Язык Ci. Описание языка.

1 ОБЩИЕ ПРИНЦИПЫ

Язык Ci предназначен для создания прикладных программ контроля и диагностики (в дальнейшем ППКД). По сравнению с языками, такими как C, Pascal, C++, Java, C# и другими, при использовании этого языка не требуются навыки системного программирования. В то же время, Ci более гибкий и мощный язык, чем Basic, часто применяемый для подобных целей. Программа на Ci выполняется достаточно быстро для решения основных задач ППКД. По сравнению с программой на C без оптимизации, процессор языка Ci примерно в 25-30 раз медленнее на тестах, содержащих арифметические вычисления. Критичные для времени выполнения операции могут быть реализованы на языке C и в дальнейшем использованы в программах на Ci.

В отличие от языка C, программа на Ci защищена от ошибок прикладного программиста. Ошибка в C программе, не обнаруженная компилятором C, приведет к фатальному завершению всего программного продукта, чаще всего, без достаточной диагностики для локализации ошибки. Аварийное завершение работы системы в ряде случаев может привести к аварии более крупного масштаба, особенно, если комплекс используется для автоматизации управления или диагностики реальных систем. Ошибка в Ci программе, не обнаруженная компилятором Ci, приведет к ее завершению, выдаче соответствующей диагностики и указанию места ошибки. При этом программная система управления или диагностики реальной системы сохранит возможность аварийного управления. Программисту на Ci нет необходимости заботиться о выделении и освобождении памяти, при этом он оперирует с объектами, имеющими определенные свойства. В этом язык Ci сходен с объектно-ориентированными языками программирования.

Чтобы упростить использование объектной модели, в Ci принят нетрадиционный для объектно-ориентированных языков способ формирования отношений между классами. Вместо того, чтобы предлагать программисту громоздкое дерево отношений между родительскими и наследующими классами, ему предлагается набор типов данных и набор операций для работы с этими типами, подобно тому, как

это сделано в обычных алгоритмических языках. Все отношения между классами

инкапсулированы внутри типов данных Ci. Типы данных в Ci могут быть сколь угодно сложными, но на них автоматически распространяются все механизмы поддержки, заложенные в компиляторе – контроль типов, автоматические преобразования.

В настоящем руководстве дается описание языка и техники программирования на Ci, приводятся справочные сведения об основных операциях и типах данных. Руководство предназначено для прикладных программистов, разрабатывающих программы на Ci. Техника подключения операций и типов данных описаны в "ASI-Ci. Руководство системного программиста".

2 Структура Ci-программы и основные понятия

2.1 Соглашение об именах и семантика

Ci-программа распознается компилятором, как поток символов, содержащий *имена*, *спецсимволы* и *разделители*.

Имена состоят из одного или нескольких символов, за исключением спецсимволов. В языке Ci, в отличие от языка C, правила формирования имен более демократичны. В имени можно использовать любые символы любого алфавита, за исключением следующих ограничений:

- имя не должно содержать один из спецсимволов: **( ) { } [ ] ` " . , ; :**
- имя не должно начинаться с цифры или знака + или -

Примечание. В базовой текущей версии Ci символы точки и двоеточия являются зарезервированными, хотя их можно использовать в именах переменных, лучше этого не делать, чтобы не порождать несовместимость с последующими версиями Ci.

Имена используются для обозначения переменных, литеральных констант и операций (подробнее об этих элементах см. далее).

Разделители – это символы, которые используются для разделения имен. Их несколько, в первую очередь, это пробел (символ с кодом 0x20) и запятая. Они про-

сто пропускаются компилятором. Запятую можно использовать для визуального разделения, например, параметров операций:

```
uint a b c d;
```

эквивалентно

```
uint a, b, c, d;
```

Также, в качестве разделителя, можно использовать невидимый символ пустой строки и символ табуляции:

```
uint a
	b
	c		d;
```

эквивалентно

```
uint a,b,c,d;
```

Общие правила использования разделителей:

- разделители должны обязательно присутствовать между именами;

- разделители должны обязательно присутствовать между числовыми значениями, а также между именами и числовыми значениями;

- разделители не обязательно включать между именами и спецсимволами, между числовыми значениями и спецсимволами, а также между спецсимволами;

- разделителей может быть сколько угодно много.

Компилятор чувствителен к регистру символов, которыми набраны имена. Это означает, что одно и то же слово, набранное в разных регистрах, может использоваться для формирования разных имен. Например, имена value, Value и VALUE – это разные имена. Кроме этого, допускается создание переменных, имена которых совпадают с именами операций, но набраны в других регистрах. Например, допустимо объявление переменных с именами Puts, CONCAT и т.д. Однако это следует использовать с большой осторожностью, чтобы не запутаться в коде программы.

В любом месте программы можно использовать комментарии. Они ограничиваются одним символом обратной кавычки ` (код 0x60). Текст между двумя такими символами компилятором пропускается. Это можно использовать для временного выключения кода из компиляции, но с осторожностью, поскольку вложенные друг в друга комментарии не поддерживаются. Если требуется сделать комментарий

в одной строке, то удобнее воспользоваться двойной обратной кавычкой. Весь текст, от двух подряд символов `` и до конца строки будет пропущен компилятором. Например:

uint a,b,c,d; ``создание переменных a b c и d

Если необходимо вставить символ обратной кавычки в строчную константу, то перед ним надо в строке указать символ обратной дробной черты. Например:
puts "символ \` обратной кавычки" ; ``пример вставки символа ` в строку.

2.2 Операции и их параметры

Ci-программа состоит из последовательности вызовов операций, выполняющихся друг за другом в рамках программного блока.

Вызов операции состоит из имени операции и списка ее параметров, разделенных друг от друга пробелами (или другими разделителями). Пример вызова операции:

operation param1 param2;

Операции бывают двух основных видов – не возвращающие значения и возвращающие значения. При этом поддерживаются операции смешанного вида – с возвратом значения, которое у некоторых операций допускается проигнорировать. Вызов операции, не возвращающей значение, должен завершаться точкой с запятой, она служит признаком окончания списка параметров. Точка с запятой может быть опущена, если операция является последней в программном блоке и следом за ней записывается закрывающая фигурная скобка, или находится конец файла текста Ci-программы. Операции, возвращающие значение, записываются вместе с их параметрами между открывающий и закрывающей круглыми скобками, и могут быть использованы на месте параметра другой операции (за исключением случаев, когда значения параметра изменяется вызываемой операцией, об этом сказано далее). Пример вызовов операций, возвращающих значения:

operation0 (operation1 param11 (operation2 param21) param13);

как видно из примера, на месте второго параметра у operation1 находится вызов опе-

рации operation2, имеющей параметр param21, и возвращающей значение. А значение, возвращаемое операцией operation1, используется в качестве параметра у operation0. Эта техника дает возможность составлять арифметические выражения любого уровня вложенности. Например:

(((a + b) / (c ^ 2)) * PI), где

a b c – переменные;

+ / ^ * – это операции;

PI – константа.

Примечание. В примере использованы вызовы операций с инфиксной записью, о них речь далее.

Необходимо понимать, что **круглые скобки имеют отношение к той операции, которая используется внутри них**. Это существенное отличие от вызова функций с параметрами в языке C. В программе на языке C в строке

```
puts( a + b );
```

скобки являются частью вызова функции puts, ограничивая список ее параметров. В Ci они являются частью операции + и сообщают компилятору о том, что вызываемая операция + должна вернуть значение. Поэтому в Ci корректнее записать эту же строку несколько иначе:

```
puts (a + b);
```

Впрочем, на результат это не повлияет. Кроме этого, в C оператор + распознается компилятором, как арифметическое действие, и он может быть записан слитно с именами или константами. В языке Ci символ + это имя операции сложения, и согласно описанным выше правилам для разделителей, оно должно быть отделено от других имен и числовых констант пробелами. В противном случае, если в последнем примере записать puts (a+b) то компилятор сочтет a+b именем операции, возвращающей значение, поскольку оно записано в круглых скобках. Если такой операции нет, компилятор выдаст сообщение об ошибке.

Возвращаемое значение некоторых операций может быть проигнорировано. Например, операция присваивания может возвращать значение, если она вызвана на

месте параметра другой операции:

a = b; `корректно`

a = (x + (c = d)); `корректно, эквивалентно c = d; a = (x + c);`

Необходимо помнить, что одна и та же операция, вызываемая с использованием, либо с игнорированием значения, **может работать по-разному**.

Операции могут иметь различное фиксированное количество параметров: от 0, до максимального числа, установленного для каждой операции в отдельности. Поддерживается также переменное число параметров у отдельных операций. Значения параметров операции вычисляются **по очереди слева направо**. Если в качестве параметров используются вызовы других операций, то сначала будут вычислены значения всех их возвратов, и только затем начнется выполнение операции. То есть, выражение вида:

op1 par11 (op2) par 13 (op3);

будет выполняться в следующем порядке:

- вызов операции op2;
- невидимой временной переменной tmp1 присваивается возвращаемое значение op2;
- вызов операции op3;
- невидимой временной переменной tmp2 присваивается возвращаемое значение op3;
- вызов операции op1 par11 tmp1 par13 tmp2.

Прикладному Ci-программисту нет необходимости заботиться о временных переменных, создаваемых компилятором для получения результатов операций и передачи их в качестве параметров другим операциям. Они создаются и удаляются автоматически.

Некоторые операции Ci могут использоваться как в *инфиксной*, так и *префиксной* форме записи. В инфиксной форме имя операции записывается после ее первого параметра, остальные параметры записываются после имени, например:

par1 op par2. В префиксной форме имя записывается перед всеми параметрами: op par1 par2. Инфиксная форма вызова операции с двумя параметрами выглядит точно также, как использование операторов в обычных языках программирования. Практически все операции, выполняющие арифметические вычисления, могут записываться в инфиксной форме, например: a = (b + c). Префиксная форма похожа на вызов функции в языке C, за тем исключением, что скобки, ограничивающие список

параметров, не используются, список ограничивается либо точкой с запятой, либо закрывающей скобкой. Все операции, для которых допустима инфиксная форма записи, могут быть записаны в префиксной форме. Это означает, что выражения
a = (b + c) ``и
= a (+ b c) ``эквивалентны.

Необходимо помнить, что одна и та же операция, вызываемая в инфиксной или постфиксной форме, **может работать по-разному**. Например:

a = (++ i); `` увеличит значение переменной i на 1, и присвоит полученное значение переменной a;

a = (i ++); `` присвоит переменной a значение переменной i и потом увеличит его на 1.

В качестве отдельных параметров многих операций можно использовать только переменные. Это означает, что операция не просто использует значение переменной, но и может его изменять. Это подобно передаче адреса переменной в языке C. Наиболее часто используемой такой операцией является присвоение:

a = b; `или` = a b;

здесь первым параметром указана переменная a, которой присваивается значение b. Операция присвоения требует первым параметром обязательно переменную, в которую она скопирует значение, полученное ей вторым параметром. Если для операции требуется только переменная, но указана константа или вызов другой операции, то будет выдано сообщение об ошибке во время компиляции. Элемент массива также является переменной, содержащей какое-то значение, и его допустимо использовать везде, где допустима простая переменная. Однако целиком массив

– это не переменная, его можно передавать только тем операциям, в описании которых сказано, что они могут принимать целиком массив.

В дальнейшем под понятием *операнд* понимается вариант параметра операции – константа, переменная или возвращаемое значение другой операцией. В случаях, когда операция принимает параметром любой из перечисленных вариантов, говорится, что допустим любой операнд.

В языке Ci поддерживаются операции с переменным числом параметров. По умолчанию максимальное число параметров у операций равно 32. Это довольно много, поскольку программы, где используются функции с числом параметров более 4-х, трудно отлаживать, и чаще выгоднее использовать в таких случаях глобальные переменные. Однако, при необходимости, в конкретной реализации процессора Ci максимальное число параметров может быть увеличено до числа, равного значению максимального целого числа для используемой платформы.

У различных операций максимальное число параметров может быть различным. По числу параметров операции можно разделить на два вида – с фиксированным числом параметров и с переменным числом от минимального до максимального. При этом могут быть операции, минимальное число параметров которых равно 0, то есть, возможен их вызов без параметров. А их максимальное число параметров меньше максимально допустимого по умолчанию. Попытка передачи параметров больше допустимого для операции числа приводит к сообщению об ошибке на этапе компиляции. Также выдается сообщение об ошибке, если операции передаются параметры, число которых меньше обязательного минимального числа. Сколько параметров может быть у каждой операции, можно узнать из ее описания. При этом следует быть внимательным, потому что **одна и та же операция, вызываемая с разным числом параметров, может работать по-разному**.

2.3 Переменные и константы

Для хранения и многократного использования различных данных в Ci программе можно использовать *именованные переменные*. Переменная содержит данное того типа, для которого она создается, причем это данное может быть сколь угодно простым или сложным, однако о его сложности программист на Ci не должен беспо-

коиться, за него это делают компилятор и библиотечные функции.

Переменная обозначается именем, которое может состоять из любых символов, допустимых в имени, но имя не должно совпадать с каким-либо именем операции, типа данных, или константы имеющейся в текущей реализации Ci.

В текущей версии Ci все имена внутри одной программы должны быть уникальны. В следующих версиях уникальность будет требоваться только внутри одного блока программы. Имя переменной не может начинаться с цифры и не может содержать спецсимволы. Все остальные имена допустимы, что дает большую свободу в их выборе, чем дает язык C. Примеры допустимых имен переменных:

```
a
b12
uint32
c/2
fail-safe
#1
*all21
```

В случае, если при вызове операции будет пропущен пробел, разделяющий ее имя и имя переменной, комбинация будет воспринята компилятором, как единое имя переменной или операции. То есть, запись

```
(a+b)
```

будет восприниматься, как вызов операции с именем a+b, и будет выдана ошибка, если такой операции нет.

Для создания (или объявления) переменных в Ci-программе вместо вызова операции указывается имя типа переменных, после которого перечисляются через разделитель имена создаваемых переменных. Допускается указание более чем одного имени для создания нескольких однотипных переменных. Например, если требуется создать три переменных целочисленного типа, то достаточно написать:

```
int x y, z;  ``запятая пропускается, как и пробел.
```

При этом имена должны быть уникальны в рамках одной Ci-программы (подробнее об этом сказано далее в описании программных блоков). В качестве имен допускается использование только сочетаний символов, интерпретируемых,

как уникальные имена. Вызовы операций, указание констант, обращение к существующим переменным при создании новых переменных не допускаются (*кроме инициализации, см. далее*). Переменные создаются и инициализируются во время выполнения программы, при этом сам факт создания и инициализации наступает именно в том месте алгоритма, где переменная объявлена программистом. Компилятор учитывает это и не разрешает использовать переменные до того, как они будут созданы. То есть:

```
i = 1; ``здесь будет выдано сообщение об ошибке – неизвестная переменная;
    int i;
    i = 1; ``а теперь все нормально.
```

Для читабельного представления данных различных типов в языке Ci предназначены *константы*. Константы многих типов внешне не отличаются от переменных, кроме того, что численные константы должны содержать только допустимые для чисел символы – это цифры, знак - (минус) у отрицательных чисел, знаки . (точка) и e (экспонента, допустимо E большое) у вещественных чисел. Целочисленные константы могут содержать не только цифровые символы и знак минус перед ними, но также могут быть записаны в шестнадцатеричной форме. Шестнадцатеричная запись константы должна начинаться с 0x или 0X. При этом регистр символов, представляющих числа от 10 до 15 не имеет значения. Примеры правильных шестнадцатеричных целочисленных констант:

```
0xa
0xBBaa
0xFF1132
0xff1132
0x20
```

0x2d

Также поддерживается двоичная запись целочисленных констант при помощи 0b, 0B, например: 0b1001000, 0B00100.

Кроме этого, в языке Ci поддерживаются *литеральные константы* – это символьные имена, которыми заменяются представления постоянных значений данных различных типов. Их можно использовать везде, где использовались бы другие данные такого же типа, но они сокращают запись и делают текст программы более понятным. Замена производится во время компиляции программы, поэтому если присвоить переменной литеральную константу, то при работе программы произойдет присвоение той величины, которой эта константа соответствует. Для некоторых типов данных может быть допустимо только литеральное представление констант – например, это тип bool (логическое), константы этого типа можно записывать только как TRUE или FALSE. Константы тесно связаны с их типами, поэтому подробнее о константах речь пойдет в описаниях типов.

Переменные при создании инициализируются неким значением по умолчанию, для чисел это 0, для строк – пустая строка. Но переменные можно инициализировать необходимыми значениями сразу после объявления. Для этого, после имени создаваемой переменной, достаточно указать константу или вызов операции, возвращающей значение соответствующего или конвертируемого типа. Инициализация переменных значениями других переменных не допускается, потому что невозможно во время компиляции определить – была ли уже создана и проинициализирована переменная, значение которой используется. Например, использование:

```
int a[100], b[(sizeof a)];
```

недопустимо, поскольку вызов sizeof во время выполнения будет производиться ДО того, как будет создан массив a. Эта ошибка обнаруживается компилятором. Но есть ситуация, которую компилятор **пока не обнаруживает** – это инициализация массива значением элемента другого массива:

```
int a[100], b[a[0]];        `НЕДОПУСТИМО!`
```

такая запись не имеет смысла, поскольку должна приводить к созданию массива нулевой длины. Но в текущей версии Ci она приведет к аварийному завершению во время выполнения Ci-программы. Пока следует избегать такой инициализации.

В следующих версиях Ci эта проблема будет устранена.

Примеры правильной инициализации переменных:

```
uint i 10;              ``эквивалентно uint i; i = 10;
int j k 0x8080;         `j останется равным 0, k получит значение 0x8080`
bool log1 TRUE;
```

При создании и инициализации нескольких переменных одного типа рекомендуется для визуального разделения пар переменная-значение использовать символ запятой:

```
float a 12.1, b 1E3, c (PI / 4);
```

`` функционально эквивалентно: float a b c; a = 12.1; b = 1E3; c = (PI / 4); но код разный, в первом случае `` более компактный и правильный

```
str s “string”, t ;
block function1 {puts s}, function2 {call function1};
```

Поскольку запятая воспринимается компилятором точно так же, как символ пробела, это просто придает программе больше читабельности.

Область жизни переменной – блок, внутри которого она создана. Каждая переменная создается в том месте блока, где она объявлена в Ci программе и существует до конца работы блока. Все переменные блока автоматически удаляются при **любом** завершении блока. Удаление переменных производится в порядке, обратном их созданию. В подавляющем большинстве случаев это не имеет значения для прикладного программиста, однако могут возникнуть ситуации, когда это будет существенно, особенно, если переменные связаны с внешними объектами – файлами на диске, портами аппаратуры и т.д.

2.4 Массивы

В текущей реализации Ci поддерживаются только одномерные массивы. Массивы, размерностью больше 1 не поддерживаются, но при необходимости их можно эмулировать на одномерных массивах*. В массив могут быть собраны данные любого, но одного и того же типа. Размерность массива можно было бы не ограничивать – для индексации используется тип беззнаковое целое, что позволяет виртуально создавать массивы размером более 4 млрд. элементов на платформе IA32. Однако в реальности создать массив такого размера не получится, это обусловлено ограниченным объемом имеющейся памяти и способом ее использования операционной системой. Поэтому в каждой конкретной реализации Ci имеется свое ограничение на размер массива. В базовой конфигурации это 2^{20} == 1048576 элементов. При необходимости (например, при переходе на архитектуру с 64-х битной адресацией) их число может быть легко увеличено в новых версиях Ci.

Массив описывается, как обычная переменная, но после имени, в квадратных скобках указывается число элементов массива:

```
uint array[2048] data[100];
str string[ 128 ];
block b[10];
```

Нетрудно заметить, что для задания размера массива используются целые числа. Это должны быть беззнаковые целые, использование других типов данных (кроме знаковых целых) вызовет сообщение об ошибке на этапе компиляции. Если используется отрицательное знаковое целое, то сообщение об ошибке будет выдано на этапе выполнения (в следующих версиях оно будет выдаваться во время компиляции). Элемент массива определяется динамически и может использоваться везде, где используется обычная переменная.

Для хранения данных в многомерных массивах используются одномерные массивы, внутри состоящие из «подмассивов». Данные «многомерного» массива адресуются при помощи суммы смещений, получаемых как произведения индексов и размеров «подмассивов». То есть для 3-х мерного массива размерностью IxJxK элементов объявляется одномерный массив a[(I J K)]. В дальнейшем элемент a[i][j][k] можно адресовать как a[(+ i (j * I) (* k J I))]. Основным минусом такой техники является то, что невозможно произвести проверку выхода отдельного индекса за границу его «подмассива». То есть, если адресовать элемент с i,j,k индексом I,0,0, то будет адресоваться тот элемент, который имеет индекс 0,1,0. Это может порождать трудные для обнаружения ошибки. Кроме этого, обращение довольно громоздкое, и приводит к генерации дополнительного кода для вычисления произведений и суммы. Также, невозможно обратиться к элементу массива, размерность которого больше максимально допустимого числа параметров у операций.

```
puts array[1] (data[100] * data[101]);
```

При этом индекс элемента вычисляется непосредственно перед получением элемента массива, после чего элемент передается параметром операции. Индекс массива является беззнаковым целым числом, он указывается внутри квадратных скобок после имени массива, точно так же, как это делается в программах на обычном языке C. Но в языке C нельзя объявить массив, используя для указания размера значение переменной, или значение, возвращаемое функцией. В языке Ci можно это делать, поскольку массивы создаются во время выполнения программы, то есть, в Ci массивы являются *динамическими*.

```
uint indx 10;
    ...
    ++ indx;
    ...
    array[ indx ];
```

Если индекс вычисляется и возвращается какой-то операцией, то ее вместе с параметрами надо указывать в круглых скобках.

```
puts array[(indx * 2)];
```

Некоторые операции могут принимать в качестве параметра не только элементы массива, но и целиком массив. Такая возможность определяется реализацией операции и только ей. Например, по умолчанию операция сложения + работает только с отдельными переменными. Но ее можно расширить (это может сделать только обученный системный программист), чтобы, например, она выполняла суммирование всех элементов всех массивов, которые передаются ей параметрами, причем независимо от их типов. Компилятор проверяет возможность передачи операции массива целиком и выдает ошибку, если такая передача невозможна. Для передачи целого массива достаточно просто не указывать индекс и квадратные скобки, только

имя массива. Наиболее часто применяемой операцией, которая может принимать массив параметром, является **clear:**

```
uint array[1000]; str string[200];
        .....
        код, содержащий изменения массивов array и string
        .....
        clear array string; ``массивы снова пусты, как после создания.
```

Необходимо отметить, что когда в описании операции указывается, что массив не допустим, это означает, что недопустима передача массива *целиком*, однако можно передавать отдельный элемент массива, поскольку он эквивалентен простой переменной.

В языке Ci принята нумерация объектов такая же, как в C, начиная с 0-го. Это означает, что первый по порядку элемент массива, первый параметр операции, первый символ строки и т.д. имеют номер 0. Последний элемент массива размерностью N имеет номер N-1. Последний символ строки длиной N, имеет номер также N-1. В описании операций параметры называются первым, вторым и т.д., но программно к ним производится обращение по номерам 0, 1, 2 и т.д. Для массива с размерностью 100 допустимые номера элементов находятся в диапазоне 0...99. При попытке обращения к элементу массива с номером, больше максимального, Ci-программа будет прервана с сообщением об ошибке. Также выполнение прекращается с ошибкой, если производится попытка создать массив с недопустимым числом элементов, то есть, равным 0 или большим максимально допустимого для текущей реализации.

3 Типы данных

Различные реализации Ci поддерживают различное количество *типов данных*. Тип данных – это абстрактное наименование группы сходных данных, определяемое тем, какие операции с этими данными являются допустимыми, как данные этого типа изображаются, и какие значения они могут принимать. В базовый комплект входят типы:

логическая величина **bool**;

беззнаковое целое число **uint**;

знаковое целое число **int**;

вещественное число одинарной точности **float**;

строка **str**;

блок **block**;

В конкретном приложении эти типы могут быть расширены практически любыми типами данных, как простыми, так и структурными (состоящими из нескольких простых или сложных типов). Простые типы данных могут быть совместимы с базовыми типами, вплоть до автоматического преобразования (см. далее «Преобразование типов»). Структурные (или сложные) типы могут иметь необходимые свойства, доступные при помощи вызовов специальных операций. Таким образом, могут быть описаны группы данных, получаемых от аппаратуры, дисковые файлы, сетевые соединения, окна, текстовые документы, базы данных и т.д. Структура данного сложного типа скрыта от прикладного программиста, подобно тому, как производится инкапсуляция в классических объектно-ориентированных языках. Базовый комплект Ci включает два сложных типа – это строка и блок.

Далее описаны базовые типы данных и приводятся сведения об автоматических преобразованиях.

3.1 Логическое (bool)

Имя типа: bool. Данные этого типа могут принимать только два значения: "ложь" и "истина". Начальное значение переменной: ложь. Непосредственно этот тип представляется литеральными константами **FALSE** (ложь) и **TRUE** (истина). Эти значения могут быть присвоены переменным типа bool, использованы в качестве параметров операций, ожидающих тип bool и т.д. Данные типа bool могут быть преобразованы в строку, при этом результатом будет строка "FALSE" или "TRUE", в зависимости от значения bool переменной. Также данные этого типа преобразуются в целые числа, в этом случае значение FALSE преобразуется в 0, а значение TRUE в 1. Аналогично при преобразовании в вещественные числа значение TRUE преобразуется в 1.0. Преобразование данного типа bool из строки в базовой версии Ci не допустимо, поскольку при ошибках прикладного программиста могут возникать неоднозначные ситуации. При грамотном программировании необходимости в таком

преобразовании не возникает.

3.2 Беззнаковое целое число (uint)

Имя типа uint. Принимает значение от 0 до максимально допустимого беззнакового целого числа для данной конкретной реализации Ci, в зависимости от платформы и используемого компилятора C. Беззнаковые целые надо использовать для хранения тех величин, которые являются целочисленными, и не могут быть отрицательными. Например, это данные, показывающие количество объектов, либо представляющие номера объектов, включая индексы элементов массивов. Для процессора семейства IA32 применяется 32-х разрядное представление беззнаковых целых, что дает максимальное число 4294967296. Начальное значение переменной: 0.

Данные типа uint являются Ci-реализацией данных типа unsigned int языка C и для них выполняются все те же правила. Они могут быть преобразованы в строку (строчное представление числа), а также в знаковое или вещественное число с округлением значения, если оно оказалось больше максимально возможного числа (аналогично таковому в языке C). При преобразовании в знаковое целое производится округление до максимально возможного знакового целого числа, при преобразовании в вещественное – до вещественного эквивалента максимально возможного целого. То есть, операции

uint a 4294967296; int b = a; float c = a;

на выходе дадут: b == 2147483647 (максимально возможное знаковое целое на IA32), c == 2.147483E9 (что эквивалентно 2147483647.0). Очевидно, происходит потеря значения, о чем необходимо помнить прикладному программисту.

Беззнаковые целые преобразуются в логические по следующему правилу – если число равно 0, то результат преобразования FALSE. Если число больше 0, то результат преобразования TRUE;

bool a; uint b 10;

a = b;

puts a; ``напечатает TRUE.

3.3 Знаковое целое число (int)

Имя типа int. Диапазон значений совпадает с диапазоном типа int языка C для конкретной аппаратной платформы (от -2147483646 до 2147483648 для IA32). Знаковые целые числа аналогичны беззнаковым, с тем отличием, что их старший разряд используется для обозначения знака числа. Эти числа следует использовать в тех случаях, когда необходимо представить данные, которые являются перечислимыми, но при этом могут быть меньше или больше 0. Например, это значения какой-либо шкалы, на которой среднее положение соответствует 0. Начальное значение целочисленной переменной: 0. Данные могут быть преобразованы в строку, беззнаковое целое и вещественное число. При преобразовании в беззнаковое целое отрицательных целых чисел, они становятся равны 0.

Примеры правильных знаковых целых чисел:

0 1234 -7689

Знаковые целые преобразуются в логические по следующему правилу – если число равно 0, то результат преобразования FALSE. Если число больше или меньше 0, то результат преобразования TRUE.

3.4 Вещественное число (float)

Имя типа float. Внутреннее представление, диапазон и точность совпадают с типом float языка C для используемой платформы. Однако функции преобразования вещественных чисел из строки и в строку поддерживают только числа от -3.4028235E38 до 3.4028235E38. Начальное значение переменной: 0.0. Данное может быть преобразовано в типы uint, int и в строку. Если строчная константа содержит число, меньше -3.40282356E38 или больше 3.40282356E38, то она преобразуется в -3.40282356E38 или 3.40282356E38 соответственно. Аналогично, если при вычислениях получилось число больше максимального или меньше минимального, то при преобразовании в строку оно ограничивается величинами, указанными ранее. При преобразовании в int производится отбрасывание дробной части без округления. При преобразовании в uint отрицательное вещественное число становится равным 0.

При преобразовании в строку по умолчанию используется экспоненциальное представление с одним значащим разрядом мантиссы и указанием порядка (преобразованием в строку можно управлять при помощи операции **floatstyle**). Это означает, что число 123.4567 при преобразовании из внутреннего представления в строку будет показываться как 1.234567E2. Значимыми в мантиссе при преобразовании из строки в вещественное число являются 7 знаков, не включая запятую. Тоже самое касается порядка числа. При преобразовании из строки регистр латинской буквы E, обозначающей экспоненту, не имеет значения. При преобразовании в строку всегда символ экспоненты будет в верхнем регистре.

Примеры правильных вещественных чисел:

1256E2
4.4E19
-1685.2e2
0.001e12
+1.12
100.0
-100.23

Вещественные числа преобразуются в логические по следующему правилу – если число равно 0.0, то результат преобразования FALSE. Если число больше или меньше 0.0, то результат преобразования TRUE.

В базовой реализации Ci имеется несколько вещественных литеральных констант для упрощения записи при математических вычислениях:
PI (число пи) == 3.141593;
PI/2 (число пи, деленное на 2) == 1.570796;
_1/PI (1 деленное на пи) == 3.183099E-1;
SQRT2 (корень из 2) == 1.414214;
_1/SQRT2 (1 деленное на корень из 2) == 7.071068E-1;
LOG2E (логарифм числа e по основанию 2) == 1.442695;
LOG10E (логарифм числа e по основанию 10) == 4.342945E-1;

LOG2 (логарифм 2 по основанию e) == 6.931472E-1;
LOG10 (логарифм 10 по основанию e) == 2.302585;
E (число e) == 2.718282.

3.5 Строка (str)

Имя типа str. Данное этого типа содержит строку символов длиной до 256 символов ASCII, каждый имеет размер 1 байт. UNICODE базовой реализацией не поддерживается. Начальное значение переменной: "" (пустая строка).

Строчная константа записывается между двойными кавычками. Если внутри строки необходимо вставить двойную или одинарную кавычку, ее следует предварить символом обратной дробной черты:

"строка с символами двойной \" и одинарной \` кавычек"
при выполнении программы будет иметь представление:
строка с символами двойной " и одинарной ` кавычек

Строка может содержать управляющие символы, аналогично строкам языка C: перевод строки '\n', табуляцию '\t', символ с десятичным или шестнадцатеричным кодом: '\100', '\0xae' и т.д. Компилятор Ci преобразует эти символы во внутреннее представление. Для включения в строку символа обратной дробной черты необходимо указывать его дважды (об этом надо помнить при написании имен файлов в ОС Microsoft Windows). Если необходимо включить в строку спецсимвол языка, его также необходимо предварять обратной дробной чертой. С особой осторожностью следует использовать внутри строк символ с кодом '\0x00', поскольку он распознается используемыми библиотечными функциями, как конец строки. В то же время, строки Ci не ограничиваются только этим символом. Общая рекомендация: следует тщательно избегать появления этого символа внутри строки.

Строки можно присваивать и сравнивать. При сравнении двух строк на больше или меньше соответствующие операции возвращают корректные значения. Сравнение строк производится слева направо, больше та строка, у которой больше код символа в текущей позиции, или та строка, которая длиннее, если символы оди-

наковы. То есть, строка "abcd" меньше, чем строка "abcf", а строки "accd" и "abcfa" больше. Символ в верхнем регистре **меньше**, чем тот же символ в нижнем регистре.

Строка хранится в буфере, выделяемом при ее создании. Длина буфера константных строк равна длине сохраняемой строки плюс один байт (в нем хранится завершающий строку ноль). Для переменных типа строка буфер выделяется с размером, установленным для строк по умолчанию, то есть 256 символов плюс один байт.

Строки в Ci динамические, это означает, что процессор и соответствующие операции заботятся о корректности сохраняемых строк при выполнении над ними сложных операций. Например, таких, как вставка одной строки в середину другой.

Строчные данные преобразуются в другие типы данных, но только в том случае, если строка содержит корректное символьное представление данных соответствующего типа. Например:

"+24.516" – корректное представление вещественного числа;
"457ф" – НЕКОРРЕКТНОЕ представление числа.

В первом случае преобразование строки в вещественное число произойдет без проблем, во втором будет выдана ошибка. Причем момент ее появления различен – если компилятор обнаружит некорректную строчную константу там, где ожидалось численное значение, он прекратит компиляцию с ошибкой. Но если строчная переменная в момент преобразования будет содержать некорректное значение, то программа будет прервана уже на этапе выполнения.

В базовом комплекте производятся преобразования строк в данные типа uint, int и float. Преобразование из строки в логическое не поддерживается, чтобы избежать возможных ошибок в Ci программе.

Примеры правильных строк:

"строка символов"
"строка\nсимволов"
"\0xaa\0xab\0x20\0xdf\n"
"строка с\tтабуляцией"

3.6 Блок (block)

Имя типа block. Блок – особый тип данных в Ci. Он используется для группировки многих операций с целью реализации ветвления и зацикливания алгоритма выполнения программы, а также для создания функций. Любой блок Ci программы изначально имеет текстовое представление, поскольку в нем задается последовательность операций. Начальное значение переменной: {} (пустой блок, «нет операций»).

Записывается блок между фигурными скобками:

{ <последовательность вызовов операций> }

Изолировано от операций блок появляться не должен, иначе возникнет ошибка при компиляции. Блочная константа указывается в качестве параметра у тех операций, которые ожидают данное типа block соответствующим параметром. Наиболее часто встречается операция if. Поэтому пример применения блоков приводится с ее использованием:

```
if logvar { a = b };
```

В примере, если переменная logvar содержит TRUE, будет выполнен блок с одной операцией присваивания. Как видно по примеру, после операции присваивания нет точки с запятой, поскольку закрывающая скобка блока завершает компиляцию блока, автоматически ограничивая параметры у присвоения. Однако после закрывающей фигурной скобки присутствует точка с запятой, она ограничивает перечисление параметров операции if. В языке C ровно наоборот – перед закрывающей фигурной скобкой у последнего оператора необходима точка с запятой, а после– не нужна.

В текстовом виде block существует при написании программы и во время ее компиляции. Компилятор преобразует блок во внутреннее представление, которое потом обрабатывается процессором Ci, производящим вызовы функций на C для выполнения операций, включенных в блок. Во внутреннем представлении программе

на Ci блок недоступен. Однако это не запрещает создавать переменные типа block, присваивать им значения и вызывать их выполнение. Нормальной является следующая запись:

```
block a;
        a = { puts "вывод строки" };
        exec a;
```

последовательность действий не требует пояснений.

Пустой блок не содержит никаких действий. Если такую переменную указать параметром, например, операции if, то ничего не произойдет. Аналогично, если для exec указать пустую блочную константу, ничего не изменится и не выполнится:

```
exec {};            `ничего полезного не делает, только процессорное время зря тратит`
```

Операция exec просто выполняет указанный ей параметром блок. Ее недостаточно для создания функций, поскольку ей нельзя передать параметры и получить возвращаемое значение. Эти возможности предоставляет операция call, описанная в справочнике по операциям.

Разумеется, блоки могут быть вложенными. Пример Ci программы, печатающей таблицу умножения:

```
int a b;
        for a `from` 1 `to` 9 `with step` 1
        {
            for b 1 9 1
            {
                stdout (a * b) " "
            };
            stdout "\n"
        };
```

Переменные, объявленные в объемлющем блоке, доступны во вложенных в него. Переменные, объявленные внутри блока, доступны внутри него, и во вложенных, но недоступны в объемлющих. Однако, *в текущей версии Ci все переменные внутри Ci-программы должны иметь уникальные имена* – это временное ограничение реализации. При выполнении программы переменные будут создаваться в тот момент, когда выполнение блока дойдет до места, где они объявлены. Но удаляются все переменные в конце блока, после выполнения его последней операции.

Данные типа блок не преобразуются в данные другого типа. В случае, если бы было реализовано преобразование из строки в блок, то функция такого преобразования включила бы в себя вызов компилятора. Для выполнения обратного преобразования из блока в строку, требуется реверсивный процессор, аналогичный дизассемблеру.

Весь текст Ci программы, является, на самом деле, главным блоком, внутри которого находится код программы. Отличие от вложенных в него блоков только в том, что ему не требуются фигурные скобки и вызов операции выполнения, он производится исполняющей системой Ci.

```
exec {                    ``выполняется Ci-системой

                          ``код программы, которую ``пишет
                          `` программист на Ci
                          puts “Hello world”;
  }
```

Наклонным шрифтом обозначен «главный блок», который выполняется автоматически.

Сравнения блоков не поддерживаются.

3.7 Преобразование типов

Как правило, операции ожидают в качестве параметров данные определенного типа. Но можно использовать данное другого типа, и, если компилятору известен способ прямого преобразования таких данных, то он создаст *скрытый* вызов опера-

ции преобразования типов. Например, если записано:

```
int a; a = 123;
        stdout a;
```

то в стандартный выходной поток будет выведено число 123. Хотя, на самом деле, операция stdout принимает своими параметрами только строки. Преобразование произойдет скрытно и автоматически. Если бы такие преобразования не поддерживались, то пришлось бы записать что-то вроде:

```
int a; a = 123;
        stdout (inttostr a);
```

Преобразование выполняется всегда, когда ожидаемый тип параметра операции не совпал с типом указанного операнда, независимо от последнего, кроме случая, когда в качестве параметра допустима только переменная или допустим массив. То есть, преобразование выполняется, если операндом указана переменная, любая константа, элемент массива или вызов другой операции. Схема примерно следующая – предположим, имеем вызов операции вида:

```
op1t1 p1t1 p2t2 (op2t2)
```

где op1t1 ожидает параметрами данные типа t1. Переменные p1t1 и p2t2 разных типов, а op2t2 возвращает значение типа t2. Но при этом, если компилятору известен способ преобразования типа t2 в t1, тогда выполнение будет происходить по следующей схеме:

сначала вызов операции op2t2;
возвращенное ей значение присваивается невидимой временной переменной tmp1t2;
значение переменной p2t2 преобразуется и записывается в невидимую временную переменную tmp2t1;
значение переменной tmp1t2 преобразуется и записывается в невидимую временную

переменную tmp3t1;
вызывается операция op1t1 p1t1 tmp2t1 tmp3t1.

Выглядит сложным, но при использовании все оказывается просто. В других языках программирования, где есть разные типы данных и не поддерживается скрытое преобразование, пришлось бы для этого вызывать специальные операции, а при необходимости изменения типа переменной перекраивать весь исходный текст. В Ci все указанные преобразования происходят незаметно для прикладного программиста и почти не требуют его внимания (за исключением ряда случаев, далее описанных в этом документе). Единственное, о чем надо знать и помнить прикладному программисту – это о возможности преобразования одних типов данных в другие. То есть, о возможности указывать вместо данных одного типа данные другого, и, в небольшом числе случаев, знать, как эти данные преобразуются. Для базовых типов данных здесь приводится сводная таблица, показывающая возможность таких преобразований:

тип из / тип в	bool	int	uint	float	str	block
bool		+	+	+	-	-
int	+		+	+	+	-
uint	+	+		+	+	-
float	+	+	+		+	-
str	+	+	+	+		-
block	-	-	-	-	-	

+ преобразование определено;
- преобразование не определено;
пусто там, где преобразование не требуется.

Поскольку такие преобразования при отладке реальных программ требуются достаточно часто, их автоматизация делает программирование более простым и быстрым. Например, чтобы вывести в stdout значения различных типов переменных вперемежку с текстом, достаточно написать:

```
float a 12.31;
        int b 101;
        bool c TRUE;
        stdout "значение a=" a ", значение b=" b ", значение c=" c "\n";
```

в результате будет выведено:

```
        значение a=1.231000E1, значение b=101, значение c=TRUE
```

Также преобразования позволяют смешивать в арифметических выражениях данные различных совместимых типов, с гарантией, что выражение будет вычислено корректно. Необходимо только знать, как при преобразованиях могут изменяться числа, чтобы не происходила потеря значения при вычислении. Если такая потеря возможна, то прямых комбинаций числовых данных различных типов в одном выражении следует избегать, либо следить за порядком вычислений. Особенно это важно, когда в одном выражении сочетаются знаковые и беззнаковые числа, либо вещественные и беззнаковые, поскольку отрицательное число, при преобразовании в беззнаковое, становится равно 0.

Иногда, особенно при вызове каких-либо операций, требуется чтобы данное было преобразовано в некоторый определенный тип. Для операций, возвращающих значения, это можно делать, указывая тип результата сразу после открывающей скобки. Например:

```
int i 200;
        puts (float i + 1);
```

будет выполнено необычным образом: сначала произойдет суммирование 200 и 1, что даст целое число 201, потом оно будет преобразовано в вещественное число, и оно, в свою очередь, будет преобразовано в строку, которая будет напечатана, в результате мы увидим, вместо 201, число 2.01E2. Для чего это нужно?

Дело в том, что в ряде случаев компилятор не знает, какого типа данное он

должен использовать. Чаще всего это возникает в ситуациях, когда надо вернуть результат арифметического выражения в качестве параметра функции при помощи операции **par@**. Например, если записать:

```
par@ 0 (12 + 20);
```

то компилятор выдаст сообщение об ошибке, поскольку он не может определить тип результата сложения – числа, указанные параметрами сложения, могут быть одного из трех типов – вещественные, целые или беззнаковые целые. Результат сложения от этого не зависит, а тип параметров определяется *во время выполнения*. Компилятору можно помочь, если написать:

```
par@ 0 (int 12 + 20)
```

в этом случае суммирование вернет целое число. То есть, *указание типа для возвращаемого значения операцией может потребоваться в тех случаях, когда какой-либо параметр операции может принимать значение любого типа, и нет другого способа его определить*. Как правило, это не требуется (и не следует) делать при написании Ci-программы до тех пор, пока компилятор не выдаст сообщение о том, что он не может определить тип возвращаемого значения, и не покажет где именно надо его указать.

В качестве побочного эффекта можно использовать такое преобразование для отбрасывания с округлением дробной части вещественных чисел при их преобразовании в строку, например:

```
float f 1234.5678;
        puts (int f + 1);
        напечатает: 1235
```

но необходимо помнить, что вещественное число может быть больше максимально допустимого целого, тогда это приведет к потере значения.

При освоении программирования на Ci первое время потребуется следить за тем, корректно ли смешиваются в одном выражении различные типы. Результат может отличаться от ожидаемого, если при преобразовании происходит потеря значения – например, если в цепочке преобразований чисел ошибочно указана логическая переменная, то после ее использования числа будут принимать значения только 0 или 1. Могут также возникнуть проблемы при смешивании в одном выражении знаковых и беззнаковых целых чисел. Неожиданный результат вычислений практически гарантирован, если производится преобразование вещественного числа в целое, и при этом вещественное число имеет значение, близкое к максимально допустимому целому. Конкретная «опасная» величина числа зависит от того, знаковое или беззнаковое целое число должно быть получено. Остановимся на этом подробнее.

Например, если имеется Ci-компилятор, созданный при помощи компилятора Microsoft из комплекта Visual Studio 2005, и для него в Ci-программе написать:

```
int i 214748360;
    puts i;
```

то будет корректно напечатано число 214748360, но если попытаться выполнить код:

```
loop {puts i} 1 i 214748360;
```

то будет напечатано число 214748352.

Это происходит из-за того, что в качестве начального значения для цикла loop может быть указано значение любого типа, а когда компилятор не знает, какой именно тип ему ожидать, то при получении строки с цифрами, он преобразует ее **сначала в вещественное число**. Затем, внутри функции loop, это число преобразуется в данное того же типа, что и переменная цикла i, то есть, в целое. В примере величина вещественного числа очень близка к максимальному знаковому целому, и при преобразовании (которое автоматически создает C-компилятор Microsoft) возникает потеря значения в младших разрядах. Для других компиляторов это может быть не так, и преобразования могут выполняться правильно.

Поэтому начинающему программисту следует быть внимательным, если в одном выражении смешиваются данные разных типов. Но любой начинающий довольно быстро приобретет необходимый опыт и понимание механизма преобразова-

ний. Тогда он сможет более свободно работать с различными типами данных. У него не будет необходимости в рутинных вызовах операций для явного преобразования данных различных типов, что приходится делать в других языках аналогичного назначения. Процессор языка Ci сделает это за него. В крайнем случае, можно использовать временные переменные для присвоения им промежуточных значений, тогда компилятор будет вынужден правильно производить преобразование.

Преобразование типов не производится в случае, если операция требует параметром обязательно переменную. В таком случае преобразование надо было бы производить **после** выполнения операции, что значительно усложнило бы компилятор. Реально таких ситуаций не много, наиболее частый случай – попытка использования беззнаковых целых в качестве параметра BASIC-подобного цикла for:

```
uint i;
        for i 0 10 1 {};
```

недопустимо, поскольку for ожидает первым параметром переменную типа знаковое целое (эта переменная изменяется во время выполнения цикла, и может принимать отрицательные значения). В таких случаях указание переменной другого типа вызывает ошибку во время компиляции, даже если определены преобразования для обоих типов. Если требуется использовать в качестве параметра цикла переменную отличного типа от int, рекомендуется применить цикл loop, который гораздо мощнее и удобнее. В других языках аналога его нет.

Преобразование выполняется скрытно и автоматически не только в случае, когда параметром указана переменная другого типа. Оно также производится, когда параметром указывается операция, возвращающая значение, или литеральная константа. Например:

```
puts PI " " (PI / 2);
```

выполнится следующим образом (курсивом обозначены скрытые автоматические действия)):
временная вещественная переменная *tmp1* = PI / 2;

временная строчная переменная *tmp2* = *вещественное-в-строку* PI;
временная строчная перменная *tmp3* = *вещественное-в-строку tmp1;*
вызов puts *tmp2*, " ", *tmp3*;

в результате в stdout будет выведено:

3.141593 1.570796

Но есть один случай, когда автоматическое преобразование не может быть выполнено, даже если операция допускает параметром не только переменную. Этот случай будет описан далее в описании операций с неизвестными типами параметров.

Типы данных в Ci жестко не ограничены перечисленными выше. При адаптации языка к конкретной программной системе они могут быть расширены типами, специфичными для этой системы. Например, если система имеет аппаратные средства, требующие набор данных для управления, то в состав Ci системы может быть включен тип данных, содержащих соответствующие значения. При этом может быть определено преобразование этого типа между любыми другими имеющимися, что позволит использовать соответствующие данные в выражениях совместно с данными другого типа. На любой новый тип автоматически распространится контроль допустимости его использования. Те операции, в описании которых сказано, что они могут работать с данными любых типов, будут его автоматически поддерживать, и компилятор будет производить преобразования, если они определены. Или наоборот – можно ограничить использование специфичного типа только набором специфичных для него операций.

3.8 Операции с неизвестными типами параметров и неизвестным типом возврата

У многих операций типы параметров изначально не зафиксированы. Такие операции называются *контекстно-зависимыми*. Контекстно-зависимая операция принимает операнды любого типа, но с рядом ограничений. Есть всего два основных вида таких операций: у одного вида тип какого-либо переданного параметра определяет остальные типы параметров, у другого вида типы параметров могут быть раз-

личными и независимы друг от друга. Это оказывает существенное влияние на автоматическое преобразование типов.

Для операций, у которых тип фиксированного (часто самого первого) параметра определяет типы остальных параметров, выполняются автоматические преобразования типов параметров. Разумеется, кроме того параметра, тип которого определяет остальные. На самой операции лежит ответственность за поддержку различных типов этого параметра, или останов программы, если полученный тип ей не поддерживается. В противном случае попытка выполнения операции над неподдерживаемым типом приведет к аварийному завершению всей системы. Тоже самое справедливо для операций, типы параметров которых могут быть различными. Автоматические преобразования над такими типами не выполняются, данные передаются операции как есть. Компилятор не может определить требуемый тип и выдать сообщение об ошибке, если преобразование невозможно. Такое сообщение выдается уже во время выполнения Ci-программы, что накладывает определенную ответственность на ее создателя. Однако поддержка различных типов данных внутри операций языка Ci не входит в рамки настоящего описания.

Важно отметить, что у многих операций тип фиксированного параметра определяет не только типы ожидаемых остальных параметров, но и тип возвращаемого операцией результата. Например, у всех арифметических операций тип первого параметра не определен, а тип второго и тип возврата такие же, как тип первого параметра. Компилятор узнает из текста программы, какого типа данное указано первым параметром, и ожидает такое же данное в качестве второго. А также формирует автоматическое преобразование, если оно требуется. Это означает, что выражение:

```
float a; int b;
        puts (a * b);
```

будет корректно выполнено, поскольку будут произведены автоматические преобразования:

```
puts (вещественное-в-строку (a * (целое-в-вещественное b));
```

Из-за того, что первый параметр операции определяет тип второго параметра, в языке Ci **не выполняется** арифметическое правило “от перемены мест слагаемых сумма не изменяется” – во всяком случае, изменяется *тип результата* суммирования:

```
float a 12.6;
        int b 1;
        floatstyle 1 FALSE;
        puts (a + b); ``напечатает вещественное число 13.6
        puts (b + a); ``напечатает целое число 13
```

Однако возможна ситуация, когда компилятор не может определить тип и параметра, и значения, возвращаемого вызываемой операцией. Это случается в арифметических выражениях с использованием параметров функций. Например:

```
puts ((par# 0) * 2); `будет выдано сообщение об ошибке – невозможно определить тип`
```

Операция par# возвращает данное не определенного заранее типа. А тип первого параметра операции умножения также не определен. Для разрешения этой проблемы в Ci предусмотрено явное указание типа возвращаемого операцией значения:

```
puts((int par# 0) * 2);
```

В таком случае операция par# вернет целочисленное значение первого параметра, который был передан в операцию call, а компилятор передаст его операции умножения и преобразует константу 2 в целое число для передачи вторым параметром. Явное указание корректно срабатывает только в случаях, когда компилятор выдает сообщение об ошибке с предложением его использовать. В остальных случаях тип определяется автоматически, а явное указание может дать не тот результат, который нужен, и его следует избегать.

Если же в качестве параметра указывается последовательность символов, и компилятор не находит подходящей переменной или литеральной константы, то он определяет тип следующим образом – если первый символ константы цифра или знак минус, то производится попытка преобразовать последовательность символов в вещественное число. При неуспехе выдается ошибка преобразования. Если же это другой символ, то последовательность символов считается строчной константой. Например:

```
puts(100 * 2);
```

напечатает

```
2.000000E2
```

поскольку константа 100 будет распознана, как вещественное число. Это определит тип другого параметра и возвращаемого результата. А при компиляции кода

```
puts(a100 * 2);
```

если переменная a100 не объявлена ранее, будет выдано сообщение об ошибке «неизвестная переменная».

4 Операции (справочник)

Далее приводится этот список операций с расшифровкой выполняемых каждой операцией действий, смысла ее параметров и возвращаемого значения. Списком можно пользоваться, как справочным руководством при написании программ на Ci. Для большинства операций приведены примеры использования. Для всех операций указан допустимый формат или форматы их вызова. При этом используется следующая форма записи:

если операция записана внутри круглых скобок, то это означает, что она возвращает значение и ее можно использовать параметром другой операции;

если операция завершена точкой с запятой, это означает, что она не возвращает зна-

чение;

для операций, возвращаемое значение которых можно игнорировать, приведены обе формы записи;

если операция может записываться в инфиксной форме, то приведены инфиксный и префиксный способы записи;

параметры приведены в виде <parameterN>, где N число от 0;

если параметр, или список параметров указан в квадратных скобках, то он может отсутствовать;

если в квадратных скобках параметры <parameterN> и <parametrM> указаны через троеточие, то это означает, что может быть от N, до M параметров, или вообще они могут отсутствовать.

Если вызов операции допускается в инфиксной форме, об этом упоминается отдельно, и в таком виде приводится пример. В противном случае допускается только префиксная форма.

Большинство исключительных ситуаций (ошибок) выявляется компилятором. Но есть множество случаев, когда компилятор не выявляет ошибку, но она выявляется на этапе выполнения. В таком случае Ci-программа будет прервана с соответствующим сообщением. Практически всегда это происходит, если операция сама проверяет допустимость выполнения запрошенных действий. Такие операции отмечены особо, к их использованию следует относиться внимательно.

Дальнейшее описание операций разделено на несколько групп:

логические операции;

операции управления данными;

арифметические операции;

поразрядные операции над целыми числами;

математические операции;

операции над строками;

операции управления алгоритмом;

отладочные операции.

4.1 Логические операции

Логические операции предназначены для помощи в управлении потоком вы-

полнения программы или изменения данных, в зависимости от различных условий. Они очень похожи на соответствующие операторы языка C, по возможности сохранена их семантика.

4.1.1 !

Формат вызова:

(! <parameter>)

возвращает bool, принимает 1 параметр;

тип параметра bool, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "нет", возвращает инверсное значение параметра, пример:

if(! boolvariable){};

4.1.2 !!

Форматы вызова:

!! <parameter>;

(!! <parameter>)

возвращает bool, принимает 1 параметр;

тип параметра bool, разрешена только переменная, массив не разрешен;

назначение операции: логическое "нет", инвертирует значение принимаемой логической переменной, и может возвращать результат, пример:

!! value; `эквивалентно value = (! value);

4.1.3 >

Форматы вызова:

(< <parameter0> <parameter1>)

(<parameter0> < <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды,

массив не разрешен;

назначение операции: логическое "больше", возвращает TRUE, если значение первого параметра больше второго, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a > b){}

4.1.4 <

Форматы вызова:

(> <parameter0> <parameter1>)

(<parameter0> > <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "меньше", возвращает TRUE, если значение первого параметра меньше второго, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a < b){}

4.1.5 ==

Форматы вызова:

(== <parameter0> <parameter1>)

(<parameter0> == <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "равно", возвращает TRUE, если значения параметров равны, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a == b){}

4.1.6 !=

Форматы вызова:

(!= <parameter0> <parameter1>)

(<parameter0> != <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "не равно", возвращает TRUE, если значения параметров не равны, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a != b){}

4.1.7 >=

Форматы вызова:

(>= <parameter0> <parameter1>)

(<parameter0> >= <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "больше или равно", возвращает TRUE, если значение первого параметра больше или равно второму, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a >= b){}

4.1.8 <=

Форматы вызова:

(<= <parameter0> <parameter1>)

(<parameter0> <= <parameter1>)

возвращает bool, принимает 2 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическое "меньше или равно", возвращает TRUE, если значение первого параметра меньше или равно второму, иначе возвращает FALSE, допустима инфиксная запись, пример:

if(a <= b){}

4.1.9 <>

Форматы вызова:

(<> <parameter0> <parameter1> <parameter2>)

(<parameter0> <> <parameter1> <parameter2>)

возвращает bool, принимает 3 параметра;

первый параметр – тип любой, разрешены любые операнды, массив не разрешен;

второй параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

третий параметр такого же типа, как первый, разрешены любые операнды, массив не разрешен;

назначение операции: логическая проверка значения на попадание в указанный диапазон, первый параметр задает проверяемое значение, второй параметр нижнюю границу, а третий верхнюю границу диапазона, функционально эквивалентно выражению

((param0 < param1) || (param0 > param2))

пример:

if(a <> lowval highval) {}

Основное назначение – проверка вещественных чисел на попадание в заданный диапазон, поскольку использование для вещественных чисел проверки на равенство не совсем корректно из-за возможного шума в младших разрядах после вычислений или проведения измерений, то есть, вместо

```
float a b;
    if( a == b ){}
```

надо использовать:

```
float a b delta 0.0001;
    if( a <> (b – delta)(b + delta) ){}
```

где delta – небольшое вещественное число, задающее погрешность сравнения, зависит от требуемой точности вычислений или измерений.

4.1.10 &&

Форматы вызова:

(&& <parameter0> <parameter1> [<parameter2>...<parameter31>])

(<parameter0> && <parameter1> [<parameter2>...<parameter31>])

возвращает bool, число параметров от 2 до максимально допустимого;

все параметры типа bool, допустимы любые операнды, массивы не допустимы;

назначение операции: логическое "И" (логическое умножение), возвращает TRUE, если значения всех параметров равны TRUE, иначе возвращает FALSE, допустима инфиксная запись, примеры:

```
if((a <= b) && (c == d )){}
bool a b c d e; c = (&& b d e a);
if( && (a ==b) (c == d) (e == 100)) {puts "something"};
```

4.1.11 ||

Форматы вызова:

(|| <parameter0> <parameter1> [<parameter2>...<parameter31>])

(<parameter0> || <parameter1> [<parameter2>...<parameter31>])

возвращает bool, число параметров от 2 до максимально допустимого;

все параметры типа bool, допустимы любые операнды, массивы не допустимы;

назначение операции: логическое "ИЛИ" (логическое сложение), возвращает TRUE, если значение хотя бы одного из параметров равно TRUE, иначе (если все FALSE) возвращает FALSE, допустима инфиксная запись, примеры:

```
if((a <= b) || (c == d)){}
bool a b c d e; c = (|| a b d e);
```

4.1.12 ?

Форматы вызова:

(? <parameter0> <parameter1> <parameter2>)

(<parameter0> ? <parameter1> <parameter2>)

возвращает значение любого типа, число параметров 3;

первый параметр типа bool, все виды операндов, массив недопустим;

второй параметр любого типа, для которого допустимо присваивание, допустимы все виды операндов, массив недопустим;

третий параметр любого типа, для которого допустимо присваивание, допустимы все виды операндов, массив недопустим;

назначение операции: логический выбор, если значение первого параметра TRUE, то возвращает значение второго, иначе третьего параметра, тип возврата такой же, как у первого параметра, допустима инфиксная запись, пример:

```
str s; bool b; s = (b ? "истина" "ложь");
```

если b == TRUE, то s присвоится слово истина, иначе присвоится слово ложь.

При использовании в выражении типа

```
var = par1 ? par2 par3;
```

эквивалентно:

```
if( par1)
```

```
{ var = par2 }
`else` {var = pa3};
```

но выполняется гораздо быстрее и позволяет не создавать промежуточные переменные при необходимости использования в сложных ситуациях (в выражениях).

4.2 Управление данными

Операции управления данными предназначены для передачи значений между переменными, инициализации, получения сведений о данных и удаления переменных.

4.2.1 =

Форматы вызова:

<parameter0> = <parameter1>;

(<parameter0> = <parameter1>)

= <parameter0> <parameter1>;

(= <parameter0> <parameter1>)

возвращает значение любого типа, возврат можно проигнорировать, число параметров 2;

первый параметр любого типа, допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: присвоение, копирует значение второго параметра в полученную первым переменную, может возвращать скопированное значение для дальнейшего использования (использования в качестве параметра другой операции),
str s1 s2; s1 = "строка"; puts (s2 = s1);
в результате s2 будет содержать такое же значение, как s1, и оно будет напечатано в стандартный вывод.

4.2.2 data

Формат вызова:

data <parameter0> <parameter1> [<parameter2>...<parameter31>];

ничего не возвращает, число параметров от 2 до максимально допустимого;

первый параметр только массив любого типа (если указан не массив, то компиляция будет прервана с сообщением об ошибке);

остальные параметры такого же типа, как первый, любые операнды, массивы не допустимы;

назначение операции: инициализация массива, принимает первым параметром имя массива любого типа, для которого допустимо присвоение, вторым и следующими параметрами значения данных такого же типа, заполняет ими элементы массива по очереди, с 0-го и до последнего, если указано меньше значений, чем размер массива, оставшиеся элементы не изменятся, если указано больше, то лишние значения будут проигнорированы, пример:

```
str message[3]; int i;
    data message "so good" "so so" "so bad";
    for i ((sizeof message) - 1) 0 -1
    {
        stdout message[ i ] " ... ";
    }
```

в результате будет напечатано: so bad ... so so ... so good

4.2.3 sizeof

Формат вызова:

(sizeof <parameter0>)

возвращает беззнаковое целое, параметр один;

параметр любого типа, только массив (если указан не массив, то компиляция будет прервана с сообщением об ошибке);

назначение операции: вычисление размера массива (числа элементов), возвращает беззнаковое целое, равное числу элементов массива, полученного параметром, примеры:

```
float array[100]; puts (sizeof array);
```

в результате будет напечатано число 100

```
float a1[10]; uint a2[(sizeof a1)];
```

в результате массив целых a2 получит такой же размер, как и массив вещественных a1.

4.2.4 sizeof-1

Формат вызова:

(sizeof-1 <parameter0>)

возвращает беззнаковое целое, параметр один;

параметр любого типа, только массив (если указан не массив, то компиляция будет прервана с сообщением об ошибке);

назначение операции: принимает параметром массив любого типа и возвращает его размер минус единица, эквивалентно ((sizeof array) - 1) но более удобно при вычислении размера массива для выполнения циклической операции над его элементами при помощи операции **for**, пример использования:

```
str a[10];
    int i;
    filereads somefile a;
    for i 0 (sizeof-1 a) 1 {puts a[i]};
```

загрузит текстовые строки из файла, имя которого содержится в somefile (подразумевается, что оно задано в Ci-программе ранее) и напечатает их.

Примечание – более удобно использовать для аналогичных целей цикл loop, но в этом случае надо указывать полное количество элементов, то есть, использовать sizeof:

```
str a[10];
    int i;
    filereads somefile a;
    loop {puts a[i]} (sizeof a) i;
```

4.2.5 clear

Формат вызова:

clear <parameter0> [<parameter1>...<parameter31>];

ничего не возвращает, число параметров от 1 до максимального;

параметры любого типа, допустимы только переменные, допустимы массивы, типы всех параметров могут быть различны;

назначение операции: очистка данных, устанавливает в одиночных переменных или во всех элементах массива значения как после их создания, пример:

int a1 a2[10]; a1 = (a2[5] = 10); clear a1 a2; puts a1 " " a2[5];

в результате будет напечатано: 0 0

4.2.6 sort

Формат вызова:

sort <parameter0> [<parameter1>];

ничего не возвращает, число параметров от 1 до 2;

первый параметр любого типа, допустим только массив;

второй параметр типа bool, допустим любой операнд;

назначение операции: сортировка массивов различных типов данных, для которых разрешено сравнение, первый параметр задает массив для сортировки, второй необязательный параметр задает направление сортировки (если опущен или FALSE то сортировка по возрастанию, если TRUE то по убыванию), если указан массив такого типа, который недопустим для сортировки, то программа прерывается с сообщением об ошибке во время выполнения, пример использования:

data fa 0.01 3e8 123.456 987E-2;

puts fa[0] fa[1] fa[2] fa[3] fa[4];

``напечатает: 1.000000E-2 3.000000E8 1.234560E2 9.870000 0.000000

sort fa TRUE;

puts fa[0] fa[1] fa[2] fa[3] fa[4];

``напечатает: 3.000000E8 1.234560E2 9.870000 1.000000E-2 0.000000

data sa "a" "-----" "123456" ". ." "message";

puts sa[0] sa[1] sa[2] sa[3] sa[4];

``напечатает: a ----- 123456 . . message

sort sa;

puts sa[0] sa[1] sa[2] sa[3] sa[4];

``напечатает ----- . . 123456 a message

4.3 Арифметика

Арифметические операции предназначены для выполнения простейших вычислений. Большинство из них поддерживает различные типы входных данных.

ВНИМАНИЕ! При выполнении операций сложения и умножения результат может переходить через максимально возможное значение для используемого типа данных. Аналогично, при выполнении вычитания и деления результат может переходить через минимально возможное значение. В базовой реализации Ci проверка таких переходов **не производится**, что может быть источником ошибок и неожиданного поведения Ci-программы.

4.3.1 +

Форматы вызова:

(<parameter0> + <parameter1> [<parameter2>...<parameter31>])

(+ <parameter0> <parameter1> [<parameter2>...<parameter31>])

тип возвращаемого значения определяется типом первого параметра, число параметров от 2 до максимального;

первый параметр в базовой реализации может быть типа int, uint и float (данные другого типа вызывают ошибку при компиляции программы), допустим любой операнд, массив не допустим;

второй и последующие параметры такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: вычисление арифметической суммы всех операндов, принимает параметрами и возвращает результат одного и того же типа, примеры использования:

puts (a + 10);

puts (100.1 + (PI / 2));

x = (+ a b c d -12 (a * b));

последнее эквивалентно выражению x = a + b + c + d - 12 + a*b

4.3.2 -

Форматы вызова:

(<parameter0> - <parameter1>)

(- <parameter0> <parameter1>)

(- <parameter0>)

тип возвращаемого значения определяется типом первого параметра, число параметров от 1 до 2;

первый параметр в базовой реализации может быть типа int, uint и float (данные другого типа вызывают ошибку при компиляции программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: при использовании с двумя параметрам возвращает значение первого операнда после вычитания из него значения второго операнда, принимает параметрами и возвращает результат одного и того же типа, примеры использования:

puts (a - 10);

puts (100.1 - 3.14);

puts ((a - b) - c);

при использовании с одним параметром возвращает значение этого параметра с изменением знака на противоположный, пример:

int i -10, a 3;

puts (- (i + a)) (- a);

напечатает: 7 -3

4.3.3 += =+

Форматы вызова:

<parameter0> += <parameter1>;

<parameter0> =+ <parameter1>;

(<parameter0> += <parameter1>)

(<parameter0> =+ <parameter1>)

+= <parameter0> <parameter1>;

=+ <parameter0> <parameter1>;

(+= <parameter0> <parameter1>)

(=+ <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет сложение операндов, записывает результат в первый операнд и может вернуть его значение, оба вида записи выполняются одинаково примеры:

a += 12;

эквивалентно a = (a + 12)

a = (b =+ c);

эквивалентно b = (b + c); a = b

4.3.4 -= =-

Форматы вызова:

<parameter0> -= <parameter1>;

<parameter0> =- <parameter1>;

(<parameter0> -= <parameter1>)

(<parameter0> =- <parameter1>)

-= <parameter0> <parameter1>;

=- <parameter0> <parameter1>;

(-= <parameter0> <parameter1>)

(=- <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет вычитание значения второго операнда из первого, записывает результат в первый операнд и может вернуть его значение, оба вида записи выполняются одинаково примеры:

a -= 12;

эквивалентно a = (a - 12)

a = (b =- c);

эквивалентно b = (b - c); a = b

4.3.5 ++

Форматы вызова:

++ <parameter0>;

<parameter0> ++;

(++ <parameter0>)

(<parameter0> ++)

тип возвращаемого значения определяется типом параметра, параметр один;

параметр может быть типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

назначение операции: арифметический инкремент, выполняет увеличение параметра на 1 (вещественное число увеличивается на 1.0) записывает результат в переменную, которая передана параметром, может возвращать результат инкремента, при этом существенно, в какой форме записана операция, в инфиксной или в префиксной:

a = (++ i); выполняется в следующем порядке:

i увеличивается на 1;

результат присваивается временной переменной;

временная переменная передается вторым параметром операции присваивания;

значение временной переменой копируется в a;

a = (i ++); выполняется в следующем порядке:

i присваивается временной переменной;

i увеличивается на 1;

временная переменная передается вторым параметром операции присваивания;

значение временной переменой копируется в a,

то есть, операция выполняется аналогично тому, как это происходит в программе на языке C – переменная инкрементируется всегда, но в зависимости от того, находится ++ слева или справа от нее, возвращаемое значение различное – либо это значение инкрементированной переменной, либо ее значение до инкремента.

4.3.6 --

Форматы вызова:

-- <parameter0>;

<parameter0> --;

(-- <parameter0>)

(<parameter0> --)

тип возвращаемого значения определяется типом параметра, параметр один;

параметр может быть типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

назначение операции: арифметический декремент, выполняет уменьшение числа на 1 (вещественное число уменьшается на 1.0) записывает результат в переменную, которая передана параметром, может возвращать результат декремента, при этом существенно, в какой форме записана операция, в инфиксной или в префиксной:

a = (-- i); выполняется в следующем порядке:

i уменьшается на 1;

результат присваивается временной переменной;

временная переменная передается вторым параметром операции присваивания.

a = (i --); выполняется в следующем порядке:

i присваивается временной переменной;

i уменьшается на 1;

временная переменная передается вторым параметром операции присваивания

то есть, операция выполняется аналогично тому, как это происходит в программе на языке C – переменная декрементируется всегда, но в зависимости от того, находится ++ слева или справа от нее, возвращаемое значение различное – либо это значение декрементированной переменной, либо ее значение до декремента.

4.3.7 *

Форматы вызова:

(<parameter0> * <parameter1> [<parameter2>...<parameter31>])

(* <parameter0> <parameter1> [<parameter2>...<parameter31>])

тип возвращаемого значения определяется типом первого параметра, число параметров 2;

первый параметр в базовой реализации может быть типа int, uint и float (данные другого типа вызывают ошибку при компиляции программы), допустим любой операнд, массив не допустим;

второй и последующие параметры такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: арифметическое произведение всех параметров, возвращает результат произведения, допустима инфиксная запись, примеры:

```
puts (a * 10);
puts (100.1 * (PI / 2));
x = (* a b c d -12 (a + b));
```

последнее эквивалентно выражению x = a * b * c * d * -12 * (a + b)

4.3.8 /

Форматы вызова:

(<parameter0> / <parameter1>)

(/ <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число

параметров 2;

первый параметр в базовой реализации может быть типа int, uint и float (данные другого типа вызывают ошибку при компиляции программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, любые операнды, массивы не допустимы, если значение второго параметра равно 0, программа будет прервана с сообщением об ошибке;

назначение операции: возвращает результат деления первого операнда на второй, допустима инфиксная запись, пример использования:

puts (a / 10); puts (100.1 / 3.14); puts ((a / b) / c);

4.3.9 %

Форматы вызова:

(<parameter0> % <parameter1>)

(% <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2;

первый параметр в базовой реализации может быть типа int, uint и float (данные другого типа вызывают ошибку при компиляции программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, любые операнды, массивы не допустимы, если значение второго параметра равно 0, программа будет прервана с сообщением об ошибке;

назначение операции: выполняет арифметическое деление делимого на делитель и возвращает остаток от деления такого же типа, как тип первого параметра, допустима инфиксная запись.

Если производится получение остатка при делении вещественных чисел, то вычисляется он следующим образом: остаток равен дополнению до величины делимого произведения делителя и целого числа, такого, что это произведение меньше делимого, но максимально близко к нему:

puts (14.1 % 3.2);

выведет 1.300000, вычисляется это следующим образом: 14.1 / 3.2 = 4.40625; округление до целого вниз 4.40625 = 4; 14.1 - 3.2 * 4 = 1.3

Примеры правильного использования:

```
puts (a % 10);
puts (100.1 % 3.14);
puts ((a % b) % c);
```

4.3.10 *= =*

Форматы вызова:

<parameter0> *= <parameter1>;
<parameter0> =* <parameter1>;
(<parameter0> *= <parameter1>)
(<parameter0> =* <parameter1>)
*= <parameter0> <parameter1>;
=* <parameter0> <parameter1>;
(*= <parameter0> <parameter1>)
(=* <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет умножение значения второго операнда на первый, записывает результат в первый операнд и может вернуть его значение, допустима инфиксная запись, оба вида записи выполняются одинаково, примеры:

```
a *= 12;
```

эквивалентно a = (a * 12)

```
a = (b =* c);
```

эквивалентно b = (b * c); a = b

/=

=/

Форматы вызова:

<parameter0> /= <parameter1>;

<parameter0> =/ <parameter1>;

(<parameter0> /= <parameter1>)

(<parameter0> =/ <parameter1>)

/= <parameter0> <parameter1>;

=/ <parameter0> <parameter1>;

(/= <parameter0> <parameter1>)

(=/ <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим, если значение второго параметра равно 0, программа будет прервана с сообщением об ошибке;

назначение операции: выполняет деление значения второго операнда на первый, записывает результат в первый операнд и может вернуть его значение, допустима инфиксная запись, оба вида записи выполняются одинаково примеры:

a /= 12;

эквивалентно a = (a / 12)

a = (b =/ c);

эквивалентно b = (b / c); a = b

4.3.11 abs

Формат вызова:

(abs <parameter0>)

тип возвращаемого значения определяется типом первого параметра, число параметров 1;

параметр типа int, uint, float (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

назначение операции: возвращает абсолютное (без учета знака) значение параметра, беззнаковые целые возвращаются без изменений, пример:

a = (abs -100); a = (abs 100); `в обоих случаях a получит значение 100`

4.4 Поразрядные операции

4.4.1 ~

Формат вызова:

(~ <parameter0>)

тип возвращаемого значения определяется типом первого параметра;

первый параметр в базовой реализации может быть типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустим любой операнд, массив не допустим;

назначение операции: возвращает значение параметра с инвертированными состояниями всех разрядов, пример:

a = (~ b);

4.4.2 |

Форматы вызова:

(<parameter0> | <parameter1>)

(| <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2;

первый параметр в базовой реализации может быть типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: возвращает значение поразрядного ИЛИ между операндами, пример использования:

a = (b | 2)

присвоит a значение b с установленным в 1 вторым битом.

4.4.3 &

Форматы вызова:

(<parameter0> & <parameter1>)

(& <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров от 2 до максимального;

первый параметр в базовой реализации может быть типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: возвращает значение поразрядного И между операндами, пример использования:

a = (b & 2)

присвоит a значение b со сброшенными в 0 всеми битами, кроме второго, его значение не изменится.

4.4.4 &~

Форматы вызова:

(<parameter0> &~ <parameter1>)

(&~ <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров от 2 до максимального;

первый параметр в базовой реализации может быть типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустим любой операнд, массив не допустим;

второй параметр такого же типа, как первый, допустимы любые операнды, массивы не допустимы;

назначение операции: возвращает значение поразрядного И между первым

операндом и инвертированным вторым, используется для сброса битов по маске, пример использования:

a = (b &~ 2)

присвоит a значение b со сброшенным вторым битом, эквивалентно недопустимому выражению a = (b & (~ 2)) поскольку позволяет использовать константу – простая инверсия допустима только над целыми числами, а константа первым параметром у ~ определяется, как вещественное число.

4.4.5 |= =|

Форматы вызова:

<parameter0> |= <parameter1>;

<parameter0> =| <parameter1>;

(<parameter0> |= <parameter1>)

(<parameter0> =| <parameter1>)

|= <parameter0> <parameter1>;

=| <parameter0> <parameter1>;

(|= <parameter0> <parameter1>)

(=| <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет поразрядное ИЛИ операндов, записывает результат в первый операнд и может вернуть его значение, применяется для установки отдельных разрядов в переменной (1 в тех разрядах, которые надо установить), допустима инфиксная запись, оба вида записи выполняются одинаково, примеры:

a |= 12;

эквивалентно a = (a | 12)

a = (b =| c);

эквивалентно b = (b | c); a = b

4.4.6 &= =&

Форматы вызова:

<parameter0> &= <parameter1>;

<parameter0> =& <parameter1>;

(<parameter0> &= <parameter1>)

(<parameter0> =& <parameter1>)

&= <parameter0> <parameter1>;

=& <parameter0> <parameter1>;

(&= <parameter0> <parameter1>)

(=& <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет поразрядное И операндов, записывает результат в первый операнд и может вернуть его значение, применяется для сброса отдельных разрядов в переменной при помощи инверсной маски (с 0 в тех разрядах, которые надо сбросить), допустима инфиксная запись, оба вида записи выполняются одинаково, примеры:

a &= 12;

эквивалентно a = (a & 12)

a = (b =& c);

эквивалентно b = (b & c); a = b

4.4.7 =&~

Форматы вызова:

<parameter0> =&~ <parameter1>;

(<parameter0> =&~ <parameter1>)

=&~ <parameter0> <parameter1>;

(=&~ <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр такого же типа, как первый, допустим любой операнд, массив не допустим;

назначение операции: выполняет поразрядное И первого операнда и инверсного второго, записывает результат в первый параметр и может вернуть его значение, применяется для сброса отдельных разрядов в переменной (первый параметр) по маске (второй параметр), маска должна быть в прямой форме (1 в тех разрядах, которые надо сбросить), допустима инфиксная запись, оба вида записи выполняются одинаково, примеры:

a =&~ 12;

эквивалентно a = (a & (~ 12)), но позволяет использовать константу, поскольку у операции ~ числовая константа определяется, как вещественное число.

a = (b =&~ c);

эквивалентно b = (b & (~c)); a = b

<<

Форматы вызова:

(<parameter0> << <parameter1>)

(<< <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустимы любые операнды, массив не допустим;

второй параметр типа uint, допустимы любые операнды, массив не допустим;

назначение операции: выполняет сдвиг значения первого параметра влево на число разрядов, заданное вторым параметром, выполняется аналогично одноимен-

ной функции языка C (младшие разряды заполняются 0-ми, старшие пропадают), возвращает значение после сдвига, допустима инфиксная запись, пример:

a = ((b | 3) << 4)

выполнит установку в значении b (но не в переменной b!) двух младших битов и сдвинет все значение влево на 4 разряда, что эквивалентно умножению на 16.

4.4.8 >>

Форматы вызова:

(<parameter0> >> <parameter1>)

(>> <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустимы любые операнды, массив не допустим;

второй параметр типа uint, допустимы любые операнды, массив не допустим;

назначение операции: выполняет сдвиг значения первого параметра вправо на число разрядов, заданное вторым параметром, выполняется аналогично одноименной функции языка C (старшие разряды заполняются 0-ми, младшие пропадают), возвращает значение после сдвига, допустима инфиксная запись, пример:

a = ((b | 64) >> 3)

выполнит установку в значении b (но не в переменной b!) 7-го бита и сдвинет все значение вправо на 3 разряда, что эквивалентно делению на 8.

4.4.9 <<= =<<

Форматы вызова:

<parameter0> <<= <parameter1>;

<parameter0> =<< <parameter1>;

(<parameter0> <<= <parameter1>)

(<parameter0> =<< <parameter1>)

<<= <parameter0> <parameter1>;

=<< <parameter0> <parameter1>;

(<<= <parameter0> <parameter1>)

(=<< <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр типа uint, допустимы любые операнды, массив не допустим;

назначение операции: выполняет сдвиг значения первого параметра влево на число разрядов, заданное вторым параметром, после чего значение записывается обратно в первый операнд, выполняется аналогично одноименной функции языка C, возвращает значение после сдвига, оба варианта записи одинаковы, допустима инфиксная запись, примеры:

b <<= 3;

выполнит присвоение переменной b значения этой переменной, сдвинутого влево на 3 разряда (эквивалентно b *= 8)

a = (b <<= 3)

выполнит присвоение переменной b значения этой переменной, сдвинутого влево на 3 разряда и присвоит результат переменной a

4.4.10 >>= =>>

Форматы вызова:

<parameter0> >>= <parameter1>;

<parameter0> =>> <parameter1>;

(<parameter0> >>= <parameter1>)

(<parameter0> =>> <parameter1>)

>>= <parameter0> <parameter1>;

=>> <parameter0> <parameter1>;

(>>= <parameter0> <parameter1>)

(=>> <parameter0> <parameter1>)

тип возвращаемого значения определяется типом первого параметра, число параметров 2, возвращаемое значение можно игнорировать;

первый параметр типа int или uint (данные другого типа вызывают ошибку при выполнении программы), допустима только переменная, массив не допустим;

второй параметр типа uint, допустимы любые операнды, массив не допустим;

назначение операции: выполняет сдвиг значения первого параметра вправо на число разрядов, заданное вторым параметром, после чего значение записывается обратно в первый параметр, выполняется аналогично одноименной функции языка C, возвращает значение после сдвига, оба варианта записи одинаковы, допустима инфиксная запись, примеры:

b >>= 3; `эквивалентно` b = (b >> 3);

выполнит присвоение переменной b значения этой переменной, сдвинутого вправо на 3 разряда (эквивалентно b /= 8).

a = (b <<= 3)

выполнит присвоение переменной b значения этой переменной, сдвинутого вправо на 3 разряда и присвоит результат переменной a.

Примечание: операции поразрядного сдвига эквивалентны операциям деления и умножения на такую же степень числа 2, на сколько разрядов сдвигается число, но они более удобны и экономичны, когда в результате вычислений или ввода внешних данных задается число разрядов, на которое необходимо сдвинуть число – в противном случае его придется вычислять возведением 2 в степень, а в языке Ci возведение в степень выполняется только над вещественными числами, что потребует выполнения операций преобразования типа числа от целого к вещественному и обратно, и даже может привести к ошибке в вычислениях.

4.5 Математика

Сюда входит набор операций для математических вычислений. Преимущественно вычисления производятся над числами типа float (вещественными).

4.5.1 sin

Формат вызова:

(sin <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает синус угла, передаваемого па-

раметром, угол в радианах, пример:

a = (sin b);

4.5.2 cos

Формат вызова:

(cos <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает косинус угла, передаваемого параметром, угол в радианах, пример:

a = (cos b);

4.5.3 arcsin

Формат вызова:

(arcsin <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает арксинус величины, передаваемой параметром, в виде угла в радианах, если параметром передается значение меньше -1.0 или больше 1.0, выполнение программы прерывается с сообщением об ошибке, пример использования:

a = (arcsin b);

4.5.4 arccos

Формат вызова:

(arccos <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает арккосинус величины, пере-

даваемой параметром, в виде угла в радианах, если параметром передается значение меньше -1.0 или больше 1.0, выполнение программы прерывается с сообщением об ошибке, пример использования:

a = (arccos 0.112);

4.5.5 tan

Формат вызова:

(tan <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает тангенс (sin/cos) угла, передаваемого параметром, угол в радианах, пример:

a = (tan b);

4.5.6 cotan

Формат вызова:

(cotan <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает котангенс (1/тангенс) угла, передаваемого параметром, угол в радианах, если при вычислении значение тангенса получилось равным 0.0, выполнение прерывается с сообщением об ошибке, пример использования:

a = (cotan b);

4.5.7 arctan

Фформат вызова:

(arctan <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допу-

стим;

назначение операции: вычисляет и возвращает арктангенс угла, передаваемого параметром, угол в радианах, пример:

a = (arctan b);

4.5.8 arctan2

Формат вызова:

(arctan2 <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает арктангенс угла, определяемого отношением параметров, угол в радианах, пример:

a = (arctan2 b c); `эквивалентно a = (arctan (b / c))`

4.5.9 ^^

Формат вызова:

(^^ <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра целое число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение первого параметра, возведенное в степень, задаваемую вторым параметром, если значение первого параметра отрицательное, а второй параметр имеет дробную часть не равную 0.0, то программа прерывается с сообщением об ошибке, поскольку должен получиться комплексным, допустима инфиксная запись, пример использования:

a = (b ^^ c);

4.5.10 sqrt

Формат вызова:

(sqrt <parameter0>)

возвращает вещественное число, принимает один параметр;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает квадратный корень из значения параметра, если параметр меньше 0.0, то выполнение программы прерывается с ошибкой, пример использования:

a = (sqrt b);

4.5.11 sqrt2

Формат вызова:

(sqrt2 <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра целое число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение корня из суммы квадратов параметров, пример использования:

a = (sqrt2 b c); `эквивалентно a = (sqrt ((b ^^ 2) + (c ^^ 2)))`

4.5.12 _1/sqrt

Формат вызова:

(_1/sqrt <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение, равное 1.0 деленное на квадратный корень из значения параметра, если значение параметра меньше или равно 0.0, выполнение программы будет прервано с ошибкой, пример использова-

ния:

a = (_1/sqrt b); `эквивалентно a = (1 / (sqrt b))`

4.5.13 e^

Формат вызова:

(e^ <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение e, возведенное в степень, задаваемую параметром, пример использования:

a = (e^ b);

4.5.14 log

Формат вызова:

(log <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение натурального логарифма величины, задаваемой параметром, если параметр меньше или равен 0.0, программа будет прервана с сообщением об ошибке, пример использования:

a = (log b);

4.5.15 log10

Формат вызова:

(log10 <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: вычисляет и возвращает значение десятичного логарифма величины, задаваемой параметром, если параметр меньше или равен 0.0, про-

грамма будет прервана с сообщением об ошибке, пример использования:

a = (log10 b);

4.5.16 floor

Формат вызова:

(floor <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение параметра, округленное до ближайшего максимального целого (округление вверх), пример использования:

a = (floor b);

4.5.17 ceil

Формат вызова:

(ceil <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение параметра, округленное до ближайшего минимального целого (округление вниз), пример использования:

a = (ceil b);

4.5.18 frexp

Формат вызова:

(frexp <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра целое число, допустима только переменная, массив не допустим;

назначение операции: возвращает нормализованную дробную часть значения

первого параметра, если первый параметр не равен 0.0, то возвращаемое значение равно X / 2^N, где X значение первого параметра, во второй параметр записывается целочисленная величина N, возвращаемое значение всегда находится между 0.5 включительно и 1 исключительно, если первый параметр равен 0.0, то возвращается 0.0 и во второй параметр записывается 0, пример использования:

a = (frexp b c);

Примечание: для получения ненормализованного значения дробной и-или целой части вещественного числа предназначена операция ***fraction.***

4.5.19 *2^

Формат вызова:

(*2^ <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение первого параметра, умноженное на 2, возведенное в степень, равную значению второго параметра, допустима инфиксная запись, пример использования:

a = (b *2^ c); `эквивалентно a = (b * (2 ^ c ))`

4.5.20 fraction

Формат вызова:

(fraction <parameter0> [<parameter1>])

возвращает вещественное число, принимает один или два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, допустима только переменная, массив не допустим;

назначение операции: возвращает значение дробной части первого параметра, если второй параметр присутствует, то в него записывается интегральная (целая)

часть первого параметра, пример использования:

```
float a b c;
        b = 123.456;
        a = (fraction b c);
        puts a c;
```

напечатает 4.560000E-1 1.230000E2

```
a = (fraction b);
```

4.5.21 sign

Формат вызова:

(sign <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает знак первого параметра, если значение первого параметра больше 0.0, возвращается 1.0, если меньше 0.0, возвращается -1.0, если равно 0.0, возвращается 0.0 (возможное в машинном представлении отрицательное значение -0.0 не учитывается), пример использования:

```
a = (sign b);
```

4.5.22 min

Формат вызова:

(min <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение минимального из двух полученных параметров, пример использования:

```
a = (min b c);
```

4.5.23 max

Формат вызова:

(max <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение минимального из двух полученных параметров, пример использования:

a = (max b c);

4.5.24 limiter

Формат вызова:

(limiter <parameter0> <parameter1>)

возвращает вещественное число, принимает два параметра;

тип первого параметра вещественное число, любой вид операнда, массив не допустим;

тип второго параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение сигмоида от первого параметра, умноженного на значение второго параметра, сигмоид вычисляется по формуле 1 / (1 + e^X), где X значение первого параметра, используется для экспоненциального ограничения амплитуды сигнала, независимо от знака у текущего отсчета, пример использования:

a = (limiter b c); `эквивалентно a = (c * (1 / (1 + (e^ b))))`

4.5.25 ^2

Формат вызова:

(^2 <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допу-

стим;

назначение операции: возвращает значение параметра, возведенного в квадрат, допустима инфиксная запись, пример использования:

a = (b ^2); `эквивалентно a = (b ^^ 2);

4.5.26 _1/

Формат вызова:

(_1/ <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение, равное 1.0 деленное на значение параметра, если параметром передается значение, равное 0.0, программа будет прервана с сообщением об ошибке, пример использования:

a = (_1/ b); `эквивалентно a = (1 / b);

4.5.27 rad

Формат вызова:

(rad <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение в радианах, равное значению угла в градусах, полученному параметром, пример использования:

a = (rad b); `эквивалентно a = (b * (PI / 180));

4.5.28 grad

Формат вызова:

(grad <parameter0>)

возвращает вещественное число, принимает один параметр;

тип параметра вещественное число, любой вид операнда, массив не допустим;

назначение операции: возвращает значение в градусах, равное значению угла в радианах, полученному параметром, пример использования:
a = (grad b); `эквивалентно a = (b * (180 / PI));`

4.5.29 norm

Формат вызова:

norm <parameter0> <parameter1> <parameter2>;

ничего не возвращает, принимает три параметра;

первый параметр типа целое, беззнаковое целое, вещественное число, допустим только массив;

второй параметр такого же типа, как первый, любой операнд, массив не допустим;

третий параметр такого же типа, как первый, любой операнд, массив не допустим;

назначение операции: производит нормализацию значений массива, принимаемого первым параметром, нормализация производится по формуле: X[i] = N * X[i] / M, где X[i] – каждый элемент нормализуемого массива, N значение первого параметра (масштаб), M значение второго параметра (делитель), если значение второго параметра равно 0.0, программа будет прервана с ошибкой, пример использования:
uint a[128]; getsignal a; norm a 1000 10;

4.5.30 random

Формат вызова:

(random [<parameter0> [<parameter1>]])

возвращает беззнаковое целое число, число параметров переменное от 0 до 2;

первый параметр, если указывается, имеет тип беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр, если указывается, имеет тип беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает псевдослучайное число, если параметров

нет, то число находится в диапазоне от 0 до максимального возможного, задаваемого библиотеками языка С, если указан один параметр, то он задает верхнюю границу случайного числа, если указаны два параметра, то первый задает верхнюю, а второй нижнюю границы:

```
a = (rand);          `случайное число от 0 до максимального знакового целого`
a = (rand 100);      `случайное число от 0 до 100`
a = (rand 10 100);          `случайное число от 10 до 100`
```

4.5.31 seed

Формат вызова:

seed;

ничего не возвращает, параметров не имеет;

назначение операции: производит сброс генератора случайных чисел, если не вызвать эту операцию перед использованием rand, то rand будет всегда возвращать одну и ту же последовательность псевдослучайных чисел, пример использования:

```
seed;
```

4.5.32 sinusoid

Формат вызова:

(sinusoid <parameter0> <parameter1> <parameter2> <parameter3>)

возвращает вещественное число, принимает четыре параметра;

все параметры вещественные числа, допустимы любые операнды, массивы не допустимы;

назначение операции: вычисляет и возвращает значение по формуле A * sin(B * C + D), где A, B, C D – параметры, соответственно от первого до четвертого, операция применяется для генерации синусоидальных колебаний, пример использования:

```
a = (sinusoid amp freq time phase);
```

4.5.33 cosinusoid

Формат вызова:

(cosinusoid <parameter0> <parameter1> <parameter2> <parameter3>)

возвращает вещественное число, принимает четыре параметра;

все параметры вещественные числа, допустимы любые операнды, массивы не допустимы;

назначение операции: вычисляет и возвращает значение по формуле: A * cos(B * C + D), где A, B, C D – параметры, соответственно от первого до четвертого, операция применяется для генерации косинусоидальных колебаний, пример использования:

a = (cosinusoid amp freq time phase);

4.6 Строки

Набор операций для работы со строчными переменными для поддержки типа данных str. Операции следят за допустимостью выполняемых действий, но большинство из них не прерывает выполнение программы. Если производится удлинение строки каким-либо способом, и результат получается длиннее буфера, то строка автоматически обрезается до длины буфера. Гарантируется правильное выполнение строчных операций только с символами ANSI, состоящими из 1 байта. UNICODE не поддерживается.

4.6.1 concat

Формат вызова:

(concat <parameter0> <parameter1> [<parameter2>...<parameter31>])

возвращает строку, принимает переменное число параметров от 2 до максимального допустимого;

первый параметр строка, допустим любой операнд, массив не допустим;

второй параметр строка, допустим любой операнд, массив не допустим;

параметры с третьего по последний такие же, как предыдущий;

назначение операции: выполняет конкатенацию (слияние) строк, полученных параметрами, возвращает результат конкатенации, пример использования:

b = 1001.3; str a = (concat “значение переменной b равно ” b);

в результате переменная a будет содержать строку “значение переменной b равно 1.0013E3”.

4.6.2 length

Формат вызова:

(length <parameter0>)

возвращает беззнаковое целое, принимает один параметр;

параметр типа строка, допустим любой операнд, массив не допустим;

назначение операции: вычисляет и возвращает длину строки, полученной параметром, возвращается число байт без учета завершающего ноля, но если в середине строки встречается нулевой байт, он тоже учитывается в общей длине, пример использования:

b = (length “длина этой строки равна 26”);

переменной b будет присвоено значение 26;

puts (length “”);

напечатает 0.

4.6.3 substr

Формат вызова:

(substr <parameter0> <parameter1> [<parameter3>])

возвращает строку, число параметров переменное от 2 до 3;

первый параметр типа строка, допустим любой операнд, массив не допустим;

второй параметр беззнаковое целое, допустим любой операнд, массив не допустим;

третий параметр беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает часть строки, полученной первым параметром (подстроку), второй параметр задает номер символа, начиная с которого извлекается подстрока, если третий параметр опущен, то извлекается только один символ, если третий параметр указан, то он задает длину подстроки, которая будет из-

влечена, если же результат переходит через границу исходной строки, то извлечение прекращается, байты нумеруются, как элементы массива, начиная с нулевого, пример использования:

puts (subs “abcdefijklmnop” 3 4);

напечатает defi

4.6.4 char

Формат вызова:

(char <parameter0> <parameter1>)

возвращает беззнаковое целое, принимает 2 параметра;

первый параметр типа строка, допустим любой операнд, массив не допустим;

второй параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает код символа ASCII, находящегося в строке в позиции, передаваемой вторым параметром, если указана позиция больше или равная длине строки, то возвращается код 0, пример:

puts (char "1a2b3c4e5f6h" 5) (hex (char "1a2b3c4e5f6h" 3)) (char "1a2b3c4e5f6h" 30);

напечатает 99 62 0

4.6.5 insubs

Формат вызова:

insubs <parameter0> <parameter1> <parameter3>;

ничего не возвращает, принимает три параметра;

первый параметр типа строка, допустима только переменная, массив не допустим;

второй параметр строка, допустим любой операнд, массив не допустим;

третий параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: заменяет в строке, полученной первым параметром,

часть текста на подстроку, полученную вторым параметром, третьим параметром указывается позиция, с которого производится замена, пример использования:

puts (insubs “abcdefijklmnop” “123456” 3);

напечатает abc123456lmnop

4.6.6 lowr

Формат вызова:

(lowr <parameter0>)

возвращает строку, принимает один параметр;

параметр типа строка, допустим любой операнд, массив не допустим;

назначение операции: переводит все символы строки в нижний регистр и возвращает результат, пример использования:

puts (lowr “abCDdeFijklMnoP”);

напечатает abcdefijklmnop

4.6.7 upr

Формат вызова:

(upr <parameter0>)

возвращает строку, принимает один параметр;

параметр типа строка, допустим любой операнд, массив не допустим;

назначение операции: переводит все символы строки в верхний регистр и возвращает результат, пример использования:

puts (lowr “abCDdeFijklMnoP”);

напечатает ABCDEFIJKLMNOP

4.6.8 getword

Формат вызова:

(getword <parameter0> <parameter1>)

возвращает строку, принимает два параметра;

первый параметр типа строка, допустим любой операнд, массив не допустим;

второй параметр типа беззнаковое целое, допустим любой операнд, массив

не допустим;

назначение операции: возвращает слово, содержащееся в строке, начиная с символа с номером, указанным вторым параметром, строка передается первым параметром, слово извлекается, пока не будет обнаружен разделитель – символ с кодом менее 0x21, пример использования:

puts (getword “abCDdeFijk lMnoP” 4);

напечатает deFijk

4.6.9 ascii

Формат вызова:

(ascii <parameter0> [<parameter1> ...<parameter31>])

возвращает строку, принимает переменное число параметров от 1 до максимального;

первый параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

второй и последующие параметры такие же, как первый;

назначение операции: формирует строку из символов, ASCII код которых задается числами, передаваемыми параметрами, завершающий ноль указывать не требуется, пример использования:

puts (ascii 0x21 0x22 0x23)

напечатает !”#

4.6.10 hex

Формат вызова:

(hex <parameter0> [<parameter1>])

возвращает строку, принимает переменное число параметров от 1 до 2;

первый параметр беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает строку, содержащую шестнадцатеричное

представление целого числа, первый параметр задает число для преобразования, второй параметр ограничивает длину строки результата указанным числом символов, причем отсечка символов производится не с конца строки, а с ее начала, это удобно, например, если надо вывести только код младшего байта или тетрады, если параметр опущен, то строка не ограничивается.

4.6.11 oct

Формат вызова:

(oct <parameter0> [<parameter1>])

возвращает строку, принимает переменное число параметров от 1 до 2;

первый параметр беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает строку, содержащую восьмеричное представление целого числа, первый параметр задает число для преобразования, второй параметр ограничивает длину строки результата указанным числом символов, причем отсечка символов производится не с конца строки, а с ее начала, это удобно, например, если надо вывести только код младшего байта или тетрады, если параметр опущен, то строка не ограничивается.

4.6.12 bin

Формат вызова:

(bin <parameter0> [<parameter1>])

возвращает строку, принимает переменное число параметров от 1 до 2;

первый параметр беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает строку, содержащую двоичное представление целого числа, первый параметр задает число для преобразования, второй пара-

метр ограничивает длину строки результата указанным числом символов, причем отсечка символов производится не с конца строки, а с ее начала, это удобно, например, если надо вывести только код младшего байта или тетрады, если параметр опущен, то строка не ограничивается.

4.6.13 stdout

Формат вызова:

stdout <parameter0> [<parameter1> ...<parameter31>];

ничего не возвращает, принимает переменное число параметров от 1 до максимального;

первый параметр типа строка, допустим любой операнд, массив не допустим;

остальные параметры такие же, как предыдущий;

назначение операции: в стандартный выходной поток исполняющей системы Ci по очереди выводит строки, получаемые параметрами, перед выводом не формируется временная строка, поэтому общая длина выводимого результата не ограничена, результат буферизуется операционной системой, поэтому может быть не выведен сразу, вывод производится как только в строке встретится символ перевода каретки ('\n') или конца потока (Ctrl-Z), для немедленного вывода (смотри операцию **puts**).

4.6.14 puts

Формат вызова:

puts <parameter0> [<parameter1> ...<parameter31>];

ничего не возвращает, принимает переменное число параметров от 1 до максимального;

первый параметр типа строка, допустим любой операнд, массив не допустим;

остальные параметры такие же, как предыдущий;

назначение операции: выводит полученные параметрами строки, через пробелы, в стандартный выходной поток исполняющей системы Ci, после чего выводит

туда символ перевода каретки, в результате буферизованная системой строка передается в стандартный выходной поток и следующий текст выводится уже с новой строки, пример использования:

puts “строка,“, “состоящая”, “из трех”;

puts “еще строка“

напечатает:

строка, состоящая из трех

еще строка

4.6.15 floatstyle

Формат вызова:

floatstyle <parameter0> [<parameter1>];

ничего не возвращает, принимает от 1 до 2 параметров;

первый параметр типа uint, допустим любой операнд, массив не допустим;

второй параметр типа bool, допустим любой операнд, массив не допустим;

назначение операции: задает формат преобразования вещественных чисел в строку, при использовании с одним параметром изменяет число знаков после запятой, независимо от текущего стиля вывода чисел, при изменении количества знаков преобразуемое число *математически округляется*, при использовании со вторым необязательным параметром, если его значение TRUE, то число будет выводиться в нормальном представлении, то есть, как 1234.56, если значение FALSE или параметр опущен, то в экспоненциальном, то есть, как 1.23456E3, общее число знаков до запятой и после нее не может быть больше 7, пример использования:

float f 9876.5432;

puts f; ``выведет 9876.5432

floatstyle 3;

puts f; ``выведет 9876.543

floatstyle 2 FALSE;

puts f; выведет 9.88E3

floatstyle 4;

puts f; выведет 9.8765E3

4.6.16 fileputs

Формат вызова:

fileputs <parameter0> <parameter1> [<parameter2>...<parameter31>];

ничего не возвращает, принимает от 2 до максимально допустимого числа параметров;

все параметры типа str, допустим любой операнд, массив не допустим;

назначение операции: запись строк в файл на постоянном носителе, первый параметр имя файла, после него от 1 до 32 строк, если файл не существовал, то он создается, если существовал, то строки последовательно дописываются в конец файла, типа файла всегда считается простым текстовым, UNICODE не поддерживается (строки записываются в ASCII), строки записываются аналогично puts, то есть, в конец каждой добавляется перевод на новую строку, а между несколькими строками, заданными параметрами, вставляется пробел.

Имя файла задается в стандарте, принятом для используемой операционной системы, оно не приводится к какому-то определенному формату, поэтому использование **fileputs** делает Ci-программу **системозависимой**.

Пример для ОС Microsoft Windows:

```
fileputs “C:\\TMP\\sample.txt“ “пример создания текстового файла“;
    puts “выполняется запись“;
    fileputs “C:\\TMP\\sample.txt“ “из“, “двух“, “строк“ ;
```

напечатает текст:

выполняется запись

и запишет в файл C:\TMP\sample.txt следующий текст:

пример создания текстового файла
из двух строк

4.6.17 filereads

Формат вызова:

filereads <parameter0> <parameter1>;

ничего не возвращает, принимает 2 параметра;

первый параметр типа str, допустим любой операнд, массив не допустим;

второй параметр типа стр, допустим только массив целиком;

назначение операции: загружает из файла, имя которого заданно первым параметром, текстовые строки в строковый массив, задаваемый вторым параметром, загружается столько строк, сколько элементов у массива, либо пока не обнаружен конец файла, загрузка производится всегда с начала файла, строки считываются без завершающего символа возврата каретки, при файловой ошибке программа прерывается (выполняется блок, заданный операцией onerror), если необходимо заполнять один и тот же массив из разных файлов, то перед каждым новым заполнением следует производить очистку этого массива вызовом операции clear, пример использования (подразумевается, что файл записан при помощи операции **fileputs**).

```
str a[2];
    filereads “C:\\TMP\\sample.txt“ a;
    puts a[0] a[1];
```

напечатает:

пример создания текстового файла из двух строк

4.7 Управление алгоритмом

Набор операций управления алгоритмом предназначен для управления последовательностью выполнения операций, из которых состоит Ci-программа. В традиционных языках это обеспечивается при помощи фиксированного набора операторов. Язык Ci позволяет расширять набор операций управления, включая в него необходимые средства. Например, базовый набор операций не содержит средств поддержки событий, вызываемых мышью, но они могут быть легко добавлены. Базовые операции не поддерживают передачу им массивов типа block. Но при необходимости их можно доработать таким образом, что они будут принимать соответствующие массивы и производить действия над всеми элементами.

Те операции, которые принимают параметр типа block, могут вызывать исполняющую систему Ci, которая производит выполнение этого блока. При этом не обязательно передавать им константу типа блок, заключенную в фигурные скобки. Можно также передавать переменные типа block, которым ранее присвоено какое-то значение соответствующего типа. При этом не страшно, если во время “выполнения” такой переменной ее значение будет изменено (выполнена операция clear или ей будет присвоен новый блок). Операция, получившая такую переменную в качестве параметра, на самом деле, получает откомпилированную константу типа блок, на которую ссылалась переменная. Выполнение кода этой константы будет продолжаться до его нормального завершения. И только при следующем вызове выполнения блочной переменной будет использоваться ее новое значение.

Входящие в базовый комплект Ci операции поддержки циклов loop и for гарантируют, что если ими получена переменная типа блок, и внутри этого блока переменной будет присвоено другое значение, то это не повлияет на работу этих операций. Они будут выполнять циклически именно тот блок, который был связан с полученной ими переменной.

Далее следует описание операций управления алгоритмом.

4.7.1 if

Формат вызова:

if <parameter0> <parameter1> [<parameter2>];

ничего не возвращает, число параметров переменное от двух до трех;

первый параметр типа bool, допустим любой операнд, массив не допустим;

второй параметр типа block, допустим любой операнд, массив не допустим;

третий параметр типа block, допустим любой операнд, массив не допустим;

назначение операции: операция условного выполнения (аналог операторов ветвления в традиционных языках), первым параметром передается логическое значение, если оно истинно (равно TRUE), то будет выполнен программный блок, передаваемый вторым параметром, если же указан третий параметр и первым параметром передано значение ложь, то будет выполнен третий параметр, примеры использования:

bool a; if a {puts “nothing” };

ничего не напечатает, т.к. после создания переменная a имеет значение FALSE.

if(100 > 1) {puts “large”};

напечатает: large

if(value == 100)

{puts “equal”}

`else` {puts “not equal”};

– выполняется так: если value равно 100, то напечатать equal, иначе напечатать not equal.

4.7.2 loop

Формат вызова:

loop <parameter0> [<parameter1> [<parameter2> [<parameter3> [parameter4]]]];

ничего не возвращает, принимает от 1 до 5 параметров;

первый параметр типа block, допустим любой операнд, массив не допустим;

второй параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

третий параметр только числовая переменная (типа int, uint или float), массив не допустим;

четвертый параметр должен быть такого же типа, как третий, допустим любой операнд, массив не допустим;

пятый параметр должен быть такого же типа, как третий, допустим любой операнд, массив не допустим;

назначение операции: выполняет циклически блок с заданным числом повторений и переменной цикла любого численного типа, с заданием начального значения переменной и шага приращения.

Первый параметр задает выполняемый блок, второй параметр задает количество повторений (если не указано, то бесконечно, если 0 то ни разу), третий параметр это автоматически изменяемая в цикле переменная (счетчик цикла), четвертый параметр задает начальное значение переменной цикла (если не указано, то 0, если

же необходимо сохранить полученное ранее значение переменной, то достаточно просто указать ее этим параметром), пятый параметр задает шаг приращения переменной (если не указано, то 1).

Ни один из параметров, кроме блока, не является обязательным, например:

```
loop {puts “message”} 3;
```

напечатает три раза слово message на новой строке, или

```
floatstyle 2 FALSE;
    float f;
    loop {puts f} 3 f;
```

напечатает

```
    0.00
    1.00
    2.00
```

Или:

```
floatstyle 2 FALSE;
    float f;
    loop {puts f} 3 f 0.1;
```

напечатает

```
    1.00E-1
    1.10
    2.10
```

Или:

```
floatstyle 2 TRUE;
    float f;
    loop {puts f} 3 f 0.1 -0.03;
```

напечатает

```
    0.10
    0.07
    0.04
```

Цикл loop с числом повторений 0 можно использовать для исключения из выполнения кода, в котором содержатся комментарии. Поскольку вложенные комментарии в Ci недопустимы, то временно отключить код, уже содержащий множество комментариев, с их помощью затруднительно. Для этого можно использовать конструкцию:

```
loop{
    ``код, содержащий комментарии
    `puts “это не будет напечатано`
    }0;
```

Причем, если в дальнейшем потребуется временно включить этот код для выполнения (при проверке алгоритма), то достаточно заменить 0 на 1 (после отладки желательно убрать цикл, чтобы не усложнять читабельность кода). Разумеется, можно использовать вложенные циклы loop{}0;

ВНИМАНИЕ!

Если изменяемая переменная имеет интегральный тип (int или uint) и достигла своего максимального (или минимального) значения, но выполнение цикла продолжается, то значение изменяемой переменной станет соответственно минимальным (или максимальным), и продолжит увеличиваться (или уменьшаться), но **цикл выполнится то число раз, которое задано вторым параметром.**

Если же изменяемая переменная вещественного типа, то достигнув максимального (или минимального) значения, она *перестанет изменяться*, но цикл также выполнится заданное число раз.

4.7.3 for

Формат вызова:

for <parameter0> <parameter1> <parameter2> <parameter3> <parameter4>;

ничего не возвращает, принимает один параметр;

первый параметр типа целое, допустима только переменная, массив не допу-

стим;

второй параметр типа целое, допустим любой операнд, массив не допустим;

третий параметр типа целое, допустим любой операнд, массив не допустим;

четвертый параметр типа целое, допустим любой операнд, массив не допустим;

пятый параметр типа block, допустим любой операнд, массив не допустим;

назначение операции: счетный цикл, первым параметром принимает переменную (только типа знаковое целое) – параметр (итератор) цикла, ее значение изменяется при каждой итерации цикла, вторым параметром принимает начальное значение для итератора, третьим параметром граничное значение для итератора, четвертым параметром приращение итератора, последним параметром блок, который выполняется на каждой итерации.

Данная реализация цикла предназначена для упрощения переноса алгоритмов, написанных на других языках, она менее мощная и гибкая, чем цикл loop.

Алгоритм работы такой – сначала проверяется, не вышло ли значение итератора за конечное значение, если не вышло, то выполняется блок, после чего итератору прибавляется приращение, и все повторяется, до тех пор, пока значение итератора не перейдет через границу.

Если в процессе выполнения блока переменная-итератор будет изменена, ее новое значение будет учитываться перед следующей итерацией, то есть, если итератор самим циклом увеличивается на величину приращения, а в теле блока он будет уменьшаться на такую же величину, это приведет к бесконечному циклу.

Значение границ, а также приращение внутри блока изменить нельзя (то есть, их изменение не скажется на выполнении цикла).

Если в результате ошибки приращение будет указано таким, что итерация начнет производится в обратном направлении, и текущее значение итератора перейдет через его начальное значение, программа будет прервана с сообщением об ошибке, например, если написано for i 0 10 -1 {}; то цикл будет прерван после первой итерации, поскольку значение i окажется равно -1, то есть, меньше 0, тоже самое произойдет, если в теле цикла значение итератора будет изменено таким образом, что она окажется меньше или больше начального значения (в зависимости от

направления счета).

Если переменная-итератор не изменяется в теле цикла, то по его окончании она содержит на 1 больше (по модулю) чем количество выполнений блока.

Примеры использования:

```
for i 0 9 1 {stdout i" "};
```

будет напечатано 0 1 2 3 4 5 6 7 8 9, после чего цикл прекратится, а переменная i будет иметь значение 10.

```
for i 0 -9 -1 {stdout i" "};
```

будет напечатано 0 -1 -2 -3 -4 -5 -6 -7 -8 -9, после чего цикл прекратится, а переменная i будет иметь значение -10.

4.7.4 continue

Формат вызова:

```
continue;
```

ничего не возвращает, параметров не имеет;

назначение операции: вызывает переход к следующей итерации цикла, при выполнении в любом месте цикла **for** или **loop** приведет к немедленному переходу к следующей итерации, цикл **for** при этом выполнит приращение итератора, если вызывается внутри **exec**, прекращает выполнение всех вложенных блоков до ближайшей сверху операции **call** или цикла, если же вызывается внутри **call**, но не внутри вложенного цикла, то прекращает выполнение функции аналогично **return** без параметров, пример использования:

```
for i 0 9 1
        {
          if( i == 5 )
              {stdout "five "; continue};
          stdout i " "
        };
```

будет напечатано 0 1 2 3 4 five 6 7 8 9

4.7.5 continueif

Формат вызова:

continueif <parameter0>;

ничего не возвращает, принимает один параметр;

параметр типа bool, допустим любой операнд, массив не допустим;

назначение операции: следующая итерация цикла при условии, что значение ее параметра равно TRUE, в остальном аналогична continue, пример использования:

```
for i 0 9 1
    {
      continueif( i == 5 )
      stdout i" "
    };
```

будет напечатано 0 1 2 3 4 6 7 8 9

4.7.6 break

Формат вызова:

break;

ничего не возвращает, параметров не имеет;

назначение операции: немедленное прерывание цикла, при выполнении в любом месте цикла **for** или **loop** (*либо* операции **switch**) приведет к завершению цикла, итератор цикла **for** при этом будет иметь текущее значение, важным отличием от языка C является то, что **break** прерывает выполнение цикла даже, если он выполняется внутри блока, вызванного операцией **exec**, если вызывается внутри **exec**, прекращает выполнение всех вложенных блоков до ближайшей сверху операции **call** или цикла, если же вызывается внутри **call**, но не внутри цикла, то прекращает выполнение функции аналогично **return** без параметров, примеры использования:

```
for i 0 9 1
    {
      if( i == 5 )
          {break};
      stdout i" "
```

```
};
```

будет напечатано 0 1 2 3 4

```
block b {stdout “breaking”; break};
    for i 0 9 1
    {
        exec b;
        stdout “not executed”;
    }
```

будет напечатано: breaking – необходимо отметить, что переменная i внутри блока b недоступна.

break внутри операции **switch** прерывает выполнение блока, в котором встречается, *но не прерывает цикл, в котором используется* **switch** – это сделано для облегчения переноса алгоритмов, написанных на C.

4.7.7 breakif

Формат вызова:

breakif <parameter0>;

ничего не возвращает, один параметр;

параметр типа bool, допустим любой операнд, массив не допустим;

назначение операции: прерывание цикла при выполнении условия, полностью аналогична **break**, но прерывает цикл только если полученное параметром логическое значение равно TRUE, пример использования:

```
for i 0 9 1
{
   breakif( i == 5 );
   stdout i” “
};
```

будет напечатано 0 1 2 3 4

4.7.8 switch

Формат вызова:

switch <parameter0> <parameter1> [<parameter2>…<parameter31>];

ничего не возвращает, число параметров переменное, от 2 до максимально допустимого;

первый параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр типа блок, допустим любой операнд, массив не допустим;

остальные параметры такие же, как предыдущий;

назначение операции: вариант условного выполнения, аналогичный switch/case в языке C, первым параметром принимает номер выполняемого блока, после чего выполняет блок, полученный соответствующим параметром (нумерация параметров от 0), если же блоков получено меньше, чем требуемый номер, то не выполняет ничего, наиболее эффективно совместное использование с **case** (смотри далее), пример:

```
for i 0 3 1
        {
            switch i
                {}
                {stdout “one ”}
                {stdout “two ”}
                {stdout “three ”};
        };
```

будет напечатано:

one two three

Выполнение любого блока может быть прервано при помощи вызова операций **break** или **breakif**.

4.7.9 case

Формат вызова:

(case <parameter0> <parameter1> [<parameter2>…<parameter31>])

возвращает беззнаковое целое, число параметров от двух до максимально допустимого;

первый параметр любого типа, допустим любой операнд, массив не допу-

стим;

последующие параметры такие же, как предыдущий;

назначение операции: значение первого параметра сравнивается по очереди со значениями второго и последующих параметров и, если обнаружено совпадение, возвращается номер совпавшего параметра минус один, если не найдено совпадения первого параметра с любым из следующих, то будет возвращено число параметров минус один, допустима инфиксная запись, если получен параметр типа, для которого недопустимо сравнение (например, блок), то выполнение прерывается с ошибкой.

Совместно с **switch** эта операция более мощная, чем комбинация switch/case в языке C, поскольку в C для switch допустимы только интегральные (целочисленные) типы значений, а для Ci такого ограничения нет, например, его можно использовать для выбора совпадающей строки, примеры использования:

puts (case (concat “ab” “cd”) “decf” “edfa” “abcd” “afdc”);

напечатает: 2

switch (floatvar case 0.01, PI, 1e12, (sin val))

{puts “case 0”}

{puts “case 1”}

{puts “case 2”}

{puts “case 3”}

{puts “default no one”};

напечатает слово case и номер параметра операции **case**, которому равна переменная floatvar, либо default no one, если floatvar не равна ни одному из указанных значений.

4.7.10 exec

Формат вызова:

exec <parameter0> [<parameter1>…<parameter31>];

ничего не возвращает, принимает число параметров от одного до максимально допустимого;

первый параметр типа блок, допустим любой операнд, массив не допустим;

второй и следующие параметры такие же, как предыдущий;

назначение операции: выполняет последовательно один за другим блоки, по-

лученные параметрами, в сами блоки передать что-либо нельзя, а если в них встретятся вызовы операций **par#** или **par@**, то будет считаться, что они имеют отношение к ближайшей сверху операции **call**, операция exec аналогична операции **call** без параметров функции, но выполняется значительно быстрее, с ее помощью реализованы операции **for** и **loop**, пример использования:

block a; a = {puts “OOPS! “}; exec a; puts “OOCH! “; exec a;

напечатает: OOPS! OOCH! OOPS!

4.7.11 call

Формат вызова:

call <parameter0> [<parameter1>…<parameter31>];

возвращает значение неопределенного заранее типа, тип возврата определяется динамически, число параметров от 1 до максимально допустимого;

первый параметр типа блок, допустим любой операнд, массив не допустим;

второй и следующие параметры любого типа, допустимы любые операнды, массивы не допустимы;

назначение операции: вызов *функции* с передачей параметров и возможным возвратом значения; первым параметром принимает блок; логично передавать заданную и инициализированную ранее переменную типа блок, поскольку в передаче константы нет смысла, но также можно передавать результат вызова функции, которая может вернуть блок, в блок можно передать параметры, причем компилятор *не контролирует* их типы, параметры кладутся на внутренний стек и становятся доступны операциям **par#** и **par@**, для контроля типов внутри функции необходимо использовать указание типа возврата.

Допускаются вложенные вызовы операции **call**, а также рекурсивные вызовы, глубина рекурсии ограничивается умолчательной величиной стека параметров функций, который рассчитан на 65535 параметров и возвращаемых значений, если стек достигает своей максимальной глубины, выполнение программы прерывается с сообщением об ошибке (то же самое для операции **exec)**.

ВНИМАНИЕ!

В текущей реализации Ci имеется существенное отличие от других языков при рекурсивных вызовах – внутренние переменные блока в текущей реализации *используются повторно* – в то время, как в C они отдельные для каждого нового вхождения в рекурсию, то есть:

```
block b
{
uint i;
puts i;
i ++;
call b;
}
```

будет печатать инкрементируемое число, поскольку значение i сохраняется при каждом входе в рекурсию, как если бы переменная i была вне блока b.

Данное свойство рекурсий не следует использовать, поскольку в дальнейшем это свойство будет ликвидировано, и будут создаваться собственные переменные для каждого уровня рекурсии;

операция **call** может вернуть значение, если в вызываемом ей блоке есть вызов операции **setret** или **return** с параметром (см. их описания), пример использования:

```
block b {
        stdout "all " (par# 0)
    };
    call b "ok\n";
```

напечатает: all ok

```
block b
    {
        stdout "b ", (par# 0), ((int par# 1) + 10), (par# 2)
    };
    call b "ok" 12 "\n";
```

напечатает: b ok22

4.7.12 par#

Формат вызова:

(par# <parameter0> [<parameter1>])

возвращает значение неопределенного заранее типа, тип возврата определяется динамически, число параметров от одного до двух;

первый параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр любого типа, допустим любой операнд, массив не допустим;

назначение операции: используется внутри функции, вызванной операцией **call**, для получения одного из параметров функции (нельзя путать параметры функции с параметрами операции), параметры нумеруются от 0.

Если вызов этой операции используется там, где не определен тип требуемого параметра (например, первым параметром арифметической операции), то необходимо указывать ее с префиксом в виде имени типа, который она должна вернуть – в этом случае компилятор сгенерирует необходимые преобразования типов, иначе он выдаст ошибку “необходимо указание типа”.

Если операция указана с двумя параметрами, то второй может быть любого типа, он задает умолчательное значение, которое будет возвращено этой операцией, если соответствующий параметр отсутствует при вызове функции – это позволяет создавать функции с переменным числом параметров и приданием им значений по умолчанию.

Рекомендуется в начале функции присвоить значения полученных ею параметров переменным с осмысленными именами, и далее использовать эти переменные, поскольку код станет не только понятнее, но и быстрее.

Примеры использования:

см. примеры операции **call**

```
block b;
        code b
        {
```

```
uint v1; v1 = (par# 1 100);
puts "value", (par# 0), (v1 + 10)
};
call b "ok" 12; ``вызов b с двумя параметрами
call b “good”;  ``вызов b с одним параметром
```

напечатает:

```
value ok 22
value good 110
```

4.7.13 par@

Формат вызова:

par@ <parameter0> <parameter1>;

ничего не возвращает, принимает два параметра;

первый параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

второй параметр любого типа, допустим любой операнд, массив не допустим;

назначение операции: используется внутри функции, вызванной операцией **call**, для изменения значения одного из параметров функции (нельзя путать параметры функции с параметрами операции), номер которого задается первым, параметры нумеруются от 0.

Второй параметр может быть любого типа, его значение присваивается переменной, которая была указана при вызове функции в качестве соответствующего параметра.

Если указанный параметр отсутствует, или указана не переменная, то программа будет прервана с сообщением об ошибке.

Пример использования:

```
block b; uint i; str s;
code b
{
par@ 0 ((int par# 2) + 100);
```

```
par@ 1 “ok”
};
call b i s 12;
puts i s;
```

напечатает: 112 ok

4.7.14 npars

Формат вызова:

(npars);

возвращает беззнаковое целое, параметров не имеет;

назначение операции: возвращает число параметров, полученное функцией, что может быть удобно, например, в случае, когда функция принимает переменное число параметров, и их надо обработать в цикле, либо с помощью **switch** выполнить различные действия, в зависимости от числа полученных параметров.

```
block b; uint i; str s;
code b
{
puts (npars);
};
call b i s 12;
```

напечатает: 3

4.7.15 ispar

Формат вызова:

(ispar <parameter0>);

возвращает тип bool, принимает один параметр;

параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает TRUE, если функции передан параметр с номером, указанным в качестве параметра этой операции, или FALSE если такого параметра передано не было, что может быть удобно для различной реализации выполняемых функцией действий, в зависимости от числа получаемых ею параметров,

пример использования:

```
block b; uint i; str s;
        code b
        {
        puts (ispar 1) (ispar 4)
        };
        call b i s 12;
```

напечатает: TRUE FALSE

4.7.16 isparvar

Формат вызова:

(isparvar <parameter0>);

возвращает тип bool, принимает один параметр;

параметр типа беззнаковое целое, допустим любой операнд, массив не допустим;

назначение операции: возвращает TRUE, если функции в качестве параметра с номером, указанным в качестве параметра этой операции, передана переменная (или элемент массива), в противном случае а также, если такого параметра вообще не было передано, возвращает FALSE, это может быть удобно, например если функция может изменить какой-либо параметр, но алгоритм не должен прерваться по ошибке, если этим параметром передана не переменная, тогда можно сначала проверить, передана ли переменная, и если да, то изменять ее значение.

```
block b; uint i; str s;
        code b
        {
        puts (isparvar 0) (isparvar 2) (isparvar 4)
        };
        call b i s 12;
```

напечатает: TRUE FALSE FALSE

4.7.17 return

Формат вызова:

return [<parameter0>];

ничего не возвращает, может вызываться без параметров, или с одним параметром;

если присутствует параметр, то он может быть любого типа, допустим любой операнд, массив не допустим;

назначение операции: прекращает выполнение операции **call**, в блоке-функции которой вызвана, управление возвращается за вызов завершаемой операции **call**.

Если указан параметр, то передает его завершаемой операции **call**, чтобы та вернула его значение, если **call** сделан без использования возвращаемого значения, то оно игнорируется, если же **call** сделан с использованием возвращаемого значения, а **return** внутри него вызван без параметра, то возвращаемое функцией значение зависит от того, была ли ранее использована операция **setret**, если не была, то функция вернет нулевое значение соответствующего типа, примеры использования:

```
block b; uint i;
        code b
        {
        puts 1; return; puts 2;
        };
        call b
```

напечатает: 1

```
block b; uint i;
        code b
        {
        puts 1; return 10; puts 2;
        };
        puts (call b)
```

напечатает:

```
        1
        1.000000E1
```

```
block b; uint i 5;
        code b
        {
        puts 1; return; puts 2;
        };
        i = (call b);
        puts i;
```

напечатает:

```
        1
        0
```

Использование **return** позволяет применять **call** для создания функций, возвращающих значения, и включения их в математические или иные выражения:

```
block diagonal ``вычисляет диагональ прямоугольного треугольника по его катетам
        {
        return( sqrt( ((par# 0) ^2 ) + ((par# 1) ^2) ) )
        };
        block i2m ``переводит дюймы в сантиметры
        {
        return( (float par# 0) * 2.54  )
        };
        puts (( call diagonal (call i2m 4) (call i2m 5) ) * 2)
```

напечатает удвоенную длину диагонали прямоугольника 4x5 дюймов, выраженную в сантиметрах.

4.7.18 setret

Формат вызова:

```
        setret <parameter0>;
```

ничего не возвращает, принимает один параметр;

параметр любого типа, допустим любой операнд, массив не допустим;

назначение операции: приготавливает значение для возврата функции, выполняемой операцией **call**, но выполнение функции не прерывает, используется совместно с последующим вызовом операции **return**, **break** или **continue**, пример использования:

```
block b; uint i; i = 5;
        code b
        {
        setret 3; stdout 1 " "; return; puts 2;
        };
        i = (call b);
        puts i;
```

напечатает: 1 3

4.7.19 code

Формат вызова:

code <parameter0> <parameter1>;

ничего не возвращает, принимает 2 параметра;

первый параметр типа блок, допустима только переменная, массив не допустим;

второй параметр типа блок, допустима переменная, константа, операция, массив не допустим;

назначение операции: эквивалентно присвоению переменной типа блок допустимого значения, включена для совместимости с будущими реализациями Ci.

4.7.20 stop

Формат вызова:

stop;

ничего не возвращает, параметров не имеет;

назначение операции: прекращает выполнение Ci программы, если ранее был задан блок завершения в операции **onstop**, то он выполняется, если **stop** встречается внутри него, то программа немедленно завершается, пример см. в описании операции **onstop**.

4.7.21 onstop

Формат вызова:

onstop <parameter0>;

ничего не возвращает, принимает 1 параметр;

параметр типа блок, допустима переменная, константа, операция, массив не допустим;

назначение операции: задает блок завершения, который будет вызван при завершении работы программы в случае выполнения операции **stop**, а также прерывания программы при выполнении других операций завершения**,** но не в случае ошибки выполнения, если внутри назначенного блока используется снова операция **stop**, то программа немедленно завершается (в конце этого блока использовать **stop** не нужно), пример:

```
onstop {puts “finished” };
        puts “finishing...”;
        stop;
```

напечатает

```
finishing...
finished
```

4.7.22 onerror

Формат вызова:

onerror <parameter0>;

ничего не возвращает, принимает 1 параметр;

параметр типа блок, допустима переменная, константа, операция, массив не допустим;

назначение операции: предназначено для отладки пользовательских Ci-программ, задает блок, который будет выполнен при возникновении ошибки на этапе выполнения программы, это предназначено для завершения критичных операций, например, для выключения какой-либо аппаратуры и выдачи соответствующих со-

общений.

Если ошибка возникнет внутри заданного блока, то выполнение программы будет немедленно прекращено, поэтому к написанию блока для **onerror** следует подходить очень внимательно, внутри блока можно получить код ошибки при помощи операции **errorcode**, например:

```
int i 100;
        onerror
        { puts "error";
                if( (errorcode) == 35 )
                        {puts "Division by zero";}
                                {puts "No parameters given";};
        };
        ``par@ 0 10;
        puts( i / 0 );
```

напечатает сообщение о делении на ноль, но если раскомментировать строку par@ 0 10; и перекомпилировать, то при выполнении возникнет ошибка обращения к несуществующему параметру блока, и будет напечатано сообщение No parameters given.

4.7.23 errorcode

Формат вызова:

(errorcode)

возвращает беззнаковое целое, параметров нет;

назначение операции: используется совместно с **onerror** для получения кода ошибки, подробнее см. в описании **onerror**;

если вызывается не в блоке обработки ошибки, то всегда возвращает 0;

возвращаемые коды ошибок:

32 операции передано данное недопустимого типа;

34 стек параметров пуст (попытка выполнить в блоке par# или par@ при том, что в блок не было ничего передано);

35 деление на ноль;

36 два массива не одинакового размера (используется в операциях обработки звука);

39 операции передано недопустимое значение;

45 индекс вышел за границу массива;

53 слишком глубокая вложенность вызовов блоков или операций с возвратом значения;

60 попытка выполнить sizeof не инициализированного массива;

61 невозможно преобразовать значение, возвращаемое операцией, способ преобразования типов не определен;

71 недопустимая инициализация;

72 недопустимое удаление;

73 переменная цикла вышла за допустимые границы;

76 массив нулевой длины;

77 надо вернуть значение;

78 невозможно открыть файл;

79 невозможно записать в файл;

80 невозможно закрыть файл;

81 невозможно прочесть файл.

4.7.24 delay

Формат вызова:

delay <parameter0>;

ничего не возвращает, принимает один параметр;

параметр типа вещественное число, допустим любой операнд, массив не допустим;

назначение операции: приостанавливает выполнение Ci-программы на указанное параметром число миллисекунд, пример использования:

delay 1000; ``задержка в 1 секунду

4.8 Отладочные операции

4.8.1 trace

Формат вызова:

trace <parameter0> [<parameter1> [<parameter2> [<parameter3>]]];

ничего не возвращает, принимает от одного до четырех параметров;

все параметры типа логическое, допустимы любые операнды, массивы не допустимы;

назначение операции: включает и выключает различные режимы трассировки выполнения Ci-программы, если первый параметр указан и имеет значение TRUE, разрешает трассировку выполнения Ci-программы, при которой в стандартный поток выводятся имена выполняемых операций, если не указан, то ничего не изменяется, если указан со значением FALSE, то трассировка отключается.

После имени выводятся значения параметров операции до и после ее вызова, а также возвращаемое ей значение, например:

```
trace TRUE;
    float b, c 1.00096, d 1.00000096;
    b = (c + d);

    в результате трассировка будет выглядеть:

    +
      before:
          00: variable float 1.000960
          01: variable float 1.000001
      after:
          result float 2.000961
          00: variable float 1.000960
          01: variable float 1.000001

    =
```

```
before:
    00: variable float 0.000000
    01: result float 2.000961
after:
    00: variable float 2.000961
    01: result float 2.000961
```

Если второй параметр указан и имеет значение TRUE, то при трассировке «раскрываются» массивы – будут показываться значения всех их элементов, этим следует пользоваться с осторожностью, поскольку трассировка массивов большого размера будет занимать много времени, если не указан, то ничего не изменяется, если указан со значением FALSE, то раскрытие массивов отключается.

Если третий параметр указан и имеет значение TRUE, то будет включен режим пошагового прохождения, который зависит от реализации в каждой конкретной версии Ci, если не указан, то ничего не изменяется, если указан со значением FALSE, то пошаговое прохождение отключается.

Если четвертый параметр указан и имеет значение TRUE, то в начале каждого выполняемого блока выводится количество переменных и размер стека вызовов функций, это предназначено для системной отладки и программистами на Ci не должно применяться, если параметр не указан, то ничего не изменяется, если указан со значением FALSE, то системная трассировка отключается.

Выключение любого режима трассировки производится вызовом **trace** и указанием на месте соответствующего параметра значения FALSE – если надо отключить только второй режим, но оставить первый, то надо вызывать trace TRUE FALSE, поскольку trace FALSE FALSE отключит оба режима, следовательно для гибкого управления трассировкой надо завести логические переменные, состояние которых менять при необходимости, и передавать их параметрами операции trace.

По-умолчанию трассировка выключена.

4.8.2 time

Формат вызова:

(time [<parameter0>])

возвращает строку, может принимать один параметр;

параметр типа беззнаковое целое, допустим любой операнд, массив недопустим;

назначение операции: возвращает в виде строки текущее время дня, с секундами, может использоваться для примерной оценки времени выполнения участков кода, либо просто для получения и выдачи текущего времени суток;

необязательный параметр задает код символа разделителя, если его нет, то используется разделитель: (двоеточие), примеры использования:

puts (time);

puts (time 0x22) `` выведет строку вида 12"30"04

4.8.3 date

Формат вызова:

(date [<parameter0>])

возвращает строку, может принимать один параметр;

параметр типа беззнаковое целое, допустим любой операнд, массив недопустим;

назначение операции: возвращает в виде строки текущую дату, месяц и год представлены двухзначным числом, формат даты международный (месяц-число-год);

необязательный параметр задает код символа разделителя, если его нет, то используется разделитель - (минус), примеры использования:

puts (date);

puts (date 0x2F) `` выведет строку вида 11/30/08

4.8.4 timer

Форматы вызова:

timer [<parameter0>];

(timer [<parameter0>])

возвращает беззнаковое целое, возврат может быть проигнорирован, может

принимать один параметр;

параметр типа логическое, допустим любой операнд, массив не допустим;

назначение операции: перезапуск таймера и измерение времени от предыдущего перезапуска, может иметь необязательный параметр, если указано TRUE, перезапускает таймер, если FALSE или параметр отсутствует, перезапуск не происходит, возвращаемое значение равно интервалу времени в миллисекундах от момента последнего перезапуска таймера, если требуется только перезапустить таймер, то возвращаемое значение можно игнорировать, в начале измерения таймер требуется перезапустить, иначе потом будет возвращено псевдослучайное число, пример использования:

.... ``код программы

timer TRUE; ``метка A, отсюда надо измерить интервал времени

....

puts (timer); ``выведет интервал времени от метки A, но таймер не перезапустится

....

puts (timer TRUE); ``метка B выведет интервал от метки A, таймер перезапустится

....

puts (timer); ``выведет интервал от метки B, таймер продолжит считать от нее.

5 СООБЩЕНИЯ

При трансляции исходных текстов ППКД в левое окно УПА АСКД СКИП-У выдаются различные диагностические сообщения. Это могут быть сообщения двух основных видов: сообщения о работе транслятора и сообщения об ошибках в транслируемой программе.

Сообщения о работе транслятора и загрузчика выдаются на английском языке.

Они могут быть следующего вида:

"Build number 1.X.Y" - номер версии и подверсий программы;

"Improper source code name - name must have .ci extension" - в поле над кнопкой Compile указано неверное имя файла программы контроля, отсутствует расширение .ci;

"Error while opening source (N): <текст сообщения>;

File: " - Ошибка при открытии исходного текста программы контроля, N код ошибки (см. руководство по ОС Microsoft Windows), <текст сообщения> формируется операционной системой Microsoft Windows и содержит расширенную информацию об ошибке на английском языке, после File: выдается полный маршрут и имя файла, который пытался открыть транслятор;

"Compiling..." - происходит компиляция программы контроля в двоичный код;

"Saving..." - происходит сохранение двоичного кода программы контроля;

"Code saved." - двоичный код программы контроля сохранен;

"NN operations compiled" - показывает количество скомпилированных операций, где NN целое число;

"Code used XX bytes (about YY byte/op), ZZZZ bytes in code page free" - показывает использование страницы кода при трансляции программы, XX количество использованных байт, YY среднее количество байт на операцию, ZZZZ количество свободных байт в кодовой странице;

"Slots left free - vars: VVVVV, returns RRRRR, constants CCCCC" - показывает количество свободных слотов в таблицах, VVVVV количество свободных слотов для переменных, RRRRR количество свободных слотов для возвратов операций, CCCCC количество свободных слотов для констант;

"Program ready to run" - при успешной компиляции и сохранении показывает готовность программы контроля к выполнению;

"Loading..." - происходит загрузка программы контроля;

"Running..." - происходит запуск программы контроля;

"Program terminated" - программа прервана, выдается при выполнении в программе контроля операции stop или при нажатии кнопки Stop test program;

"Program successfully finished" - программа успешно завершена.

Сообщения об ошибках в транслируемой программе выдаются на английском языке. Они имеют следующий вид:

Error NN (line XX): <текст сообщения>

где NN – код ошибки, XX – номер строки, в которой обнаружена ошибка, <текст сообщения> может быть один из:

"Not enougth memory or error unknown" - нет памяти, или ошибка не определена;

"No parameters in call of this block" - блок был вызван без параметров;

"Block parameters stack overflow" - переполнение стека передачи параметров в блок;

"No parameter with given number" - нет параметра с указанным номером;

"String or constant doesn't match expected type" - строка или константа не совпадает с ожидаемым типом;

"Bad new variable type" - недопустимый тип новой переменной;

"With same name and another type variable exists" - существует переменная с таким же именем и другим типом;

"Variable table full" - слишком много переменных, переполнение таблицы;

"String too long" - строка слишком длинная;

"Operation name expected" - ожидалось имя операции;

"Name not found in desired table" - имя в таблице не найдено ;

"Unexpected end of program" - неожиданный конец текста программы;

"Wrong parameters number" - неверное число параметров;

"Must be variable" - допустима только переменная;

"Variable not found OR missing ; in previous operation OR operation call without ()" - переменная не найдена, или отсутствует ; в конце предыдущей операции, или вызов возвращающей операции не внутри ();

"Wrong data or operation type" - неверные данные или тип операции;

"Unterminated or too long string const" - незавершенная или слишком длинная строковая константа;

"Type already defined" - тип уже определен;

"Extra parameter in operation call" - лишний параметр при вызове операции;

"Too many parameters OR possible call without ()s" - слишком много параметров, или возможен вызов не внутри ();

"Must be type" - должен быть указан тип;

"Absent begin { brace" - отсутствует открывающая скобка {;

"Absent end } brace or ; at block end" - отсутствует закрывающая скобка или неверный символ ; в конце блока;

"String should be continued" - строка должна быть продолжена;

"Needed name of operation to call" - ожидалось имя вызываемой операции;

"Unknown operation" - неизвестная операция;

"Type convertion not defined" - преобразование типа не определено;

"Returned value should be used" - возвращаемое значение должно быть использовано;

"Function returns nothing" - операция ничего не возвращает;

"Shold be) in operation call" - должна быть закрывающая скобка) при вызове операции;

"Not an allowed type for operation" - недопустимый тип данного для этой операции;

"Cannot define type, please use type name before operation" - невозможно определить тип, необходимо использование имени типа перед именем операции;

"Internal error - empty parameters stack" - внутреннаяя ошибка, пустой стек параметров;

"Division by zero" - деление на ноль;

"Sizes of two arrays in operation not the same" - размеры двух массивов не совпадают;

"Unterminated comment" - незакрытый комментарий;

"Program terminated" - программа прервана;

"Unavailable value given to operation" - недопустимое значение передано операции;

"Only array element available" - допустим только элемент массива;

"Array not available" - массив не допустим;

"Only entire array can be given" - возможна только передача целиком массива;

"Index brace] expected" - ожидалась закрывающая скобка];

"With another size array already exists" - уже существует массив другого размера;

"Array subscript out off range" - индекс вышел за границу массива;

"Constant unavailable" - константа недопустима;

"Code page overflow" - переполнение страницы кода, слишком большая программа;

"misplaced [" - открывающая квадратная скобка [в неверном месте;

"misplaced]" - закрывающая квадратная скобка] в неверном месте;

"misplaced ; or end of stream" - символ ; в неверном месте или обнаружен неожиданный конец программы;

"misplaced)" - закрывающая круглая скобка) в неверном месте;

"Unavailable parameter" - недопустимый параметр;

"Too deeply nested calls" - слишком много вложенных вызовов, переполнение внутреннего стека;

"Same name operation exists" - существует операция с таким именем;

"Same name type exists" - существует тип данных с таким именем;

"Same name constant exists" - существует литеральная константа с таким именем;

"Same name variable already exists" - уже есть объявление переменной с таким именем;

"Too many constants" - слишком много констант, переполнение таблиц;

"Too many variables" - слишком много переменных, переполнение таблиц;

"Variable or array not initialized" - переменная или массив не проициализированы;

"Cannot convert data types" - не могу преобразовать данные этих типов;

"Operation should be used in prefix form" - операцию можно использовать только в префиксной форме;

"Block unavailable" - блок недопустим;

"Bad program label" - неверная метка программы, файл Ci-программы испорчен;

"Bad record in program file" - неверная запись в файле программы, файл Ci-программы испорчен;

"Code check sume failure" - неверная контрольная сумма кода в файле программы, файл Ci-программы испорчен;

"Code too large" - загружаемая программа не помещается в кодовую страницу;

"Data check sume failure" - неверная контрольная сумма данных в файле программы, файл Ci-программы испорчен;

"Too many datas" - слишком много данных;

"Variable is not an array" - переменная не является массивом;

"INTERNAL ERROR - Bad initialization" - внутренняя ошибка, неправильная инициализация;

"Unexpected end of block, deleting non initialized variables" - внутренняя ошибка, неожиданный конец блока, удаление неинициализированных переменных;

"\"for\" cycle index out of range" - индекс цикла for вышел за допустимые границы;

"Zero size array not available" - массив нулевого размера недопустим;

"Function must return value" - функция должна вернуть значение;

"Error when open file" - ошибка при открытии файла;

"Error when write to file" - ошибка при записи в файл;

"Error when close file" - ошибка при закрытии файла;

За строкой с сообщением об ошибке выводится строка программы, в которой обнаружена ошибка. Под ней символ ^ указывает наиболее вероятное место ошибки.