ЭЛЕМЕНТЫ ЯЗЫКА СИ

Используемые символы

Для образования ключевых слов и идентификаторов используются прописные и строчные буквы английского алфавита, арабские цифры, а также символ подчеркивания. Одинаковые прописные и строчные буквы считаются различными символами, так как имеют различные коды. В комментариях можно использовать буквы русского алфавита.

Знаки нумерации и специальные символы используются, с одной стороны, для организации процесса вычислений, а с другой - для передачи компилятору определенного набора инструкций;

Управляющие и разделительные символы. К ним относятся: пробел, символы табуляции, перевода строки, возврата каретки, новая страница и новая строка. Эти символы отделяют друг от друга объекты, определяемые пользователем, к которым относятся константы и идентификаторы. Последовательность разделительных символов рассматривается компилятором как один символ.

В функциях ввода и вывода информации широко используются управляющие последовательности. Такая последовательность состоит из обратной дробной черты (\) (обязательный первый символ) и комбинации латинских букв и цифр.

Константы

В языке СИ разделяют четыре типа констант: целые константы, константы с плавающей запятой, символьные константы и строковые литералы.

Целая константа - это десятичное, восьмеричное или шестнадцатеричное число, которое представляет целую величину.

Десятичная константа состоит из одной или нескольких десятичных цифр, причем первая цифра не должна быть нулем (в противном случае число будет воспринято как восьмеричное).

Восьмеричная константа состоит из обязательного нуля и одной или нескольких восьмеричных цифр (0 ÷ 7).

Шестнадцатеричная константа начинается с обязательной последовательности 0x или 0X и содержит одну или несколько шестнадцатеричных цифр (0 ÷ 9,A ÷ F).

Примеры целых констант:

Десятичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	-020	0x10
-127	0117	0x2B
240	0360	-0XF0

Десятичные константы рассматриваются как величины со знаком, и им присваивается тип int (целая) или long (длинная целая) в соответствии со значением константы. Если константа меньше 32768, то ей присваивается тип int, в противном случае - long.

Восьмеричным и шестнадцатеричным константам присваивается тип int, unsigned int (беззнаковая целая), long или unsigned long в зависимости от значения константы.

Для того, чтобы любую целую константу определить типом long, достаточно в конце константы поставить букву "l" или "L". Пример:

5l, 6l, 128L, 0105L, 0X2A11L.

Константа с плавающей точкой - десятичное число, представленное в виде действительной величины с десятичной точкой или экспонентой. Константы с плавающей точкой представляют положительные величины удвоенной точности (имеют тип double)

Примеры: 115.75, 1.5E-2, -0.025, .075, -0.85E2

Символьная константа представляется символом, заключенным в апострофы. Управляющая последовательность рассматривается как одиночный символ, допустимо ее использовать в символьных константах. Значением символьной константы является числовой код символа. Примеры:

' '- пробел ,

'Q'- буква Q ,

'\n' - символ новой строки ,

'\\' - обратная дробная черта ,

'\v' - вертикальная табуляция .

Символьные константы имеют тип int и при преобразовании типов дополняются знаком.

Строковая константа (литерал) - последовательность символов (включая строковые и прописные буквы русского и латинского алфавитов, а также цифры), заключенных в кавычки (") . Например: "Школа N 35", "город Тамбов", "YZPT КОД". В конец каждого строкового литерала компилятором добавляется нулевой символ, представляемый управляющей последовательностью \0. Строковый литерал имеет тип char[], т. е. рассматривается как массив символов. Число элементов этого массива = число символов в строке + 1, так как символ конца строки также является элементом массива.

Идентификатор

Идентификатором называется последовательность цифр и букв, а также специальных символов, при условии, что первой стоит буква или специальный символ. Для образования идентификаторов могут быть использованы строчные или прописные буквы латинского алфавита. В качестве специального символа может использоваться символ подчеркивания (_). Два идентификатора, для образования которых используются совпадающие строчные и прописные буквы, считаются различными. Например: abc, ABC, A128B, a128b .

Компилятор допускает любое количество символов в идентификаторе, но значимыми являются первые 31 символ. Идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций библиотеки компилятора языка СИ.

Ключевые слова

Ключевые слова - это зарезервированные идентификаторы, которые можно использовать только в соответствии со значением, известным компилятору языка СИ.

Список ключевых слов:

auto double int struct break else long switch
register typedef char extern return void case float
unsigned default for signed union do if sizeof
volatile continue enum short while

Использование комментариев в тексте программы

Комментарий - это набор символов между специальными символами начала (/*) и конца (*/) комментария, который игнорируется компилятором. Комментарий может занимать несколько строк, например:

```
/* комментарии к программе */
/* начало алгоритма */
```

или

```
/* комментарии можно записать в следующем виде, однако надо быть осторожным, чтобы внутрь комментария не попались операторы программы, которые также будут игнорироваться */
```

Типы данных и их объявление

Объявления переменной имеет следующий формат:

[спецификатор-класса-памяти] спецификатор-типа описатель [=инициатор] [,описатель [= инициатор]]...

Описатель - идентификатор простой переменной либо более сложная конструкция с квадратными скобками, круглыми скобками или звездочкой (набором звездочек).

Спецификатор типа - одно или несколько ключевых слов, определяющие тип объявляемой переменной. В языке СИ имеется стандартный набор типов данных, используя который можно сконструировать новые (уникальные) типы данных.

Инициатор - задает начальное значение или список начальных значений, которые (которое) присваивается переменной при объявлении.

Спецификатор класса памяти - определяется одним из четырех ключевых слов языка СИ: auto, extern, register, static, и указывает, каким образом будет распределяться память под объявляемую переменную, с одной стороны, а с другой, область видимости этой переменной, т.е., из каких частей программы можно к ней обратиться.

Категории типов данных

Ключевые слова для определения основных типов данных

Целые типы	Плавающие типы:
char	float

int	double
short	long double
long	
signed	
unsigned	

Переменная любого типа может быть объявлена как немодифицируемая. Это достигается добавлением ключевого слова const к спецификатору типа. Если после слова const отсутствует спецификатор типа, то подразумевается спецификатор типа int. Если ключевое слово const стоит перед объявлением составных типов (массив, структура, смесь, перечисление), то каждый элемент также должен являться немодифицируемым, т.е. значение ему может быть присвоено только один раз.

Примеры:

const double A=2.128E-2;

const B=286; (подразумевается const int B=286)

Инициализация данных

При объявлении переменной ей можно присвоить начальное значение, присоединяя инициатор к описателю. Инициатор начинается со знака "=" и имеет следующие формы.

Формат 1: = инициатор;

Формат 2: = { список - инициаторов };

Формат 1 используется при инициализации переменных основных типов и указателей, а формат 2 - при инициализации составных объектов.

Примеры:

char tol = 'N';

Переменная tol инициализируется символом 'N'.

const long megabute = (1024 * 1024);

Немодифицируемая переменная megabute инициализируется константным выражением, после чего она не может быть изменена.

static int b[2][2] = {1,2,3,4};

Выражения и присваивания

Операнды и операции

Комбинация знаков операций и операндов, результатом которой является определенное значение, называется выражением. Знаки операций определяют действия, которые должны быть выполнены над операндами. Значение выражения зависит от расположения знаков операций и круглых скобок, а также от приоритета выполнения операций.

При вычислении выражений тип каждого операнда может быть преобразован к другому типу. Преобразования типов могут быть неявными (при выполнении операций и вызовов функций), или явными, при выполнении операций приведения типов.

Операнд - это константа, литерал, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное комбинацией операндов, знаков операций и круглых скобок. Каждый операнд имеет тип.

Если в качестве операнда используется константа, то ему соответствует значение и тип представляющей его константы. Целая константа может быть типа int, long, unsigned int, unsigned long, в зависимости от ее значения и от формы записи. Символьная константа имеет тип int. Константа с плавающей точкой всегда имеет тип double.

Строковый литерал состоит из последовательности символов, заключенных в кавычки, и представляется в памяти как массив элементов типа char, инициализируемый указанной последовательностью символов. Значением строкового литерала является адрес первого элемента строки и синтаксически строковый литерал является немодифицируемым указателем на тип char. Строковые литералы могут быть использованы в качестве операндов в выражениях, допускающих величины типа указателей. Однако, так как строки не являются переменными, их нельзя использовать в левой части операции присваивания.

Идентификаторы переменных и функций. Каждый идентификатор имеет тип, который устанавливается при его объявлении. Значение идентификатора зависит от типа следующим образом:

идентификаторы объектов целых и плавающих типов представляют значения соответствующего типа;

идентификатор объекта типа enum представлен значением одной константы из множества значений констант в перечислении. Значением идентификатора является константное значение. Тип значения есть int, что следует из определения перечисления.

Вызов функций состоит из выражения, за которым следует необязательный список выражений в круглых скобках:

выражение-1 ([список выражений])

Значением выражения-1 должен быть адрес функции (например, идентификатор функции). Значения каждого выражения из списка выражений передаются в функцию в качестве фактического аргумента. Операнд, являющийся вызовом функции, имеет тип и значение возвращаемого функцией значения.

Приведение типов - это изменение (преобразование) типа объекта. Для выполнения преобразования необходимо перед объектом записать в скобках нужный тип:

(имя-типа) операнд.

Приведение типов используются для преобразования объектов одного скалярного типа в другой скалярный тип. Однако выражению с приведением типа не может быть присвоено другое значение.

Пример:

```
int i;
double x;
b = (double)i+2.0;
```

В этом примере целая переменная i с помощью операции приведения типов приводится к плавающему типу, а затем уже участвует в вычислении выражения.

Константное выражение - это выражение, результатом которого является константа. Операндом константного выражения могут быть целые константы, символьные константы, константы с плавающей точкой, константы перечисления, выражения приведения типов, выражения с операцией sizeof и другие константные выражения. Однако на использование знаков операций в константных выражениях налагаются следующие ограничения:

1. В константных выражениях нельзя использовать операции присваивания и последовательного вычисления (,) .

2. Операция "адрес" (&) может быть использована только при некоторых инициализациях.

Выражения со знаками операций могут участвовать в выражениях как операнды. Выражения со знаками операций могут быть унарными (с одним операндом), бинарными (с двумя операндами) и тернарными (с тремя операндами).

Унарное выражение состоит из операнда и предшествующего ему знаку унарной операции и имеет следующий формат:

знак-унарной-операции операнд .

Бинарное выражение состоит из двух операндов, разделенных знаком бинарной операции:

операнд1 знак-бинарной-операции операнд2 .

Тернарное выражение состоит из трех операндов, разделенных знаками тернарной операции (?) и (:), и имеет формат:

операнд1 ? операнд2 : операнд3 .

Операции. По количеству операндов, участвующих в операции, операции подразделяются на унарные, бинарные и тернарные.

В языке Си имеются следующие унарные операции:

- арифметическое отрицание (отрицание и дополнение);

~ побитовое логическое отрицание (дополнение);

! логическое отрицание;

* разадресация (косвенная адресация);

& вычисление адреса;

+ унарный плюс;

++ увеличение (инкремент);

-- уменьшение (декремент);

sizeof размер .

Унарные операции выполняются справа налево.

Операции увеличения и уменьшения увеличивают или уменьшают значение операнда на единицу и могут быть записаны как справа, так и слева от операнда. Если знак операции записан перед операндом (префиксная форма), то изменение операнда происходит до его использования в выражении. Если знак операции записан после операнда (постфиксная форма), то операнд вначале используется в выражении, а затем происходит его изменение.

В отличие от унарных, бинарные операции (Таблица 1) выполняются слева направо.

Таблица 1. Основные операции языка Си

Знак операции	**Операция**	**Группа операций**
*	Умножение	Мультипликативные
/	Деление	
%	Остаток от деления	
+	Сложение	Аддитивные
-	Вычитание	
<	Меньше	Операции отношения
<=	Меньше или равно	
>=	Больше или равно	
==	Равно	
!=	Не равно	
&&	Логическое И	Логические операции
||	Логическое ИЛИ	
,	Последовательное вычисление	Последовательного вычисления
=	Присваивание	Операции присваивания
*=	Умножение с присваиванием	
/=	Деление с присваиванием	
%=	Остаток от деления с присваиванием	

-=	Вычитание с присваиванием	
+=	Сложение с присваиванием	

Левый операнд операции присваивания должен быть выражением, ссылающимся на область памяти (но не объектом, объявленным с ключевым словом const), такие выражения называются леводопустимыми. К ним относятся:

идентификаторы данных целого и плавающего типов, типов указателя, структуры, объединения;

индексные выражения, исключая выражения имеющие тип массива или функции;

выражения выбора элемента (->) и (.), если выбранный элемент является леводопустимым;

выражения унарной операции разадресации (*), за исключением выражений, ссылающихся на массив или функцию;

выражение приведения типа если результирующий тип не превышает размера первоначального типа.

При записи выражений следует помнить, что символы (*), (&), (!), (+) могут обозначать унарную или бинарную операцию.

Преобразования при вычислении выражений

При выполнении операций производится автоматическое преобразование типов, чтобы привести операнды выражений к общему типу или чтобы расширить короткие величины до размера целых величин, используемых в машинных командах. Выполнение преобразования зависит от специфики операций и от типа операнда или операндов.

При вычислении выражений операнды преобразуются к типу того операнда, который имеет наибольший размер.

Операнды типа float преобразуются к типу double.

Если один операнд long double, то второй преобразуется к этому же типу.

Если один операнд double, то второй также преобразуется к типу double.

Любые операнды типа char и short преобразуются к типу int.

Любые операнды unsigned char или unsigned short преобразуются к типу unsigned int.

Если один операнд типа unsigned long, то второй преобразуется к типу unsigned long.

Если один операнд типа long, то второй преобразуется к типу long.

Если один операнд типа unsigned int, то второй операнд преобразуется к этому же типу.

Пример:

```
double         ft,sd;
unsigned char  ch;
unsigned long  in;
int            i;
....
  sd=ft*(i+ch/in);
```

При выполнении этого оператора присваивания правила преобразования будут использоваться следующим образом. Операнд ch преобразуется к unsigned int. Затем он преобразуется к типу unsigned long. По этому же правилу i преобразуется к unsigned long и результат операции, заключенной в круглые скобки будет иметь тип unsigned long. Затем он преобразуется к типу double, и результат всего выражения будет иметь тип double.

Операции отрицания и дополнения

Операция арифметического отрицания (-) вырабатывает отрицание своего операнда. Пример:

```
double  u = 5;
u = -u;          /* переменной u присваивается ее отрицание,
                    т.е. u  принимает значение -5     */
```

Операция логического отрицания "НЕ" (!) вырабатывает значение 0, если операнд есть истина (не нуль), и значение 1, если операнд равен нулю (0). Результат имеет тип int. Пример:

```
int t, z=0;
t=!z;
```

Переменная t получит значение, равное 1, так как переменная z имела значение равное 0 (ложно).

Операции разадресации и адреса

Эти операции используются для работы с переменными типа указатель.

Операция разадресации (*) осуществляет косвенный доступ к адресуемой величине через указатель. Операнд должен быть указателем. Результатом операции является величина, на которую указывает операнд. Типом результата является тип величины, адресуемой указателем.

Операция адреса (&) дает адрес своего операнда. Операндом может быть любое именуемое выражение. Имя функции или массива также может быть операндом операции “адрес”. Результатом операции адреса является указатель на операнд. Тип, адресуемый указателем, является типом операнда.

Примеры:

```
int t, f=0,  * adress;
adress = &t  /*переменной adress, объявляемой как
             указатель, присваивается адрес переменной t*/
* adress =f; /* переменной находящейся по адресу, содержащемуся в переменной adress, присваивается значение переменной f,  т.е. 0 , что эквивалентно t=f;      т.е.   t=0;  */
```

Операция sizeof

С помощью операции sizeof можно определить размер памяти, которая соответствует идентификатору или типу. Операция sizeof имеет следующий формат:

sizeof(выражение).

В качестве выражения может быть использован любой идентификатор, либо имя типа, заключенное в скобки

Если в качестве выражения указано имя массива, то результатом является размер всего массива (т.е. произведение числа элементов на длину типа), а не размер указателя, соответствующего идентификатору массива.

Когда sizeof применяются к имени типа структуры или объединения (или к идентификатору, имеющему тип структуры или объединения), то результатом будет фактический размер структуры или объединения, который может включать участки памяти, используемые для выравнивания элементов структуры или объединения. Таким об-

разом, этот результат может не соответствовать размеру, получаемому путем простого сложения размеров элементов структуры.

Пример:

```
struct    { char   h;
            int    b;
            double f;
          } str;
int a1;
a1  = sizeof(str);
```

Переменная a1 получит значение, равное 12, в то же время, если сложить длины всех используемых в структуре типов, то получим, что длина структуры str равна 7. Несоответствие имеет место ввиду того, что после размещения в памяти первой переменной h длиной 1 байт, добавляется 1 байт для выравнивания адреса переменной b на границу слова (слово имеет длину 2 байта для машин серии IBM PC), далее осуществляется выравнивание адреса переменной f на границу двойного слова (4 байта). Таким образом, в результате операций выравнивания для размещения структуры в оперативной памяти требуется на 5 байт больше.

В связи с этим целесообразно рекомендовать при объявлении структур и объединения располагать их элементы в порядке убывания длины типов, т.е. приведенную выше структуру следует записать в следующем виде:

```
struct  { double f;
          int    b;
          char   h;
        } str;
```

Мультипликативные операции

К этому классу операций относятся операции умножения (*), деления (/) и получения остатка от деления (%). Операндами операции (%) должны быть целые числа. Типом результата является тип операндов после преобразования.

Операция умножения (*) выполняет умножение операндов.

```
int  i=5;
float f=0.2;
```

```
double g,z;
   g=f*i;
```

Тип произведения i и f преобразуется к типу double, затем результат присваивается переменной g.

Операция деления (/) выполняет деление первого операнда на второй. Если две целые величины не делятся нацело, то результат округляется в сторону нуля.

При попытке деления на ноль выдается сообщение во время выполнения.

```
int i=49, j=10,  n, m;
n = i/j;              /* результат   4   */
m = i/(-j);           /* результат  -4   */
```

Операция (%) дает остаток от деления первого операнда на второй.

Знак результата зависит от конкретной реализации.

```
int n = 49, m = 10, i, j, k, l;
i = n % m;            /*   9   */
j = n % (-m);         /*   9   */
k = (-n) % m;         /*  -9   */
l = (-n) % (-m);      /*  -9   */
```

Аддитивные операции

К аддитивным операциям относятся сложение (+) и вычитание (-). Операнды могут быть целого или плавающего типов. В некоторых случаях над операндами аддитивных операций выполняются общие арифметические преобразования. Однако преобразования, выполняемые при аддитивных операциях, не обеспечивают обработку ситуаций переполнения и потери значимости. Информация теряется, если результат аддитивной операции не может быть представлен типом операндов после преобразования. При этом сообщение об ошибке не выдается.

Пример:

```
int   i=30000, j=30000, k;
      k=i+j;
```

В результате сложения k получит значение, равное -5536.

Логические операции

К логическим операциям относятся операция логического И (&&) и операция логического ИЛИ (||). Операнды логических операций могут быть целого типа, плавающего типа или типа указателя, при этом в каждой операции могут участвовать операнды различных типов.

Операнды логических выражений вычисляются слева направо. Если значения первого операнда достаточно, чтобы определить результат операции, то второй операнд не вычисляется.

Результатом логической операции является 0 или 1, тип результата int.

Операция логического И (&&) вырабатывает значение 1, если оба операнда имеют нулевые значения. Если один из операндов равен 0, то результат также равен 0. Если значение первого операнда равно 0, то второй операнд не вычисляется.

Операция логического ИЛИ (||) выполняет над операндами операцию включающего ИЛИ. Она вырабатывает значение 0, если оба операнда имеют значение 0, если какой-либо из операндов имеет ненулевое значение, то результат операции равен 1. Если первый операнд имеет ненулевое значение, то второй операнд не вычисляется.

Условная операция

В языке СИ имеется одна тернарная операция - условная операция, которая имеет следующий формат:

Операнд 1 ? операнд 2 : операнд 3

Операнд 1 должен быть целого или плавающего типа или быть указателем. Если операнд 1 не равен 0, то вычисляется операнд 2 и его значение является результатом операции. Если операнд 1 равен 0, то вычисляется операнд 3 и его значение является результатом операции. Следует отметить, что вычисляется либо операнд 2, либо операнд 3, но не оба. Тип результата зависит от типов операнда 2 и операнда 3 следующим образом.

1. Если операнд 2 (или операнд 3) имеет целый или плавающий тип, то выполняются обычные арифметические преобразования. Типом результата является тип операнда после преобразования.

2. Если операнд 2 и операнд 3 имеют один и тот же тип структуры, объединения или указателя, то тип результата будет тем же самым типом структуры, объединения или указателя.

3. Если оба операнда имеют тип void, то результат имеет тип void.

4. Если один операнд является указателем на объект любого типа, а другой операнд является указателем на void, то указатель на объект преобразуется к указателю на void, который и будет типом результата.

5. Если один из операндов является указателем, а другой константным выражением со значением 0, то типом результата будет тип указателя.

Пример:

max = (d<=b) ? b : d;

Переменной max присваивается максимальное значение переменных d и b.

Операции увеличения и уменьшения

Операции увеличения (++) и уменьшения (--) являются унарными операциями присваивания. Они соответственно увеличивают или уменьшают значения операнда на единицу. Операнд может быть целого или плавающего типа или типа указатель и должен быть модифицируемым. Операнд целого или плавающего типа увеличивается (уменьшаются) на единицу. Тип результата соответствует типу операнда. Операнд адресного типа увеличивается или уменьшается на размер объекта, который он адресует. В языке допускается префиксная или постфиксная формы операций увеличения (уменьшения), поэтому значение выражения, использующего операции увеличения (уменьшения), зависит от того, какая из форм указанных операций используется.

Если знак операции стоит перед операндом (префиксная форма записи), то изменение операнда происходит до его использования в выражении и результатом операции является увеличенное или уменьшенное значение операнда.

В том случае если знак операции стоит после операнда (постфиксная форма записи), то операнд вначале используется для вычисления выражения, а затем происходит изменение операнда.

Примеры:

```
int t=1, s=2, z, f;
    z=(t++)*5;
```

Вначале происходит умножение t*5, а затем увеличение t. В результате получится z=5, t=2.

```
f=(++s)/3;
```

Вначале значение s увеличивается, а затем используется в операции деления. В результате получим s=3, f=1.

В случае, если операции увеличения и уменьшения используются как самостоятельные операторы, префиксная и постфиксная формы записи становятся эквивалентными.

```
z++; /* эквивалентно */ ++z;
```

Простое присваивание

Операция простого присваивания используется для замены значения левого операнда значением правого операнда. При присваивании производится преобразование типа правого операнда к типу левого операнда по правилам, упомянутым раньше. Пример:

```
int t;
char f;
long z;
t=f+z;
```

Значение переменной f преобразуется к типу long, вычисляется f+z, результат преобразуется к типу int и затем присваивается переменной t.

Составное присваивание

Имеется целая группа операций присваивания, которые объединяют простое присваивание с одной из бинарных операций. Такие операции называются составными операциями присваивания и имеют вид:

(операнд 1) (бинарная операция) = (операнд 2) .

Составное присваивание по результату эквивалентно следующему простому присваиванию:

(операнд 1) = (операнд 1) (бинарная операция) (операнд 2) .

Отметим, что выражение составного присваивания с точки зрения реализации не эквивалентно простому присваиванию, так как в последнем операнд 1 вычисляется дважды.

Каждая операция составного присваивания выполняет преобразования, которые осуществляются соответствующей бинарной опера-

цией. Левым операндом операций (+=) (-=) может быть указатель, в то время как правый операнд должен быть целым числом.

Примеры:

```
double  arr[4]={ 2.0, 3.3, 5.2, 7.5 } ;
double  b=3.0;
b+=arr[2];     /*  эквивалентно   b=b+arr[2]          */
arr[3]/=b+1;   /*  эквивалентно   arr[3]=arr[3]/(b+1)   */
```

Заметим, что при втором присваивании использование составного присваивания дает более заметный выигрыш во времени выполнения, так как левый операнд является индексным выражением.

Приоритеты операций и порядок вычислений

В языке СИ операции с высшими приоритетами вычисляются первыми. Наивысшим приоритетом является приоритет, равный 1. Приоритеты и порядок операций приведены в Таблица 2.

Таблица 2

Приоритет	**Знак операции**	**Типы операции**	**Порядок выполнения**
1	() [] . ->	Выражение	Слева направо
2	- ~ ! * & ++ -- sizeof приведение типов	Унарные	Справа налево
3	* / %	Мультипликативные	Слева направо
4	+ -	Аддитивные	
5	< > <= >=	Отношение	
6	== !=	Отношение (равенство)	
7	&&	Логическое И	
8	||	Логическое ИЛИ	
9	? :	Условная	
10	= *= /= %= += -= &= |= >>=	Простое и составное присваивание	Справа налево

	<<= ^=		
11	,	Последовательное вычисление	Слева направо

Побочные эффекты

Операции присваивания в сложных выражениях могут вызывать побочные эффекты, так как они изменяют значение переменной. Побочный эффект может возникать и при вызове функции, если он содержит прямое или косвенное присваивание (через указатель). Это связано с тем, что аргументы функции могут вычисляться в любом порядке. Например, побочный эффект имеет место в следующем вызове функции:

prog (a,a=k*2);

В зависимости от того, какой аргумент вычисляется первым, в функцию могут быть переданы различные значения.

Порядок вычисления операндов некоторых операций зависит от реализации и поэтому могут возникать разные побочные эффекты, если в одном из операндов используется операции увеличения или уменьшения, а также другие операции присваивания.

Например, выражение i*j+(j++)+(--i) может принимать различные значения при обработке разными компиляторами. Чтобы избежать недоразумений при выполнении побочных эффектов, необходимо придерживаться следующих правил.

1. Не использовать операции присваивания переменной в вызове функции, если эта переменная участвует в формировании других аргументов функции.

2. Не использовать операции присваивания переменной в выражении, если эта переменная используется в выражении более одного раза.

Преобразование типов

При выполнении операций происходят неявные преобразования типов в следующих случаях:

- при выполнении операций осуществляются обычные арифметические преобразования (которые были рассмотрены выше);

- при выполнении операций присваивания, если значение одного типа присваивается переменной другого типа;

- при передаче аргументов функции.

Кроме того, в Си есть возможность явного приведения значения одного типа к другому.

В операциях присваивания тип значения, которое присваивается, преобразуется к типу переменной, получающей это значение. Допускается преобразования целых и плавающих типов, даже если такое преобразование ведет к потере информации.

Преобразование целых типов со знаком. Целое со знаком преобразуется к более короткому целому со знаком, посредством усечения старших битов. Целое со знаком преобразуется к более длинному целому со знаком, путем размножения знака. При преобразовании целого со знаком к целому без знака, целое со знаком преобразуется к размеру целого без знака, и результат рассматривается как значение без знака.

Преобразование целого со знаком к плавающему типу происходит без потери информации, за исключением случая преобразования значения типа long int или unsigned long int к типу float, когда точность может быть потеряна.

Преобразование целых типов без знака. Целое без знака преобразуется к более короткому целому без знака или со знаком путем усечения старших битов. Целое без знака преобразуется к более длинному целому без знака или со знаком путем дополнения нулей слева.

Когда целое без знака преобразуется к целому со знаком того же размера, битовое представление не изменяется. Поэтому значение, которое оно представляет, изменяется, если знаковый бит установлен (равен 1), т.е. когда исходное целое без знака больше чем максимальное положительное целое со знаком, такой же длины.

Целые значения без знака преобразуются к плавающему типу, путем преобразования целого без знака к значению типа signed long, а затем значение signed long преобразуется в плавающий тип. Преобразования из unsigned long к типу float, double или long double производятся с потерей информации, если преобразуемое значение больше, чем максимальное положительное значение, которое может быть представлено для типа long.

Преобразования плавающих типов. Величины типа float преобразуются к типу double без изменения значения. Величины double и long double преобразуются к float с некоторой потерей точности. Если значение слишком велико для float, то происходит потеря значимости, о чем сообщается во время выполнения.

При преобразовании величины с плавающей точкой к целым типам она сначала преобразуется к типу long (дробная часть плавающей величины при этом отбрасывается), а затем величина типа long преобразуется к требуемому целому типу. Если значение слишком велико для long, то результат преобразования не определен.

Преобразования из float, double или long double к типу unsigned long производится с потерей точности, если преобразуемое значение больше, чем максимально возможное положительное значение, представленное типом long.

Преобразования при приведении типов. Явное преобразование типов может быть осуществлено посредством операции приведения типов, которая имеет формат:

(имя-типа) операнд.

В приведенной записи имя-типа задает тип, к которому должен быть преобразован операнд.

Пример:

```
int    i=2;
long   l=2;
double   d;
float    f;
d=(double)i * (double)l;
f=(float)d;
```

В данном примере величины i,l,d будут явно преобразовываться к указанным в круглых скобках типам.

Операторы

Все операторы языка СИ могут быть условно разделены на следующие категории:

- условные операторы, к которым относятся оператор условия if и оператор выбора switch;
- операторы цикла (for,while,do while);
- операторы перехода (break, continue, return, goto);
- другие операторы (оператор "выражение", пустой оператор).

Операторы в программе могут объединяться в составные операторы с помощью фигурных скобок. Любой оператор в программе может быть помечен меткой, состоящей из имени и следующего за ним двоеточия.

Все операторы языка СИ, кроме составных операторов, заканчиваются точкой с запятой ";".

Оператор «выражение»

Любое выражение, которое заканчивается точкой с запятой, является оператором.

Выполнение оператора выражение заключается в вычислении выражения. Примеры:

++ i;

Этот оператор представляет выражение, которое увеличивает значение переменной i на единицу.

a=cos(b * 5);

Этот оператор представляет выражение, включающее в себя операции присваивания и вызова функции.

a(x,y);

Этот оператор представляет выражение, состоящее из вызова функции.

Пустой оператор

Пустой оператор состоит только из точки с запятой. При выполнении этого оператора ничего не происходит. Он обычно используется в следующих случаях:

- в операторах do, for, while, if в строках, когда место оператора не требуется, но по синтаксису требуется хотя бы один оператор;
- при необходимости пометить фигурную скобку.

Синтаксис языка СИ требует, чтобы после метки обязательно следовал оператор. Фигурная же скобка оператором не является. Поэтому, если надо передать управление на фигурную скобку, необходимо использовать пустой оператор.

Пример:

```
int main ( )
{
    :
     { if (...) goto a;    /* переход на скобку */
        { ...
        }
 a:;  }
     return 0;
 }
```

Составной оператор

Составной оператор представляет собой несколько операторов и объявлений, заключенных в фигурные скобки:

```
{  [объявление]
   :
   оператор; [оператор];
   :
}
```

В конце составного оператора точка с запятой не ставится.

Выполнение составного оператора заключается в последовательном выполнении составляющих его операторов.

Пример:

```
int main ()
{
  int    q,b;
  double t,d;
    :
    if (...)
      {
        int    e,g;
        double f,q;
        :
      }
    :
    return (0);
}
```

Отметим, что переменная q является локальной в составном операторе, т.е. она никоим образом не связана с переменной q, объявленной в начале функции main с типом int. Отметим также, что выражение, стоящее после return, может быть заключено в круглые скобки, хотя их наличие необязательно.

Оператор if

Формат оператора:

if (выражение) оператор 1; [else оператор 2;]

Выполнение оператора if начинается с вычисления выражения.

Далее выполнение осуществляется по следующей схеме:

- если выражение истинно (т.е. отлично от 0), то выполняется оператор 1;

- если выражение ложно (т.е. равно 0),то выполняется оператор 2;

- если выражение ложно и отсутствует оператор 2 (в квадратные скобки заключена необязательная конструкция), то выполняется следующий за if оператор.

После выполнения оператора if управление передается следующему оператору программы, если последовательность выполнения операторов программы не будет принудительно нарушена использованием операторов перехода.

Пример:

```
if (i < j)   i++:
else { j = i-3;   i++; }
```

Этот пример иллюстрирует также и тот факт, что на месте «оператор 1», так же как и на месте «оператор 2» могут находиться сложные конструкции.

Допускается использование вложенных операторов if. Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

Примеры:

```
int main ( )
 {
  int t=2, b=7, r=3;
    if (t>b)
     {
        if (b < r)  r=b;
     }
    else r=t;
  return (0);
 }
```

В результате выполнения этой программы r станет равным 2.

Если же в программе опустить фигурные скобки, стоящие после оператора if, то программа будет иметь следующий вид:

```
int main  ( )
```

```
{
 int t=2,b=7,r=3;
 if ( a>b )
    if ( b < c ) t=b;
    else      r=t;
 return (0);
}
```

В этом случае r получит значение равное 3, так как ключевое слово else относится ко второму оператору if, который не выполняется, поскольку не выполняется условие, проверяемое в первом операторе if.

Следующий фрагмент иллюстрирует вложенность операторов if:

```
char ZNAC;
int x,y,z;
 :
if (ZNAC == '-') x = y - z;
else  if (ZNAC == '+') x = y + z;
     else if (ZNAC == '*') x = y * z;
          else if (ZNAC == '/') x = y / z;
               else ...
```

Из рассмотрения этого примера можно сделать вывод, что конструкции, использующие вложенные операторы if, являются довольно громоздкими и не всегда достаточно надежными

Оператор switch

Оператор switch предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch ( выражение )
{   [объявление]
       :
    [ case  константное-выражение1]: [ список-операторов1]
    [ case  константное-выражение2]: [ список-операторов2]
      :
      :
    [ default: [ список операторов ]]
}
```

Выражение, следующее за ключевым словом switch в круглых скобках, может быть любым выражением, но его значение должно быть целым. Значение выражения является ключевым для выбора из нескольких вариантов. Тело оператора switch состоит из нескольких

операторов, помеченных ключевым словом case с последующим константным выражением целого типа. Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы.

Все константные выражения в операторе switch должны быть уникальны. Может присутствовать один фрагмент, помеченный ключевым словом default.

Список операторов может быть пустым, либо содержать один или более операторов. Причем в операторе switch не требуется заключать последовательность операторов в фигурные скобки.

Схема выполнения оператора switch следующая:

- вычисляется выражение в круглых скобках;
- вычисленные значения последовательно сравниваются с константными выражениями, следующими за ключевыми словами case;
- если одно из константных выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом case. После этого последовательно выполняются все остальные ветви;
- если ни одно из константных выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом default, а в случае его отсутствия управление передается на следующий после switch оператор.

Таким образом, программист должен сам позаботиться о досрочном выходе из switch, если это необходимо. Чаще всего для этого используется оператор break.

Пример:

```
int i=2;
switch (i)
  {
   case 1: i += 2;
   case 2: i *= 3;
   case 0: i /= 2;
   case 4: i -= 5;
   default:     ;
}
```

Выполнение оператора switch начинается с оператора, помеченного case 2. Таким образом, переменная i получает значение, равное 6, далее выполняется оператор, помеченный ключевым словом case 0, а

затем case 4, переменная i примет значение 3, а затем значение -2. Оператор, помеченный ключевым словом default, не изменяет значения переменной.

Рассмотрим ранее приведенный пример использования вложенных операторов if, переписанный теперь с использованием оператора switch.

```
char ZNAC;
int x,y,z;
switch (ZNAC)
{
    case '+':  x = y + z;    break;
    case '-':  x = y - z;    break;
    case '*':  x = y * z;    break;
    case '/':  x = u / z;    break;
    default : ;
  }
```

Использование оператора break позволяет в необходимый момент прервать последовательность выполняемых операторов в теле оператора switch путем передачи управления оператору, следующему за switch.

Отметим, что в теле оператора switch можно использовать вложенные операторы switch, при этом в ключевых словах case можно использовать одинаковые константные выражения.

Пример:

```
  :
switch (a)
 {
  case 1: b=c; break;
  case 2:
     switch (d)
     {   case 0:  f=s;  break;
         case 1:  f=9;  break;
         case 2:  f-=9; break;
     }
   case 3: b-=c; break;
    :
 }
```

Оператор break

Оператор break обеспечивает прекращение выполнения самого внутреннего из объединяющих его операторов switch, do, for, while. После выполнения оператора break управление передается оператору, следующему за прерванным.

Оператор for

Оператор for - это наиболее общий способ организации цикла. Он имеет следующий формат:

for (выражение 1 ; выражение 2 ; выражение 3) тело

Выражение 1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение 2 - это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение 3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла.

Схема выполнения оператора for:

1. Вычисляется выражение 1.

2. Вычисляется выражение 2.

3. Если значения выражения 2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение 3 и осуществляется переход к пункту 2, если выражение 2 равно нулю (ложь), то управление передается на оператор, следующий за оператором for.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным.

Пример:

```
int main()
{ int i,b;
  for (i=1; i<10; i++) b="i*i;" return 0; }
```

В этом примере вычисляются квадраты чисел от 1 до 9.

Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом.

Пример:

```
int main()
{  int top,  bot;
   char string[100],  temp;
   for ( top=0, bot=100 ; top < bot ; top++, bot--)
```

```
  {  temp=string[top];
     string[bot]=temp;
  }
  return 0;
}
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные top и bot. Отметим, что на месте выражение 1 и выражение 3 здесь используются несколько выражений, записанных через запятую, и выполняемых последовательно.

Другим вариантом использования оператора for является бесконечный цикл. Для организации такого цикла можно использовать пустое условное выражение, а для выхода из цикла обычно используют дополнительное условие и оператор break.

Пример:

```
for (;;)
{ ...
  ... break;
  ...
}
```

Так как, согласно синтаксису языка Си, оператор может быть пустым, тело оператора for также может быть пустым. Такая форма оператора может быть использована для организации поиска.

Пример:

```
for (i=0; t[i]<10 ; i++) ;
```

В данном примере переменная цикла i принимает значение номера первого элемента массива t, значение которого больше 10.

Оператор while

Оператор цикла while называется циклом с предусловием и имеет следующий формат:

while (выражение) тело ;

В качестве выражения допускается использовать любое выражение языка Си, а в качестве тела любой оператор, в том числе пустой или составной. Схема выполнения оператора while следующая:

1. Вычисляется выражение.

2. Если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело оператора while.

3. Процесс повторяется с пункта 1.

Оператор цикла вида

for (выражение 1; выражение 2; выражение 3) тело ;

может быть заменен оператором while следующим образом:

```
выражение 1;
while (выражение 2)
{  тело
   выражение 3;
}
```

Так же, как и при выполнении оператора for, в операторе while вначале происходит проверка условия. Поэтому оператор while удобно использовать в ситуациях, когда тело оператора не всегда нужно выполнять.

Внутри операторов for и while можно использовать локальные переменные, которые должны быть объявлены с определением соответствующих типов.

Оператор do while

Оператор цикла do while называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид:

do тело while (выражение);

Схема выполнения оператора do while:

1. Выполняется тело цикла (которое может быть составным оператором).

2. Вычисляется выражение.

3. Если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break.

Операторы while и do while могут быть вложенными.

Пример:

```
int i,j,k;
...
i=0;  j=0;  k=0;
do { i++;
    j--;
    while (a[k] < i)  k++;
```

```
    }
while (i<30 && j<-30);
```

Оператор continue

Оператор continue, как и оператор break, используется только внутри операторов цикла, но в отличие от него выполнение программы продолжается не с оператора, следующего за прерванным оператором, а с начала прерванного оператора. Формат оператора следующий:

continue;

Пример:

```
int main()
{   int a,b;
    for (a=1,b=0; a<100; b+="a,a++)" { if (b%2) continue; ... /*
```

обработка четных сумм */ } return 0; }

Когда сумма чисел от 1 до a становится нечетной, оператор continue передает управление на очередную итерацию цикла for, не выполняя операторы обработки четных сумм.

Оператор continue, как и оператор break, прерывает самый внутренний из объемлющих его циклов.

Оператор return

Оператор return завершает выполнение функции, в которой он задан, и возвращает управление в вызывающую функцию, в точку, непосредственно следующую за вызовом. Функция main передает управление операционной системе. Формат оператора:

return [выражение];

Значение выражения, если оно задано, возвращается в вызывающую функцию в качестве значения вызываемой функции. Если выражение опущено, то возвращаемое значение не определено. Выражение может быть заключено в круглые скобки, хотя их наличие не обязательно.

Если в какой-либо функции отсутствует оператор return, то передача управления в вызывающую функцию происходит после выполнения последнего оператора вызываемой функции. При этом возвращаемое значение не определено. Если функция не должна иметь возвращаемого значения, то ее нужно объявлять с типом void.

Таким образом, использование оператора return необходимо либо для немедленного выхода из функции, либо для передачи возвращаемого значения.

Пример:

```
int sum (int a, int b)
{   return (a+b);    }
```

Функция sum имеет два формальных параметра a и b типа int, и возвращает значение типа int, о чем говорит описатель, стоящий перед именем функции. Возвращаемое оператором return значение равно сумме фактических параметров.

Пример:

```
void prov (int a, double b)
{  double c;
  if (a<3) return; else if (b>10) return;
      else { c=a+b;
           if ((2*c-b)==11) return;
           }
 }
```

В этом примере оператор return используется для выхода из функции в случае выполнения одного из проверяемых условий.

Оператор goto

Использование оператора безусловного перехода goto в практике программирования на языке СИ настоятельно не рекомендуется, так как он затрудняет понимание программ и возможность их модификаций.

Формат этого оператора следующий:

```
goto имя-метки;
        ...
имя-метки: оператор;
```

Оператор goto передает управление на оператор, помеченный меткой имя-метки. Помеченный оператор должен находиться в той же функции, что и оператор goto, а используемая метка должна быть уникальной, т.е. одно имя-метки не может быть использовано для разных операторов программы. Имя-метки - это идентификатор.

Любой оператор в составном операторе может иметь свою метку. Используя оператор goto, можно передавать управление внутрь составного оператора. Но нужно быть осторожным при входе в составной оператор, содержащий объявления переменных с инициализацией,

так как объявления располагаются перед выполняемыми операторами и значения объявленных переменных при таком переходе будут не определены.

Структура программы

Исходные файлы и объявление переменных

Обычная СИ-программа представляет собой определение функции main, которая для выполнения необходимых действий вызывает другие функции. Приведенные выше примеры программ представляли собой один исходный файл, содержащий все необходимые для выполнения программы функции. Связь между функциями осуществлялась по данным посредством передачи параметров и возврата значений функций.

Для того, чтобы определяемая функция могла выполнять какие либо действия, она должна использовать переменные. В языке СИ все переменные должны быть объявлены до их использования. Объявления устанавливают соответствие имени и атрибутов переменной, функции или типа. Определение переменной вызывает выделение памяти для хранения ее значения.

В языке СИ программным блоком считается последовательность объявлений, определений и операторов, заключенная в фигурные скобки. Существуют два вида блоков - составной оператор и определение функции, состоящее из составного оператора, являющегося телом функции, и предшествующего телу заголовка функции (в который входят имя функции, типы возвращаемого значения и формальных параметров). Блоки могут включать в себя составные операторы, но не определения функций. Внутренний блок называется вложенным, а внешний блок - объемлющим.

Все функции существуют в течение всего времени выполнения программы.

Область видимости - это часть текста программы, в которой может быть использован данный объект. Объект считается видимым в блоке или в исходном файле, если в этом блоке или файле известны имя и тип объекта. Объект может быть видимым в пределах блока, исходного файла или во всех исходных файлах, образующих программу. Это зависит от того, на каком уровне объявлен объект: на внутреннем, т.е. внутри некоторого блока, или на внешнем, т.е. вне всех блоков.

Если объект объявлен внутри блока, то он видим в этом блоке, и во всех внутренних блоках. Если объект объявлен на внешнем уровне, то он видим от точки его объявления до конца данного исходного файла.

Время жизни и область видимости программных объектов

Время жизни переменной (глобальной или локальной) определяется по следующим правилам.

1. Переменная, объявленная глобально (т.е. вне всех блоков), существует на протяжении всего времени выполнения программы.

2. Локальные переменные (т.е. объявленные внутри блока) с классом памяти register или auto, имеют время жизни только на период выполнения того блока, в котором они объявлены. Если локальная переменная объявлена с классом памяти static или extern, то она имеет время жизни на период выполнения всей программы.

Видимость переменных и функций в программе определяется следующими правилами.

1. Переменная, объявленная или определенная глобально, видима от точки объявления или определения до конца исходного файла. Можно сделать переменную видимой и в других исходных файлах, для чего в этих файлах следует ее объявить с классом памяти extern.

2. Переменная, объявленная или определенная локально, видима от точки объявления или определения до конца текущего блока. Такая переменная называется локальной.

3. Переменные из объемлющих блоков, включая переменные, объявленные на глобальном уровне, видимы во внутренних блоках. Эту видимость называют вложенной. Если переменная, объявленная внутри блока, имеет то же имя, что и переменная, объявленная в объемлющем блоке, то это разные переменные, и переменная из объемлющего блока во внутреннем блоке будет невидимой.

4. Функции с классом памяти static видимы только в исходном файле, в котором они определены. Всякие другие функции видимы во всей программе.

Метки в функциях видимы на протяжении всей функции.

Имена формальных параметров, объявленные в списке параметров прототипа функции, видимы только от точки объявления параметра до конца объявления функции.

Директивы препроцессора

Директивы препроцессора представляют собой инструкции, записанные в тексте программы на СИ и выполняемые до трансляции программы. Директивы препроцессора позволяют изменить текст программы, например, заменить некоторые лексемы в тексте, вставить текст из другого файла, запретить трансляцию части текста и т.п. Все директивы препроцессора начинаются со знака #. После директив препроцессора точка с запятой не ставится.

Директива #include

Директива #include включает в текст программы содержимое указанного файла. Эта директива имеет две формы:

```
#include "имя файла"
#include <имя файла>
```

Если имя файла указано в кавычках, то поиск файла осуществляется в соответствии с заданным маршрутом, а при его отсутствии в текущем каталоге. Если имя файла задано в угловых скобках, то поиск файла производится в стандартных директориях операционной системы, задаваемых командой PATH.

Директива #include может быть вложенной, т.е. во включаемом файле тоже может содержаться директива #include, которая замещается после включения файла, содержащего эту директиву.

Директива #include широко используется для включения в программу так называемых заголовочных файлов, содержащих прототипы библиотечных функций, и поэтому большинство программ на СИ начинаются с этой директивы.

Директива #define

Директива #define служит для замены часто использующихся констант, ключевых слов, операторов или выражений некоторыми идентификаторами. Идентификаторы, заменяющие текстовые или числовые константы, называют именованными константами. Идентификаторы, заменяющие фрагменты программ, называют макроопределениями, причем макроопределения могут иметь аргументы.

Директива #define имеет две синтаксические формы:

```
#define  идентификатор текст
#define  идентификатор (список параметров) текст
```

Эта директива заменяет все последующие вхождения идентификатора на текст. Текст может представлять собой любой фрагмент программы на СИ, а также может и отсутствовать. В последнем случае все экземпляры идентификатора удаляются из программы.

Пример:

```
#define WIDTH 80
#define LENGTH (WIDTH+10)
```

Эти директивы изменят в тексте программы каждое слово WIDTH на число 80, а каждое слово LENGTH на выражение (80+10) вместе с окружающими его скобками.

Во второй синтаксической форме в директиве #define имеется список формальных параметров, который может содержать один или несколько идентификаторов, разделенных запятыми. Формальные параметры в тексте макроопределения отмечают позиции, на которые должны быть подставлены фактические аргументы макровызова. Каждый формальный параметр может появиться в тексте макроопределения несколько раз.

При макровызове вслед за идентификатором записывается список фактических аргументов, количество которых должно совпадать с количеством формальных параметров.

Пример:

```
#define  MAX(x,y) ((x)>(y))?(x):(y)
```

Эта директива заменит фрагмент

```
t=MAX(i,s[i]);
```

на фрагмент

```
t=((i)>(s[i])?(i):(s[i]);
```

Как и в предыдущем примере, круглые скобки, в которые заключены формальные параметры макроопределения, позволяют избежать ошибок, связанных с неправильным порядком выполнения операций, если фактические аргументы являются выражениями.

Например, при наличии скобок фрагмент

```
t=MAX(i&j,s[i]||j);
```

будет заменен на фрагмент

```
t=((i&j)>(s[i]||j)?(i&j):(s[i]||j);
```

а при отсутствии скобок - на фрагмент

```
t=(i&j>s[i]||j)?i&j:s[i]||j;
```

в котором условное выражение вычисляется в совершенно другом порядке.

Директива #undef

Директива #undef используется для отмены действия директивы #define. Синтаксис этой директивы следующий:

#undef идентификатор

Директива отменяет действие текущего определения #define для указанного идентификатора. Пример:

```
#undef WIDTH
#undef MAX
```

Эти директивы отменяют определение именованной константы WIDTH и макроопределения MAX.

ЛИТЕРАТУРА

1. М.Болски. Язык программирования Си. Справочник: М.: Радио и связь, 1988. – 96 с.
2. Р.Берри. Язык Си. Введение для программистов. / Р.Берри, Б.Микинз. - М.: Финансы и статистика, 1988.
3. М.Дансмур. ОС UNIX и программирование на языке Си. М.: Радио и связь, 1989.
4. Джехани Н. Программирование на языке СИ./Пер. с англ. И.Г.Шестакова под ред. Б.А.Кузьмина. М.: Радио и связь, 1988, - 270 с.
5. Дейтел, Х. М. Как программировать на С. М.: «Бином», 2000.
6. Дуглас Т. Программирование на языке СИ для персонального компьютера IBM PC. М.: Радио и связь, 1991, - 428 с.
7. Жешке Р. Толковый стандарт языка Си: СПб: Питер, 1994. 223 с.
8. Керниган Б. Язык программирования Си./ Керниган Б., Ритчи Д. М.: «Вильямс», 2007. – 304 с.
9. Белецкий Ян. Энциклопедия языка СИ. М: Мир, 1992, -686 с.
10. М.Уэйт. Язык Си. Руководство для начинающих./М.Уэйт, С.Прата, Д.Мартин. М.: Мир, 1988. – 512 с.
11. Р. Хазфилд, К. Лоуренс и др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: М.: «ДиаСофт», 2001. - 736 с.
12. Л.Хэнкок. Введение в программирование на языке Си./ Л.Хэнкок, М.Кригер. - М.: Радио и связь, 1986. - 192 с.
13. Шилдт Г. Полный справочник по С. М.: «Вильямс», 2004. – 704 с.