реализации. Считается, что *строка имени* **непосредственно связана** с *определяющим вхождением* в области определения воображаемого внешнего процесса при ее предопределенном значении.

Строка имени, не являющаяся **сильно видимой** в области, **слабо видима** в ней, если она **неявно** задана *строкой имени*, **явно видимой в области**. *Строка имени* в области **непосредственно связана** с **неявным** *определяющим вхождением* (см. раздел 12.2.4).

12.2.3.2 Операторы видимости

Синтаксис:

< оператор видимости > :: = (1)
< оператор разрешения > (1.1)
| *< оператор занятия >* (1.2)

Семантика: Операторы видимости допустимы только в областях модульона и при управлении видимостью *строк имен*, упомянутых в областях, и косвенно их **неявными** *строками имен*.

Статические свойства: *Оператор видимости* имеет одну или две **исходные** области (см. раздел 10.2) и одну или две **приписанные** области назначения, определяемые следующим образом:

- Если *оператор видимости* — это *оператор занятия*, то его область **назначения** — это область, непосредственно объемлющая *оператор занятия*, а его **исходные** области — это области, непосредственно объемлющие данную область.

- Если *оператор видимости* — это *оператор разрешения*, то его **исходная** область — это область, непосредственно объемлющая *оператор разрешения*, а его области **назначения** — это области, непосредственно объемлющие данную область.

12.2.3.3 Предложение переименования префикса

Синтаксис:

< предложение переименования префикса > :: = (1)
(< старый префикс > —> < новый префикс >) ! < постфикс > (1.1)

< старый префикс > :: = (2)
< префикс > (2.1)
| *< пусто >* (2.2)

< новый префикс > :: = (3)
< префикс > (3.1)
| *< пусто >* (3.2)

< постфикс > :: = (4)
< постфикс оператора занятия > { , < постфикс оператора занятия > }* (4.1)
| *< постфикс оператора разрешения > { , < постфикс оператора разрешения > }** (4.2)

Производный синтаксис: *Предложение переименования префикса*, где *постфикс* состоит из нескольких *постфиксов оператора занятия* (*оператора разрешения*), является производным синтаксисом для нескольких предложений переименования префикса — по одному для каждого *постфикса оператора занятия* (*оператора разрешения*), — отделенных запятыми, с тем же *старым префиксом* и *новым префиксом*.

Например:

GRANT *(p —>q) ! a, b;*

является производным синтаксисом для

GRANT *(p —>q) ! a, (p —>q) ! b;*

Семантика: Предложения переименования префикса используются в операторах видимости для выражения изменения префикса в префиксных строках имени, которые разрешены или заняты. (Поскольку предложения переименования префикса могут быть использованы без изменений префикса — когда *старый префикс* и *новый префикс* пусты, — поэтому они могут быть взяты в качестве семантической базы для операторов видимости.)

Статические свойства: *Предложения переименования префикса* имеет одну или две присоединенные **исходные** области, являющиеся **исходными** областями *оператора видимости*, в который оно записано.

Предложение переименования префикса имеет одну или две **приписанные** области назначения, которые являются областями **назначения** *оператора видимости*, в которых оно записано.

Постфикс имеет набор присоединенных *строк имен*, присоединенных к его *постфиксу оператора занятия* или присоединенных к его *постфиксу оператора разрешения*. Эти *строки имен* являются постфиксными *строками имен предложения переименования префикса*.

Предложение переименования префикса имеет набор **старых** *строк имен* и набор **новых** *строк имен*. Каждая постфиксная *строка имени*, присоединенная к *предложению переименования префикса*, дает как **старую** *строку имени*, так и **новую** строку имени, присоединенную к *предложению переименования префикса*, следующим образом: **новая** *строка имени* получается в результате снабжения постфиксной *строки имени новым префиксом*; **старая** *строка имени* получается в результате снабжения постфиксной *строки имени старым префиксом*.

Когда **новая** *строка имени* и **старая** *строка имени* получены из одной и той же постфиксной *строки имени*, то считают, что **старая** *строка имени* является источником **новой** *строки имени*.

Правила видимости: **Новые** *строки имен*, присоединенные к *предложению переименования префикса*, **сильно видимы** в их областях **назначения** и **непосредственно связаны** в этих областях с их источниками в **исходных** областях. Если *предложение переименования префикса* является частью *оператора занятия* (*разрешения*), то эти *строки имен* занимаются (разрешаются) в их области(ях) **назначения.**

Считается, что *строка имени* NS **занимаема** модульоном M, непосредственно заключенным в области R, тогда и только тогда, когда она **сильно видима** в R, не **связана** в R ни с какой *строкой имени* в области R модульона M, и не **связана непосредственно** с *определяющим вхождением строки* предопределенного *имени*.

Считается, что *строка имени* NS **разрешаема** модульоном M, непосредственно включенным в область R, тогда и только тогда, когда она **сильно видима** в области модульона M, не **связана** в нем ни с какой *строкой имени* в R, не **связана непосредственно** в нем с *определяющим вхождением строки* предопределенного *имени*.

Статические условия: Если *предложение переименования префикса* находится в *операторе занятия*, непосредственно объемлемом в области модульона M, тогда каждая из его **старых** *строк имен* должна быть:

- **связанной** в области, непосредственно объемлющей область модульона M, и
- **занимаемой** модульоном M.

Если *предложение переименования префикса* находится в *операторе разрешения*, непосредственно объемлемым в области модульона M, тогда каждая из его старых *строк имен* должна быть:

- **связанной** в области модульона M и
- **разрешаемой** модульоном M.

Предложение переименования префикса, встречающееся в *операторе занятия* (*разрешения*), должно иметь *постфикс*, то есть *постфикс оператора занятия* (*разрешения*).

Примеры:

25.35 *(stack ! int —>stack) !* **ALL** *(1.1)*

12.2.3.4 Оператор разрешения

Синтаксис:

< *оператор разрешения* > :: = *(1)*

GRANT < *предложение переименования префикса* >

{ , < *предложение переименования префикса* > }* ; *(1.1)*

| **GRANT** < *окно оператора разрешения* > [< *префиксное предложение* >] ; *(1.2)*

< *окно оператора разрешения* > :: = (2)

< *постфикс оператора разрешения* > { , < *постфикс оператора разрешения* > }* (2.1)

< *постфикс оператора разрешения* > :: = (3)

< *строка имени* > (3.1)

| < *строка имени нового вида* > < *предложение запрета* > (3.2)

| [< *префикс* > !] **ALL** (3.3)

< *префиксное предложение* > :: = (4)

PREFIXED [< *префикс* >] (4.1)

< *предложение запрета* > :: = (5)

FORBID {< *список запрещенных имен* > | **ALL**} (5.1)

< *список запрещенных имен* > :: = (6)

(< *имя поля* > { , < *имя поля* > }*) (6.1)

Семантика: Операторы разрешения являются средством распространения видимости строк имен в области модульона на непосредственно объемлющие области. **FORBID** может быть описан только для имен **нового вида**, являющихся структурными видами. Это означает, что все ячейки и значения данного вида имеют поля, выбираемые только внутри разрешающего модульона, а не вне его.

Действуют следующие правила видимости:

- Если *оператор разрешения* содержит *предложение(я) переименования префикса*, то результатом он имеет действие его *предложения(ий) переименования префикса* (см. раздел 12.2.3.3).

- Если *оператор разрешения* содержит *окно оператора разрешения*, то это — сокращенное обозначение набора *операторов разрешения* с *предложениями переименования префикса*, образованными следующим образом:

  — для каждого *постфикса оператора разрешения* в *окне оператора разрешения* имеется соответствующий *оператор разрешения*;

  — *старый префикс* в их *предложении переименования префикса* пуст;

  — *новый префикс* в их *предложении переименования префикса* является *префиксом*, присоединенным к *префиксному предложению* в *операторе разрешения*, или же он пуст, если в исходном *операторе разрешения* нет *префиксного предложения*;

  — *постфикс* в *предложении переименования префикса* — это соответствующий *постфикс* в *окне оператора разрешения*.

- Обозначения **FORBID ALL** — это сокращенное обозначение запрета всех *имен полей* имени **нового вида** (см. раздел 12.2.5).

- Если *предложение переименования префикса* в *операторе разрешения* содержит *постфикс оператора разрешения*, содежащий в свою очередь *префикс* и **ALL**, тогда оно имеет вид:

  (*OP* —>*NP*) ! *P* ! **ALL**,

  где *OP* и *NP* — возможно, пустые *старый префикс* и *новый префикс*, соответственно, *а* P — *префикс* в *постфиксе оператора разрешения*. Тогда *предложение переименования префикса* является сокращенным обозначением предложения вида:

  (*OP* ! *P* —> *NP* ! *P*) ! **ALL**.

Статические свойства: *Префиксное предложение* имеет присоединенный *префикс*, определяемый следующим образом:

- Если *префиксное предложение* содержит *префикс*, тогда этот *префикс* присоединяется.

- В противном случае присоединенный *префикс* — это *простой префикс*, *строка имени* которого определяется следующим образом:

  — если область, непосредственно объемлющая *префикс*, является *модулем* или *регионом*, тогда *строка имени* является такой же, как *строка имени* для имени модульона этого модульона;

  — если область, непосредственно объемлющая *префикс*, является *регионом спецификации* или *модулем спецификации*, тогда строка имени — это строка имени перед **SPEC**.

Постфикс оператора разрешения имеет набор присоединенных *строк имен*, определяемых следующим образом:

- если *строка имени* или набор содержит *строку имени нового вида*, тогда набор содержит только одну *строку имени*;
- в противном случае, пусть *OP* — (возможно, пустой) *старый префикс предложения переименования префикса*, в котором находится *постфикс оператора разрешения*, тогда набор содержит все *строки имен* вида *OP* ! *N* (то есть полученные в результате снабжения *N* префиксом *OP*) для любой такой *строки имени* N, что *OP* ! *N* **сильно видима** в области модульона, в котором находится *оператор разрешения*, и разрешаема этим модульоном.

Статические условия: *Строка имени нового вида* с предложением запрета должна быть **сильно видима** в области R модульона, в которой находится *оператор разрешения*. *Строка имени нового вида* должна быть **связана** в R с *определяющим вхождением* нового вида, который должен быть структурным видом, а каждое *имя поля* в *списке имен полей* должно быть *именем поля* этого вида. *Определяющее вхождение* нового вида должно быть непосредственно объемлемо в R. Все имена **полей** в *списке запрещенных имен* должны иметь различные строки имен.

Если *оператор разрешения* находится в области *региона* или *региона спецификации*, он не должен разрешать *строку имени*, **связанную** в этой области с *определяющим вхождением*:

- имени **ячейки**, или
- имени **опознавателя-ячейки**, где *ячейка* в его описании является **внутрирегиональной**, или
- имени **синонима**, *значение* которого является **внутрирегиональным**.

Предложение переименования префикса в *операторе разрешения* должно иметь *постфикс оператора разрешения*.

Если *оператор разрешения* содержит *префиксное предложение*, не содержащее *префикса*, то его непосредственно объемлющий модульон не должен быть *контекстом* и,

- если его непосредственно объемлющий модульон — это *модуль* или *регион*, тогда оно должно быть именованным (то есть оно должно начинаться с *определяющего вхождения*, за которым следует двоеточие);
- если его непосредственно объемлющий модульон — это *модуль спецификации* или *регион спецификации*, то оно должно начинаться со *строки простого имени*.

Примеры:

25.7 **GRANT** *(—> stack ! char) !* **ALL**; *(1.1)*

6.44 *gregorian_date, julian_day_number* *(2.1)*

12.2.3.5 Оператор занятия

Синтаксис:

< *оператор занятия* > :: = *(1)*
SEIZE < *предложение переименования префикса* >
{ , < *предложение переименования префикса* > }*; *(1.1)*
| SEIZE < *окно оператора занятия* > [< *префиксное предложение* >] ; *(1.2)*

< *окно оператора занятия* > :: = *(2)*
< *постфикс оператора занятия* > { , < *постфикс оператора занятия* > }* *(2.1)*

< *постфикс оператора занятия* > :: = *(3)*
< *строка имени* > *(3.1)*
| [< *префикс* >/] **ALL** *(3.2)*

Семантика: Операторы занятия — это средство распространения видимости строк имен в группе областей на области непосредственно объемлемых модульонов.

Действуют следующие правила видимости:

- Если *оператор занятия* содержит *предложение(ия) переименования префикса*, то результатом он имеет действие его *предложения(ий) переименования префикса* (см. раздел 12.2.3.3).
- Если *оператор занятия* содержит *окно оператора занятия*, то это — сокращенное обозначение набора *операторов занятия* с *предложениями переименования префикса*, образованными следующим образом:
  - для каждого *постфикса оператора занятия* в *окне оператора занятия* имеется соответствующий *оператор занятия*;
  - *старый префикс* в их *предложении переименования префикса* является *префиксом*, присоединенным к *префиксному предложению* в *операторе занятия*, или же он пуст, если в исходном *операторе занятия* нет *префиксного предложения*;
  - *новый префикс* в их *предложении переименования префикса* пуст;
  - *постфикс* в их *предложении переименования префикса* — это соответствующий *постфикс окна оператора занятия*.
- Если *предложение переименования префикса* в *операторе занятия* имеет *постфикс оператора занятия*, содержащий *префикс* и **ALL**, тогда оно имеет вид:

  (*OP* —>*NP*) ! *P* ! **ALL**,

  где *OP* и *NP* — возможно, пустые *старый* и *новый префиксы* соответственно, а *P* — *префикс* в *постфиксе оператора занятия*. Тогда *предложение переименования префикса* является сокращенным обозначением предложения вида:

  (*OP* ! *P* —>*NP* ! *P*) ! **ALL**

Статические свойства: *Постфикс оператора занятия* имеет набор присоединенных *строк имен*, определяемых следующим образом:

- Если *постфикс оператора занятия* — это *строка имени*, то набор содержит только *строку имени*.
- Иначе, если *постфикс оператора занятия* есть **ALL**, а *OP* — (возможно, пустой) *старый префикс предложения переименования префикса*, частью которого является *постфикс оператора занятия*, тогда набор содержит все *строки имен* вида *OP ! S* для любой такой строки имени *S*, что *OP ! S* **сильно видима** в области, непосредственно объемлющей модульон, в котором находится *оператор занятия*, и **занимаема** этим модульоном.

Статические условия: *Предложение переименования префикса* в *операторе занятия* должно иметь *постфикс оператора занятия*.

Если *оператор занятия* содержит *префиксное предложение*, не содержащее *префикса*, тогда его непосредственно объемлющий модульон не должен быть *контекстом*, а также:

- если его непосредственно объемлющий модульон — это *модуль* или *регион*, тогда оно должно быть именованным (то есть должно начинаться с *определяющего вхождения*, за которым следует двоеточие);
- если его непосредственно объемлющий модульон — это *модуль спецификации* или *регион спецификации*, тогда оно должно начинаться *строкой простого имени*.

Примеры:

25.35 **SEIZE** *(stack ! int —> stack) !* **ALL**; *(1.1)*

12.2.4 Строки неявных имен

Каждая *строка имени*, **сильно видимая** в области R, имеет набор **строк неявных** *имен*, которые могут быть **слабо видимы** в R.

Каждый *вид* имеет, возможно, пустой набор присоединенных **неявных** *определяющих вхождений* в область, как указано в таблице 2.

Каждая *строка имени* NS, **сильно видимая** в области R, имеет набор **неявных** *определяющих вхождений*, определяемых ниже, где D — одно из *определяющих вхождений*, с которым **связана** NS в R:

- Если D определяет имя **доступа** вида M, тогда **неявные** *определяющие вхождения* для *NS* в R — это *определяющие вхождения*, **неявно** задаваемые видом M в R;
- Если D определяет имя **вида**, то **неявные** *определяющие вхождения* для *NS* в R — это *определяющие вхождения*, **неявно** заданные в R определяющим видом имени **вида**.

- Если D определяет имя **процедуры**, то **неявные** *определяющие вхождения* для NS в R — это *определяющие вхождения*, **неявно** заданные в R видами **спецификаций параметров** и **спецификацией результата**, если имеется.

- Если D определяет имя **процесса**, то **неявные** *определяющие вхождения* для NS в R — это *определяющие вхождения*, **неявно** заданные в R видами **спецификаций параметров**, если таковые имеются.

- Если D определяет имя **сигнала**, то **неявные** *определяющие вхождения* NS в R — это все *определяющие вхождения*, **неявно** заданные в R всеми видами, приписанными к сигналу.

- В противном случае набор пуст.

Виды	**Набор неявных определяющих вхождений**
INT, BOOL, CHAR, **RANGE** (...) **BIN** *(n), PTR, INSTANCE,* **EVENT,** *ASSOCIATION, TIME, DURATION,* **BOOLS** *(n),* **CHARS** *(n)*	Пусто
имя вида	Набор *определяющих вхождений*, **неявно** заданных в R его **определяющим** видом.
имя вида (...) (параметризованное)	Набор *определяющих вхождений*, **неявно** заданных в R *именем вида*.
M(*m:n*), **REF** *M*, **ROW** *M*, **READ** *M* **POWERSET** *M*, **BUFFER** *M*, **TEXT** (...) *M*	Набор *определяющих вхождений*, заданных **неявно** с помощью *M* в R.
SET *(...)*	Набор *определяющих вхождений* **элементов перечисления** в виде.
PROC (M_1, ..., M_n) (M_n+1)	Объединение наборов *определяющих вхождений*, **неявно** заданных в R с помощью M_1—M_{n+1}.
ARRAY (*M*) *N*, **ACCESS** (*M*) *N*	Объединение наборов *определяющих вхождений*, **неявно** заданных в R с помощью *M* и *N*.
STRUCT (N_1 M_1 ,..., N_n M_n)	Объединение наборов *определяющих вхождений*, **неявно** заданных в R с помощью M_i для полей, **видимых** в R. Для вариантных структур это — объединение *определяющих вхождений*, **неявно** заданных в R полями вариантной структуры, **видимыми** в R.

Таблица 2. Неявные определяющие вхождения видов в область R

Если *строка имени* NS, **сильно видимая** в области R, имеет **неявные** *определяющие вхождения*; то каждое из них описывает *строку* **неявных** *имен* для NS в R: пусть D будет *определяющим вхождением*, **подразумеваемым** NS в R и пусть Ni — это *строка имени* для D. Имеются два случая:

- NS — *строка простого имени*. Тогда Ni — *строка* **неявного** *имени* для NS.

- NS имеет вид *P ! S*, где *S* — *строка простого имени*. Тогда *P ! Ni* — *строка* **неявного** *имени* для NS.

Примеры:

```
m: MODULE
        DCL x SET (on, off);
        GRANT x PREFIXED;
END;

/*m ! x видимо здесь с неявным m ! on, m ! off*/
```

12.2.5 Видимость имен полей

Следующие контексты могут содержать *имена полей*:

- *поля структур* и *поля структур значений*;
- *кортежи помеченной структуры*;
- *предложения запрета* в *операторах разрешения*.

В каждом из этих случаев *строка имени имени поля* может быть **связана** с *определяющим вхождением имени поля* в виде M или в определяющем виде M, которые получаются следующим образом:

- M — это вид *ячейки структуры* или (**строгого**) *примитивного значения структуры*;
- M — это вид *кортежа структуры*;
- M — это вид *определяющего вхождения*, с которым **связана** *строка имени нового вида* в области, в которой находится *предложение запрета*.

Однако, если **новизна** вида M является *определяющим вхождением*, устанавливающим имя **нового вида**, которое разрешено *оператором разрешения* в модульоне как *постфикс оператора разрешения* с *предложением запрета*, тогда имена полей, упомянутые в списке запрещенных имен, являются **видимыми** только:

- в группе разрешающего модульона;
- в группе области, в которой **квазиновизна** N непосредственно объемлема, если **новизна** вида M **связана новизной** с **квазиновизной** N;
- в области **соответствующего** модульона, если модульон — это *спецификация модуля* или *спецификация региона*.

Вне этих областей *имена полей*, упомянутые в *списке запрещенных имен*, **невидимы** и использоваться не могут.

12.3 ВЫБОР ВАРИАНТА

Синтаксис:

```
< спецификация метки выбора > :: =                                   (1)
        < список меток выбора > { , < список меток выбора > }*       (1.1)

< список меток выбора > :: =                                         (2)
        (< метка выбора > { , < метка выбора > }*)                   (2.1)
      | < несоответствующая >                                        (2.2)

< метка выбора > :: =                                                (3)
        < дискретное постоянное выражение >                          (3.1)
      | < постоянный диапазон >                                      (3.2)
      | < имя дискретного вида >                                     (3.3)
      | ELSE                                                         (3.4)

< несоответствующая > :: =                                           (4)
        (*)                                                          (4.1)
```

Семантика: Выбор варианта — это средство выбора варианта из списка вариантов. Выбор осуществляется согласно определенному списку значений селекторов. Выбор варианта может применяться к:

- альтернативным полям (см. раздел 3.12.4), в этом случае выбирается список вариантных полей;
- кортежам помеченных массивов (см. раздел 5.2.5), в этом случае выбирается значение элемента массива;
- условным выражениям (см. раздел 5.3.2), в этом случае выбирается выражение;
- оператору выбора (см. раздел 6.4), в этом случае выбирается список операторов действия.

В первом, третьем и четвертом случаях каждый вариант помечается спецификацией метки выбора; в кортеже помеченного массива каждое значение помечается списком меток выбора. Для облегчения описания список меток выбора в кортеже помеченного массива будет рассматриваться в этом разделе в качестве спецификации метки выбора только с одним вхождением списка меток выбора.

При выборе варианта выбирается та альтернатива, которая помечена спецификацией меток выбора, соответствующей списку значений селектора. (Количество значений селектора всегда будет тем же, что и количество вхождений списка меток выбора в спецификации меток выбора.) Считается, что список значений совпадает со спецификацией меток выбора тогда и только тогда, когда каждое значение совпадает с соответствующим списком меток выбора (по позиции) в спецификации меток выбора.

Считается, что значение совпадает со списком меток выбора тогда и только тогда, когда:

- список меток выбора состоит из меток выбора, а значение — это одно из значений, явно указанное одной из меток выбора или неявно указанное в случае атрибута ELSE;
- список меток выбора содержит *несоответствующую* метку.

Значения, явно указанные меткой выбора, являются значениями, вырабатываемыми любым *дискретным постоянным выражением* или определяемыми *постоянным диапазоном* или *именем дискретного вида*. Все значения, неявно указанные атрибутом ELSE, являются возможными значениями селектора, которые явно не указываются любым ассоциативным списком меток выбора (то есть принадлежащим тому же значению селектора) в любой спецификации меток выбора.

Статические свойства:

- *Альтернативные поля* со *спецификацией меток выбора*, *кортеж помеченного массива*, *условное выражение* или *оператор выбора* имеют список присоединенных спецификаций меток выбора, образованный выбором *спецификации меток выбора* перед каждой *вариантной альтернативой*, *значением* или *альтернативой выбора* соответственно.
- *Метка выбора* имеет присоединенный класс, который, если это *дискретное постоянное выражение*, является классом *дискретного постоянного выражения*; если это *постоянный диапазон*, то присоединенный класс является **результирующим классом** из классов каждого *дискретного постоянного выражения* в *постоянном диапазоне*; если это *имя дискретного вида*, то присоединенный класс является **результирующим классом** класса М-значения, где М — *имя дискретного вида*; если это атрибут ELSE, тогда присоединенный класс является **полным** классом.
- *Список меток выбора* имеет присоединенный класс, который является **полным** классом, если список *несоответствующий*, в противном случае присоединенный класс — это **результирующий класс** из классов каждой *метки выбора*.
- *Спецификация меток выбора* имеет список присоединенных классов, которые являются классами списков меток выбора.
- Список спецификаций меток выбора имеет присоединенный **результирующий список классов**. Этот **результирующий список классов** образуется конструкцией для каждой позиции в списке **результирующего класса** для всех классов с данной позицией.

Список спецификаций меток выбора является **полным** тогда и только тогда, если для всех списков возможных значений селектора имеется спецификация меток выбора, совпадающая со списком значений селектора. Набор всех возможных значений селектора определяется контекстом следующим образом:

- для **тегового вариантного** структурного вида — это набор значений, определяемых видом соответствующего поля **признака**;
- для **беспризнакового вариантного** структурного вида — это набор значений, определяемых **корневым** видом соответствующего **результирующего класса** (этот класс не является **полным** классом; см. раздел 3.12.4);
- для кортежа массива — это набор значений, определяемых **индексным** видом вида кортежа массива;
- для оператора выбора со списком диапазонов — это набор значений, определяемых соответствующим дискретным видом в списке диапазонов;
- для оператора выбора без списка диапазонов или условного выражения — это набор значений, определяемых М, где класс соответствующего селектора — это класс М-значения или М-производный класс.

Статические условия: Для каждой *спецификации меток выбора* количество вхождений *списка меток выбора* должно быть одинаковым.

Для любых двух вхождений *спецификации меток выбора* их списки классов должны быть **совместимыми**.

Список вхождений *спецификации меток выбора* должен быть **непротиворечивым**, то есть каждый список возможных значений селектора совпадает самое большее с одной спецификацией меток выбора.

Примеры:

11.9	*(occupied)*	*(2.1)*
11.58	*(rook),(*)*	*(1.1)*
8.26	*(***ELSE***)*	*(2.1)*

12.4 ОПРЕДЕЛЕНИЕ И ПЕРЕЧЕНЬ СЕМАНТИЧЕСКИХ КАТЕГОРИЙ

В данном разделе приведен перечень всех семантических категорий, выделенных в описании синтаксиса подчеркнутой частью. Если эти категории не определены в соответствующих разделах, то они определяются в настоящем разделе; в остальных случаях на соответствующие разделы имеются ссылки.

12.4.1 Имена

Имена видов:

имя абсолютно-временного вида:	*имя* определено как абсолютно-временной вид;
имя доступного вида:	*имя* определено как доступный вид;
имя массивного вида:	*имя* определено как массивный вид;
имя ассоциативного вида:	*имя* определено как ассоциативный вид;
имя булевского вида:	*имя* определено как булевский вид;
имя закрепленного ссылочного вида:	*имя* определено как закрепленный ссылочный вид;
имя буферного вида:	*имя* определено как буферный вид;
имя символьного вида:	*имя* определено как символьный вид;
имя дискретного вида:	*имя* определено как дискретный вид;
имя продолжительного вида:	*имя* определено как продолжительный вид;
имя событийного вида:	*имя* определено как событийный вид;
имя свободного ссылочного вида:	*имя* определено как свободный ссылочный вид;
имя экземплярного вида:	*имя* определено как экземплярный вид;
имя целого вида:	*имя* определено как целый вид;
имя вида:	см. раздел 3.2.1;
имя нового вида:	см раздел 3.2.3;
имя параметризованного массивного вида:	*имя* определено как **параметризованный** массивный вид;
имя параметризованного строкового вида:	*имя* определено как **параметризованный** строковой вид;
имя параметризованного структурного вида:	*имя* определено как **параметризованный** структурный вид;
имя множественного вида:	*имя* определено как множественный вид;
имя процедурного вида:	*имя* определено как процедурный вид;
имя ограниченного вида:	*имя* определено как ограниченный вид;
имя рядового вида:	*имя* определено как рядовой вид;
имя перечислимого вида:	*имя* определено как перечислимый вид;
имя строкового вида:	*имя* определено как строковой вид;
имя структурного вида:	*имя* определено как структурный вид;
имя синонимического вида:	см. раздел 3.2.2;
имя вариантного структурного вида:	*имя* определено как **вариантный** структурный вид.

Имена доступа:

имя ячейки:	см. разделы 4.1.2;
имя ячейки с присоединением:	см. раздел 6.5.4;
имя перечисления ячеек:	см. раздел 6.5.2;
имя опознавателя-ячейки:	см. разделы 4.1.3.

Имена значений:

имя булевской константы:	см. раздел 5.2.4.3;
имя пустой константы:	см. раздел 5.2.4.6;
имя синонима:	см. раздел 5.1;
имя значения с присоединением:	см. раздел 6.5.4;
имя перечисления значений:	см. раздел 6.5.2;
имя принятого значения:	см. разделы 6.19.2, 6.19.3.

Смешанные имена:

имя ячейки закрепленной ссылки: *имя ячейки* с закрепленным ссылочным видом;

имя встроенной программы: любое имя в языке CHILL или имя, определяемое реализацией и обозначающее встроенную программу;

имя ячейки свободной ссылки: *имя ячейки* со свободным ссылочным видом;

имя общей процедуры: *имя процедуры*, универсальность которой **общая**;

имя метки: см. разделы 6.1, 10,6;

строка имени нового вида: *строка имени*, **связанная** с *определяющим вхождением* имени **нового вида**;

нерезервируемое имя: *имя*, не являющееся ни одним из зарезервированных имен добавления C.1;

имя процедуры: см. раздел 10.4;

имя процесса: см. раздел 10.5;

имя элемента перечисления: см. раздел 3.4.5;

имя сигнала: см. раздел 11.5;

имя поля признака: см. раздел 3.12.4;

имя неопределенного синонима: см. раздел 5.1.

12.4.2 Ячейки

ячейка доступа: *ячейка* доступного вида;

ячейка массива: *ячейка* массивного вида;

ассоциативная ячейка: *ячейка* ассоциативного вида;

ячейка символьной строки: *ячейка* **символьного** строкового вида;

буферная ячейка: *ячейка* буферного вида;

дискретная ячейка: *ячейка* дискретного вида;

ячейка события: *ячейка* событийного вида;

ячейка экземпляра: *ячейка* экземплярного вида;

ячейка статического вида: *ячейка* статического вида;

ячейка строки: *ячейка* строкового вида;

ячейка структуры: *ячейка* структурного вида;

ячейка текста: *ячейка* текстового вида.

12.4.3 Выражения и значения

примитивное значение абсолютного времени: *примитивное значение*, класс которого **совместим** с абсолютно-временным видом;

выражение массива: *выражение*, класс которого **совместим** с массивным видом;

примитивное значение массива: *примитивное значение*, класс которого **совместим** с массивным видом;

булевское выражение: *выражение*, класс которого **совместим** с булевским видом;

примитивное значение закрепленной ссылки: *примитивное значение*, класс которого **совместим** с закрепленным ссылочным видом;

выражение символьной строки: *выражение*, класс которого **совместим** с **символьным** строковым видом;

постоянное значение:
значение, являющееся **постоянной**;

дискретное выражение:
выражение, класс которого **совместим** с дискретным видом;

дискретное постоянное выражение:
дискретное выражение, являющееся **константой**;

примитивное значение продолжительности:
примитивное значение, класс которого **совместим** с продолжительным видом;

примитивное значение свободной ссылки:
примитивное значение, класс которого **совместим** со свободным ссылочным видом;

примитивное значение экземпляра:
примитивное значение, класс которого **совместим** с экземплярным видом;

целое выражение:
выражение, класс которого **совместим** с целым видом;

целое постоянное выражение:
целое выражение, являющееся **константой**;

множественное выражение:
выражение, класс которого **совместим** с множественным видом;

примитивное значение процедуры:
примитивное значение, класс которого **совместим** с процедурным видом;

примитивное значение ссылки:
примитивное значение, класс которого **совместим** либо с закрепленным ссылочным видом, либо со свободным ссылочным видом, либо с рядовым видом;

примитивное значение ряда:
примитивное значение, класс которого **совместим** с рядовым видом;

выражение строки:
выражение, класс которого **совместим** со строковым видом;

примитивное значение строки:
примитивное значение, класс которого **совместим** со строковым видом;

примитивное значение структуры:
примитивное значение, класс которого **совместим** со структурным видом.

12.4.4 Смешанные семантические категории:

массивный вид:
вид, в котором *составной* вид является *массивным* видом;

дискретный вид:
вид, в котором *несоставной* вид является *дискретным* видом;

вызов встроенной программы выдачи ячейки:
см. раздел 6.7;

вызов процедуры возврата ячейки:
см. раздел 6.7;

нерезервируемый символ:
символ, не являющийся ни кавычками ("), ни диакритическим знаком (^);

неспециальный символ:
символ, не являющийся ни диакритическим знаком (^), ни открывающей скобкой (();

строковой вид:
вид, в котором *составной* вид является *строковым* видом;

вызов встроенной программы выдачи значения:
см. раздел 6.7;

вызов процедуры возврата значения:
см.раздел 6.7.

13 ВАРИАНТЫ РЕАЛИЗАЦИЙ

13.1 ВСТРОЕННЫЕ ПРОГРАММЫ, ОПРЕДЕЛЯЕМЫЕ РЕАЛИЗАЦИЕЙ

Семантика: Реализация может обеспечивать ряд своих встроенных программ в дополнение к встроенным программам, определяемым в языке.

Механизм передачи параметров также определяется реализацией.

Предопределенные имена: Имя встроенной программы, определяемой реализацией, предопределено как имя **встроенной программы.**

Статические свойства: Имя **встроенной программы** может сопровождаться связанным с ним набором имен исключений, определяемых реализацией. *Вызов встроенной программы* есть *вызов встроенной программы выдачи* **значения (ячейки)** тогда и только тогда, когда в реализации задано, что в результате вызова встроенной программы выдается значение (ячейка) для данного набора статических свойств параметров и данного статического контекста вызова.

В реализации также описывается **региональность** значения (ячейки).

13.2 ЦЕЛЫЕ ВИДЫ, ОПРЕДЕЛЯЕМЫЕ РЕАЛИЗАЦИЕЙ

Реализация определяет **верхнюю границу** и **нижнюю границу** целого вида *INT*. Реализация может определять целые виды, отличающиеся от целых видов, определяемых *INT*; например, короткие целые, длинные целые, целые без знака. Эти целые виды должны обозначаться именами целых **видов**, определяемых реализацией. Эти имена рассматриваются как имена **нового вида, подобного** *INT*. Их диапазоны значений определяются реализацией. Эти целые виды могут определяться как **корневые** виды соответствующих классов.

13.3 ИМЕНА ПРОЦЕССОВ, ОПРЕДЕЛЯЕМЫХ РЕАЛИЗАЦИЕЙ

Реализация может определять набор имен **процессов**, определяемых реализацией, то есть имен **процессов**, определение которых не специфицировано в языке CHILL. Считается, что определение помещается в область воображаемого внешнего процесса или любой контекст. Процессы с данным именем могут запускаться, и допустимо оперирование значениями экземпляров, обозначающими эти процессы.

13.4 ПРОГРАММЫ ОБРАБОТКИ ИСКЛЮЧЕНИЙ, ОПРЕДЕЛЯЕМЫЕ РЕАЛИЗАЦИЕЙ

К каждому определению процесса в реализации может добавляться программа обработки исключений; такая программа может обрабатывать любое исключение.

13.5 ИМЕНА ИСКЛЮЧЕНИЙ, ОПРЕДЕЛЯЕМЫХ РЕАЛИЗАЦИЕЙ

В реализации может быть определен ряд имен исключений.

13.6 ПРОЧИЕ ЭЛЕМЕНТЫ, ОПРЕДЕЛЯЕМЫЕ РЕАЛИЗАЦИЕЙ

- Статическая проверка динамических условий (см. раздел 2.1.2);
- *директива реализации* (см. раздел 2.6);
- *имя ссылки на текст* (см. разделы 2.7 и 10.10.1);
- **рекурсивность** и **универсальность** по умолчанию (см. разделы 3.7 и 10.4);
- набор значений продолжительных видов (см. раздел 3.11.2);
- набор значений абсолютно-временных видов (см. раздел 3.11.3);
- **формат элемента** по умолчанию (см. раздел 3.12.3);

- сравнение значений **беспризнаковой вариантной структуры** (см. раздел 3.12.4);
- количество битов в слове (см. раздел 3.12.5);
- минимальное битовое заполнение (см. раздел 3.12.5);
- дополнительные **ссылочные** (субъ)ячейки (см. раздел 4.2.1);
- семантика *имени ячейки с присоединением* и *имени значения с присоединением*, являющихся **вариантным** полем ячейки **беспризнаковой вариантной** структуры (см. разделы 4.2.2 и 5.2.3);
- семантика **вариантных** полей **беспризнаковых вариантных** структур (см. разделы 4.2.10, 5.2.13 и 6.2);
- семантика *преобразования ячейки* (см. раздел 4.2.13);
- семантика *преобразования выражения* и дополнительные условия (см. раздел 5.2.11);
- дополнительные *фактические параметры* в *выражении запуска* (см. раздел 5.2.14);
- диапазоны значений для выражений **констант** и **постоянных** (см. раздел 5.3.1);
- алгоритм расписания (см. разделы 6.15, 6.18.2, 6.18.3, 6.19.2 и 6.19.3);
- освобождение памяти в *TERMINATE* (см. раздел 6.20.4);
- обозначения файлов (см. раздел 7.1);
- операции над ассоциациями (см. разделы 7.1 и 7.2.1);
- неисключающие ассоциации (см. раздел 7.1);
- дополнительные атрибуты ассоциативных значений (см. раздел 7.2.2);
- семантика *ассоциативных параметров* (см. раздел 7.4.2);
- исключение *ASSOCIATEFAIL* (см. раздел 7.4.2);
- семантика *параметров модификации* (см. раздел 7.4.5);
- исключения *CREATEFAIL*, *DELETEFAIL* и *MODIFYFAIL* (см. раздел 7.4.5);
- исключение *CONNECTFAIL* (см. раздел 7.4.6.);
- семантика чтения записей, не являющихся допустимыми значениями согласно виду записи (см. раздел 7.4.9);
- дополнительные **прерываемые** действия (см. раздел 9.2);
- исключение *TIMERFAIL* (см. разделы 9.3.1, 9.3.2 и 9.3.3);
- точность значений продолжительности (см. разделы 9.4.1 и 9.4.2);
- индикация **постоянного** значения в *определениях квазисинонима* (см. раздел 10.10.3);
- **региональность** встроенных программ (см. раздел 11.2.2).

ДОБАВЛЕНИЕ A: НАБОР СИМВОЛОВ ЯЗЫКА CHILL

Набор символов языка CHILL является расширением алфавита № 5 МККТТ (Международная эталонная версия, Рекомендация V3). Для тех значений, представление которых превышает 127, графического представления не предусмотрено.

Представление целого — это двоичное число, образуемое битами от b_8 до b_1, где b_1 — младший значащий бит.

	$b_7b_6b_5$	000	001	010	011	100	101	110	111
$b_4b_3b_2b_1$		0	1	2	3	4	5	6	7
0000	0	NUL	TC_7 (DLE)	SP	0	@	P	'	p
0001	1	TC_1 (SOH)	DC_1	!	1	A	Q	a	q
0010	2	TC_2 (STX)	DC_2	"	2	B	R	b	r
0011	3	TC_3 (ETX)	DC_3	#	3	C	S	c	s
0100	4	TC_4 (EOT)	DC_4	$	4	D	T	d	t
0101	5	TC_5 (ENQ)	TC_8 (NAK)	%	5	I	U	e	u
0110	6	TC_6 (ACK)	TC_9 (SYN)	&	6	F	V	f	v
0111	7	BEL	TC_{10} (ETB)	'	7	G	W	g	w
1000	8	FE_0 (BS)	CAN	(	8	H	X	h	x
1001	9	FE_1 (HT)	EM	)	9	I	Y	i	y
1010	10	FE_2 (LF)	SUB	*	:	J	Z	j	z
1011	11	FE_3 (VT)	ESC	+	;	K	[	k	{
1100	12	FE_4 (FF)	IS_4 (FS)	, <	L	\	l	|	
1101	13	FE_5 (CR)	IS_3 (GS)	-	=	M	]	m	}
1110	14	SO	IS_2 (RS)	.	>	N		n	~
1111	15	SI	IS_1 (US)	/	?	O	_	o	DEL

ДОБАВЛЕНИЕ B: СПЕЦИАЛЬНЫЕ СИМВОЛЫ И КОМБИНАЦИИ ЗНАКОВ

	Имя	Употребление
;	точка с запятой	окончание операторов и т.д.
,	запятая	разделитель в различных конструкциях
(	левая круглая скобка	открывающая круглая скобка в различных конструкциях
)	правая круглая скобка	закрывающая круглая скобка в различных конструкциях
[	левая квадратная скобка	открывающая скобка кортежа
]	правая квадратная скобка	закрывающая скобка кортежа
(:	левая скобка кортежа	открывающая скобка кортежа
:)	правая скобка кортежа	закрывающая скобка кортежа
:	двоеточие	указатель метки, указатель диапазона
.	точка	символ выделения поля
:=	символы присваивания	присваивание, инициализация
<	меньше чем	операция сравнения
< =	меньше чем или равно	операция сравнения,
=	равно	операция сравнения, присваивание, инициализация, указатель определения
/=	не равно	операция сравнения
> =	больше чем или равно	операция сравнения
>	больше чем	операция сравнения
+	плюс	операция сложения
–	минус	операция вычитания
*	звездочка	операция умножения, неопределенное значение, неименованное значение, несоответствующий символ
/	косая черта	операция деления
//	двойная косая черта	операция конкатенации
—>	стрелка	ссылка и снятие ссылки, переименование префикса
< >	ромб	начало или конец предложения директивы
/*	открытие комментария	скобка начала комментария
*/	закрытие комментария	скобка конца комментария
'	апостроф	начальный или конечный символ в различных константах
"	кавычки	начальный или конечный символ в символьных строковых константах
""	двойные кавычки	кавычки в символьных строковых константах
!	префиксная операция	снабжение имен префиксами
B'	классификатор константы	двоичное основание константы
D'	классификатор константы	десятичное основание константы
H'	классификатор константы	шестнадцатеричное основание константы
O'	классификатор константы	восьмеричное основание константы
– –	конец строки	разграничитель конца строки внутренних комментариев

ДОБАВЛЕНИЕ C: СТРОКИ СПЕЦИАЛЬНЫХ ПРОСТЫХ ИМЕН

C.1 СТРОКИ ЗАРЕЗЕРВИРОВАННЫХ ПРОСТЫХ ИМЕН

ACCESS
AFTER
ALL
AND
ANDIF
ARRAY
ASSERT
AT

BEGIN
BIN
BODY
BOOLS
BUFFER
BY

CASE
CAUSE
CHARS
CONTEXT
CONTINUE
CYCLE

DCL
DELAY
DO
DOWN
DYNAMIC

ELSE
ELSIF
END
ESAC
EVENT
EVER
EXCEPTIONS
EXIT

FI
FOR
FORBID

GENERAL
GOTO
GRANT

IF
IN
INIT
INLINE
INOUT

LOC

MOD
MODULE

NEWMODE
NONREF
NOPACK
NOT

OD
OF
ON
OR
ORIF
OUT

PACK
POS
POWERSET
PREFIXED
PRIORITY
PROC
PROCESS

RANGE
READ
RECEIVE
RECURSIVE
REF
REGION
REM
REMOTE
RESULT
RETURN
RETURNS
ROW

SEIZE
SEND
SET
SIGNAL
SIMPLE
SPEC
START
STATIC
STEP
STOP
STRUCT
SYN
SYNMODE

TEXT
THEN
THIS
TIMEOUT
TO

UP

VARYING

WHILE
WITH

XOR

C.2 СТРОКИ ПРЕДОПРЕДЕЛЕННЫХ ПРОСТЫХ ИМЕН

ABS
ABSTIME
ALLOCATE
ASSOCIATE
ASSOCIATION

BOOL

CARD
CHAR
CONNECT
CREATE

DAYS
DELETE
DISCONNECT
DISSOCIATE
DURATION

EOLN
EXISTING
EXPIRED

FALSE
FIRST

GETASSOCIATION
GETSTACK
GETTEXTACCESS
GETTEXTINDEX
GETTEXTRECORD
GETUSAGE

HOURS

INDEXABLE
INSTANCE
INT
INTTIME
ISASSOCIATED

LAST
LENGTH
LOWER

MAX
MILLISECS
MIN
MINUTES
MODIFY

NULL
NUM

OUTOFFILE

PRED
PTR

READABLE
READONLY
READRECORD
READTEXT
READWRITE

SAME
SECS
SEQUENCIBLE
SETTEXTACCESS
SETTEXTINDEX
SETTEXTRECORD
SIZE
SUCC

TERMINATE
TIME
TRUE

UPPER
USAGE

VARIABLE

WAIT
WHERE
WRITEABLE
WRITEONLY
WRITERECORD
WRITETEXT

C.3 ИМЕНА ИСКЛЮЧЕНИЙ

ALOCATEFAIL
ASSERTFAIL
ASSOCIATEFAIL
CONNECTFAIL
CREATEFAIL
DELAYFAIL
DELETEFAIL
EMPTY
MODIFYFAIL
NOTCONNECTED
NOTASSOCIATED
OVERFLOW
RANGEFAIL
READFAIL
SENDFAIL
SPACEFAIL
TAGFAIL
TEXTFAIL
TIMERFAIL
WRITEFAIL

ДОБАВЛЕНИЕ D: ПРИМЕРЫ ПРОГРАММ

1. Операции над целыми

```
integer_operations:
MODULE

    add:
    PROC (i,j INT) RETURNS (INT) EXCEPTIONS (OVERFLOW);
            RESULT i+j;
    END add;

    mult:
    PROC (i,j INT) RETURNS (INT) EXCEPTIONS (OVERFLOW);
            RESULT i*j;
    END mult;

    GRANT add, mult;
    SYNMODE operand_mode=INT;
    GRANT operand_mode;
    SYN neutral_for_add=0,
        neutral_for_mult=1;
    GRANT neutral_for_add,
            neutral_for_mult;

END integer_operations;
```

2. Те же операции над дробями

```
fraction_operations:
MODULE
    NEWMODE fraction=STRUCT (num,denum INT);

    add:
    PROC (f1,f2 fraction) RETURNS (fraction) EXCEPTIONS (OVERFLOW);
            RETURN [f1.num*f2.denum+f2.num*f1.denum,f1.denum*f2.denum];
    END add;

    mult:
    PROC (f1,f2 fraction) RETURNS (fraction) EXCEPTIONS (OVERFLOW);
            RETURN [f1.num*f2.num,f2.denum*f1.denum];
    END mult;

    GRANT add, mult;
    SYNMODE operand_mode=fraction;
    GRANT operand_mode;
    SYN neutral_for_add fraction=[ 0,1 ],
        neutral_for_mult fraction=[ 1,1 ];
    GRANT neutral_for_add,
            neutral_for_mult;

END fraction_operations;
```

3. Те же операции над комплексными числами

```
1   complex_operations:
2   MODULE
3       NEWMODE complex=STRUCT (re,im INT);
4
5       add:
6       PROC (c1,c2 complex) RETURNS (complex) EXCEPTIONS (OVERFLOW);
7               RETURN [c1.re+c2.re,c1.im+c2.im];
8       END add;
9
10      mult:
11      PROC (c1,c2 complex) RETURNS (complex) EXCEPTIONS (OVERFLOW);
12              RETURN [c1.re*c2.re-c1.im*c2.im,c1.re*c2.im+c1.im*c2.re];
13      END mult;
14
15      GRANT add, mult;
16      SYNMODE operand_mode=complex;
17      GRANT operand_mode;
18      SYN neutral_for_add=complex [ 0,0 ],
19          neutral_for_mult=complex [ 1,0 ];
20      GRANT neutral_for_add,
21              neutral_for_mult;
22
23  END complex_operations;
```

4. Правила общей арифметики

```
1   general_order_arithmetic:  /* из сводных алгоритмов CACM № 93 */
2   MODULE
3       op:
4       PROC (a INT INOUT, b,c,order INT)
5               EXCEPTIONS (wrong_input) RECURSIVE;
6           DCL d INT;
7           ASSERT b>0 AND c>0 AND order>0
8               ON (ASSERTFAIL):
9                   CAUSE wrong_input;
10              END;
11          CASE order OF
12              (1):    a := b+c;
13                      RETURN;
14              (2):    d := 0;
15              (ELSE): d := 1;
16          ESAC;
17          DO FOR i := 1 TO c;
18              op (a,b,d,order-1);
19              d := a;
20          OD;
21          RETURN;
22      END op;
23
24      GRANT op;
25
26  END general_order_arithmetic;
```

5. Побитовое сложение и проверка результата

```
1   add_bit_by_bit:
2   MODULE
3       adder:
4       PROC (a STRUCT (a2,a1 BOOL) IN, b STRUCT (b2,b1 BOOL) IN)
5               RETURNS (STRUCT (c4,c2,c1 BOOL));
6           DCL c STRUCT (c4,c2,c1 BOOL);
7           DCL k2,x,w,t,s,r BOOL;
8           DO WITH a,b,c;
9               k2 := a1 AND b1;
10              c1 := NOT k2 AND (a1 OR b1);
11              x := a2 AND b2 AND k2;
12              w := a2 OR b2 OR k2;
13              t := b2 AND k2;
14              s := a2 AND k2;
15              r := a2 AND b2;
16              c4 := r OR s OR t;
17              c2 := x OR (w AND NOT c4);
18          OD;
19          RETURN c;
20      END adder;
21      GRANT adder;
22  END add_bit_by_bit;
23
24  exhaustive_checker:
25  MODULE
26      SEIZE adder;
27      DCL a STRUCT (a2,a1 BOOL),
28            b STRUCT (b2,b1 BOOL);
29      SYNMODE res=ARRAY (1:16) STRUCT (c4,c2,c1 BOOL);
30      DCL r INT, results res;
31      DO WITH a,b;
32          r := 0;
33          DO FOR a2 IN BOOL;
34              DO FOR a1 IN BOOL;
35                  DO FOR b2 IN BOOL;
36                      DO FOR b1 IN BOOL;
37                          r+ := 1;
38                          results (r) := adder (a,b);
39                      OD;
40                  OD;
41              OD;
42          OD;
43      OD;
44      ASSERT
45          results=res [[FALSE,FALSE,FALSE ],[FALSE,FALSE,TRUE ],
46                       [FALSE,TRUE,FALSE ] ,[FALSE,TRUE,TRUE ],
47                       [FALSE,FALSE,TRUE ] ,[FALSE,TRUE,FALSE ],
48                       [FALSE,TRUE,TRUE ]  ,[TRUE,FALSE,FALSE ],
49                       [FALSE,TRUE,FALSE ] ,[FALSE,TRUE,TRUE ],
50                       [TRUE,FALSE,FALSE ] ,[TRUE,FALSE,TRUE ],
51                       [FALSE,TRUE,TRUE ]  ,[TRUE,FALSE,FALSE ],
52                       [TRUE,FALSE,TRUE ]  ,[TRUE,TRUE,FALSE ]];
53  END exhaustive_checker;
```

```
playing_with_dates:
MODULE  /* из сводных алгоритмов CACM № 199 */
    SYNMODE month=SET (jan,feb,mar,apr,may,jun,
                jul,aug,sep,oct,nov,dec);
    NEWMODE date=STRUCT (day INT (1:31), mo month, year INT);

    gregorian_date:
    PROC (julian_day_number INT) RETURNS (date);
        DCL j INT := julian_day_number,
              d,m,y INT;
        j- := 1_721_119;
        y := (4 * j - 1) / 146_097;
        j := 4 * j - 1 - 146_097 * y;
        d := j / 4;
        j := (4 * d + 3) / 1_461;
        d := 4 * d + 3 - 1_461 * j;
        d := (d + 4) / 4;
        m := (5 * d - 3) / 153;
        d := 5 * d - 3 - 153 * m;
        d := (d + 5) / 5;
        y := 100 * y + j;
        IF m<100 THEN m + := 3;
                 ELSE m - := 9;
                      y + := 1;
        FI;
        RETURN [d,month (m+1), y];
    END gregorian_date;

    julian_day_number:
    PROC (d date) RETURNS (INT);
        DCL c,y,m INT;
        DO WITH d;
            m := NUM (mo)+1;
            IF m>2 THEN m - := 3;
                   ELSE m + := 9;
                        year - := 1;
            FI;
            c := year/100;
            y := year-100*c;
            RETURN (146_097*c)/4+(1_461*y)/4
                      +(153+m+c)/5+day+1_721_119;
        OD;
    END julian_day_number;
    GRANT gregorian_date, julian_day_number;
END playing_with_dates;

test:
MODULE
    SEIZE gregorian_date, julian_day_number;
    ASSERT julian_day_number ([ 10,dec,1979 ])=julian_day_number
            (gregorian_date(julian_day_number([ 10,dec,1979 ])));
END test;
```

```
1   Roman:
2   MODULE
3       SEIZE n,rn;
4       GRANT convert;
5       convert:
6       PROC () EXCEPTIONS (string_too_small);
7           DCL r INT := 0;
8           DO WHILE n>=1_000;
9               rn(r) := 'M';
10              n - := 1_000;
11              r + := 1;
12          OD;
13          IF n>500 THEN rn(r) := 'D';
14                        n - := 500;
15                        r + := 1;
16          FI;
17          DO WHILE n>=100;
18              rn(r) := 'C';
19              n - := 100;
20              r + := 1;
21          OD;
22          IF n>=50 THEN rn(r) := 'L';
23                        n - := 50;
24                        r + := 1;
25          FI;
26          DO WHILE n>=10;
27              rn(r) := 'X';
28              n - := 10;
29              r + := 1;
30          OD;
31          IF n>=5 THEN rn(r) := 'V';
32                       n - := 5;
33                       r + := 1;
34          FI;
35          DO WHILE n>=1;
36              rn(r) := 'I';
37              n - := 1;
38              r + := 1;
39          OD;
40          RETURN;
41      END ON (RANGEFAIL): DO FOR i := 0 TO UPPER (rn);
42                              rn(i) := '.';
43                          OD;
44                          CAUSE string_too_small;
45      END convert;
46  END Roman;
47  test:
48  MODULE
49      SEIZE convert;
50      DCL n INT INIT := 1979;
51      DCL rn CHARS (20) INIT := (20)' ';
52      GRANT n,rn;
53      convert ();
54      ASSERT rn="MDCCCCLXXVIIII"//(6)' ';
55  END test;
```

8. **Подсчет букв в символьной строке произвольной длины**

```
1   letter_count:
2   MODULE
3       SEIZE max;
4       DCL letter POWERSET CHAR INIT := ['A' : 'Z'];
5       count:
6       PROC (input ROW CHARS (max) IN, output ARRAY ('A':'Z') INT OUT);
7           output := [(ELSE) : 0];
8           DO FOR i := 0 TO UPPER (input ->);
9               IF input -> (i) IN letter
10                  THEN
11                      output (input -> (i)) + := 1;
12              FI;
13          OD;
14      END count;
15      GRANT count;
16  END letter_count;
17  test:
18  MODULE
19      SYNMODE results=ARRAY ('A':'Z')INT;
20      DCL c CHARS (10) INIT := "A-B<ZAA9K' ";
21      DCL output results;
22      SYN max=10_000;
23      GRANT max;
24      SEIZE count;
25      count (-> c,output);
26      ASSERT output=results [('A') : 3,('B','K','Z') : 1, (ELSE) : 0];
27  END test;
```

9. **Простые числа**

```
1   prime:
2   MODULE
3
4       SYN max = H'7FFF;
5       NEWMODE number_list =POWERSET INT (2:max);
6       SYN empty = number_list [ ];
7       DCL sieve number_list INIT := [ 2:max ],
8           primes number_list INIT := empty;
9       GRANT primes;
10      DO WHILE sieve/=empty;
11          primes OR := [MIN (sieve)];
12          DO FOR j := MIN (sieve) BY MIN (sieve) TO max;
13              sieve - := [j];
14          OD;
15      OD;
16  END prime;
```

10. **Реализация стеков двумя разными способами, прозрачными для пользователя**

```
1   stack: MODULE
2       NEWMODE element =STRUCT (a INT, b BOOL);
3       stacks_1:
4       MODULE
```

```
5           SEIZE element;
6           SYN max=10_000,min=1;
7           DCL stack ARRAY (min : max) element,
8                 stackindex INT INIT := min;
9
10          push:
11          PROC (e element) EXCEPTIONS (overflow);
12                IF stackindex=max
13                   THEN CAUSE overflow;
14                FI;
15                stackindex + := 1;
16                stack (stackindex) := e;
17                RETURN;
18          END push;
19
20          pop:
21          PROC () EXCEPTIONS (underflow);
22                IF stackindex=min
23                   THEN CAUSE underflow;
24                FI;
25                stackindex - := 1;
26                RETURN;
27          END pop;
28
29          elem:
30          PROC (i INT) RETURNS (element LOC) EXCEPTIONS (bounds);
31                IF i<min OR i>max
32                   THEN CAUSE bounds;
33                FI;
34                RETURN stack (i);
35          END elem;
36
37          GRANT push,pop,elem;
38    END stacks_1;
39    stacks_2:
40    MODULE
41          SEIZE element;
42          NEWMODE cell=STRUCT (pred,succ REF cell,info element);
43          DCL p,last,first REF cell INIT := NULL;
44
45          push:
46          PROC (e element) EXCEPTIONS (overflow);
47              p := ALLOCATE (cell) ON
48                                   (ALLOCATEFAIL) : CAUSE overflow;
49                                   END;
50              IF last=NULL
51                 THEN first := p;
52                      last := p;
53                 ELSE last ->. succ := p;
54                      p ->. pred := last;
55                      last := p;
56              FI;
57              last ->. info := e;
58              RETURN;
59          END push;
60
61          pop:
62          PROC () EXCEPTIONS (underflow);
63              IF last=NULL
64                 THEN CAUSE underflow;
65              FI;
```

```
66              p := last;
67              last := last ->. pred;
68              IF last = NULL
69                  THEN first := NULL;
70                  ELSE last ->. succ := NULL;
71              FI;
72              TERMINATE(p);
73              RETURN;
74          END pop;
75
76          elem:
77          PROC (i INT) RETURNS (element LOC) EXCEPTIONS (bounds);
78              IF first=NULL
79                  THEN CAUSE bounds;
80              FI;
81              p := first;
82              DO FOR j := 2 TO i;
83                  IF p ->. succ=NULL
84                      THEN CAUSE bounds;
85                  FI;
86                  p := p ->. succ;
87              OD;
88              RETURN p ->. info;
89          END elem;
90
91          / * GRANT push, pop, elem; * /
92      END stacks_2;
93  END stack;
```

11. Фрагмент игры в шахматы

```
1   chess_fragments:
2   MODULE
3       NEWMODE piece=STRUCT (color SET (white,black),
4                                 kind SET (pawn,rook,knight,bishop,queen,king));
5       NEWMODE column=SET (a,b,c,d,e,f,g,h);
6       NEWMODE line=INT (1 : 8);
7       NEWMODE square=STRUCT (status SET (occupied,free),
8                                  CASE status OF
9                                  (occupied) : p piece,
10                                 (free) :
11                                 ESAC);
12      NEWMODE board=ARRAY (line) ARRAY (column) square;
13      NEWMODE move=STRUCT (lin_1,lin_2 line,
14                                 col_1,col_2 column);
15
16      initialise:
17      PROC (bd board INOUT);
18          bd := [ (1): [(a,h):   [.status: occupied, .p : [white,rook]],
19                       (b,g):   [.status: occupied, .p : [white,knight]],
20                       (c,f):   [.status: occupied, .p : [white,bishop]],
21                       (d):     [.status: occupied, .p : [white,queen]],
22                       (e):     [.status: occupied, .p : [white,king]]],
23                  (2): [(ELSE):[.status: occupied, .p : [white,pawn]]],
24                  (3:6):[(ELSE):[.status: free]],
25                  (7): [(ELSE):[.status: occupied, .p : [black,pawn]]],
26                  (8): [(a,h):   [.status: occupied, .p : [black,rook]],
27                       (b,g):   [.status: occupied, .p : [black,knight]],
```

```
28                  (c,f):    [.status: occupied, .p : [black,bishop]],
29                  (d):      [.status: occupied, .p : [black,queen]],
30                  (e):      [.status: occupied, .p : [black,king]]]
31              ];
32          RETURN;
33      END initialise;
34      register_move:
35      PROC (b board LOC,m move) EXCEPTIONS (illegal);
36          DCL starting square LOC := b (m.lin_1)(m.col_1),
37              arriving square LOC := b (m.lin_2)(m.col_2);
38          DO WITH m;
39              IF starting.status=free THEN CAUSE illegal; FI;
40              IF arriving.status/=free THEN
41                  IF arriving.p.kind=king THEN CAUSE illegal; FI;
42              FI;
43              CASE starting.p.kind, starting.p.color OF
44                  (pawn),(white):
45                  IF col_1 = col_2 AND (arriving.status/=free
46                      OR NOT (lin_2=lin_1+1 OR lin_2=lin_1+2 AND lin_2=2))
47                      OR (col_2=PRED (col_1) OR col_2=SUCC (col_1))
48                      AND arriving.status=free THEN CAUSE illegal; FI;
49                      IF arriving.status/=free THEN
50                          IF arriving.p.color=white THEN CAUSE illegal; FI; FI;
51                  (pawn),(black):
52                  IF col_1=col_2 AND (arriving.status/=free
53                      OR NOT (lin_2=lin_1-1 OR lin_2=lin_1-2 AND lin_1=7))
54                      OR (col_2=PRED (col_1) OR col_2=SUCC (col_1))
55                      AND arriving.status=free THEN CAUSE illegal; FI;
56                      IF arriving.status/=free THEN
57                          IF arriving.p.color=black THEN CAUSE illegal; FI; FI;
58                  (rook),(*):
59                  IF NOT ok_rook (b,m)
60                      THEN CAUSE illegal;
61                  FI;
62                  (bishop),(*):
63                  IF NOT ok_bishop (b,m)
64                      THEN CAUSE illegal;
65                  FI;
66                  (queen),(*):
67                  IF NOT ok_rook (b,m) AND NOT ok_bishop (b,m)
68                      THEN CAUSE illegal;
69                  FI;
70                  (knight),(*):
71                  IF ABS (ABS (NUM (col_2)-NUM (col_1))
72                          -ABS (lin_2- lin_1)) /= 1
73                      OR ABS (NUM (col_2)-NUM (col_1))
74                          +ABS (lin_2- lin_1) =/ 3 THEN CAUSE illegal; FI;
75                  IF arriving.status/=free THEN
76                          IF arriving.p.color=starting.p.color THEN
77                          CAUSE illegal; FI; FI;
78                  (king),(*):
79                  IF ABS (NUM (col_2)-NUM (col_1)) > 1
80                      OR ABS (lin_2- lin_1) > 1
81                      OR lin_2=lin_1 AND col_2=col_1 THEN CAUSE illegal; FI;
82                  IF arriving.status/=free THEN
83                      IF arriving.p.color=starting.p.color THEN
84                      CAUSE illegal; FI; FI; / * проверка движения короля не реализована * /
85              ESAC;
86          OD;
87          arriving := starting;
88          starting := [.status:free];
```

```
89          RETURN;
90      END register_move;
91      ok_rook:
92      PROC (b board,m move) RETURNS (BOOL);
93          DCL starting square := b (m.lin_1)(m.col_1),
94              arriving square := b (m.lin_2)(m.col_2);
95
96          DO WITH m;
97              IF NOT (col_2=col_1 OR lin_1=lin_2) THEN RETURN FALSE; FI;
98              IF arriving.status/=free THEN
99                  IF arriving.p.color=starting.p.color THEN;
100                 RETURN FALSE; FI; FI;
101             IF col_1=col_2
102                 THEN IF lin_1<lin_2
103                         THEN DO FOR lin := lin_1+1 TO lin_2-1;
104                                 IF b (lin)(col_1).status/=free
105                                     THEN RETURN FALSE;
106                                 FI;
107                              OD;
108                         ELSE DO FOR lin := lin_1-1 DOWN TO lin_2+1;
109                                 IF b (lin)(col_1).status/=free
110                                     THEN RETURN FALSE;
111                                 FI;
112                              OD;
113                      FI;
114             ELSIF col_1<col_2
115                         THEN DO FOR col := SUCC (col_1) TO PRED (col_2);
116                                 IF b (lin_1)(col).status/=free
117                                     THEN RETURN FALSE;
118                                 FI;
119                              OD;
120                         ELSE DO FOR col := SUCC (col_2) DOWN TO PRED (col_1);
121                                 IF b (lin_1)(col).status/=free
122                                     THEN RETURN FALSE;
123                                 FI;
124                              OD;
125             FI;
126             RETURN TRUE;
127         OD;
128     END ok_rook;
129     ok_bishop:
130     PROC (b board,m move) RETURNS (BOOL);
131         DCL starting square := b (m.lin_1)(m.col_1),
132             arriving square := b (m.lin_2)(m.col_2),
133             col column;
134
135         DO WITH m;
136             CASE lin_2>lin_1,col_2>col_1 OF
137                 (TRUE),(TRUE): col := col_1;
138                                DO FOR lin := lin_1+1 TO lin_2-1;
139                                       col := SUCC (col);
140                                       IF b (lin)(col).status/=free
141                                           THEN RETURN FALSE;
142                                       FI;
143                                OD;
144                                IF SUCC (col)/=col_2
145                                       THEN RETURN FALSE;
146                                FI;
147                 (TRUE),(FALSE): col := col_1;
148                                DO FOR lin := lin_1+1 TO lin_2-1;
149                                       col := PRED (col);
```

```
150                                     IF b (lin)(col).status/=free
151                                         THEN RETURN FALSE;
152                                     FI;
153                                 OD;
154                                 IF PRED (col)/=col_2
155                                         THEN RETURN FALSE;
156                                 FI;
157                         (FALSE),(TRUE): col := col_1;
158                                 DO FOR lin := lin_1-1 DOWN TO lin_2+1;
159                                         col := SUCC (col);
160                                         IF b (lin)(col).status/=free
161                                             THEN RETURN FALSE;
162                                         FI;
163                                 OD;
164                                 IF SUCC (col)/=col_2
165                                         THEN RETURN FALSE;
166                                 FI;
167                         (FALSE),(FALSE): col := col_1;
168                                 DO FOR lin := lin_1-1 DOWN TO lin_2+1;
169                                         col := PRED (col);
170                                         IF b (lin)(col).status/=free
171                                             THEN RETURN FALSE;
172                                         FI;
173                                 OD;
174                                 IF PRED (col)/=col_2
175                                         THEN RETURN FALSE;
176                                 FI;
177                     ESAC;
178                     IF arriving.status=free THEN RETURN TRUE;
179                     ELSE RETURN arriving.p.color/=starting.p.color; FI;
180             OD;
181         END ok_bishop;
182         END chess_fragments;
```

12. Построение и оперирование циклически связанным списком

```
1   circular_list:
2   MODULE
3       handle_list:
4       MODULE
5           GRANT insert, remove, node;
6           NEWMODE node=STRUCT (pred, suc REF node, value INT);
7           DCL pool ARRAY (1:1000)node;
8           DCL head node := (: NULL,NULL,0 :);
9
10          insert: PROC (new node);
11                  /* вставки */
12          END insert;
13
14          remove: PROC ();
15                  /* удаления */
16          END remove;
17
18          initialize_list:
19          BEGIN
20              DCL last REF node := ->head;
21              DO FOR new IN pool;
22                  new.pred := last;
```

```
23                  last->.suc := ->new;
24                  last := ->new;
25                  new.value := 0;
26              OD;
27              head.pred := last;
28              last->.suc := ->head;
29          END initialize_list;
30
31      END handle_list;
32      manipulate:
33      MODULE
34          SEIZE node, remove, insert;
35          DCL node_a node := (: NULL,NULL,536 :);
36          remove();
37          remove();
38          insert(node_a);
39      END manipulate;
40  END circular_list;
```

13. Регион управления конкурентным доступом к ресурсам

```
1   allocate_resources:
2   REGION
3       GRANT allocate, deallocate;
4       NEWMODE resource_set = INT (0:9);
5       DCL allocated ARRAY (resource_set)BOOL := (: (resource_set): FALSE :);
6       DCL resource_freed EVENT;
7
8       allocate:
9       PROC () RETURNS (resource_set);
10          DO FOR EVER;
11              DO FOR i IN resource_set;
12                  IF NOT allocated(i)
13                      THEN
14                          allocated(i) := TRUE;
15                          RETURN i;
16                  FI;
17              OD;
18              DELAY resource_freed;
19          OD;
20      END allocate;
21
22      deallocate:
23      PROC (i resource_set);
24          allocated(i) := FALSE;
25          CONTINUE resource_freed;
26      END deallocate;
27
28  END allocate_resources;
```

14. Постановка вызовов в очередь к коммутатору

```
1   switchboard:
2   MODULE
3
4       /* В данном примере рассматривается коммутатор, ставящий в очередь входящие
           вызовы и передающий их равномерно оператору. Всякий раз оператор может
```

```
5           обслужить один и только один вызов. Вызов обслуживается распределителем
6           вызовов, посылающим вызовы через фиксированные интервалы времени. Если
7           оператор не готов принять вызов или имеются другие вызовы, ожидающие
8           обслуживания, то новый вызов ставится в очередь на ожидание обслуживания. */
9       DCL operator_is_ready,
10            switch_is_closed EVENT;
11
12      call_distributor:
13      PROCESS ();
14          wait:
15          PROC (x INT);
16              /* некоторое действие ожидания */
17          END wait;
18          DO FOR EVER;
19              wait (10/* секунды */ ) ;
20              CONTINUE operator_is_ready;
21          OD;
22      END call_distributor;
23
24      call_process:
25      PROCESS ();
26          DELAY CASE
27          (operator_is_ready): /* некоторые действия */;
28          (switch_is_closed): DO FOR i IN INT (1:100);
29                                  CONTINUE operator_is_ready;
30                                   /* очистить очередь */
31                              OD;
32          ESAC;
33      END call_process;
34
35      operator:
36      PROCESS ();
37          DCL time INT;
38          DO FOR EVER;
39              IF time = 1700
40                  THEN CONTINUE switch_is_closed;
41              FI;
42          OD;
43      END operator;
44
45      START call_distributor();
46      START operator();
47      DO FOR i IN INT (1:100);
48          START call_process();
49      OD;
50  END switchboard;
```

15. Распределение и освобождение ресурсов

```
1   definitions:
2   MODULE
3   SIGNAL
4       acquire,
5       release=(INSTANCE),
6       congested,
7       ready,
8       advance,
9       readout=(INT);
```

```
10      GRANT ALL;
11  END definitions;
12  counter_manager:
13  MODULE
14
15  /* Для иллюстрации использования сигналов и варианта получения (вместо
16     сигналов могут использоваться буферы) мы рассмотрим пример, в
17     котором распределитель управляет ресурсами, в данном случае
18     счетчиками. Модуль является частью большой системы, в которой
19     имеются пользователи, запрашивающие услуги от counter_manager.
20     Модуль имеет два определения процессов, по одному для распределения и
21     для счетчиков. Initiate (инициировать) и terminate (завершать) — это
22     внутренние сигналы, посылаемые от распределителя в счетчики. Все
23     прочие сигналы, посылаемые от пользователей или к пользователям,
24     являются внешними. */
25
26      SEIZE /* внешние сигналы */
27          acquire, release, congested,ready,advance,readout;
28      SIGNAL initiate = (INSTANCE),
29             terminate;
30      allocator:
31      PROCESS ();
32          NEWMODE no_of_counters = INT (1:100);
33      DCL counters ARRAY (no_of_counters)
34                              STRUCT (counter INSTANCE,status SET (busy,idle));
35      DO FOR each IN counters;
36          each := (: START counter(), idle :);
37      OD;
38      DO FOR EVER;
39      BEGIN
40          DCL user INSTANCE;
41          await_signals:
42          RECEIVE CASE SET user;
43          (acquire):
44              DO FOR each IN counters;
45                  DO WITH each;
46                      IF status = idle
47                          THEN
48                              status := busy;
49                              SEND initiate (user) TO counter;
50                              EXIT await_signals;
51                      FI;
52                  OD;
53              OD;
54              SEND congested TO user;
55          (release IN this_counter):
56              SEND terminate TO this_counter;
57              find_counter:
58              DO FOR each IN counters;
59                  DO WITH each;
60                      IF this_counter = counter
61                          THEN
62                              status := idle;
63                              EXIT find_counter;
64                      FI;
65                  OD;
66              OD find_counter;
67          ESAC await_signals;
68      END;
69      OD;
70      END allocator;
        counter:
```

```
71      PROCESS ();
72          DO FOR EVER;
73          BEGIN
74              DCL user INSTANCE,
75                  count INT := 0;
76              RECEIVE CASE
77                  (initiate IN received_user):
78                      SEND ready TO received_user;
79                      user := received_user;
80              ESAC;
81              work_loop:
82              DO FOR EVER;
83                  RECEIVE CASE
84                      (advance): count + := 1;
85                      (terminate):
86                          SEND readout(count) TO user;
87                          EXIT work_loop;
88                  ESAC;
89              OD work_loop;
90          END;
91          OD;
92      END counter;
93      START allocator();
94  END counter_manager;
```

16. Распределение и освобождение ресурсов с использованием буферов

```
1
2
3   user_world:
4   MODULE
5   /* Этот пример аналогичен примеру программы 15, только для связи
6      здесь вместо сигналов используются буферы. Основное отличие
7      состоит в том, что процессы теперь идентифицируются посредством
8      ссылок на буферы локальных сообщений, а не на значения
9      экземпляров. Для каждого процесса существует один буфер сообщений,
10     объявленный локальным. Для каждого определения процесса имеется
11     один набор типов сообщений. При запуске каждый процесс должен
12     идентифицировать адрес своего буфера для запускающего процесса.
13     Модуль user_world определяет некоторую операционную среду, в
14     которой используется counter_manager. */
15
16  SEIZE allocator;
17  GRANT user_buffers,user_messages,
18            allocator_messages, allocator_buffers,
19            counter_messages, counters_buffers;
20  NEWMODE
21      user_messages =
22          STRUCT (type SET (congested, ready,
23                      readout, allocator_id),
24                              CASE type OF
25                                  (congested) : ,
26                                  (ready) : counter REF counters_buffers,
27                                  (readout) : count INT,
28                                  (allocator_id): allocator REF allocator_buffers
29                              ESAC),
30      user_buffers = BUFFER (1) user_messages,
31      allocator_messages =
```

```
32          STRUCT (type SET (acquire, release, counter_id),
33                  CASE type OF
34                      (acquire) : user REF user_buffers,
35                      (release,
36                      counter_id): counter REF counters_buffers
37                  ESAC),
38      allocator_buffers = BUFFER (1) allocator_messages,
39      counter_messages =
40          STRUCT (type SET (initiate, advance, terminate),
41                  CASE type OF
42                      (initiate) : user REF user_buffers,
43                      (advance,
44                      terminate):
45                  ESAC),
46      counters_buffers = BUFFER (1) counter_messages;
47  DCL user_buffer user_buffers,
48        allocator_buf REF allocator_buffers,
49        counter_buf REF counters_buffers;
50  START allocator(->user_buffer);
51  allocator_buf := (RECEIVE user_buffer).allocator;
52  END user_world;
53  counter_manager:
54  MODULE
55  SEIZE user_buffers,user_messages,
56          allocator_messages, allocator_buffers,
57          counter_messages, counters_buffers;
58  GRANT allocator;
59
60  allocator:
61  PROCESS (starter REF user_buffers);
62      DCL allocator_buffer allocator_buffers;
63      NEWMODE no_of_counters = INT (1:10);
64      DCL counters ARRAY (no_of_counters)
65                              STRUCT (counter REF counters_buffers,
66                                      status SET (busy, idle)),
67            message allocator_messages;
68      SEND starter->([allocator_id, ->allocator_buffer]);
69      DO FOR each IN counters;
70          START counter(->allocator_buffer);
71          each := [(RECEIVE allocator_buffer).counter, idle];
72      OD;
73      DO FOR EVER;
74          BEGIN
75          DCL user REF user_buffers;
76          message := RECEIVE allocator_buffer;
77          handle_messages:
78          CASE message.type OF
79          (acquire):
80              user := message.user;
81              DO FOR each IN counters;
82                  DO WITH each;
83                      IF status= idle
84                          THEN status := busy;
85                                  SEND counter->([initiate, user]);
86                                  EXIT handle_messages;
87                      FI;
88                  OD;
89              OD;
90              SEND user->([congested]);
91          (release):
92              SEND message.counter->([terminate]);
```

```
93              find_counter:
94              DO FOR each IN counters;
95                  DO WITH each;
96                      IF message.counter = counter
97                          THEN status := idle;
98                                  EXIT find_counter;
99                      FI;
100                 OD;
101             OD find_counter;
102         (counter_id): ;
103         ESAC handle_messages;
104         END;
105     OD;
106 END allocator;
107 counter:
108 PROCESS (starter REF allocator_buffers);
109     DCL counter_buffer counters_buffers;
110     SEND starter->([counter_id, ->counter_buffer]);
111     DO FOR EVER;
112         BEGIN
113             DCL user REF user_buffers,
114                 count INT := 0,
115                 message counter_messages;
116             message := RECEIVE counter_buffer;
117             CASE message.type OF
118                 (initiate): user := message.user;
119                             SEND user->([ready, ->counter_buffer]);
120                 ELSE /* некоторое ошибочное действие */
121             ESAC;
122         work_loop:
123         DO FOR EVER;
124             message := RECEIVE counter_buffer;
125             CASE message.type OF
126                 (advance) : count + := 1;
127                 (terminate):SEND user->([readout, count]);
128                             EXIT work_loop;
129                 ELSE /* некоторое ошибочное действие */
130             ESAC;
131         OD work_loop;
132     END;
133     OD;
134 END counter;
135 END counter_manager;
```

17. Сканер 1 строк

```
1   string_scanner1: /* В данной программе строки реализуются посредством
2                       упакованных массивов символов. */
3   MODULE
4       SYN
5           blanks ARRAY (0:9)CHAR PACK = [(*):' ' ], linelength = 132;
6       SYNMODE
7           stringptr = ROW ARRAY (lineindex)CHAR PACK,
8           lineindex = INT (0:linelength-1);
9
10  scanner:
11  PROC (string stringptr, scanstart lineindex INOUT,
12          scanstop lineindex, stopset POWERSET CHAR)
```

```
13          RETURNS (ARRAY (0:9)CHAR PACK);
14      DCL count INT := 0,
15          res ARRAY (0:9)CHAR PACK := blanks;
16      DO
17          FOR c IN string->(scanstart:scanstop)
18          WHILE NOT (c IN stopset);
19              count + := 1;
20      OD;
21      IF count>0
22          THEN
23              IF count>10
24                  THEN
25                      count := 10;
26              FI;
27              res(0:count-1) := string->(scanstart:scanstart+count-1);
28      FI;
29      RESULT res;
30      IF scanstart+count < scanstop
31          THEN
32              scanstart := scanstart+count+1;
33      FI;
34  END scanner;
35
36  GRANT scanner;
37
38  END string_scanner1;
```

18. **Сканер 2 строк**

```
1   string_scanner2:  /* Этот пример аналогичен примеру программы, но в нем вместо
2                        упакованных массивов используется символьная строка. */
3   MODULE
4       SYN
5           blanks = (10)' ', linelength = 132;
6       SYNMODE
7           stringptr = ROW CHARS (linelength),
8           lineindex = INT (0:linelength-1);
9
10      scanner:
11          PROC (string stringptr, scanstart lineindex INOUT,
12                scanstop lineindex, stopset POWERSET CHAR)
13                RETURNS (CHARS (10));
14              DCL count INT := 0;
15              DO FOR i := scanstart TO scanstop
16                  WHILE NOT (string->(i) IN stopset);
17                      count + := 1;
18              OD;
19              IF count>0
20                  THEN
21                      IF count>=10
22                          THEN
23                              RESULT string->(scanstart UP 10);
24                          ELSE
25                              RESULT string->(scanstart:scanstart+count-1)
26                                      //blanks(count:9);
27                      FI;
28                  ELSE
29                      RESULT blanks;
```

```
30              FI;
31              IF scanstart+count < scanstop
32                  THEN
33                      scanstart := scanstart+count+1;
34              FI;
35          END scanner;
36
37      GRANT scanner;
38
39  END string_scanner2;
```

19. Исключение элемента из двойного связного списка

```
1   queue: MODULE
2       SYNMODE info=INT;
3       queue_removal:
4       MODULE
5           SEIZE info;
6           GRANT remove;
7           remove:
8           PROC (p PTR) RETURNS (info) EXCEPTIONS (EMPTY);
9                   /* Эта процедура исключает элемент, к которому есть обращение от p,
10                     из очереди и возвращает информацию о содержимом данного элемента
11                     очереди. */
12                  SYNMODE element = STRUCT (
13                                  i info POS (0,8:31),
14                                  prev PTR POS (1,0:15),
15                                  next PTR POS (1,16:31));
16                  DCL x REF element LOC := element(p), prev, next PTR;
17                  prev := x->.prev;
18                  next := x->.next;
19                  x->.prev, x->.next := NULL;
20                  RESULT x->.i;
21                  p := prev;
22                  x->.next := next;
23                  p := next;
24                  x->.prev := prev;
25          END remove;
26      END queue_removal;
27  END queue;
```

20. Коррекция записи файла

```
1   read_modify_write:
2   MODULE
3
4   /* этот пример показывает, как могут быть использованы концепции ввода—вывода языка CHILL */
5   /* при написании прикладной программы, где запись файла с произвольным доступом      */
6   /* может быть скорректирована или добавлена, если до этого момента не использовалась. */
7
8     NEWMODE
9       index_set = INT (1:1000),
10      record_type = STRUCT (
11                              free   BOOL,
12                              count  INT,
13                              name   CHARS (20));
```

```
14
15    DCL
16      curindex          index_set,
17      file_association  ASSOCIATION,
18      record_file       ACCESS (index_set) record_type,
19      record_buffer     record_type;
20
21    ASSOCIATE (file_association,"DSK:RECORDS.DAT");   /* создать ассоциацию   */
22    CONNECT (record_file,file_association,READWRITE); /* соединить с файлом   */
23    curindex := 123;                                  /* расположить запись   */
24    READRECORD (record_file,curindex,record_buffer);  /* считать запись       */
25      IF record_buffer.free                           /* если запись свободна */
26        THEN                                          /* отыскать и           */
27          record_buffer.free := FALSE                 /* инициализировать ее  */
28          record_buffer.count := 0;
29          record_buffer.name := "CHILL I/O concept  ";
30      FI;
31    record_buffer.count + := 1;                       /* увеличить ее счетчик */
32    WRITERECORD (record_file, curindex, record_buffer); /* записать запись    */
33    DISSOCIATE (file_association);                    /* конец ассоциации     */
34
35  END read_modify_write;
```

21. Слияние двух отсортированных файлов

```
1   merge_sorted_files:
2   MODULE
3
4    /* этот пример показывает, как два отсортированных файла могут быть объединены в один файл  */
5    /* новый отсортированный файл, где поле 'key' (ключ) используется для сортировки           */
6    /* после слияния старые отсортированные файлы исключаются                                  */
7
8     NEWMODE
9       record_type = STRUCT (
10                                key   INT,
11                                name CHARS (50));
12
13    DCL
14      flag       BOOL,
15      infiles    ARRAY (BOOL) ACCESS record_type,
16      outfile    ACCESS record_type,
17      buffers    ARRAY (BOOL) record_type,
18      innames    ARRAY (BOOL) CHARS (10) INIT := ["FILE.IN.1 ","FILE.IN.2 "],
19      outname    CHARS (10) INIT := "FILE.OUT  ",
20      inassocs   ARRAY (BOOL) ASSOCIATION,
21      outassoc   ASSOCIATION;
22
23    /* соединить оба отсортированных файла ввода, связать с доступом для ввода  */
24    /* и считать их первую запись в буфер                                        */
25
26    DO
27      FOR curfile IN infiles,
28            curbuffer IN buffers,
29            curassoc IN inassocs,
30            curname IN innames;
31        CONNECT (curfile, ASSOCIATE (curassoc,curname), READONLY);
32        READRECORD (curfile, curbuffer);
33    OD;
```

```
34
35    /* соединить файл вывода, создать файл для ассоциации */
36    /* и связать его с доступом для вывода                */
37
38    ASSOCIATE (outassoc,outname);
39    CREATE (outassoc);
40    CONNECT (outfile, outassoc, WRITEONLY);
41    merge_files:
42      DO FOR EVER
43
44        /* установить, какой файл, если вообще такой имеется, обрабатывает следующий */
45        /* 'флаг', указывающий на файл                                                */
46
47        CASE OUTOFFILE (infiles(FALSE)),OUTOFFILE (infiles(TRUE)) OF
48          (TRUE),  (TRUE):                          /* оба файла пусты   */
49                   EXIT merge_files;
50          (TRUE),  (FALSE):                         /* один файл пуст    */
51                   flag := TRUE;
52          (FALSE), (TRUE):                          /* один файл пуст    */
53                   flag := FALSE;
54          (FALSE), (FALSE):                         /* нет пустых файлов */
55                   flag := buffers(FALSE).key>buffers(TRUE).key;
56        ESAC;
57
58            /* вывести буфер, содержащий в настоящий момент запись с      */
59            /* наименьшим значением 'ключа', записать в буфер новую запись */
60
61            WRITERECORD (outfile,buffers(flag));
62            READRECORD (infiles(flag), buffers(flag));
63    OD merge_files;
64
65    /* исключить файлы ввода и закрыть файл вывода */
66
67    DO
68      FOR curassoc IN inassocs;
69        DELETE (curassoc);                          /* исключить файл    */
70        DISSOCIATE (curassoc);                      /* завершить ассоциацию */
71    OD;
72    DISSOCIATE (outassoc);                          /* разъединить и закончить */
73
74  END merge_sorted_files;
```

22. Чтение файла с записями переменной длины

```
1   variable_length_records:
2   MODULE
3
4     /* Этот пример показывает, как может быть обработан файл,          */
5     /* состоящий из записей переменной длины.                          */
6     /* Файл состоит из ряда строк переменной длины;                    */
7     /* алгоритм считывает строку, выделяет для нее соответствующую ячейку */
8     /* и заносит ссылку на эту ячейку в стек                           */
9
10    NEWMODE
11      string = CHARS (80),
12       link_record = STRUCT (
13                                 next_record   REF link_record,
14                                 string_row    ROW string);
```

```
15
16    DCL
17      pushdownlist   REF link_record INIT := NULL,
18      length         INT (1:80),
19      temporaryrow   ROW string,
20      fileaccess     ACCESS string DYNAMIC,
21      association    ASSOCIATION;
22      filename       CHARS (20) VARYING INIT := "INPUT.DATA";
23    ASSOCIATE (association,filename);                    /* ассоциировать файл ввода          */
24    CONNECT (fileaccess, association, READONLY);         /* соединить доступ ввода            */
25    temporaryrow := READRECORD (fileaccess);             /* считать первую запись             */
26    DO                                                   /* пока не встретится end-of-file
27      WHILE NOT(OUTOFFILE(fileaccess));                  (признак конца файла)             */
28        pushdownlist := ALLOCATE (link_record,           /* получить новую запись компоновки  */
29                                  [pushdownlist,NULL ]); /* и инициализировать ее             */
30        length := 1 + UPPER (temporaryrow->);            /* определить длину строки           */
31        DO
32          WITH pushdowlist->;                            /* добавить новую строку к списку    */
33            string_row := ALLOCATE (CHARS (length),      /* выделить место для строки         */
34                                    temporaryrow->);     /* и занять его                      */
35        OD;
36        temporaryrow := READRECORD (fileaccess);         /* получить следующую запись в файле */
37    OD;
38    DISSOCIATE (association);                            /* конец ассоциации                  */
39
40  END variable_length_records;
```

23. Использование модулей спецификации

```
1     /* Примеры 23 и 24 являются примером 8, разделенным на две части. */
2   letter_count:
3   SPEC MODULE
4       /* Это модуль спецификации для соответствующего модуля в примере 8. */
5       SEIZE max;
6       count:
7       PROC (input ROW CHARS (max) IN, output ARRAY ('A':'Z') INT OUT) END;
8       GRANT count;
9   END letter_count;
10  letter_count: REMOTE "example 24";
11  test:
12  MODULE
13      /* Это модуль 'test' из примера 8.                    */
14      /* Он может быть скомпилирован модульным образом с    */
15      /* упомянутым выше модулем спецификации               */
16      SYNMODE results = ARRAY ('A':'Z') INT;
17      DCL c CHARS (10) INIT := "A-B<ZAA9K' ";
18      DCL output results;
19      SYN max = 10_000;
20      GRANT max;
21      SEIZE count;
22      count (-> c, output);
23      ASSERT output = results [('A') : 3, ('B', 'K', 'Z') : 1, (ELSE) : 0 ];
24  END test;
```

24. Примеры контекста

```
1   CONTEXT
2       /* Это контекст модуля "letter_count",                    */
3       /* используемый в примере 23 и позволяющий выполнить */
4       /* модульную компиляцию "letter_count".                   */
5       SYN max = 10_000;
6   FOR
7   letter_count:
8   MODULE
9       SEIZE max;
10      DCL letter POWERSET CHAR INIT := ['A' : 'Z'];
11      count:
12      PROC (input ROW CHARS (max) IN, output ARRAY ('A':'Z') INT OUT);
13          output := [(ELSE) : 0];
14          DO FOR i := 0 TO UPPER (input ->);
15              IF input -> (i) IN letter THEN
16                  output (input -> (i)) + := 1;
17              FI;
18          OD;
19      END count;
20      GRANT count;
21  END letter_count;
```

25. Использование префиксов и удаленных модулей

```
1   /* В этом примере используется модуль 'stack' из примера 27 или 28.                  */
2   /* Он показывает, как можно использовать префиксы в случае конфликтов на уровне имен.  */
3   /* Он использует удаленную конструкцию для совместного использования исходной программы.*/
4   char_stack:
5   MODULE
6       SYNMODE element = CHAR;
7       GRANT (-> stack ! char) ! ALL;
8       stack: SPEC REMOTE  "example 29";
9       stack: REMOTE       "example 27 or 28";
10  END char_stack;
11
12  int_stack:
13  MODULE
14      SYNMODE element = INT;
15      GRANT (-> stack ! int) ! ALL;
16      stack: SPEC REMOTE  "example 29";
17      stack: REMOTE       "example 27 or 28";
18  END int_stack;
19      /* Здесь 'push', 'pop' и 'element' являются видимыми, но                          */
20      /* с префиксами 'stack ! char' и 'stack ! int' для                                */
21      /* реализаций с элементом = CHAR и                                                */
22      /* элементом = INT соответственно.                                                */
23      /* Ниже приведены некоторые возможности использования разрешенных */
24      /* имен внутри модулей.                                                           */
25  MODULE
26      SEIZE ALL PREFIXED stack ;
27      DCL c CHAR;
28      int ! push (123) ;
29      char ! push ('a') ;
30      int ! pop ( ) ;
31      c = char ! elem (1) ;
```

```
32  END;
33
34  MODULE
35      SEIZE (stack ! int -> stack) ! ALL;
36      stack ! push (345) ;
37      stack ! pop ( ) ;
38  END;
```

26. Использование ввода—вывода текста

```
1   textio:
2   MODULE
3
4      /* Этот пример демонстрирует использование средств ввода—вывода текста */
5
6       DCL
7           outfile   ASSOCIATION,
8           output    TEXT (80) DYNAMIC,
9           size      INT := 12345,
10          flag      BOOL := FALSE,
11          set       SET (a,b,c) := b,
12          s1        CHARS (5) := "CHILL",
13          s2        CHARS (5) DYNAMIC := "text";
14
15      ASSOCIATE (outfile,"OUTPUT.DATA");          -- ассоциировать файл вывода
16      CREATE (outfile);                           -- создать его
17      CONNECT (output,outfile,WRITEONLY);         -- затем соединить с ячейкой текста
18      WRITETEXT (output,"%B%/",10);               -- 1010
19      WRITETEXT (output,"%C%/",set);              -- b
20      WRITETEXT (output,"size = %C%/",size);      -- размер = 12345
21      WRITETEXT (output,"%CL6%C i/o%/",s1,s2);    -- ввод—вывод текста на языке CHILL
22      WRITETEXT (output,"flag =%X%C",flag);       -- флаг = FALSE
23      size := GETTEXTINDEX (output);              -- 12
24      DISSOCIATE (outfile);
25  END textio;
```

27. Родовой стек

```
1       /* В этом примере используется родовой стек. Следует      */
2       /* заметить, что  вид элемента отсутствует.               */
3       /* Вид элемента определяется окружением.                  */
4       /* Контекст является виртуально введенным контекстом,     */
5       /* и он не имеет источника.                               */
6   CONTEXT REMOTE FOR
7   stack:
8   MODULE
9       SEIZE element;
10      NEWMODE cell = STRUCT (pred,succ REF cell,info element);
11      DCL p,last,first REF cell INIT := NULL;
12
13      push:
14      PROC (e element) EXCEPTIONS (overflow)
15          p := ALLOCATE (cell) ON (ALLOCATEFAIL): CAUSE overflow; END;
16          IF last = NULL THEN
17              first := p;
18              last := p;
```

```
19          ELSE
20              last -> .succ := p;
21              p -> .pred := last;
22              last := p;
23          FI;
24          last -> .info := e;
25          RETURN;
26      END push;
27
28      pop:
29      PROC () EXCEPTIONS (underflow)
30          IF last = NULL THEN
31              CAUSE underflow;
32          FI;
33          p:= last;
34          last := last -> .pred;
35          IF last = NULL THEN
36              first := NULL;
37          ELSE
38              last -> .succ := NULL;
39          FI;
40          TERMINATE (p);
41          RETURN;
42      END pop;
43
44      elem:
45      PROC (i INT) RETURNS (element LOC) EXCEPTIONS (bounds)
46          IF first = NULL THEN
47              CAUSE bounds;
48          FI;
49          p := first;
50          DO FOR j := 2 TO i;
51              IF p -> .succ = NULL THEN
52                  CAUSE bounds;
53              FI;
54              p := p -> .succ;
55          OD;
56          RETURN p -> .info;
57      END elem;
58
59      GRANT push,pop,elem;
60  END stack;
```

28. Абстрактный тип данных

```
1       /* В этом примере используются функции примера 27,                  */
2       /* при этом демонстрируется, как абстрактный тип данных             */
3       /* может быть реализован в языке CHILL двумя различными способами.  */
4   CONTEXT REMOTE FOR
5   stack:
6   MODULE
7       SEIZE element;
8       SYN max = 10_000, min = 1;
9       DCL  stack ARRAY (min : max) element,
10               stackindex INT INIT := min-1;
11      push:
12      PROC (e element) EXCEPTIONS (overflow)
13          IF stackindex = max THEN
```

```
14              CAUSE overflow;
15          FI;
16          stackindex +:= 1;
17          stack(stackindex) := e;
18          RETURN;
19      END push;
20      pop:
21      PROC () EXCEPTIONS (underflow)
22          IF stackindex = min THEN
23              CAUSE underflow;
24          FI;
25          stackindex-:= 1;
26          RETURN;
27      END pop;
28
29      elem:
30      PROC (i INT) RETURNS (element LOC) EXCEPTIONS (bounds)
31          IF i < min OR i > max THEN
32              CAUSE bounds;
33          FI;
34          RETURN stack(i);
35      END elem;
36
37      GRANT push,pop,elem;
38  END stacks;
```

29. Пример модуля спецификации

```
1     /* Этот модуль спецификации (SPEC MODULE) определяет интерфейс в примерах 27 и 28. */
2   stack: SPEC MODULE
3       SEIZE element;
4       push: PROC (e element) EXCEPTIONS (overflow) END;
5       pop: PROC () EXCEPTIONS (underflow) END;
6       elem: PROC (i INT) RETURNS (element LOC) EXCEPTIONS (bounds) END;
7       GRANT push,pop,elem;
8   END stack;
```

ДОБАВЛЕНИЕ E: ИЗЪЯТЫЕ ЭЛЕМЕНТЫ

Описанные ниже элементы не входят в настоящую Рекомендацию Z.200, однако они включены в Рекомендацию Z.200 Красной книги (том VI, Выпуск VI.12, 1984 год). Ниже приводится краткое описание этих элементов; полное их определение содержится в соответствующих разделах Рекомендации Z.200 1984 года, на которые имеются ссылки. Эти элементы могут также содержаться в реализациях.

1. Директива освобождения (см. раздел 2.6)

Директива освобождения освобождала строки **зарезервированных** простых имен, описанных в *списке строк зарезервированных простых* имен, поэтому их можно было определять вновь.

2. Синтаксис целых видов (см. раздел 3.4.2)

BIN был производным синтаксисом для *INT*.

3. Перечислимые виды с пропусками (см. раздел 3.4.5)

Перечислимый вид определял набор именованных и неименованных значений. Перечислимый вид был перечислимым видом **с пропусками** тогда и только тогда, когда количество его имен **элементов перечисления** было меньше **количества значений** перечислимого вида.

4. Синтаксис процедурных видов (см. раздел 3.7)

Спецификация результата без необязательной строки **зарезервированного** простого имени **RETURNS** была производным синтаксисом для *спецификации результата* со строкой имени **RETURNS**.

5. Синтаксис массивных видов (см. раздел 3.11.3)

Строка **зарезервированного** простого имени **ARRAY** была необязательной.

6. Обозначения уровневой структуры (см. раздел 3.11.5)

Уровневый структурный вид был производным синтаксисом для *вложенного структурного вида*. В обозначениях уровневой структуры полям предшествовали номера уровней. Если структура содержала поля, которые сами являются структурами или массивами структур, то была сформирована иерархия структур, а номер уровня можно было связать с каждым полем. Вместо записи вложенных структурных видов допускалась запись номера уровня перед именем поля в *уровневом структурном виде*.

7. Имена ссылок отображения (см. раздел 3.11.6)

Имена ссылок отображения могли использоваться для спецификации отображения способом, определяемым реализацией.

8. Базированные описания (см. раздел 4.1.4)

Базированное описание без *имени ячейки закрепленной или свободной ссылки* было производным синтаксисом для оператора определения синонимического вида. Базированное описание с *именем ячейки закрепленной или свободной ссылки* определяло одно или несколько имен доступа. Эти имена служили в качестве альтернативного доступа к ячейке путем разыменования ссылочного значения, содержавшегося в описанной ссылочной ячейке. Эта операция разыменования выполнялась всякий раз, когда доступ осуществлялся только через описанное **базированное** имя.

9. Символьные строковые константы (см. раздел 5.2.4.6)

Символьные строковые константы разделялись знаками апострофа. Кроме представления в печатном виде, могло использоваться шестнадцатеричное представление. Символьные строковые константы единичной длины выступали в качестве символьных литералов.

10. Обозначение адреса (см. раздел 5.3.8)

ADDR (< *ячейка* >) являлось производным синтаксисом для —> < *ячейка* >.

11. Синтаксис присваивания (см. раздел 6.2)

Символ = был производным синтаксисом для символа :=.

12. Синтаксис оператора выбора (см. раздел 6.4)

Список диапазонов оператора выбора мог быть описан в более общем виде с помощью *дискретного вида*, а не только с помощью *имени дискретного вида*.

13. Синтаксис оператора цикла с управлением по счетчику (см. раздел 6.5.2)

Диапазон в *перечислении диапазонов* оператора цикла с управлением по счетчику мог быть описан в более общем виде с помощью *дискретного вида*, а не только с помощью *имени дискретного вида*.

14. Явные счетчики циклов (см. раздел 6.5.2)

Если имя доступа было видимым в области, где находился *оператор цикла*, равный одному из имен, определяемых *счетчиками цикла*, тогда *счетчик цикла* был **явным**; в противном случае он был **неявным**. В первом случае значение счетчика цикла записывалось в обозначенную ячейку непосредственно перед аварийным завершением.

Было установлено различие между **нормальным** и **аварийным** завершением. Имело место нормальное завершение, если вычисление по крайней мере одного из счетчиков цикла указывало на завершение. Имело место аварийное завершение, если результатом выполнения while-условия было *FALSE* или если выход из оператора цикла совершался при передаче управления из оператора.

15. Синтаксис оператора вызова (см. раздел 6.7)

Строка **зарезервированного** простого имени **CALL** была необязательной. *Оператор вызова* с **CALL** был производным от *оператора вызова* без **CALL**.

16. Исключение RECURSEFAIL (см. раздел 6.7)

Если **нерекурсивная** процедура вызывала себя рекурсивным образом, то это было причиной исключения *RECURSEFAIL*.

17. Синтаксис оператора запуска (см. раздел 6.13)

Оператор запуска с опцией **SET** был производным синтаксисом для оператора однократного присваивания: < *ячейка экземпляра* > := < *выражение запуска* >

18. Явные имена принятых значений (см. раздел 6.19)

Оператор варианта получения сигнала и оператор варианта получения из буфера могли вводить имена **принятых значений**. Если имя было видимым в области, где находился *оператор варианта получения сигнала*, равный одному из имен, вводимых после атрибута **IN**, тогда имя **принятого значения** было **явным**; в противном случае оно было **неявным**. В первом случае принятое значение записывалось в обозначенную ячейку сразу же перед выполнением списка операторов действия.

19. Блоки (см. раздел 8.1)

Условный оператор, оператор выбора, оператор цикла и *оператор выбора задержки* не были определены как блоки.

20. Оператор входа (см. раздел 8.4)

Процедура могла иметь множественные точки входа, что обеспечивалось операторами входа. Эти операторы рассматривались как дополнительные определения процедуры. Определяющее вхождение в операторе входа определяло имя точки входа в процедуре, в область которой оно было помещено. Точка входа определялась текстовой позицией оператора входа.

21. Имена регистров (см. раздел 8.4)

Спецификация регистра могла даваться в формальном параметре процедуры и в спецификации результата. В случае передачи значением это означало, что фактический параметр содержался в описанном регистре; в случае передачи ячейкой это означало, что (невидимый) указатель на фактическую ячейку содержался в описанном регистре. Если спецификация содержалась в спецификации результата, это означало, что возвращаемое значение или (невидимый) указатель на возвращаемую ячейку содержался в описанном регистре.

22. Слабо видимые имена и операторы видимости (см. раздел 10.2.4.3)

Считается, что *строка имени* NS, слабо видимая в области R, занимаема модульоном M, непосредственно объемлемым в R, если NS была связана в R с *определяющим вхождением*, не окруженным областью M. Считается, что *строка имени* NS, слабо видимая в области R модульона M, разрешаема областью M, если NS была связана в R с *определяющим вхождением*, окруженным областью R.

23. Занятие именем модульона (см. раздел 10.2.4.5)

Если *предложение переименования префикса* в *операторе занятия* имело *постфикс занятия*, содержавший *строку имени* модульона и атрибут ALL, тогда *предложение переименования префикса* было эквивалентным набору *операторов занятия* для любой строки имени, которая была сильно видимой в области, непосредственно объемлющей модульон, в котором находился *оператор занятия* и был занимаем этим модульоном и которая была разрешена модульоном, приписанным *имени модульона* в области, непосредственно объемлющей модульон, в котором находился *оператор занятия*.

24. Строки предопределенных простых имен (см. раздел C.2)

AND, NOT, OR, REM, MOD, THIS и *XOR* были строками предопределенных простых имен.

ДОБАВЛЕНИЕ F: СВОДНЫЙ СИНТАКСИС

2 ПРЕДВАРИТЕЛЬНЫЕ ЗАМЕЧАНИЯ

< строка простого имени > :: =
*< буква > {< буква > | < цифра > | - }**

< буква > :: =
A | B | C | D | E | F | G | H | I | J | K | L | M
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| a | b | c | d | e | f | g | h | i | j | k | l | m
| n | o | p | q | r | s | t | u | v | w | x | y | z

< цифра > :: =
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

< комментарий > :: =
< комментарий в скобках >
| < комментарий в конце строки >

< комментарий в скобках > :: =
/ < символьная строка > */*

< комментарий в конце строки > :: =
– – < символьная строка > < конец-строки >

< символьная строка > :: =
*{ < символ > }**

< предложение директивы > :: =
< > < директива > { , < директива >} < >*

< директива > :: =
< директива реализации >

< имя > :: =
< строка имени >

< строка имени > :: =
< строка простого имени >
| < строка префиксного имени >

< строка префиксного имени > :: =
< префикс > ! < строка простого имени >

< префикс > :: =
*< простой префикс > { ! < простой префикс > }**

< простой префикс > :: =
< строка простого имени >

< определяющее вхождение > :: =
< строка простого имени >

< список определяющих вхождений > :: =
*< определяющее вхождение > { , < определяющее вхождение > }**

< имя поля > :: =
< строка простого имени >

< определяющее вхождение имени поля > :: =
< строка простого имени >

< список определяющих вхождений имен полей > :: =
*< определяющее вхождение имени поля > { ,< определяющее вхождение имени поля > }**

< имя исключения > :: =
< строка простого имени >
| < строка префиксного имени >

< имя ссылки на текст > :: =
< строка простого имени >
| < строка префиксного имени >

< определение вида > :: =
< список определяющих вхождений > = < определяющий вид >

< определяющий вид > :: =
< вид >

< оператор определения синонимического вида > :: =
SYNMODE *< определение вида >* { , *< определение вида >* }*;

< оператор определения нового вида > :: =
NEWMODE *< определение вида >* { , *< определение вида >* }*;

< вид > :: =
[READ] *< несоставной вид >*
| **[READ]** *< составной вид >*

< несоставной вид > :: =
< дискретный вид >
| *< множественный вид >*
| *< ссылочный вид >*
| *< процедурный вид >*
| *< экземплярный вид >*
| *< синхронизационный вид >*
| *< обменный вид >*
| *< временной вид >*

< дискретный вид > :: =
< целый вид >
| *< булевский вид >*
| *< символьный вид >*
| *< перечислимый вид >*
| *< ограниченный вид >*

< целый вид > :: =
< имя целого вида >

< булевский вид > :: =
< имя булевского вида >

< символьный вид > :: =
< имя символьного вида >

< перечислимый вид > :: =
SET *(< список перечисления >)*
| *< имя перечислимого вида >*

< список перечисления > :: =
< нумеруемый список перечисления >
| *< ненумеруемый список перечисления >*

< нумеруемый список перечисления > :: =
< нумеруемый элемент перечисления > { , *< нумеруемый элемент перечисления >* }*

< нумеруемый элемент перечисления > :: =
< определяющее вхождение > = < целое постоянное выражение >

< ненумеруемый список перечисления > :: =
< элемент перечисления > { , *< элемент перечисления >* }*

< элемент перечисления > :: =
< определяющее вхождение >

< ограниченный вид > :: =
< имя дискретного вида > (< постоянный диапазон >)
| **RANGE** *(< постоянное выражение >)*
| **BIN** *(< целое постоянное выражение >)*
| *< имя ограниченного вида >*

< постоянный диапазон > :: =
< нижняя граница > : < верхняя граница >

< *нижняя граница* > :: =
< *дискретное постоянное выражение* >

< *верхняя граница* > :: =
< *дискретное постоянное выражение* >

< *множественный вид* > :: =
POWERSET < *элементный вид* >
| < *имя множественного вида* >

< *элементный вид* > :: =
< *дискретный вид* >

< *ссылочный вид* > :: =
< *закрепленный ссылочный вид* >
| < *свободный ссылочный вид* >
| < *рядовый вид* >

< *закрепленный ссылочный вид* > :: =
REF < *ссылаемый вид* >
| < *имя закрепленного ссылочного вида* >

< *ссылаемый вид* > :: =
< *вид* >

< *свободный ссылочный вид* > :: =
< *имя свободного ссылочного вида* >

< *рядовый вид* > :: =
ROW < *строковый вид* >
| **ROW** < *массивный вид* >
| **ROW** < *вариантный структурный вид* >
| < *имя рядового вида* >

< *процедурный вид* > :: =
PROC ([< *список параметров* >]) [< *спецификация результата* >]
[**EXCEPTIONS** (< *список исключений* >)] [**RECURSIVE**]
| < *имя процедурного вида* >

< *список параметров* > :: =
< *спецификация параметра* > { , < *спецификация параметра* > }*

< *спецификация параметра* > :: =
< *вид* > [< *атрибут параметра* >]

< *атрибут параметра* > :: =
IN | **OUT** | **INOUT** | **LOC** [**DYNAMIC**]

< *спецификация результата* > :: =
RETURNS (< *вид* > [< *атрибут результата* >])

< *атрибут результата* > :: =
[**NONREF**] **LOC** [**DYNAMIC**]

< *список исключений* > :: =
< *имя исключения* > { ,< *имя исключения* > }*

< *экземплярный вид* > :: =
< *имя экземплярного вида* >

< *синхронизационный вид* > :: =
< *событийный вид* >
| < *буферный вид* >

< *событийный вид* > :: =
EVENT [(< *длина события* >)]
| < *имя событийного вида* >

< *длина события* > :: =
< *целое постоянное выражение* >

< *буферный вид* > :: =
BUFFER [(< *длина буфера* >)] < *буферный элементный вид* >
| < *имя буферного вида* >

< *длина буфера* > :: =
< *целое постоянное выражение* >

< *буферный элементный вид* > :: =
< *вид* >

< *обменный вид* > :: =
< *ассоциативный вид* >
| < *доступный вид* >
| < *текстовый вид* >

< *ассоциативный вид* > :: =
< *имя ассоциативного вида* >

< *доступный вид* > :: =
ACCESS [(< *индексный вид* >)] [< *вид записи* > [**DYNAMIC**]
| < *имя доступного вида* >

< *вид записи* > :: =
< *вид* >

< *индексный вид* > :: =
< *дискретный вид* >
| < *постоянный диапазон* >

< *текстовый вид* > :: =
TEXT (< *длина текста* >) [< *индексный вид* >] [**DYNAMIC**]

< *длина текста* > :: =
< *целое постоянное выражение* >

< *временной вид* > :: =
< *продолжительный вид* >
| < *абсолютно-временной вид* >

< *продолжительный вид* > :: =
< *имя продолжительного вида* >

< *абсолютно-временной вид* > :: =
< *имя абсолютно-временного вида* >

< *составной вид* > :: =
< *строковый вид* >
| < *массивный вид* >
| < *структурный вид* >

< *строковый вид* > :: =
< *тип строки* > (< *длина строки* >) [**VARYING**]
| < *параметризованный строковый вид* >
| < *имя строкового вида* >

< *параметризованный строковый вид* > :: =
< *имя начального строкового вида* > (< *длина строки* >)
| < *имя параметризованного строкового вида* >

< *имя начального строкового вида* > :: =
< *имя строкового вида* >

< *тип строки* > :: =
BOOLS
| **CHARS**

< *длина строки* > :: =
< *целое постоянное выражение* >

< *массивный вид* > :: =
ARRAY (< *индексный вид* > { , < *индексный вид* > } *)
< *вид элемента* > { < *формат элемента* > }*
| < *параметризованный массивный вид* >
| < *имя массивного вида* >

< *параметризованный массивный вид* > :: =
< *имя начального массивного вида* > (< *верхний индекс* >)

| < имя параметризованного массивного вида >

< имя начального массивного вида > :: =
< имя массивного вида >

< верхний индекс > :: =
< дискретное постоянное выражение >

< вид элемента > :: =
< вид >

< структурный вид > :: =
STRUCT (< поле > { , < поле > }*)
| < параметризованный структурный вид >
| < имя структурного вида >

< поле > :: =
< фиксированное поле >
| < альтернативное поле >

< фиксированное поле > :: =
< список определяющих вхождений имен полей > < вид > [< формат поля >]

< альтернативное поле > :: =
CASE [< список признаков >] **OF**
< вариантная альтернатива > {,< вариантная альтернатива >}*
[**ELSE** [< вариантное поле > { , < вариантное поле > }*]] **ESAC**

< вариантная альтернатива > :: =
[< спецификация метки выбора >] : [< вариантное поле > { , < вариантное поле > }*]

< список признаков > :: =
< имя поля признака > { , < имя поля признака >}*

< вариантное поле > :: =
< список определяющих вхождений имен поля > < вид > [< формат поля >]

< параметризованный структурный вид > :: =
< имя начального вариантного структурного вида > (< список постоянных выражений >)
| < имя параметризованного структурного вида >

< имя начального вариантного структурного вида > :: =
< имя вариантного структурного вида >

< список постоянных выражений > :: =
< дискретное постоянное выражение > { ,< дискретное постоянное выражение > }*

< формат элемента > :: =
PACK | **NOPACK** | < шаг >

< формат поля > :: =
PACK | **NOPACK** | < позиция >

< шаг > :: =
STEP (< позиция > [, < величина шага >])

< позиция > :: =
POS (< слово >, < начальный бит >, < длина >)
| **POS** (< слово > [,< начальный бит > [: < конечный бит >]])

< слово > :: =
< целое постоянное выражение >

< величина шага > :: =
< целое постоянное выражение >

< начальный бит > :: =
< целое постоянное выражение >

< конечный бит > :: =
< целое постоянное выражение >

< длина > :: =
< целое постоянное выражение >

< *оператор описания* > :: =
DCL < *описание* > { , < *описание* > }*;

< *описание* > :: =
< *описание ячейки* >
| < *описание опознавателя-ячейки* >

< *описание ячейки* > :: =
< *список определяющих вхождений* > < *вид* > [**STATIC**] [< *инициализация* >]

< *инициализация* > :: =
< *инициализация границей области* >
| < *инициализация границей времени жизни* >

< *инициализация границей области* > :: =
< *символ присваивания* > < *значение* > [< *программа обработки исключений* >]

< *инициализация границей времени жизни* > :: =
INIT < *символ присваивания* > < *постоянное значение* >

< *описание опознавателя-ячейки* > :: =
< *список определяющих вхождений* > < *вид* > **LOC** [**DYNAMIC**]
< *символ присваивания* > < *ячейка* > [< *программа обработки исключений* >]

< *ячейка* > :: =
< *имя доступа* >
| < *разыменованная закрепленная ссылка* >
| < *разыменованная свободная ссылка* >
| < *разыменованный ряд* >
| < *элемент строки* >
| < *подстрока* >
| < *элемент массива* >
| < *подмассив* >
| < *поле структуры* >
| < *вызов процедуры возврата ячейки* >
| < *вызов встроенной программы выдачи ячейки* >
| < *преобразование ячейки* >

< *имя доступа* > :: =
< *имя ячейки* >
| < *имя опознавателя-ячейки* >
| < *имя перечисления ячеек* >
| < *имя ячейки с присоединением* >

< *разыменованная закрепленная ссылка* > :: =
< *примитивное значение закрепленной ссылки* > —> [< *имя вида* >]

< *разыменованная свободная ссылка* > :: =
< *примитивное значение свободной ссылки* > —> < *имя вида* >

< *разыменованный ряд* > :: =
< *примитивное значение ряда* > - >

< *элемент строки* > :: =
< *ячейка строки* > (< *начальный элемент* >)

< *начальный элемент* > :: =
< *целое выражение* >

< *подстрока* > :: =
< *ячейка строки* > (< *левый элемент* > : < *правый элемент* >)
| < *ячейка строки* > (< *начальный элемент* > **UP** < *размер подстроки* >)

< *левый элемент* > :: =
< *целое выражение* >

< *правый элемент* > :: =
< *целое выражение* >

< *размер подстроки* > :: =
< *целое выражение* >

< элемент массива > :: =
< ячейка массива > (< список выражений >)

< список выражений > :: =
< выражение > { , < выражение > }*

< подмассив > :: =
< ячейка массива > (< нижний элемент > : < верхний элемент >)
| < ячейка массива > (< первый элемент > **UP** < размер подмассива >

< нижний элемент > :: =
< выражение >

< верхний элемент > :: =
< выражение >

< первый элемент > :: =
< выражение >

< поле структуры > :: =
< ячейка структуры > . < имя поля >

< вызов процедуры возврата ячейки > :: =
< вызов процедуры возврата ячейки >

< вызов встроенной программы выдачи ячейки > :: =
< вызов встроенной программы выдачи ячейки >

< преобразование ячейки > :: =
< имя вида > (< ячейка статического вида >)

5 ЗНАЧЕНИЯ И ОПЕРАЦИИ НАД НИМИ

< оператор определения синонима > :: =
SYN < определение синонима > { , < определение синонима > }*;

< определение синонима > :: =
< список определяющих вхождений > [< вид >] = < постоянное значение >

< примитивное значение > :: =
< содержимое ячейки >
| < имя значения >
| < константа >
| < кортеж >
| < элемент строки значений >
| < подстрока значений >
| < элемент массива значений >
| < подмассив значений >
| < поле структуры значений >
| < преобразование выражения >
| < вызов процедуры возврата значения >
| < вызов встроенной программы выдачи значения >
| < выражение запуска >
| < операция индикации >
| < выражение в круглых скобках >

< содержимое ячейки > :: =
< ячейка >

< имя значения > :: =
< имя синонима >
| < имя перечисления значений >
| < имя значения с присоединением >
| < имя получения значения >
| < имя общей процедуры >

< константа > :: =
< целая константа >
| < булевская константа >
| < символьная константа >

| < перечислимая константа >
| < пустая константа >
| < символьная строковая константа >
| < битовая строковая константа >

< целая константа > :: =
< десятичная целая константа >
| < двоичная целая константа >
| < восьмеричная целая константа >
| < шестнадцатеричная целая константа >

< десятичная целая константа > :: =
[{ D | d }'] { < цифра > | _ }+

< двоичная целая константа > :: =
{ B | b }' { 0 | 1 | _ }+

< восьмеричная целая константа > :: =
{ O | o }' { < восьмеричная цифра > | _ }+

< шестнадцатеричная целая константа > :: =
{ H | h }' { < шестнадцатеричная цифра > | _ }+

< шестнадцатеричная цифра > :: =
< цифра > | A | B | C | D | E | F | a | b | c | d | e | f

< восьмеричная цифра > :: =
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

< булевская константа > :: =
< имя булевской константы >

< символьная константа >
' < символ > | < управляющая последовательность > '

< перечислимая константа > :: =
< имя элемента перечисления >

< пустая константа > :: =
< имя пустой константы >

< символьная строковая константа > :: =
" { < нерезервируемый символ > | < кавычки > | < управляющая последовательность > }* "

< кавычки > :: =
""

< управляющая последовательность > :: =
^ (< целое постоянное выражение > { , < целое постоянное выражение > }*)
| ^ < неспециальный символ >
| ^^

< битовая строковая константа > :: =
< двоичная битовая строковая константа >
| < восьмеричная битовая строковая константа >
| < шестнадцатеричная битовая строковая константа >

< двоичная битовая строковая константа > :: =
{ B | b }' { 0 | 1 | _ }*

< восьмеричная битовая строковая константа > :: =
{ O | o }' { < восьмеричная цифра > | _ }*

< шестнадцатеричная битовая строковая константа > :: =
{ H | h }' { < шестнадцатеричная цифра > | _ }*

< кортеж > :: =
[< имя вида >] (: {< кортеж множества > |
< кортеж массива > | < кортеж структуры > } :)

< кортеж множества > :: =
[{ < выражение > | < диапазон > } { , { < выражение > | < диапазон > } }*]

< диапазон > :: =
< выражение > : < выражение >

< *кортеж массива* > :: =
< *кортеж непомеченного массива* >
| < *кортеж помеченного массива* >

< *кортеж непомеченного массива* > :: =
< *значение* > { , < *значение* > }*

< *кортеж помеченного массива* > :: =
< *список меток выбора* > : < *значение* > { , < *список меток выбора* > : < *значение* > }*

< *кортеж структуры* > :: =
< *кортеж непомеченной структуры* >
| < *кортеж помеченной структуры* >

< *кортеж непомеченной структуры* > :: =
< *значение* > { , < *значение* > }*

< *кортеж помеченной структуры* > :: =
< *список имен полей* > : < *значение* > { , < *список имен полей* > : < *значение* > }*

< *список имен полей* > :: =
< *имя поля* > {,. < *имя поля* > }*

< *элемент строки значений* > :: =
< *примитивное значение строки* > (< *начальный элемент* >)

< *подстрока значений* > :: =
< *примитивное значение строки* > (< *левый элемент* > : < *правый элемент* >)
| < *примитивное значение строки* > (< *начальный элемент* > **UP** < *размер подстроки* >)

< *элемент массива значений* > :: =
< *примитивное значение массива* > (< *список выражений* >)

< *подмассив значений* > :: =
< *примитивное значение массива* > (< *нижний элемент* > : < *верхний элемент* >)
| < *примитивное значение массива* > (< *первый элемент* > **UP** < *размер подмассива* >)

< *поле структуры значений* > :: =
< *примитивное значение структуры* > . < *имя поля* >

< *преобразование выражения* > :: =
< *имя вида* > (< *выражение* >)

< *вызов процедуры возврата значения* > :: =
< *вызов процедуры возврата значения* >

< *вызов встроенной программы выдачи значения* > :: =
< *вызов встроенной программы выдачи значения* >

< *выражение запуска* > :: =
START < *имя процесса* > ([< *список фактических параметров* >])

< *оператор индикации* > :: =
THIS

< *выражение в круглых скобках* > :: =
(< *выражение* >)

< *значение* > :: =
< *выражение* >
| < *неопределенное значение* >

< *неопределенное значение* > :: =
*
| < *имя неопределенного синонима* >

< *выражение* > :: =
< *операнд-0* >
| < *условное выражение* >

< *условное выражение* > :: =
| **IF** < *булевское выражение* > < *then-альтернатива* >
< *else-альтернатива* > **FI**
CASE < *список селекторов выбора* > **OF** { < *альтернатива выбора значения* > }+
[**ELSE** < *субвыражение* >] **ESAC**

< *then-альтернатива* > :: =
THEN < *субвыражение* >

< *else-альтернатива* > :: =
ELSE < *субвыражение* >
| **ELSIF** < *булевское выражение* >
< *then-альтернатива* > < *else-альтернатива* >

< *субвыражение* > :: =
< *выражение* >

< *альтернатива выбора значения* > :: =
< *спецификация метки выбора* > : < *субвыражение* >

< *операнд-0* > :: =
< *операнд-1* >
| < *субоперанд-0* > { **OR** | **ORIF** | **XOR** } < *операнд-1* >

< *субоперанд-0* > :: =
< *операнд-0* >

< *операнд-1* > :: =
< *операнд-2* >
| < *субоперанд-1* > { **AND** | **ANDIF** } < *операнд-2* >

< *субоперанд-1* > :: =
< *операнд-1* >

< *операнд-2* > :: =
< *операнд-3* >
| < *субоперанд-2* > < *операция-3* > < *операнд-3* >

< *субоперанд-2* > :: =
< *операнд-2* >

< *операция-3* > :: =
< *операция сравнения* >
| < *операция принадлежности* >
| < *операция включения множества* >

< *операция сравнения* > :: =
= | /= | > | >= | < | < =

< *операция принадлежности* > :: =
IN

< *операция включения множества* > :: =
< = | >= | < | >

< *операнд-3* > :: =
< *операнд-4* >
| < *субоперанд-3* > < *операция-4* > < *операнд-4* >

< *субоперанд-3* > :: =
< *операнд-3* >

< *операция-4* > :: =
< *арифметическая аддитивная операция* >
| < *операция конкатенации строк* >
| < *операция разности множеств* >

< *арифметическая аддитивная операция* > :: =
+ | -

< *операция конкатенации строк* > :: =
//

< *операция разности множеств* > :: =
–

< *операнд-4* > :: =
< *операнд-5* >
| < *субоперанд-4* > < *арифметическая мультипликативная операция* > < *операнд-5* >

< *субоперанд-4* > :: =

< операнд-4 >

< арифметическая мультипликативная операция > :: =
| | / | **MOD** | **REM**

< операнд-5 > :: =
[< унарная операция >] < операнд-6 >

< унарная операция > :: =
— | **NOT**
| < операция повторения строки >

< операция повторения строки > :: =
(< целое постоянное выражение >)

< операнд-6 > :: =
< ссылаемая ячейка >
| < выражение получения >
| < примитивное значение >

< ссылаемая ячейка > :: =
—> < ячейка >

< выражение получения > :: =
RECEIVE < буферная ячейка >

6 ДЕЙСТВИЯ

< оператор действия > :: =
[< определяющее вхождение > :] < действие > [< программа обработки исключений >]
[< строка простого имени >];
| < модуль >
| < модуль спецификации >
| < контекстовый модуль >

< действие > :: =
< сложный оператор >
| < оператор присваивания >
| < оператор вызова >
| < оператор выхода >
| < оператор возврата >
| < оператор результата >
| < оператор перехода >
| < оператор контроля >
| < пустой оператор >
| < оператор запуска >
| < оператор останова >
| < оператор задержки >
| < оператор продолжения >
| < оператор посылки >
| < оператор причины >

< сложный оператор > :: =
< условный оператор >
| < оператор выбора >
| < оператор цикла >
| < блок begin-end >
| < оператор выбора задержки >
| < оператор варианта получения >
| < оператор временных соотношений >

< оператор присваивания > :: =
< оператор однократного присваивания >
| < оператор многократного присваивания >

< оператор однократного присваивания > :: =
< ячейка > < символ присваивания > < значение >
| < ячейка > < операция присваивания > < выражение >

< *оператор многократного присваивания* > :: =
< *ячейка* > { , < *ячейка* > }$^{+}$ < *символ присваивания* > < *значение* >

< *операция присваивания* > :: =
< *замкнутая двоичная операция* > < *символ присваивания* >

< *замкнутая двоичная операция* > :: =
OR | **XOR** | **AND**
| < *операция разности множеств* >
| < *арифметическая аддитивная операция* >
| < *арифметическая мультипликативная операция* >
| < *операция конкатенации строк* >

< *символ присваивания* > :: =
:=

< *условный оператор* > :: =
IF < *булевское выражение* > < *then-часть* > [< *else-часть* >] **FI**

< *then-часть* > :: =
THEN < *список операторов действия* >

< *else-часть* > :: =
ELSE < *список операторов действия* >
| **ELSIF** < *булевское выражение* > < *then-часть* > [< *else-часть* >]

< *оператор выбора* > :: =
CASE < *список селекторов выбора* > **OF** [< *список диапазонов* >;] { < *альтернатива выбора* > }$^{+}$
[**ELSE** < *список операторов действия* >] **ESAC**

< *список селекторов выбора* > :: =
< *дискретное выражение* > { , < *дискретное выражение* > }*

< *список диапазонов* > :: =
< *имя дискретного вида* > { , < *имя дискретного вида* > }*

< *альтернатива выбора* > :: =
< *спецификация метки выбора* > : < *список операторов действия* >

< *оператор цикла* > :: =
DO [< *управляющая часть* > ;] < *список операторов действия* > **OD**

< *управляющая часть* > :: =
< *управление по счетчику* > [< *управление по условию* >]
| < *управление по условию* >
| < *присоединение* >

< *управление по счетчику* > :: =
FOR { < *итерация* > { , < *итерация*}* | **EVER** }

< *итерация* > :: =
< *перечисление значений* >
| < *перечисление ячеек* >

< *перечисление значений* > :: =
< *перечисление шагов* >
| < *перечисление диапазонов* >
| < *перечисление множеств* >

< *перечисление шагов* > :: =
< *счетчик цикла* > < *символ присваивания* >
< *начальное значение* > [< *значение шага* >] [**DOWN**] < *конечное значение* >

< *счетчик цикла* > :: =
< *определяющее вхождение* >

< *начальное значение* > :: =
< *дискретное выражение* >

< *значение шага* > :: =
BY < *целое выражение* >

< *конечное значение* > :: =
TO < *дискретное выражение* >

< *перечисление диапазонов* > :: =
< *счетчик цикла* > [**DOWN**] **IN** < *имя дискретного вида* >

< *перечисление множеств* > :: =
< *счетчик цикла* > [**DOWN**] **IN** < *выражение множеств* >

< *перечисление ячеек* > :: =
< *счетчик цикла* > [**DOWN**] **IN** < *составной объект*>

< *составной объект* > :: =
< *ячейка массива* >
| < *выражение массива* >
| < *ячейка строки* >
| < *выражение строки* >

< *управление по условию* > :: =
WHILE < *булевское выражение* >

< *присоединение* > :: =
WITH < *управление присоединением* > { , < *управление присоединением* > }*

< *управление присоединением* > :: =
< *ячейка структуры* >
| < *примитивное значение структуры* >

< *оператор выхода* > :: =
EXIT < *имя метки* >

< *оператор вызова* > :: =
< *вызов процедуры* >
| < *вызов встроенной программы* >

< *вызов процедуры* > :: =
{< *имя процедуры* > | < *примитивное значение процедуры* > }
([< *список фактических параметров* >])

< *список фактических параметров* > :: =
< *фактический параметр* > { , < *фактический параметр* > }*

< *фактический параметр* > :: =
< *значение* >
| < ячейка >

< *вызов встроенной программы* > :: =
< *имя встроенной программы* > ([< *список параметров встроенной программы* >])

< *список параметров встроенной программы* > :: =
< *параметр встроенной программы* > { , < *параметр встроенной программы* > }*

< *параметр встроенной программы* > :: =
< *значение* >
| < *ячейка* >
| < *нерезервируемое имя* > [(*список параметров встроенной программы* >)]

< *оператор возврата* > :: =
RETURN [< *результат* >]

< *оператор результата* > :: =
RESULT < *результат* >

< *результат* > :: =
< *значение* >
| < *ячейка* >

< *оператор перехода* > :: =
GOTO < *имя метки* >

< *оператор контроля* > :: =
ASSERT < *булевское выражение* >

< *пустой оператор* > :: =
< *пусто* >
< *пусто* > :: =

< *оператор причины* > :: =
CAUSE < *имя исключения* >

< *оператор запуска* > :: =
< *выражение запуска* >

< *оператор останова* > :: =
STOP

< *оператор продолжения* > :: =
CONTINUE < *ячейка события* >

< *оператор задержки* > :: =
DELAY < *ячейка события* > [< *приоритет* >]

< *приоритет* > :: =
PRIORITY < *целое постоянное выражение* >

< *оператор выбора задержки* > :: =
DELAY CASE [**SET** < *ячейка экземпляра* > [< *приоритет* >]; | < *приоритет* > ;]
{ < *вариант задержки* > }$^{+}$
ESAC

< *вариант задержки* > :: =
(< *список событий* >) : < *список операторов действия* >

< *список событий* > :: =
< *ячейка события* > { , < *ячейка события* > }*

< *оператор посылки* > :: =
< *оператор посылки сигнала* >
| < *оператор посылки в буфер* >

< *оператор посылки сигнала* > :: =
SEND < *имя сигнала* > [(< *значение* >] { , < *значение* > }*)
[**TO** < *примитивное значение экземпляра* >] [< *приоритет* >]

< *оператор посылки в буфер* > :: =
SEND < *буферная ячейка* > (< *значение* >) [< *приоритет* >]

< *оператор варианта получения* > :: =
< *оператор варианта получения сигнала* >
| < *оператор варианта получения из буфера* >

< *оператор варианта получения сигнала* > :: =
RECEIVE CASE [**SET** < *ячейка экземпляра* >;]
{ < *альтернатива получения сигнала* > }$^{+}$
[**ELSE** < *список операторов действия* >] **ESAC**

< *альтернатива получения сигнала* > :: =
(< *имя сигнала* > [**IN** < *список определяющих вхождений* >]) : < *список операторов действия* >

< *оператор варианта получения из буфера* > :: =
RECEIVE CASE [**SET** < *ячейка экземпляра* >;]
{ < *вариант получения из буфера* > }$^{+}$
[**ELSE** < *список операторов действия* >]
ESAC

< *вариант получения из буфера* > :: =
(< *буферная ячейка* > **IN** < *определяющее вхождение* >): < *список операторов действия* >

< *вызов встроенной CHILL-программы* > :: =
< *вызов простой встроенной CHILL-программы* >
| < *вызов встроенной CHILL-программы выдачи ячейки* >
| < *вызов встроенной CHILL-программы выдачи значения* >

< *вызов простой встроенной CHILL-программы* > :: =
< *вызов встроенной программы окончания* >
| < *вызов простой встроенной программы ввода—вывода* >
| < *вызов простой встроенной программы временных соотношений* >

< *вызов встроенной CHILL-программы выдачи ячейки* > :: =
< *вызов встроенной программы выдачи ячейки ввода—вывода* >

< вызов встроенной CHILL-программы выдачи значения > :: =
NUM (< дискретное выражение >)
| PRED (< дискретное выражение >)
| SUCC (< дискретное выражение >)
| ABS (< целое выражение >)
| CARD (< множественное выражение >)
| MAX (< множественное выражение >)
| MIN (< множественное выражение >)
| SIZE (< ячейка > { < аргумент вида > })
| UPPER (< верхний—нижний аргумент >)
| LOWER (< верхний—нижний аргумент >)
| LENGTH (< аргумент длины >)
| < вызов встроенной программы размещения >
| < вызов встроенной программы выдачи значения ввода—вывода >
| < вызов встроенной программы выдачи значения времени >

< аргумент вида > :: =
< имя вида >
| < имя массивного вида > (< выражение >)
| < имя строкового вида > (< целое выражение >)
| < имя вариантного структурного вида > (< список выражений >)

< верхний—нижний аргумент > :: =
< ячейка массива >
| < выражение массива >
| < имя массивного вида >
| < ячейка строки >
| < выражение строки >
| < имя строкового вида >
| < дискретная ячейка >
| < дискретное выражение >
| < имя дискретного вида >

< аргумент длины > :: =
< ячейка строки >
| < выражение строки >

< вызов встроенной программы размещения > :: =
GETSTACK (< аргумент вида > [, < значение >])
| ALLOCATE (< аргумент вида > [, < значение >])

< вызов встроенной программы окончания > :: =
TERMINATE (< ссылочное примитивное значение >)

7 ВВОД И ВЫВОД

< вызов встроенной программы выдачи значения ввода—вывода > :: =
< вызов встроенной программы выдачи атрибутов ассоциации >
| < вызов встроенной программы индикации ассоциации >
| < вызов встроенной программы выдачи атрибутов доступа >
| < вызов встроенной программы переноса данных из файла >
| < вызов встроенной программы получения текста >

< вызов простой встроенной программы ввода—вывода > :: =
< вызов встроенной программы диссоциации >
| < вызов встроенной программы модификации >
| < вызов встроенной программы соединения >
| < вызов встроенной программы разъединения >
| < вызов встроенной программы переноса данных в файл >
| < вызов встроенной программы переноса текста >
| < вызов встроенной программы хранения текста >

< вызов встроенной программы выдачи ячейки ввода—вывода > :: =
< вызов встроенной программы ассоциации >

< вызов встроенной программы ассоциации > :: =

ASSOCIATE (< ассоциативная ячейка > [, < список ассоциативных параметров >])

< вызов встроенной программы индикации ассоциации > :: =
ISASSOCIATED (< ассоциативная ячейка >)

< список ассоциативных параметров > :: =
*< ассоциативный параметр > { , < ассоциативный параметр > }**

< ассоциативный параметр > :: =
< ячейка >
| < значение >

< вызов встроенной программы диссоциации > :: =
DISSOCIATE (< ассоциативная ячейка >)

< вызов встроенной программы выдачи атрибутов ассоциации > :: =
EXISTING (< ассоциативная ячейка >)
| RFADABLE (< ассоциативная ячейка >)
| WRITEABLE (< ассоциативная ячейка >)
| INDEXABLE (< ассоциативная ячейка >)
| SEQUENCIBLE (< ассоциативная ячейка >)
| VARIABLE (< ассоциативная ячейка >)

< вызов встроенной программы модификации > :: =
CREATE (< ассоциативная ячейка >)
| DELETE (< ассоциативная ячейка >)
| MODIFY (< ассоциативная ячейка >) [, < список параметров модификации >])

< список параметров модификации > :: =
*< параметр модификации > { , < параметр модификации > }**

< параметр модификации > :: =
< значение >
| < ячейка >

< вызов встроенной программы соединения > :: =
CONNECT (< ячейка переноса >, < ассоциативная ячейка >,
< выражение использования > [, < где-выражение > [, < индексное выражение >]])

< ячейка переноса > :: =
< ячейка доступа >
| < ячейка текста >

< выражение использования > :: =
< выражение >

< где-выражение > :: =
< выражение >

< индексное выражение > :: =
< выражение >

< вызов встроенной программы разъединения > :: =
DISCONNECT (< ячейка переноса >)

< вызов встроенной программы выдачи атрибутов доступа > :: =
GETASSOCIATION (< ячейка переноса >)
| GETUSAGE (< ячейка переноса >)
| OUTOFFILE (< ячейка переноса >)

< вызов встроенной программы переноса данных из файла > :: =
READRECORD (< ячейка доступа > [, < индексное выражение >]
[, < ячейка памяти >])

< вызов встроенной программы переноса данных в файл > :: =
WRITERECORD (< ячейка доступа > [, < индексное выражение >],
< выражение записи >)

< ячейка памяти > :: =
< ячейка статического вида >

< выражение записи > :: =
< выражение >

< вызов встроенной программы переноса текста > :: =
READTEXT (< список аргументов ввода—вывода текста >)
| WRITETEXT (< список аргументов ввода—вывода текста >)

< список аргументов ввода—вывода текста > :: =
< аргумент текста > [, < индексное выражение >],
< аргумент формата > [, < список ввода—вывода >]

< аргумент текста > :: =
< ячейка текста >
| < ячейка символьной строки >
| < выражение символьной строки >

< аргумент формата > :: =
< выражение символьной строки >

< список ввода—вывода > :: =
< элемент списка ввода—вывода > { , < элемент списка ввода—вывода > }*

< элемент списка ввода—вывода > :: =
< аргумент значения >
| < аргумент ячейки >

< аргумент ячейки > :: =
< дискретная ячейка >
| < ячейка строки >

< аргумент значения > :: =
< дискретное выражение >
| < выражение строки >

< строки управления форматом > :: =
[< текст формата >] {< спецификация формата > [< текст формата >] }*

< текст формата > :: =
{< символ, не обозначающий проценты > | < процент > }

< процент > :: =
% %

< спецификация формата > :: =
% [< коэффициент повторения >] < элемент формата >

< коэффициент повторения > :: =
{< цифра > }+

< элемент формата > :: =
< предложение формата >
| < предложение в круглых скобках >

< предложение формата > :: =
< управляющий код > [% .]

< управляющий код > :: =
< предложение преобразования >
| < предложение редактирования >
| < предложение ввода—вывода >

< предложение в круглых скобках > :: =
(< строка управления формата > %)

< предложение преобразования > :: =
< код преобразования > { < спецификатор преобразования > }*
[< размер предложения >]

< код преобразования > :: =
B | O | H | C

< спецификатор преобразования > :: =
L | E | P < символ >

< размер предложения > ::+
{ < цифра > }+ | V

< предложение редактирования > :: =

< код редактирования > [< размер предложения >]

< код редактирования > ::=
X | < | > | T

< предложение ввода—вывода > ::=
< код ввода—вывода >

< код ввода—вывода > ::=
/ | — | + | ? | ! | =

< вызов встроенный программы получения текста > ::=
GETTEXTRECORD (< ячейка текста >)
| GETTEXTINDEX (< ячейка текста >)
| GETTEXTACCESS (< ячейка текста >)
| EOLN (< ячейка текста >)

< вызов встроенный программы хранения текста > ::=
SETTEXTRECORD (< ячейка текста > , < ячейка символьной строки >)
| SETTEXTINDEX (< ячейка текста > , < целое выражение >)
| SETTEXTACCESS (< ячейка текста > , < ячейка доступа >)

8 ОБРАБОТКА ИСКЛЮЧЕНИЙ

< программа обработки исключений > ::=
ON {< выбор исключения > }* [**ELSE** < список операторов действия >] **END**

< выбор исключения > ::=
(< список исключений >) : < список операторов действия >

9 КОНТРОЛЬ ВРЕМЕНИ

< оператор временных соотношений > ::=
< оператор относительных временных соотношений >
| < оператор абсолютных временных соотношений >
| < оператор циклических временных соотношений >

< оператор относительных временных соотношений > ::=
AFTER < примитивное значение продолжительности > [**DELAY**] **IN**
< список операторов действия > < программа обработки исключений временных соотношений > **END**

< программа обработки исключений временных соотношений > ::=
TIMEOUT < список операторов действия >

< оператор абсолютных временных соотношений > ::=
AT < примитивное значение абсолютного времени > **IN**
< список операторов действия > < программа обработки исключений временных соотношений > **END**

< оператор циклических временных соотношений > ::=
CYCLE < примитивное значение продолжительности > **IN**
< список операторов действия > **END**

< вызов встроенной программы выдачи значения времени > ::=
< вызов встроенной программы продолжительности >
| < вызов встроенной программы абсолютного времени >

< вызов встроенной программы продолжительности > ::=
MILLISECS (< целое выражение >)
| SECS (< целое выражение >)
| MINUTES (< целое выражение >)
| HOURS (< целое выражение >)
| DAYS (< целое выражение >)

< вызов встроенной программы абсолютного времени > ::=
ABSTIME ([[[[[[< выражение года > ,] < выражение месяца > ,]
< выражение дня > ,]
< выражение часа > ,]
< выражение минут > ,]
< выражение секунд >])

< выражение года > ::=

< целое выражение >

< выражение месяца > :: =
< целое выражение >

< выражение дня > :: =
< целое выражение >

< выражение часа > :: =
< целое выражение >

< выражение минут > :: =
< целое выражение >

< выражение секунд > :: =
< целое выражение >

< вызов простой встроенной программы временных соотношений > :: =
WAIT ( )
| *EXPIRED* ( )
| *INTTIME* (< примитивное значение абсолютного времени > , [[[[< ячейка года >
< ячейка месяца > ,] < ячейка дня > ,]
< ячейка часа > ,] < ячейка минут > ,]
< ячейка секунд >)

< ячейка года > :: =
< целое выражение >

< ячейка месяца > :: =
< целое выражение >

< ячейка дня > :: =
< целое выражение >

< ячейка часа > :: =
< целое выражение >

< ячейка минут > :: =
< целое выражение >

< ячейка секунд > :: =
< целое выражение >

10 СТРУКТУРА ПРОГРАММ

< тело блока begin-end > :: =
< список операторов данных > < список операторов действия >

< тело процедуры > :: =
< список операторов данных > < список операторов действия >

< тело процесса > :: =
< список операторов данных > < список операторов действия >

< тело модуля > :: =
{ < оператор данных > | < оператор видимости > < регион > |
< регион спецификации > }* < список операторов действия >

< тело региона > :: =
{ < оператор данных > | < оператор видимости > }*

< тело модуля спецификации > :: =
{ < оператор квазиданных > | < оператор видимости > * |
< модуль спецификации > | < регион спецификации > }*

< тело региона спецификации > :: =
{ < оператор квазиданных > | < оператор видимости > }*

< тело контекста > :: =
{ < оператор квазиданных > | < оператор видимости >
| < модуль спецификации > | < регион спецификации > }*

< список операторов действия > :: =
{ < оператор действия > }*

< *список операторов данных* > :: =
{ < *оператор данных* > }*

< *оператор данных* > :: =
< *оператор описания* >
| < *оператор определения* >

< *оператор определения* > :: =
< *оператор определения синонимического вида* >
| < *оператор определения нового вида* >
| < *оператор определения синонима* >
| < *оператор определения процедуры* >
| < *оператор определения процесса* >
| < *оператор определения сигнала* >
| < *пусто* >;

< *блок begin-end* > :: =
BEGIN < *тело блока begin-end* > **END**

< *оператор определения процедуры* > :: =
< *определяющее вхождение* > : < *определение процедуры* >
[< *программа обработки исключений* >] [< *строка простого имени* >] ;

< *определение процедуры* > :: =
PROC *(*[*список формальных параметров* >] *)* [< *спецификация результата* >]
[**EXCEPTIONS** *(* < *список исключений* >] < *список атрибутов процедуры* >
< *тело процедуры* > **END**

< *список формальных параметров* > :: =
< *формальный параметр* > { , < *формальный параметр* > }*

< *формальный параметр* > :: =
< *список определяющих вхождений* > < *спецификация параметра* >

< *список атрибутов процедуры* > :: =
[< *универсальность* >] [**RECURSIVE**]

< *универсальность* > :: =
GENERAL
| **SIMPLE**
| **INLINE**

< *оператор определения процесса* > :: =
< *определяющее вхождение* > : < *определение процесса* >
[< *программа обработки исключений* >] [< *строка простого имени* >] ;

< *определение процесса* > :: =
PROCESS *(*[< *список формальных параметров* >]*)* < *тело процесса* > **END**

< *модуль* > :: =
[< *контекстовый список* >] [< *определяющее вхождение* > :]
MODULE [BODY] < *тело модуля* > **END**
[< *программа обработки исключений* >] [< *строка простого имени* >] ;
| < *удаленный модульон* >

< *регион* > :: =
[< *контекстовый список* >] [< *определяющее вхождение* > :]
REGION [BODY] < *тело региона* > **END**
[< *программа обработки исключений* >] [< *строка простого имени* >] ;
| < *удаленный модульон* >

< *программа* > :: =
{ < *модуль* > | < *модуль спецификации* > < *регион* > | < *регион спецификации* > }+

< *удаленный модульон* > :: =
[< *строка простого имени* > :] **REMOTE** < *указатель модуля* >;

< *удаленная спецификация* > :: =
[< *строка простого имени* > :] **SPEC REMOTE** < *указатель модуля* >;

< *удаленный контекст* > :: =
CONTEXT REMOTE < *указатель модуля* >

[< *тело модуля контекста* >] **FOR**

< *контекстовый модуль* > :: =
CONTEXT MODULE REMOTE < *указатель модуля* > ;

< *указатель модуля* > :: =
< *символьная строковая константа* >
| < *имя ссылки на текст* >
| < *пусто* >

< *модуль спецификации* > :: =
< *модуль простой спецификации* >
| < *спецификация модуля* >
| < *удаленная спецификация* >

< *модуль простой спецификации* > :: =
[< *контекстовый список* >] [< *строка простого имени* > :] **SPEC MODULE**
< *тело модуля спецификации* > **END**
[< *строка простого имени* >] ;

< *спецификация модуля* > :: =
[< *контекстовый список* >] < *строка простого имени* > : **MODULE SPEC**
< *тело модуля спецификации* > **END** [< *строка простого имени* >] ;

< *регион спецификации* > :: =
< *регион простой спецификации* >
| < *спецификация региона* >
| < *удаленная спецификация* >

< *регион простой спецификации* > :: =
{ < *контекстовый список* >] [< *строка простого имени* > :] **SPEC REGION**
< *тело региона спецификации* > **END** [< *строка простого имени* >] ;

< *спецификация региона* > :: =
[< *контекстовый список* >] < *строка простого имени* > : **REGION SPEC**
< *тело региона спецификации* > **END** [< *строка простого имени* >] ;

< *контекстовый список* > :: =
< *контекст* > {< *контекст* > }*
| < *удаленный контекст* >

< *контекст* > :: =
CONTEXT < *тело модуля контекста* > **FOR**

< *оператор квазиданных* > :: =
< *оператор квазиописания* >
| < *оператор квазиопределения* >

< *оператор квазиописания* > :: =
DCL < *квазиописание* > { , < *квазиописание* > }*;

< *квазиописание* > :: =
< *описание квазиячейки* >
| < *описание опознавателя-квазиячейки* >

< *описание квазиячейки* > :: =
< *список определяющих вхождений* > < *вид* > **[STATIC]**

< *описание опознавателя-квазиячейки* > :: =
< *список определяющих вхождений* > < *вид* >
LOC [NONREF] [DYNAMIC]

< *оператор квазиопределения* > :: =
< *оператор определения синонимического вида* >
| < *оператор определения нового вида* >
| < *оператор определения синонима* >
| < *оператор определения квазисинонима* >
| < *оператор определения квазипроцедуры* >
| < *оператор определения квазипроцесса* >
| < *оператор определения квазисигнала* >
| < *пусто* >;

< *оператор определения квазисинонима* > :: =

SYN < *определение квазисинонима* > { , < *определение квазисинонима* > }* ;

< *определение квазисинонима* > :: =
< *список определяющих вхождений* > {< *вид* > = [< *постоянное значение* >] |
[< *вид* >] = < *постоянное выражение* > }

< *оператор определения квазипроцедуры* > :: =
< *определяющее вхождение* > : **PROC** (< *список квазиформальных параметров* >])
[< *спецификация результата* >] [**EXCEPTIONS** (< *список исключений* >)]
< *список атрибутов процедуры* > **END** [< *строка простого имени* >] ;

< *список квазиформальных параметров* > :: =
< *квазиформальный параметр* > { , < *квазиформальный параметр* > }*

< *квазиформальный параметр* > :: =
< *строка простого имени* > { , < *строка простого имени* > }* < *спецификация параметра* >

< *оператор определения квазипроцесса* > :: =
< *определяющее вхождение* > : **PROCESS** ([< *список квазиформальных параметров* >])
END [< *строка простого имени* >];

< *оператор определения квазисигнала* > :: =
SIGNAL < *определение квазисигнала* > { , < *определение квазисигнала* > }* ;

< *определение квазисигнала* > :: =
< *определяющее вхождение* > [= (< *вид* > { , < *вид* > }*)] [**TO**]

11 ПАРАЛЛЕЛЬНОЕ ВЫПОЛНЕНИЕ

< *оператор определения сигнала* > :: =
SIGNAL < *определение сигнала* > { , < *определение сигнала* > }*;

< *определение сигнала* > :: =
< *определяющее вхождение* > [= (< *вид* > { , < *вид* > }*)] [**TO** < *имя процесса* >]

12 ОБЩИЕ СЕМАНТИЧЕСКИЕ СВОЙСТВА

< *оператор видимости* > :: =
< *оператор разрешения* >
| < *оператор занятия* >

< *предложение переименования префикса* > :: =
(< *старый префикс* > < *новый префикс* >) ! < *постфикс* >

< *старый префикс* > :: =
< *префикс* >
| < *пусто* >

< *новый префикс* > :: =
< *префикс* >
| < *пусто* >

< *постфикс* > :: =
< *постфикс оператора занятия* > { , < *постфикс оператора занятия* > }*
| < *постфикс оператора разрешения* > { , < *постфикс оператора разрешения* > }*

< *оператор разрешения* > :: =
GRANT < *предложение переименования префикса* > { , < *предложение переименования префикса* > }*;
| **GRANT** < *окно оператора разрешения* > [< *префиксное предложение* >] ;

< *окно оператора разрешения* > :: =
< *постфикс оператора разрешения* > { , < *постфикс оператора разрешения* > }*

< *постфикс оператора разрешения* > :: =
< *строка имени* >
| < *строка имени нового вида* > < *предложение запрета* >
| [< *префикс* > !] **ALL**

< *префиксное предложение* > :: =
PREFIXED [< *префикс* >]

< *предложение запрета* > :: =
FORBID { < *список запрещенных имен* > | **ALL** }

< *список запрещенных имен* > :: =
(< *имя поля* > { , < *имя поля* > }*)

< *оператор занятия* > :: =
SEIZE < *предложение переименования префикса* > { , < *предложение переименования префикса* > }*;
| **SEIZE** < *окно оператора занятия* > [< *префиксное предложение* >;

< *окно оператора занятия* > :: =
< *постфикс оператора занятия* > { , < *постфикс оператора занятия* > }*

< *постфикс оператора занятия* > :: =
< *строка имени* >
| [< *префикс* > *!*] **ALL**

< *спецификация метки выбора* > :: =
< *список меток выбора* > { , < *список меток выбора* > }*

< *список меток выбора* > :: =
(< *метка выбора* > { , < *метка выбора* > }*)
| < *несоответствующая* >

< *метка выбора* > :: =
< *дискретное постоянное выражение* >
| < *постоянный диапазон* >
| < *имя дискретного вида* >
| **ELSE**

< *несоответствующая* > :: =
(*)

ДОБАВЛЕНИЕ G: УКАЗАТЕЛЬ ПРАВИЛ ВЫВОДА

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<абсолютно-временной вид>	3.11.3	27	27
<альтернатива выбора значения>	5.3.2	67	67
<альтернатива выбора>	6.4	78	78
<альтернатива получения сигнала>	6.19.2	93	93
<альтернативное поле>	3.12.4	31	31
<аргумент вида>	6.20.3	96	96, 98
<аргумент длины>	6.20.3	96	96
<аргумент значения>	7.5.3	111	111
<аргумент текста>	7.5.3	111	111
<аргумент формата>	7.5.3	111	111
<аргумент ячейки>	7.5.3	111	111
<арифметическая аддитивная операция>	5.3.6	71	71, 76
<арифметическая мультипликативная операция>	5.3.7	72	72, 76
<ассоциативный вид>	3.10.2	25	25
<ассоциативный параметр>	7.4.2	103	103
<атрибут параметра>	3.7	22	22
<атрибут результата>	3.7	22	22
<битовая строковая константа>	5.2.4.8	56	52
<блок begin-end>	10.3	130	75
<буква>	2.2	8	8
<булевская константа>	5.2.4.3	53	52
<булевский вид>	3.4.3	17	16
<буферный вид>	3.9.3	24	23
<буферный элементный вид>	3.9.3	24	24
<вариант задержки>	6.17	90	90
<вариант получения из буфера>	6.19.3	94	94
<вариантная альтернатива>	3.12.4	31	31
<вариантное поле>	3.12.4	31	31
<величина шага>	3.12.5	34	34
<верхний индекс>	3.12.3	29	29
<верхний элемент>	4.2.9	46	46, 62
<верхний—нижний аргумент>	6.20.3	96	96
<верхняя граница>	3.4.6	19	19
<вид записи>	3.10.3	25	25
<вид редактирования>	7.5.6	116	113
<вид элемента>	3.12.3	29	29
<вид>	3.3	15	13, 20, 21, 22, 24, 25, 29, 31, 39, 40, 50, 139, 140, 145
<восьмеричная битовая строковая константа>	5.2.4.8	56	56
<восьмеричная целая константа>	5.2.4.2	53	53
<восьмеричная цифра>	5.2.4.2	53	53, 56
<временной вид>	3.11.1	27	15
<выбор исключения>	8.2	120	120
<вызов встроенной CHILL-программы выдачи значения>	6.20.3	96	95
<вызов встроенной CHILL-программы выдачи ячейки>	6.20.2	95	95
<вызов встроенной CHILL-программы>	6.20	95	
<вызов встроенной программы абсолютного времени>	9.4.2	124	124
<вызов встроенной программы ассоциации>	7.4.2	103	102
<вызов встроенной программы выдачи атрибутов ассоциации>	7.4.4	104	102
<вызов встроенной программы выдачи атрибутов доступа>	7.4.8	107	102

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<вызов встроенной программы выдачи значения ввода—вывода>	7.4.1	102	96
<вызов встроенной программы выдачи значения времени>	9.4	124	96
<вызов встроенной программы выдачи значения>	5.2.13	64	51
<вызов встроенной программы выдачи ячейки ввода—вывода>	7.4.1	102	95
<вызов встроенной программы выдачи ячейки>	4.2.12	48	41
<вызов встроенной программы диссоциации>	7.4.3	103	102
<вызов встроенной программы индикации ассоциации>	7.4.2	103	102
<вызов встроенной программы модификации>	7.4.5	104	102
<вызов встроенной программы окончания>	6.20.4	98	95
<вызов встроенной программы переноса данных в файл>	7.4.9	108	102
<вызов встроенной программы переноса данных из файла>	7.4.9	108	102
<вызов встроенной программы переноса текста>	7.5.3	111	102
<вызов встроенной программы получения текста>	7.5.8	118	102
<вызов встроенной программы продолжительности>	9.4.1	124	124
<вызов встроенной программы размещения>	6.20.4	98	96
<вызов встроенной программы разъединения>	7.4.7	107	102
<вызов встроенной программы соединения>	7.4.6	105	102
<вызов встроенной программы хранения текста>	7.5.8	118	102
<вызов встроенной программы>	6.7	84	48, 64
<вызов простой встроенной CHILL-программы>	6.20.1	95	95
<вызов простой встроенной программы ввода—вывода>	7.4.1	102	95
<вызов простой встроенной программы временных соотношений>	9.4.3	125	95
<вызов процедуры возврата значения>	5.2.12	64	51
<вызов процедуры возврата ячейки>	4.2.11	48	41
<вызов процедуры>	6.7	84	48, 64, 84
<выражение в круглых скобках>	5.2.16	65	51
<выражение года>	9.4.2	124	124
<выражение дня>	9.4.2	124	124
<выражение записи>	7.4.9	108	108
<выражение запуска>	5.2.14	65	51, 88
<выражение использования>	7.4.6	105	105
<выражение месяца>	9.4.2	124	124
<выражение минут>	9.4.2	125	124
<выражение получения>	5.3.9	74	74
<выражение секунд>	9.4.2	125	124
<выражение часа>	9.4.2	125	124
<выражение>	5.3.2	67	18, 19, 24, 26, 28, 29, 31, 34, 44, 45, 46, 55, 56, 63, 65, 66, 67, 73, 75, 77, 78, 80, 82, 87, 89, 96, 105, 108, 111, 118, 124, 125, 140, 164
<где-выражение>	7.4.6	105	105
<двоичная битовая строковая константа>	5.2.4.8	56	56
<двоичная целая константа>	5.2.4.2	53	53
<действие>	6.1	75	75
<десятичная целая константа>	5.2.4.2	53	53
<диапазон>	5.2.5	56	56

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<директива реализации>			10
<директива>	2.6	10	10
<дискретный вид>	3.4.1	16	15
<длина буфера>	3.9.3	24	24
<длина события>	3.9.2	24	24
<длина строки>	3.12.2	28	28
<длина текста>	3.10.4	26	26
<длина>	3.12.5	34	34
<доступный вид>	3.10.3	25	25
<закрепленный ссылочный вид>	3.6.2	21	20
<замкнутая двоичная операция>	6.2	76	76
<значение шага>	6.5.2	80	80
<значение>	5.3.1	66	39, 50, 56, 57, 75, 84, 86, 91, 92, 98, 103, 104, 140
<имя доступа>	4.2.2	42	41
<имя значения>	5.2.3	51	50
<имя исключения>	2.7	10	22, 88
<имя начального вариантного структурного вида>	3.12.4	31	31
<имя начального массивного вида>	3.12.3	29	29
<имя начального строкового вида>	3.12.2	28	28
<имя поля>	2.7	10	31, 47, 57, 63, 160
<имя ссылки на текст>	2.7	10	137
<имя>	2.7	10	16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 31, 42, 43, 49, 51, 53, 54, 56, 63, 65, 66, 78, 80, 83, 84, 87, 91, 93, 96, 145, 164
<индексное выражение>	7.4.6	105	105, 108, 111
<индексный вид>	3.10.3	25	25, 26, 29
<инициализация границей времени жизни>	4.1.2	39	39
<инициализация границей области>	4.1.2	39	39
<инициализация>	4.1.2	39	39
<итерация>	6.5.2	80	80
<кавычки>	5.2.4.7	55	55
<квазиописание>	10.10.3	139	139
<квазиформальный параметр>	10.10.3	140	140
<код ввода—вывода>	7.5.7	117	117
<код преобразования>	7.5.5	114	114
<код редактирования>	7.5.6	116	116
<комментарий в конце строки>	2.4	9	9
<комментарий в скобках>	2.4	9	9
<комментарий>	2.4	9	
<конец-строки>			9
<конечное значение>	6.5.2	80	80
<конечный бит>	3.12.5	34	34
<константа>	5.2.4.1	52	50
<контекст>	10.10.2	138	138
<контекстовый модуль>	10.10.1	137	75
<контекстовый список>	10.10.2	138	134, 135, 138
<кортеж массива>	5.2.5	56	56
<кортеж множества>	5.2.5	56	56
<кортеж непомеченного массива>	5.2.5	56	56
<кортеж непомеченной структуры>	5.2.5	56	56
<кортеж помеченного массива>	5.2.5	56	56
<кортеж помеченной структуры>	5.2.5	57	56

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<кортеж структуры>	5.2.5	56	56
<кортеж>	5.2.5	56	50
<коэффициент повторения>	7.5.4	113	113
<левый элемент>	4.2.7	45	45, 60
<массивный вид>	3.12.3	29	28
<метка выбора>	12.3	164	164
<множественный вид>	3.5	20	15
<модуль простой спецификации>	10.10.2	138	138
<модуль спецификации>	10.10.2	138	75, 128, 135
<модуль>	10.6	134	75, 135
<начальное значение>	6.5.2	80	80
<начальный бит>	3.12.5	34	34
<начальный элемент>	4.2.6	44	44, 45, 60
<ненумеруемый список перечисления>	3.4.5	18	18
<неопределенное значение>	5.3.1	66	66
<несоответствующая>	12.3	164	164
<несоставной вид>	3.3	15	15
<нижний элемент>	4.2.9	46	46, 62
<нижняя граница>	3.4.6	19	19
<новый префикс>	12.2.3.3	158	158
<нумеруемый список перечисления>	3.4.5	18	18
<нумеруемый элемент перечисления>	3.4.5	18	18
<обменный вид>	3.10.1	25	15
<ограниченный вид>	3.4.6	19	16
<окно оператора занятия>	12.2.3.5	161	161
<окно оператора разрешения>	12.2.3.4	160	159
<операнд-0>	5.3.3	68	67, 68
<операнд-1>	5.3.4	69	68, 69
<операнд-2>	5.3.5	69	69
<операнд-3>	5.3.6	71	69, 71
<операнд-4>	5.3.7	72	71, 72
<операнд-5>	5.3.8	73	72
<операнд-6>	5.3.9	74	73
<оператор абсолютных временных соотношений>	9.3.2	123	122
<оператор варианта получения из буфера>	6.19.3	94	92
<оператор варианта получения сигнала>	6.19.2	93	92
<оператор варианта получения>	6.19.1	92	75
<оператор видимости>	12.2.3.2	158	128
<оператор возврата>	6.8	86	75
<оператор временных соотношений>	9.3	122	75
<оператор выбора задержки>	6.17	90	75
<оператор выбора>	6.4	78	75
<оператор вызова>	6.7	84	75
<оператор выхода>	6.6	83	75
<оператор данных>	10.2	128	128
<оператор действия>	6.1	75	128
<оператор задержки>	6.16	89	75
<оператор занятия>	12.2.3.5	161	158
<оператор запуска>	6.13	88	75
<оператор квазиданных>	10.10.3	139	128
<оператор квазиописания>	10.10.3	139	139
<оператор квазиопределения>	10.10.3	139	139
<оператор контроля>	6.10	87	75
<оператор многократного присваивания>	6.2	75	75
<оператор однократного присваивания>	6.2	75	75

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<оператор описания>	4.1.1	39	128
<оператор определения квазипроцедуры>	10.10.3	140	139
<оператор определения квазипроцесса>	10.10.3	140	139
<оператор определения квазисигнала>	10.10.3	140	139
<оператор определения квазисинонима>	10.10.3	140	139
<оператор определения нового вида>	3.2.3	14	128, 139
<оператор определения процедуры>	10.4	131	128
<оператор определения процесса>	10.5	133	128
<оператор определения сигнала>	11.5	145	128
<оператор определения синонима>	5.1	50	128, 139
<оператор определения синонимического вида>	3.2.2	14	128, 139
<оператор определения>	10.2	128	128
<оператор останова>	6.14	88	75
<оператор относительных временных соотношений>	9.3.1	122	122
<оператор перехода>	6.9	87	75
<оператор посылки в буфер>	6.18.3	92	91
<оператор посылки сигнала>	6.18.2	91	91
<оператор посылки>	6.18.1	91	75
<оператор присваивания>	6.2	75	75
<оператор причины>	6.12	88	75
<оператор продолжения>	6.15	88	75
<оператор разрешения>	12.2.3.4	159	158
<оператор результата>	6.8	86	75
<оператор цикла>	6.5.1	79	75
<оператор циклических временных соотношений>	9.3.3	123	122
<операция включения множества>	5.3.5	70	69
<операция индикации>	5.2.15	65	51
<операция конкатенации строк>	5.3.6	71	71, 76
<операция повторения строки>	5.3.8	73	73
<операция принадлежности>	5.3.5	70	69
<операция присваивания>	6.2	76	75
<операция разности множеств>	5.3.6	71	71, 76
<операция сравнения>	5.3.5	70	69
<операция-3>	5.3.5	69	69
<операция-4>	5.3.6	71	71
<описание квазиячейки>	10.10.3	139	139
<описание опознавателя квазиячейки>	10.10.3	139	139
<описание опознавателя-ячейки>	4.1.3	40	39
<описание ячейки>	4.1.2	39	39
<описание>	4.1.1	39	39
<определение вида>	3.2.1	13	14
<определение квазисигнала>	10.10.3	140	140
<определение квазисинонима>	10.10.3	140	140
<определение процедуры>	10.4	131	131
<определение процесса>	10.5	133	133
<определение сигнала>	11.5	145	145
<определение синонима>	5.1	50	50
<определяющее вхождение имени поля>	2.7	10	10
<определяющее вхождение>	2.7	10	10, 18, 75, 80, 94, 131, 133, 134, 135, 140, 145
<определяющий вид>	3.2.1	13	13
<параметр встроенной программы>	6.7	84	84
<параметр модификации>	7.4.5	104	104
<параметризованный массивный вид>	3.12.3	29	29
<параметризованный строковой вид>	3.12.2	28	28
<параметризованный структурный вид>	3.12.4	31	31
<первый элемент>	4.2.9	46	46, 62
<перечисление диапазонов>	6.5.2	80	80

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<перечисление значений>	6.5.2	80	80
<перечисление множеств>	6.5.2	80	80
<перечисление шагов>	6.5.2	80	80
<перечисление ячеек>	6.5.2	80	80
<перечислимая константа>	5.2.4.5	54	52
<перечислимый вид>	3.4.5	18	16
<подмассив значений>	5.2.9	62	50
<подмассив>	4.2.9	46	41
<подстрока значений>	5.2.7	60	50
<подстрока>	4.2.7	45	41
<позиция>	3.12.5	34	34
<поле структуры значений>	5.2.10	63	50
<поле структуры>	4.2.10	47	41
<поле>	3.12.4	31	31
<постоянный диапазон>	3.4.6	19	19, 25, 164
<постфикс оператора занятия>	12.2.3.5	161	158, 161
<постфикс оператора разрешения>	12.2.3.4	160	158, 160
<постфикс>	12.2.3.3	158	158
<правый элемент>	4.2.7	45	45, 60
<предложение в круглых скобках>	7.5.4	113	113
<предложение ввода—вывода>	7.5.7	117	113
<предложение директивы>	2.6	10	
<предложение запрета>	12.2.3.4	160	160
<предложение переименования префикса>	12.2.3.3	158	159, 161
<предложение преобразования>	7.5.5	114	113
<предложение формата>	7.5.4	113	113
<преобразование выражения>	5.2.11	63	50
<преобразование ячейки>	4.2.13	49	41
<префикс>	2.7	10	10, 158, 160, 161
<префиксное предложение>	12.2.3.4	160	159, 161
<примитивное значение>	5.2.1	50	42, 43, 60, 61, 62, 63, 74, 83, 84, 91, 98, 122, 123, 125
<приоритет>	6.16	89	89, 90, 91, 92
<присоединение>	6.5.4	83	79
<программа обработки исключений временных соотношений>	9.3.1	122	122, 123
<программа обработки исключений>	8.2	120	39, 40, 75, 131, 133, 134, 135
<программа>	10.8	135	
<продолжительный вид>	3.11.2	27	27
<простой префикс>	2.7	10	10
<процедурный вид>	3.7	22	15
<процент>	7.5.4	113	113
<пустая константа>	5.2.4.6	54	52
<пусто>	6.11	87	87, 128, 137, 139, 158
<пустой оператор>	6.11	87	75
<размер подстроки>, <размер подмассива>	4.2.7, 4.2.9	45, 46	45, 46, 60, 62
<размер предложения>	7.5.5	114	114, 116
<разыменованная закрепленная ссылка>	4.2.3	42	41
<разыменованная свободная ссылка>	4.2.4	43	41
<разыменованный ряд>	4.2.5	43	41
<регион простой спецификации>	10.10.2	138	138
<регион спецификации>	10.10.2	138	128, 135
<регион>	10.7	135	128, 135
<результат>	6.8	86	86
<рядовый вид>	3.6.4	21	20
<свободный ссылочный вид>	3.6.3	21	20
<символ присваивания>	6.2	76	39, 40, 75, 76, 80

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<символ>			9, 54, 55, 113, 114
<символьная константа>	5.2.4.4	54	52
<символьная строка>	2.4	9	9
<символьная строковая константа>	5.2.4.7	55	52, 137
<символьный вид>	3.4.4	17	16
<синхронизационный вид>	3.9.1	23	15
<слово>	3.12.5	34	34
<сложный оператор>	6.1	75	75
<событийный вид>	3.9.2	24	23
<содержимое ячейки>	5.2.2	51	50
<составной вид>	3.12.1	28	15
<составной объект>	6.5.2	80	80
<спецификатор преобразования>	7.5.5	114	114
<спецификация метки выбора>	12.3	164	31, 67, 78
<спецификация модуля>	10.10.2	138	138
<спецификация параметра>		22	22, 131, 140
<спецификация региона>	10.10.2	138	138
<спецификация результата>	3.7	22	22, 131, 140
<спецификация формата>	7.5.4	113	113
<список аргументов ввода—вывода текста>	7.5.3	111	111
<список ассоциативных параметров>	7.4.2	103	103
<список атрибутов процедуры>	10.4	131	131, 140
<список ввода—вывода>	7.5.3	111	111
<список выражений>	4.2.8	46	46, 61, 96
<список диапазонов>	6.4	78	78
<список запрещенных имен>	12.2.3.4	160	160
<список имен полей>	5.2.5	57	57
<список исключений>	3.7	22	22, 120, 131, 140
<список квазиформальных параметров>	10.10.3	140	140
<список меток выбора>	12.3	164	56, 164
<список операторов данных>	10.2	128	128
<список операторов действия>	10.2	128	77, 78, 79, 90, 93, 94, 120, 122, 123, 128
<список определяющих вхождений имен полей>	2.7	10	31
<список определяющих вхождений>	2.7	10	13, 39, 40, 50, 93, 131, 139, 140
<список параметров встроенной программы>	6.7	84	84
<список параметров модификации>	7.4.5	104	104
<список параметров>	3.7	22	22
<список перечисления>	3.4.5	18	18
<список постоянных выражений>	3.12.4	31	31
<список признаков>	3.12.4	31	31
<список селекторов выбора>	6.4	78	67, 78
<список событий>	6.17	90	90
<список фактических параметров>	6.7	84	65, 84
<список формальных параметров>	10.4	131	131, 133
<ссылаемая ячейка>	5.3.9	74	74
<ссылаемый вид>	3.6.2	21	21
<ссылочный вид>	3.6.1	20	15
<старый префикс>	12.2.3.3	158	158
<строка имени>	2.7	10	10, 160, 161
<строка префиксного имени>	2.7	10	10
<строка простого имени>	2.2	8	10, 75, 131, 133, 134, 135, 136, 138, 140
<строка управления форматом>	7.5.4	113	113
<строковой вид>	3.12.2	28	28
<структурный вид>	3.12.4	31	28
<субвыражение>	5.3.2	67	67
<субоперанд-0>	5.3.3	68	68

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<субоперанд-1>	5.3.4	69	69
<субоперанд-2>	5.3.5	69	69
<субоперанд-3>	5.3.6	71	71
<субоперанд-4>	5.3.7	72	72
<счетчик цикла>	6.5.2	80	80
<текст формата>	7.5.4	113	113
<текстовый вид>	3.10.4	26	25
<тело блока begin-end>	10.2	128	130
<тело модуля контекста>	10.2	128	137, 138
<тело модуля спецификации>	10.2	128	138
<тело модуля>	10.2	128	134
<тело процедуры>	10.2	128	131
<тело процесса>	10.2	128	133
<тело региона спецификации>	10.2	128	138
<тело региона>	10.2	128	135
<тип строки>	3.12.2	28	28
<удаленная спецификация>	10.10.1	136	138
<удаленный контекст>	10.10.1	137	138
<удаленный модульон>	10.10.1	136	134, 135
<указатель модуля>	10.10.1	137	136, 137
<унарная операция>	5.3.8	73	73
<универсальность>	10.4	131	131
<управление по счетчику>	6.5.2	80	79
<управление по условию>	6.5.3	82	79
<управление присоединением>	6.5.4	83	83
<управляющая последовательность>	5.2.4.7	55	54, 55
<управляющая часть>	6.5.1	79	79
<управляющий код>	7.5.4	113	113
<условное выражение>	5.3.2	67	67
<условный оператор>	6.3	77	75
<фактический параметр>	6.7	84	84
<фиксированное поле>	3.12.4	31	31
<формальный параметр>	10.4	131	131
<формат поля>	3.12.5	34	31
<формат элемента>	3.12.5	34	29
<целая константа>	5.2.4.2	53	52
<целый вид>	3.4.2	16	16
<цифра>	2.2	8	8, 53, 113, 114
<шаг>	3.12.5	34	34
<шестнадцатеричная битовая строковая константа>	5.2.4.8	56	56
<шестнадцатеричная целая константа>	5.2.4.2	53	53
<шестнадцатеричная цифра>	5.2.4.2	53	53, 56
<экземплярный вид>	3.8	23	15
<элемент массива значений>	5.2.8	61	50
<элемент массива>	4.2.8	46	41
<элемент перечисления>	3.4.5	18	18
<элемент списка ввода—вывода>	7.5.3	111	111
<элемент строки значений>	5.2.6	60	50
<элемент строки>	4.2.6	44	41
<элемент формата>	7.5.4	113	113
<элементный вид>	3.5	20	20
<ячейка года>	9.4.3	125	125
<ячейка дня>	9.4.3	125	125
<ячейка месяца>	9.4.3	125	125
<ячейка минут>	9.4.3	125	125
<ячейка памяти>	7.4.9	108	108
<ячейка переноса>	7.4.6	105	105, 107
<ячейка секунд>	9.4.3	125	125

Нетерминальные символы	Раздел	Страница определения	Используется на стр.
<ячейка часа>	9.4.3	125	125
<ячейка>	4.2.1	41	40, 44, 45, 46, 47, 49, 51, 74, 75, 80, 83, 84, 86, 88, 89, 90, 92, 93, 94, 96, 103, 104, 105, 108, 111, 118, 125
<else-альтернатива>	5.3.2	67	67
<else-часть>	6.3	77	77
<then-альтернатива>	5.3.2	67	67
<then-часть>	6.3	77	77

ДОБАВЛЕНИЕ H: АЛФАВИТНЫЙ УКАЗАТЕЛЬ

На странице, указанной жирным шрифтом, содержится определяющее вхождение данного элемента алфавитного указателя; обычным шрифтом обозначены страницы, содержащие вхождение с использованием данного элемента алфавитного указателя.

ISBN 92-61-03804-2