Лекция 1: Основы программирования на языке С. Данные и команды.

Общая характеристика языка С/С++

Язык С создан в 1970 г. для *системного программирования*, т. е. для разработки систем программного обеспечения, первоначально - для создания операционной системы UNIX. В дальнейшем он приобрел большую популярность при решении самых разных задач и входит в десятку наиболее распространенных в мире языков. Основные характерные черты языка С следующие.

1. Язык С является *процедурно-ориентированным* языком высокого уровня. Процедурные языки предназначены для описания процедуры, т. е. алгоритма решения задачи. Этот класс включает наиболее распространенные языки программирования: Pascal, BASIC, FORTRAN, PL/1 и др.

Менее известны *непроцедурные языки*: LISP, PROLOG и др. Их называют еще *декларативными*, *проблемно-ориентированными* или даже *неалгоритмическими* языками программирования. На них описывается не столько процесс решения задачи, сколько характеристики искомого результата. По этому описанию транслятор сам строит нужный алгоритм, что возможно только при достаточно узкой специализации языка.

2. *Универсальность*. Язык С содержит большой набор операторов и операций. Наряду с характерными для языков высокого уровня средствами (структурность, модульность, определяемые типы данных), включает средства низкого (близкого к машинному) уровня: указатели, битовые операции, сдвиги. Это позволяет во многих случаях обойтись без языка ассемблера.

3. *Ориентация на структурное программирование* - наличие операторов **if**, **while**, **do-while** для реализации структурных алгоритмов. Подобные операторы имеются в Algol-60, PL/1, Pascal и в большинстве современных языков, но отсутствуют, например, в старых версиях языков Basic и Fortran, в которых для реализации циклов и ветвлений необходимы операторы перехода **goto.**

4. *Простой и удобный синтаксис*. Программа, как правило, получается короче, чем на других языках.

5. *Ориентация на профессионалов*. Программисту предоставлен максимум свободы, что позволяет писать компактные и эффективные программы, но в то же время требует высокой квалификации, чтобы избегать ошибок и не делать программу излишне запутанной.

Язык Pascal, напротив, разработан для обучения программированию. Поэтому при внешнем сходстве эти языки имеют принципиальные различия. Pascal содержит много ограничений. Они оберегают от возможных ошибок начинающих программистов, но затрудняют профессиональное программирование, при котором часто приходится обходить эти ограничения.

6. Язык С получил развитие в семействе популярных языков: C++, Java, C#. Эти языки сохраняют в основном синтаксис и базовые средства языка С.

Правила записи программы

Обычно *алфавит* языков программирования, в том числе С, включает латинские буквы, цифры и *специальные символы*: знаки операций, скобки, *разделители* (запятая, точка с запятой, точка) и другие.

Во многих языках, например Pascal, соответствующие заглавная и строчная буквы считаются одинаковыми. В языке С/С++ они различаются.

Русские буквы и другие не входящие в алфавит символы можно использовать в комментариях и обрабатываемых данных (символьных и строковых константах).

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

цифра ::= 0|1|2|3|4|5|6|7|8|9
имя ::= {буква | _} [буква | _ | цифра]...

Имена (идентификаторы) используются для обозначения переменных, констант, функций, меток и типов данных. В языке С имя может быть произвольной длины, но некоторые компиляторы могут учитывать ограниченное количество символов.

В подобных ситуациях, когда некоторый аспект языка (в данном случае длина имен) может быть по-разному реализован в трансляторах, говорят, что он *зависит от реализации* языка.

В любом языке имеются *зарезервированные (служебные* или *ключевые) слова*, которые нельзя использовать как имена. В учебнике служебные слова языка С/С++ для наглядности выделены полужирным шрифтом.

Служебные слова языка С/С++:

- типы данных: **char**, **const**, **double**, **enum**, **float**, **int**, **long**, **short**, **signed**, **struct**, **union**, **unsigned**, **void**;
- классы памяти: **auto, extern, register, static**;
- для операторов: **break**, **case**, **continue**, **default**, **do**, **else**, **for**, **goto**, **if**, **return**, **switch**, **while**;
- другие: **sizeof**, **typedef**.

Программа состоит из лексем. Лексемой является константа, имя переменной или другого объекта, служебное слово, знак операции и др. Для разделения лексем служат не влияющие на смысл программы *пробельные символы*: пробелы, символы табуляции и новой строки. Вместо одного пробельного символа для наглядности программы можно использовать любое их количество.

Между лексемами разрешено также вставлять комментарий для пояснения. Он не влияет на выполнение программы.

Комментарий ::= /* [символ **...**] */ | // [символ **...**] конец-строки

Операторы

В языке С имеются следующие операторы.

оператор ::= пустой-оператор | оператор-выражение | блок |
составной-оператор | условный-оператор |
цикл-с-предусловием | цикл-с-постусловием |
цикл-с-параметром | переход | переключатель |
оператор-продолжения | оператор-завершения |
оператор-возврата

пустой-оператор ::= ;

Пустой оператор используется в тех случаях, когда по правилам синтаксиса должен присутствовать оператор (например, внутри цикла или условного оператора), а делать ничего не требуется.

оператор-выражение ::= выражение ;

В языке С выражение, заканчивающееся точкой с запятой, образует *оператор-выражение*. Частными случаями этого оператора являются оператор присваивания и вызов подпрограммы, которые в других языках обычно рассматриваются как отдельные виды операторов.

составной-оператор ::= { [оператор...] }

Составной оператор используется, когда необходимо написать последовательность из нескольких операторов там, где по правилам синтаксиса должен присутствовать один оператор, например, внутри цикла или условного оператора.

условный-оператор ::= **if** (выражение) оператор [**else** оператор]

При выполнении *условного оператора* сначала проверяется условие - выражение, значением которого должно быть целое число. Если это значение не равно нулю, то выполняется следующий за выражением оператор.

Пример – получение абсолютного значения x:

if (x < 0) x = -x; // x = | x |

Если присутствует конструкция “**else** оператор”, то оператор, следующий за словом **else**, выполняется при нулевом значении выражения.

Пример: **if** (a > b) x = a; **else** x = b; // x = max (a, b);

цикл-с-предусловием ::= **while** (выражение) оператор

При выполнении *цикла с предусловием* сначала проверяется условие - выражение, значением которого должно быть целое число. Если это значение не равно нулю, то выполняется следующий за выражением оператор, снова проверяется выражение и т. д., пока выражение не примет

значение ноль. В таком цикле оператор может иметь нулевое число повторений.

цикл-с-постусловием ::= **do** оператор **while** (выражение) ;

При выполнении *цикла с постусловием* сначала выполняется оператор, затем проверяется условие - выражение, значением которого должно быть целое число. Если это значение не равно нулю, то снова выполняется оператор и т. д., пока выражение не примет значение ноль. Оператор в этом цикле выполняется не менее одного раза.

оператор-возврата ::= **return** [выражение] ;

Оператор возврата прекращает выполнение вызова функции. Происходит возврат из тела функции в точку вызывающей программы за вызовом функции. Если присутствует выражение, то его значение становится значением функции и подставляется на место вызова функции.

Цикл с параметром описан в разделе 2.6.1. Операторы перехода, переключателя, продолжения и завершения рассмотрены в разделе 2.6.2.

2.6.1. Цикл с параметром

Такой цикл, при повторении которого некоторая переменная (*параметр цикла*) изменяется с определенным шагом, называется *циклом с параметром*. Цикл с параметром часто встречается в программах, и обозначения для такого цикла предусмотрены практически во всех языках программирования.

цикл-с-параметром ::= **for** ([выражение]; [выражение]; [выражение])
оператор

В языке С цикл с параметром записывается с помощью оператора **for** (для), имеющего вид:

for (выражение-1; выражение-2; выражение-3) оператор

Оператор **for** эквивалентен фрагменту программы:

```
выражение-1;
while (выражение-2) {
   оператор
   выражение-3;
}
```

Таким образом, выражение-1 обозначает подготовительные операции, выполняемые один раз перед циклом; выражение-2 - условие повторения, выражение-3 - действия, выполняемые после каждого повторения оператора в теле цикла.

Например,

фрагмент программы: можно записать через цикл **for:**

```
j=2;
```

```
while (j <= k) {                    for (j=2; j <= k; j++)
   f = f * j;                           f = f * j;
   j++;
}
```

Каждое выражение может содержать несколько выражений, разделенных запятыми - *операция-запятая* (в выражении 2 при этом проверяется значение последнего из входящих в него выражений). Это позволяет, например, организовать цикл с двумя параметрами. Так, в следующем цикле

for (k=1, m=200; k <= 100; k++, m--) оператор

оператор выполняется для значений k=1, m=200, затем k=2, m=199, …, затем k=100, m=101.

Вместо *f=1; **for** (j=2; j <= k; j++)
можно написать **for** (*f=1, j=2; j <= k; j++) .

Дополнительные операторы управления

Перед любым оператором можно поставить *метку* и выполнять к ней *переход* из любой точки той функции, где расположен *помеченный оператор*.

оператор ::= метка: оператор
метка ::= имя
переход ::= **goto** метка;

Без переходов можно обойтись, и их не следует использовать для организации циклов и ветвлений, поскольку они затрудняют понимание программы. Имеет смысл использовать переходы, если они делают программу короче и понятнее, например, для обработки особых ситуаций, когда приходится выходить сразу из нескольких вложенных циклов (рассмотренный ниже оператор **break** выходит только из самого внутреннего цикла).

Пример 2.1.

```
while ( ... )
{  while ( ... )
   {  ...
        if (обнаружена ошибка)
            goto oshibka;
        S1;
    }
}
S2;
oshibka: устранение последствий;
```

Устранение подобных переходов обычно загромождает программу дополнительными проверками и переменными:

```
int osh;
. . .
osh=0;
while ( ... && !osh)
{  while ( ... && !osh)
   {  . . .
        if (обнаружена ошибка) osh=1;
        if ( !osh )
        {
           S1;
        }
   }
}
if ( !osh )
{
   S2;
}
oshibka: устранение последствий;
```

оператор-продолжения ::= **continue**;

Оператор продолжения **continue** (продолжать) вызывает пропуск оставшейся части тела ближайшего охватывающего цикла **while**, **do-while** или **for** и переход к началу следующего повторения (эквивалентен переходу к концу тела цикла - **goto** m;, если в конце тела цикла поставить метку m с пустым оператором):

```
while (...)          do                    for  ( ... )
{ . . .              {  . . .              { . . .
   continue;              continue;             continue;
   ...                    ...                   ...
   m: ;                   m: ;                  m: ;
}                     } while (...);         }
```

Оператор **continue** затрудняет понимание программы (хотя и меньше, чем переход) и его использование желательно ограничить.

оператор-завершения ::= **break**;

Оператор завершения **break** (прекратить) прекращает выполнение ближайшего охватывающего его оператора цикла **while**, **do-while**, **for** или переключателя **switch** и вызывает переход к следующему за ним оператору (эквивалентен **goto** m; в следующих примерах). Его использование часто помогает избежать сложных проверок.

```
while(...)       do              for (...)          switch(...)
{  . . .         {  . . .        {  . . .           { . . .
   break;           break;          break;             break;
   . . .            . . .           . . .                    . . .
```

```
}                  } while(...);      }                  }
m: ;               m: ;               m: ;               m: ;
```

```
переключатель ::= switch (выражение)
                  {  [case цел-конст-выраж:  [оператор...]] ...
                     [ default: оператор...]
                     [ case цел-конст-выраж: [оператор...]] ...
                  }
```

Оператор переключателя **switch** (переключить) предназначен для организации многовариантного ветвления. Каждый вариант **case** (случай) можно пометить целой или символьной константой (или константным выражением). Переключатель осуществляет переход к *метке* **case**, содержащей константу, равную значению выражения.

Если ни одна из констант не совпадает со значением выражения, выполняется переход к *метке* **default** (умолчание). При отсутствии нужной метки и варианта **default** переключатель эквивалентен пустому оператору. Вариант **default** не обязательно должен быть последним.

Каждый вариант обычно заканчивается оператором **break**. В противном случае после него выполняются последующие варианты до конца переключателя или до ближайшего **break.**

Например, для изменения символьного значения z по схеме ‘+’ → ‘*’ → ‘-‘ → ‘+’ можно использовать следующий оператор

```
switch (z) {
  case ‘+’:  z = ‘*’; break;
  case ‘*’:  z = ‘-‘; break;
  case ‘-‘:  z = ‘+’;
}
```

Базовые типы данных

Тип данных – это множество значений с набором операций над ними.

```
тип  ::= [const] основной-тип | имя-типа | производный-тип |
                enum имя-типа | struct имя-типа | union имя-типа
имя-типа          ::= имя
основной-тип      ::= целый-тип | вещественный-тип
```

Пустой тип **void** в заголовке и прототипе функции указывает на отсутствие у нее значения или параметров (по умолчанию, когда не указан тип значения функции, подразумевается тип **int**). Остальные типы описывают возможные значения переменных и функций.

```
определение-данных   ::=     тип список-описат-иниц ;
список-описат-иниц   ::=     описатель-иниц[, описатель-иниц]...
описатель-иниц   ::=  [const] описатель [= нач-значение]
описатель            ::=  имя | описатель [ [колич-элементов] ]
```

Целый тип

Целый тип включает целые числа, диапазон которых ограничен размером отведенной для числа памяти.

целый-тип ::= [**signed** | **unsigned**] [символьный-тип | **int** |
short [**int**] | **long** [**int**]]

По умолчанию, когда не указан ни один из возможных вариантов, подразумевается подчеркнутый вариант: **signed** или **int**.

Размер памяти и диапазон значений зависят от реализации языка. Типичные для IBM PC-совместимых 16-битовых компьютеров характеристики разновидностей целого типа приведены ниже. На 32-битовых компьютерах обычно используют четырехбайтовые целые числа типа **int**.

	Размер	Диапазон
int	2 байта 16 бит	от -2^{15} до $2^{15}-1$
		т. е. от -32768 до 32767
short (<= **int**)	2 байта 16 бит	от -32768 до 32767
long (>= **int**)	4 байта 32 бита	от -2^{31} до $2^{31}-1$
		т. е. приблизительно $\pm 2*10^9$
unsigned	2 байта 16 бит	от 0 до $2^{16}-1$
		т. е. от 0 до 65535
unsigned short	2 байта 16 бит	от 0 до 65535
unsigned long	4 байта 32 бита	от 0 до $2^{32}-1$
		т. е. приблизительно от 0 до $4*10^9$
unsigned char	1 байт 8 бит	от 0 до 255

Служебные слова имеют следующий смысл:
int (integer) - целое, **short** - короткое, **long** - длинное, **signed** - число со знаком, **unsigned** - *беззнаковое* (неотрицательное) число, **char** (character) - символьное (целочисленный код символа).

Константы целого типа

целое-число ::= 10-е-число | 8-е-число | 16-е-число |
символьная-константа
10-е-число ::= цифра-не-0 [цифра **...**][L | l]
цифра-не-0 ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
8-е-число ::= 0 цифра-8-я **...** [L | l]
цифра-8-я ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
16-е-число ::= 0{X | x} цифра-16-я **...** [L | l]
цифра-16-я ::= цифра | A | B | C | D | E | F | a | b | c | d | e | f

Шестнадцатеричные цифры со значениями от 10 до 15 изображаются первыми шестью малыми или большими буквами латинского алфавита:

a=A=10, b=B=11, c=C=12, d=D=13, e=E=14, f=F=15.

Примеры:

0 34 034 = 3*8+4 =28 0x3B8 =3*256+11*16+8=936

0L 034L 0X25L - буква "L" или "l" обозначает константу типа **long.**

Восьмеричные и *шестнадцатеричные числа* служат для краткой записи двоичных чисел. Перевод чисел из этих систем в двоичную систему и обратно очень прост, т. к. их основания являются степенями двойки. Каждая восьмеричная цифра соответствует трем двоичным цифрам, а шестнадцатеричная – четырем двоичным цифрам.

Допустимые операции над целыми числами: + сложение, - вычитание и изменение знака, * умножение, / целочисленное деление, % остаток от целочисленного деления, ++ увеличение на единицу (*инкремент*), -- уменьшение на единицу (*декремент*).

Один и тот же знак "-" обозначает разные операции: *бинарную операцию* вычитания (с двумя операндами) и *унарную операцию* "изменить знак" (с одним операндом). Более того, забегая вперед, заметим, что знаки: "+", "-", "*" и "/" в языке С обозначают аналогичные арифметические операции не только для целых, но и для вещественных чисел, а ведь эти операции выполняются совершенно иначе, чем для целых чисел (другими машинными командами), поскольку у каждого типа данных свой способ представления значений в памяти ЭВМ и свой способ выполнения операций над ними.

Использование одинакового обозначения для разных операций называют *перегрузкой*. В языке С перегружены знаки: "+", "-", "*", "/" и некоторые другие. Их смысл зависит от количества и типов операндов. Это явление часто встречается в языках программирования.

При *целочисленном делении с остатком* дробная часть частного отбрасывается, а остаток получается из формулы

Делимое = Частное * Делитель + Остаток

Пример:

14 / 3 = 4	14 % 3 = 2	14 = 4 * 3 + 2
(-14)/ 3 = -4	(-14)% 3 = -2	-14 = (-4)* 3 + (-2)

При таком определении в языке С/С++ абсолютная величина частного и остатка не зависит от знаков делимого и делителя. Остаток по абсолютной величине меньше делителя. Частное отрицательно, если делимое и делитель имеют разные знаки. Знак остатка совпадает со знаком делимого.

Заметим, что некоторые математики определяют остаток по другим правилам. При появлении сомнений в подобных ситуациях полезно провести эксперимент на компьютере.

Кроме рассмотренных операций, в языке С имеются нетрадиционные для языков унарные операции ++ и --. Они увеличивают или уменьшают на 1 значение переменной, являющейся операндом.

Если операция ++ или -- записана в *префиксной форме* (знак операции - перед операндом), то значением выражений

++x, --x

является новое значение переменной x. Для *постфиксной формы* (знак операции - после операнда) значение выражений

x++, x—

равно старому значению переменной x.

Таким образом, порядок использования и изменения значения в операциях ++ и -- совпадает с порядком записи переменной и операции.

Пример. Пусть x равен 5.

Выражение	Новое значение x	Значение выражения
++x	6	6
x++	6	5
--x	4	4
x--	4	5

Представление целых чисел и операции над ними производятся точно (если нет переполнения)!

Вещественный тип

вещественный-тип ::= **float** | **double** | **long double**

float - плавающее (с плавающей точкой), **double -** двойной точности.
long double – длинное двойной точности.

Тип	Размер	Точность	Диапазон значений
float	4 байта	6..7 значащих цифр	± (от 10^{-38} до 10^{+38})
double	8 байт	15 значащих цифр	± (от 10^{-308} до 10^{+308})
long double	10 байт	19 значащих цифр	± (от $3.4 \cdot 10^{-4932}$ до $1.1 \cdot 10^{+4932}$)

Константы вещественного типа имеют либо точку, либо букву E (или e), обозначающую умножение на степень десяти.

Примеры.
Запись *с фиксированной точкой:* 3.14
Запись *с плавающей точкой (экспоненциальная):* 31.4E-1 или 31.4e-1

Операции над вещественными числами: + сложение, - вычитание и изменение знака, * умножение, / деление, ++ увеличение на единицу, -- уменьшение на единицу .

Представление вещественных чисел и операции над ними производятся приближенно (даже если нет переполнения)!

Нельзя, например, писать:

x=1; while (x != 2) {... x = x + 0.1;}

В этом случае программа *зациклится* (будет повторять цикл бесконечно), т. к. 1 + 10 * 0.1 лишь приблизительно равно 2.

Вместо (x != 2) следует, например, писать (abs(x-2) > 0.001).

Символьный тип

Значением *символьного типа* является одиночный символ. Обычно в языках над символами допускаются только операции присваивания и сравнения.

В языке С символьные данные рассматриваются как разновидность целых чисел (со знаком или без знака, в зависимости от реализации). Числовым значением символа является его код. Поэтому в языке С над символами разрешаются еще арифметические операции.

Код символа обычно занимает один байт, но иногда и два байта (в международном коде UNICODE).

символьный-тип ::= **char**
символьная-константа **::=** 'символ' | '\n' | '\t' | '\v' |
'\b' | '\r' | '\f' | '\\' | '\'' |
'\"' | '\0' | '\цифра8 цифра8 цифра8'

Специальные (управляющие) символьные константы:

'\n' Новая строка (new line) - переход в начало следующей строки,
'\t' Табуляция горизонтальная,
'\v' Табуляция вертикальная,
'\b' Возврат на шаг (backspace) - стирание предыдущего символа,
'\r' Возврат каретки (carriage return),
'\f' Перевод формата,
'\\' \
'\'' '
'\"' "
'\0' Нулевой символ (байт с нулевым кодом),
'\цифра8[цифра8][цифра8]' Символ с данным восьмеричным кодом,
(частный случай - '\0')

Примеры символьных констант: 'D' '*' '5'
('5' обозначает код цифры "5", который не равен числу 5).

Обычно в персональных компьютерах для символов используется *американский стандартный код ASCII*, ставший и международным стандартом. В ASCII, например, код латинской буквы 'D' равен 68 и поэтому символьные константы 'D' и '\104' в языке С обозначают одно и то же и эквивалентны числовым константам: 68, 0104, 0x44.

В зависимости от конкретной ситуации выбирается наиболее понятный из этих вариантов записи с учетом того, что числовая запись делает

программу менее мобильной: при другой кодировке символов ее работа нарушится.

Конкретный набор символов и их кодировка зависят от реализации языка, но во всех кодировках (и для всех языков) соблюдаются следующие условия.

1. Цифры закодированы возрастающими на 1 целыми числами:

'1' = '0' + 1
'2' = '0' + 2
. . .
'9' = '0' + 9

Отсюда важные соотношения:

Код цифры = '0' + Значение цифры а

Значение цифры = Код цифры - '0' б

2. Коды заглавных латинских букв возрастают по алфавиту, но не обязательно подряд:

'A' < 'B' < ... < 'Z' , в

но может быть, что 'B' ≠ 'A'+1 или 'C' ≠ 'B'+1 и т. д.

3. Коды строчных латинских букв (если они есть в наборе символов) также обладают свойствами (в):

'a' < 'b' < ... < 'z' и, возможно, 'b' ≠ 'a'+1 или 'c' ≠ 'b'+1 и т. д.

Русские буквы (они имеются не во всех кодировках) кодируются не всегда по алфавиту и могут не обладать этими свойствами.

Рассмотрим некоторые стандартные функции ввода-вывода символа (для их использования необходим #**include** <stdio.h>).

Пусть имеется определение переменной x:

char x; (или **int** x;)

Тогда ввод символа из *стандартного входного файла* (клавиатуры) в x:

scanf ("%c", &x);

можно заменить присваиванием

x = getchar ();

Функция getchar() вводит очередной символ из стандартного входного файла и возвращает в виде значения код этого символа.

В языке C, в отличие от других языков, ввод часто пишется не только в виде самостоятельного оператора, но и внутри условия в операторах **if**, **while**, **do-while** и **for**.

Например, цикл ввода символов до конца файла может иметь вид

```
while ((x=getchar()) != EOF)
    Обработка x;
```

При попытке ввода данных за концом файла (после нажатия клавиш Ctrl-Z или Ctrl-z, затем Enter) переменной x присваивается код конца файла, обозначаемый символической константой EOF. Ее определение зависит от реализации и находится в файле stdio.h. Для проверки конца входного файла введенный символ сравнивается с EOF.

Сравнение x с EOF правильно выполнится во всех реализациях языка C, если x определена как **int**. Использование типа **char** может приводить к неверному сравнению в некоторых реализациях (когда код символа рассматривается как беззнаковое число, он не может равняться константе EOF, обычное значение которой -1).

Функция getch() - ввод символа без эхо-вывода, отличается от getchar() только тем, что вводимый с клавиатуры символ не отображается на экране дисплея. Это удобно, например, при вводе пароля, который не должен появляться на экране.

Вывод символа x в *стандартный выходной файл* (на экран)

```
printf ("%c", x);
```

эквивалентен оператору

```
putchar (x);
```

Стандартный входной и выходной файлы, вместо клавиатуры и экрана, можно переадресовать на любой файл магнитного диска.

2.7.4. Логический (булевский) тип

Логический (булевский) тип – это тип данных из двух значений: "истина" и "ложь", над которыми выполняются логические операции, рассмотренные в подразделе 2.8.3.

Обычно в языках программирования логический тип относится к основным типам данных. В языке C самостоятельный логический тип отсутствует. Вместо этого в логических операциях целое нулевое число рассматривается как *значение ложь*, а ненулевое значение - как *истина*.

Таким же образом проверяется условие в операторах **if** и **while**, которое поэтому можно записывать в виде произвольного выражения, а не только в виде особого логического выражения, как в других языках.

Выражения

Выражение - это формула, определяющая последовательность операций для получения значения.

Выражения в языках программирования высокого уровня аналогичны алгебраическим выражениям, только записываются в одну строку, без

дробной черты, нижних и верхних индексов и т. п. В языке С форма записи выражений задается следующими грамматическими правилами.

выражение ::= операнд | префиксная-операция выражение |
++ адрес | -- адрес | адрес ++ | адрес -- |
sizeof { выражение | (тип) } |
выражение бинарная-операция выражение |
выражение ? выражение : выражение |
адрес операция-присваивания выражение

префиксная-операция ::= ! | ~ | - | & | * | (тип)
бинарная-операция ::= * | / | % | + | - | << | >> | < | > |
<= | >= | == | != | & | ^ | ¦ | && | ¦¦ | ,
операция-присваивания ::= = | *= | /= | %= | += | -= |
<<= | >>= | &= | ^= | ¦=
адрес ::= переменная | * выражение
операнд ::= константа | переменная | вызов-функции |
(выражение)

вызов-функции ::= операнд ([выражение [, выражение] ...])
переменная ::= имя | поле-структуры | поле-объединения |
элемент-массива
элемент-массива ::= переменная [выражение]

константа ::= строка | целое-число | вещественное-число |
имя-конст

Переменная и постоянная величина. Присваивание, ввод и вывоД

Информация в алгоритмах представляется в виде величин. *Величина* имеет *обозначение* в алгоритме или программе и обладает *значением*, которое при выполнении программы хранится в некоторой области оперативной памяти ЭВМ, имеющей определенный *адрес* – номер первой из составляющих ее ячеек памяти.

Постоянная величина (константа) не изменяет своего значения в процессе выполнения алгоритма. Обычно константы записываются в алгоритме в виде чисел и других обозначений, явно показывающих их значения.

Примеры констант языков С/С++:

- *целочисленные константы*: 48 -123;

- *вещественные константы* , т. е. действительные (целые и нецелые) числа:
57.6 -2.0 (дробная часть отделяется точкой),
3.8E-6 (число 3.8, умноженное на 10 в степени -6);
- *символьные константы* (одиночные символы) в С/С++ заключаются в апострофы - одиночные кавычки: 'A' , '5' , '*' , '-';

- *строковые константы* (текст) в C/C++ пишутся в двойных кавычках: "КАИ", "Сообщение 1", "С".

Переменная величина (или просто - *переменная*) обозначается в алгоритме именем и в процессе выполнения алгоритма может принимать различные значения.

В математике имя переменной обычно состоит из одной буквы. В языках программирования имя переменной (как и имена других объектов программы) обычно представляет собой *идентификатор* - последовательность латинских букв и цифр, начинающуюся с буквы, например: X, kod2, Sort.

Каждая константа и переменная относится к определенному *типу данных,* определяющему множество возможных значений величины, форму записи констант и набор допустимых операций над значениями. Типы данных отличаются также способом *представления* значений в памяти ЭВМ и выполнения операций над ними. Поэтому в программе необходимо определить тип каждой использованной в ней величины.

В жизни нам также важно знать тип объекта, чтобы правильно обращаться с ним. Так, операция “построить дом” будет выполняться по-разному в зависимости от типа дома: от того, возводится ли этот дом, например, для жилья, или собирается из детского конструктора или создается в какой-нибудь компьютерной игре.

Основные типы данных в языке C/C++ обозначаются следующими служебными словами:

int - целый: целые числа;
float - вещественный: вещественные числа;
char - символьный: одиночные символы.

Например, *определения переменных*

```
float x, y;    int t=5, kol;    char sim;
```

обозначают, что x и y являются вещественными переменными, t, kol – целочисленные переменные, sim - символьная переменная. Начальное значение переменной t равно 5.

Константа тоже может иметь имя. Именованную константу называют *символической константой***.** Примеры способов определения символических констант на языке C/C++:

```
#define     NMAX  100
const  float eps = 0.000001;
```

Здесь NMAX=100 - целочисленная константа, eps=0.000001 - вещественная константа.

Использование подобных символических констант делает программу более наглядной и легко изменяемой. По традиции, в языке C/C++ имена символических констант обычно пишут заглавными буквами.

Переменную можно сравнить с классной доской, на которой записано некоторое значение. Это значение можно читать и переписывать (копировать) в другое место сколько угодно раз. Его можно стереть и на том же месте написать новое значение. Прежнее значение при этом теряется.

В программе для хранения значения переменной или константы выделяется одна или несколько ячеек оперативной памяти ЭВМ. Ячейка памяти как раз и работает по принципу классной доски: чтение из ячейки происходит без разрушения информации в ней, а при записи новой информации старое содержимое ячейки автоматически стирается.

Задание нового значения переменной называется *присваиванием*. Присваивание является одной из важнейших операций большинства алгоритмических языков.

В языке С, как и во многих языках программирования, *оператор присваивания* имеет вид

переменная = выражение;

В языке С символ "=" обозначает *операцию присваивания* и читается "присвоить", а операция сравнения "равно" записывается двумя знаками равенства "==". Это сделано для сокращения программы, т. к. присваивание используется чаще, чем операция "равно". В ряде языков, в том числе Pascal, чтобы отличать от операции "равно" =, знаком присваивания служит ":=".

Примеры операторов присваивания C/C++:

X = 5;	читается "X присвоить 5"
A = X - 2;	читается "A присвоить X-2"
X = X + 1;	читается "X присвоить X+1"

При выполнении оператора присваивания сначала вычисляется значение выражения, записанного справа от знака присваивания, а потом это значение заменяет собой прежнее значение переменной, указанной в левой части оператора. Таким образом, присваивание можно понимать как операцию "заменить на". Последний из приведенных операторов, например, увеличивает на единицу значение переменной X.

В отличие от других языков, в языке C/C++ присваивание не обязательно образует самостоятельный оператор. Присваивание можно использовать и внутри выражения наравне с другими операциями. Значение операции присваивания равно новому значению переменной.

Выражение языков программирования пишется подобно обычным алгебраическим выражениям, но, в отличие от них, не допускает многоэтажной записи дробей, индексов и степеней, поскольку вводится с клавиатуры ЭВМ. Поэтому дробная черта заменяется знаком деления '/', индексы пишутся в квадратных скобках после имени переменной. Знаком умножения служит звездочка '*'.

Перед решением задачи программа вводит исходные данные (значения переменных). *Ввод* - это пересылка (копирование) данных в оперативную

память из внешнего носителя информации (из файла): с клавиатуры, магнитного диска и других устройств ввода, и присваивание их переменным.

Ввод аналогичен присваиванию, только значение переменной не вычисляется, а читается с внешнего носителя.

Обрабатываемые данные (значения переменных и констант) во время выполнения программы вместе с ней находятся в оперативной памяти. Полученные результаты (значения переменных, констант и выражений) *выводятся* программой, т. е. копируются из оперативной памяти на внешний носитель (в файл): на экран, бумагу печатающего устройства (принтера), магнитный диск и другие устройства вывода.

В каждом языке программирования имеются свои операторы ввода-вывода. Например, при выполнении оператора ввода языка С

```
scanf ("%f %d %c", &x, &kol, &sim);
```

компьютер останавливается и ждет, пока с клавиатуры не будут введены три значения, а затем присваивает их, соответственно, переменным x, kol и sim, определения которых даны выше. Приведенный оператор ввода эквивалентен последовательности из трех операторов:

```
scanf ("%f", & x);
scanf ("%d", & kol);
scanf ("%c", & sim);
```

Записанные в кавычках форматы определяют вид элементов данных в файле, т.е. во входном тексте. Формат %f обозначает вещественное число (от слова float), %d - десятичное целое число (decimal), %c - один символ (char). Вводимые числа разделяются произвольным количеством пробелов и/или переходов на новую строку. Порядок форматов и переменных соответствует порядку ввода данных, т. е. их размещения в файле.

Знак & называется *амперсенд*. Перед именем переменной амперсенд обозначает операцию определения адреса этой переменной, указывающего, куда должны записываться вводимые значения. Он может отделяться от имени переменной пробелами.

Если, например, определенным выше переменным присвоены значения: t=11, sim='A', kol=10, то оператор вывода языка С

```
printf ("%d%c класс - %d часов", t, sim, kol+5);
```

выведет на экран дисплея заданный в кавычках текст, в который вместо форматов %d, %c и %d подставит, соответственно, значения заданных после текста выражений: t, sim и kol+5.

На экране появится текст:

11A класс - 15 часов

Рассмотренный оператор вызова функции scanf вводит данные из стандартного входного файла, а функция printf выводит данные в

стандартный выходной файл. Обычно стандартному входному и выходному файлам соответствуют клавиатура и экран, но можно переадресовать их на любой файл (см. главу 8).

Пример 1.1. Рассмотрим на языках C и C++ программу, которая вычисляет площадь прямоугольника - вводит два вещественных числа с клавиатуры дисплея, получает их произведение и выводит его на экран дисплея.

Программа 1.1 на C/C++ в стиле языка C:

```
/*        Программа 1.1. Площадь прямоугольника
  */
#include <stdio.h>              /* Вставка заголовочного файла    */
float    x, y, s;               /* Описание переменных            */
int main()                      /*  Заголовок программы           */
{  scanf ("%f %f", &x, &y);     /*  Ввод значений X и  Y          */
   s = x * y;                   /*  Получение результата          */
   printf ("%f", s);            /*  Вывод результата              */
   return 0;                    /* Успешное завершение            */
}
```

Программа 1.1 в стиле C++:

```
//        Программа 1.1. Площадь прямоугольника
#include <iostream.h>           // Вставка заголовочного файла
float    x, y, s;               // Описание переменных
int main()                      //  Заголовок программы
{  cin >> x >> y;               //  Ввод значений x и  y
   s = x * y;                   //  Получение результата
   cout  <<  s;                 //  Вывод результата s
   return 0;                    //  Успешное завершение
}
```

Пояснения к программе 1.1

1. Текст, ограниченный как бы скобками из символов /* и */, в языке C/C++ служит *комментарием* для пояснения программы. Он не влияет на выполнение программы, но значительно повышает ее наглядность: делает программу *самодокументированной*. Такой комментарий может занимать несколько строк. В языке C++ можно использовать еще и *строчный комментарий*, который начинается двумя символами косой черты // и продолжается до конца строки.

2. Команда препроцессора - *директива*

#include <stdio.h>

вставляет в программу текст из файла stdio.h, содержащего прототипы (заголовки) стандартных функций ввода/вывода данных.

Эта директива пишется с новой строки и размещается в начале программы, использующей стандартные функции ввода или вывода данных: scanf(), printf() и др.

3. Строки вида

```
int main ()
{
    . . .                    /*  Тело функции main */
}
```

составляют определение главной функции программы (main - главный). Программа может состоять из нескольких функций. Одной из них всегда должна быть функция main, с которой начинается выполнение программы.

4. Оператор **return** 0; предписывает завершить выполнение функции, считать ее значением число 0 и возвратиться к продолжению программы, запустившей эту функцию (return - возвратиться). Слово **int** в заголовке функции **int** main () говорит о том, что ее значением является целое число.

Из функции main произойдет возврат в операционную систему на запуск следующей программы. Значением функции main является *код завершения*: 0 обозначает успешное завершение, ненулевое значение – неудачную работу.

5. В языке C++ кроме форматного ввода-вывода с помощью стандартных функций scanf и printf возможен также *потоковый ввод-вывод*, использованный в варианте программы 1.1 на языке C++. В этом случае в программе необходима директива #**include** <iostream.h>. В операторе потокового ввода cin >> x >> y; и вывода cout << s; имена cin и cout обозначают, соответственно, стандартный входной и стандартный выходной поток данных (файл), а знаки >> и << играют роль “стрелок”, показывающих направление передачи данных.

При потоковом вводе-выводе программист не указывает вид данных в файле, используются стандартные форматы. Форматный ввод и вывод выглядит более громоздко, зато позволяет программисту самому управлять представлением информации в файле.

Порядок выполнения операций

В выражениях сначала выполняются операции в скобках, затем вне их. Порядок выполнения определяется *приоритетами* (старшинством) операций. В таблице 2.1 операции разделены горизонтальной чертой на группы. Внутри каждой группы приоритеты одинаковы. Группы расположены по убыванию приоритета: от старших операций к младшим.

В отсутствие скобок операции одной группы обычно выполняются слева направо. Поэтому в таблице отмечены только группы с обратным порядком выполнения операций - справа налево.

Для четырех операций (&& || ?: ,) первым вычисляется левый операнд. Для остальных операций порядок обработки операндов зависит от реализации.

Таблица. Операции в выражениях языка C/C++

имя() [] . ->	Вызов функции Получение элемента массива Получение элемента структуры (или объединения) Получение элемента структуры (или объединения), адресуемой указателем	
! ~ - ++ -- & * (тип) **Sizeof**	Логическое отрицание Битовое отрицание Изменение знака Увеличение на единицу Уменьшение на единицу Определение адреса Обращение по адресу (косвенная адресация) Преобразование типа Определение размера в байтах	Справа налево
* / %	Умножение Деление Остаток от деления	
+ -	Сложение Вычитание	
<< >>	Сдвиг влево Сдвиг вправо	
< <= > >=	Меньше Меньше или равно Больше Больше или равно	
== !=	Равно Не равно	
&	Битовая операция И	
^	Битовая операция исключающее ИЛИ (сложение по модулю 2, неэквивалентность	
|	Битовая операция ИЛИ	
&&	Логическая операция И	
||	Логическая операция ИЛИ	
? **:**	Условная операция	Справа налево
= *= /= %= += -= <<= >>= &= ^= |=	Присваивание	Справа налево
,	Операция запятая	

Пример. Рассмотрим оператор-выражение:

$$y = (x = 5) + (++x);$$

Если операнды сложения вычислять слева направо, то x станет равным 6, а y получит значение 5 + 6 = 11.

Если в другом компиляторе операнды вычисляются справа налево, то x получит значение 5, а y будет равным 5 + 1 + старое значение x.

Из этого примера видно, что для повышения мобильности программы не следует использовать переменную в том же выражении, где ей присваивается значение.

Операции присваивания

Знаки *= /= %= += -= <<= >>= &= ^= |= обозначают сочетание соответствующей операции (* / % + - << >> & ^ |) с присваиванием. Пусть op - такая операция. Тогда выражение вида x op= операнд эквивалентно

x = x op (операнд)

Например, выражение x += 5 эквивалентно x = x + 5, а выражение x *= y + 2 эквивалентно x = x * (y + 2), но не x = x * y + 2, как было бы, если бы операнд не был заключен в скобки.

Логические операции

В результате *операций сравнения (отношения)*: < <= > >= == != и логических операций: ! && || в качестве *логического значения* получается число 0, обозначающее ложь, или 1 - истина.

При этом операндом логических операций может быть не только число 0 или 1, но и любое целое число: нулевой операнд рассматривается как ложь, ненулевой операнд - как истина.

Логические операции языка С: ! - *отрицание (инверсия)*, && - логическая операция И (*логическое умножение, конъюнкция*) и || - логическая операция ИЛИ (*логическое сложение, дизъюнкция*) определяются следующей *таблицей истинности* (1 - истина, 0 - ложь).

x	y	~x !x	~y !y	x & y x && y	x | y x || y
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Логические операции позволяют из простых условий составлять сложные составные условия.

Логические операции имеют следующие свойства (доказываются перебором всех значений по таблице истинности).

1. *Закон отрицания* ! (! x) = x

2. *Коммутативность* x && y = y && x

x || y = y || x

3. *Ассоциативность* (x && y) && z = x && (y && z)

(x || y) || z = x || (y || z)

4. *Дистрибутивность* x && (y || z) = (x && y) || (x && z)

x || (y && z) = (x || y) && (x || z)

5. *Законы де Моргана* ! (x || y) = !x && !y

! (x && y) = !x || !y

Из законов де Моргана следует полезное правило: чтобы получить отрицание любого логического выражения (инвертировать его), необходимо заменить в нем все операции И и ИЛИ друг на друга, а каждый их операнд - на его отрицание.

Иногда легче бывает сформулировать условие, противоположное требуемому, а затем формально получить его отрицание.

Особенностью C/C++ по сравнению с другими языками является правило вычисления операндов логических операций: если по значению одного операнда уже можно определить результат операции, то значение другого операнда не вычисляется. Так, если левый операнд логической операции И - ложь (у логической ИЛИ - истина), результат получается ложь (истина) и правый операнд не вычисляется. Таким образом в некоторых случаях нарушается переместительный закон - свойство коммутативности.

Это важно и удобно, когда правый операнд имеет смысл только при определенных значениях левого операнда, например, в следующем операторе **if**:

```
if (x != 0 && y / x > 5)  . . .
```

При этой проверке деления на ноль никогда не будет, т. к. при x==0 левый операнд операции && получает значение "ложь" и правый операнд этой операции (выражение y/x > 5) не вычисляется.

В других языках перед выполнением операции И или ИЛИ обычно вычисляются оба операнда, и поэтому подобная запись была бы недопустимой из-за возможности деления на ноль. Приведенную проверку пришлось бы записать более громоздко:

```
if (x != 0)
   if (y / x > 5)  . . .
```

Примеры

1. Поскольку цифры кодируются последовательными целыми числами, условие "значение символьной переменой x является цифрой" на языке C запишется так:

x >= '0' && x <= '9'

Его отрицание (противоположное ему условие) "значение x не является цифрой" можно получить заменой && на ||, а каждого неравенства - на противоположное:

```
x < '0' || x > '9'
```

Ошибкой было бы написать первое условие, как в математике, в виде двойного неравенства '0' <= x <= '9'. Хотя формально это выражение не противоречит правилам C, оно бесполезно, так как тождественно равно 1, поскольку результат первого сравнения (1 или 0) всегда меньше кода цифры 9.

2. Если заглавные и строчные латинские буквы кодируются отдельными диапазонами последовательных целых чисел (как, например, в коде ASCII), то условие "значение символьной переменой x является латинской буквой" можно записать так:

```
(x>='A' && x<='Z') || (x>='a' && x<='z')
```

Скобки здесь поставлены для наглядности. Они не обязательны, поскольку операция && имеет более высокий приоритет, чем операция ||. Однако в подобных случаях скобки не только повышают наглядность, но и помогают избежать возможных ошибок.

Противоположное условие "значение x - не является латинской буквой" сразу записать труднее, чем получить формально по правилу де Моргана:

```
(x<'A' || x>'Z') && (x<'a' || x>'z')
```

Здесь скобки уже обязательны.

3. Год является високосным (содержит 366 дней вместо обычных 365), если он делится на 400 или делится на 4, но не делится на 100:

```
if (god % 400 == 0 || god % 4 == 0 && god % 100 != 0)
    . . .                          /* год високосный */
```

Битовые операции

Битовые операции (поразрядные логические операции и сдвиги) выполняются над целыми и символьными значениями, рассматриваемыми как последовательности битов (двоичных разрядов 1 или 0). В отличие от более сложных логических операций, битовая операция выполняется быстро - одна машинная команда. Обычно битовые операции бывают только в языках ассемблера, C является исключением.

Битовые операции: ~ - битовое отрицание, & - битовая операция И, | - битовая операция ИЛИ, ^ - битовое исключающее ИЛИ, подобны логическим операциям, но выполняются поразрядно: каждый разряд (бит) результата получается независимо от других разрядов применением операции к соответствующим битам операндов (1 - истина, 0 - ложь).

Операция *исключающее ИЛИ* (*неэквивалентность, сложение по модулю 2, XOR*) имеет следующую таблицу истинности:

x	y	x ^ y	Свойства операции ^:
0	0	0	x ^ y = y ^ x
1	0	1	x ^ x = 0, x ^ ~x = 1
0	1	1	x ^ 0 = x, x ^ 1 = ~x
1	1	0	

При выполнении *логического сдвига* влево:

X << n

или вправо:

X >> n

биты операнда X перемещаются на n разрядов в соответствующую сторону, n битов выходят за пределы операнда и таким образом теряются, освобождающиеся с противоположной стороны n битов заполняются нулями.

Пример 4

Пусть: A = 1 0 1 0 0 0 1 1
B = 1 1 0 0 1 0 0 1

Тогда: ~A = 0 1 0 1 1 1 0 0
A & B = 1 0 0 0 0 0 0 1
A | B = 1 1 1 0 1 0 1 1
A ^ B = 0 1 1 0 1 0 1 0
A << 2 = 1 0 0 0 1 1 0 0
B >> 3 = 0 0 0 1 1 0 0 1

Арифметический сдвиг числа на n битов влево соответствует его умножению на 2^n, а вправо - делению на 2^n (но выполняется в несколько раз быстрее):

$$x << n = x * 2^n \qquad x >> n = x / 2^n = x * 2^{-n}$$

Логический сдвиг вправо обеспечивает такое соответствие только для положительных и беззнаковых чисел.

При арифметическом сдвиге вправо чисел со знаком освобождающиеся слева биты заполняются не нулями, как при логическом сдвиге, а значением знакового разряда (крайнего левого бита). Отрицательные числа имеют в знаковом разряде 1, неотрицательные 0. Арифметический сдвиг влево не отличается от логического.

В некоторых реализациях языка С сдвиг вправо чисел со знаком может быть арифметическим. Сдвиг беззнаковых целых в любой реализации является логическим.

Поразрядные логические операции и сдвиги образуют универсальный набор операций, позволяющий выполнить любое преобразование битов.

Таким способом, в отличие от других языков высокого уровня, в С можно реализовать нестандартные операции, в том числе с частями ячеек памяти, что обычно бывает возможно лишь в языках ассемблера.

Поразрядное И (&) часто используют для обнуления некоторого множества битов и выделения таким образом остальных битов.

Пример 5. Выделение младших 7 битов значения x. Оператор

```
x = x & 0177;                    /* или  x &= 0177;
        */
                                 /*        1 111 111  в двоичной системе */
```

выделяет младшие 7 битов значения x (они не изменяются, т. к. логически умножаются на 1) , обнуляя остальные биты логическим умножением на 0.

Пример 6. Обнуление младшие 6 битов значения x. Оператор

```
x = x & ~0x3F;      /* или  x &= ~077;                              */
                    /* 0...0 0011 1111  в двоичной системе          */
                    /* 1...1 1100 0000  после инвертирования
                       */
```

обнуляет младшие 6 битов значения x (не изменяя остальные биты). Эта операция мобильна, т. к. не зависит от размера памяти для x в разных реализациях, в отличие, например, от оператора

```
x = x & 0xFFC0;          /* 1111 1111 1100 0000  в двоичной системе */
```

предполагающего, что x занимает два байта.

Подобным образом поразрядное ИЛИ (|) позволяет установить в единицу нужное множество битов, логически складываемых с 1, без изменения остальных битов, которые складываются с 0.

Поразрядное исключающее ИЛИ ^ (сложение по модулю два) позволяет инвертировать - получить отрицание нужного множества битов, складываемых по модулю два с 1, без изменения остальных битов, которые складываются с 0.

Пример 7. Запись 1 в младшие k битов целой переменной x без изменения остальных разрядов (где k - переменная). Решение имеет вид:

```
int k, x, y;
    . . .
y = ~(~0 << k);          /* k младших битов y = 1...1, остальные 0 */
x = x | y;               /* запись 1 в k младших битов x           */
```

Пример 8. Упаковка и распаковка данных. Пусть X < 32, Y < 64 - беззнаковые целые. Для значения X хватило бы 5 битов, для Y - 6 битов, и их можно хранить в одной переменной Z типа **int** или **unsigned**:

unsigned X, Y, Z;

а) Упаковка: X поместить в младшие 5 битов, а Y - в следующие 6 битов переменной Z, сохранив без изменения старшие биты Z:

Z = Z – Z % 2048 + 32*Y+X; или (что быстрее):

```
Z = Z & ~0x7FF;                 /* очистка (обнуление) места для X, Y */
Z = Z | X | Y << 5;             /* пересылка X, Y ==> Z               */
```

б) Распаковка: присвоить переменной X младшие 5 битов, а Y - следующие 6 битов переменной Z:

X = Z % 32; Y = Z % 2048 /32; или Y = Z /32 % 64; или (что быстрее):

```
X = Z & 037;                    /* выделение 5 младших битов 0…4 из Z */
Y = Z >> 5 & 077;               /* запись в Y битов 5 …10 из Z        */
```

Условная операция

Условное выражение

e1 ? e2 : e3

равно выражению e2, если выражение e1 != 0 (истинно), и равно выражению e3 в противном случае.

Пример 9. Максимум из двух значений:

```
z = (a > b) ? a : b;            /* z = max (a, b); */
```

Определение размера памяти

Значением операции

sizeof (тип) или **sizeof** выражение

является количество байтов, занимаемое значением данного типа или данного выражения **(**size of - размер от).

Пример 2.10. Размер памяти для данных типа **int**:

sizeof (**int**) равно 2 в большинстве реализаций.

Преобразования типа

Для каждого типа данных используется свой способ хранения значений в памяти. Иногда возникает необходимость преобразовать значение одного типа в соответствующее значение другого типа данных. В таких случаях в программу вставляются команды, обеспечивающие требуемое преобразование во время ее работы.

Если тип операнда не соответствует операции, он преобразуется к нужному для операции типу (требуется, чтобы его можно было так преобразовать).

Преобразование типа данных должно происходить также, если операнды некоторой операции имеют разные типы, например, один операнд арифметической операции - целый, а другой - вещественный. В этом случае обычно операнд более частного типа преобразуются к более общему типу другого операнда (целый - в вещественный).

Преобразование (приведение) типа может быть явным - по указанию программиста, или неявным (автоматическим). В языке С действуют следующие правила *неявного преобразования типа* транслятором, записанные неформально на псевдокоде (стрелка --> обозначает преобразование типа).

1. Перед арифметическими операциями:

[**unsigned**] **char** или **short** --> [**unsigned**] **int**;
float --> **double**;
if (один из операндов **double**)
другой --> **double** и результат будет **double**
else if (один из операндов [**unsigned**] **long**)
другой --> [**unsigned**] **long**, результат - [**unsigned**] **long**
else if (один из операндов **unsigned**)
другой --> **unsigned** и результат - **unsigned**
else; /* оба операнда и результат - **int** */

Таким образом, в языке С все операции с плавающей точкой автоматически выполняются с двойной точностью.

2. Перед присваиванием правая часть --> тип левой части:

char --> **int** с размножением знака или без него: дополнительные биты заполняются знаковым (старшим) битом или нулями;
int --> **char** с отбрасыванием старших разрядов;
float --> **int** с отбрасыванием дробной части;
double --> **float** с округлением.

3. При передаче параметров (аргументов) функции:

[**unsigned**] **char** или **short** --> [**unsigned**] **int**;
float --> **double**;
/* поэтому формальные параметры надо описывать как **int**
или double, даже если фактические - **char** или **float** */

if (есть описание типов аргументов)
преобразование, как перед присваиванием ;

Принудительное (явное) преобразование типа выполняется по указанию программиста с помощью *операции приведения типа*:

(тип) операнд

Значением этой операции является заданный операнд, преобразованный к указанному типу, как при присваивании.

Пример 11. Преобразование аргумента функции к нужному типу:

```
int n;
  . . .
sqrt ((double) n)                    /* параметр sqrt должен быть double */
```

Стандартные функции

В любом языке имеются *стандартные* (*встроенные*) *библиотечные функции*, используемые в программе без описания. В языке С богатый набор стандартных функций - свыше 200 (см. приложение 2). Наиболее важные из них рассматриваются в разных разделах пособия. Ниже приводится лишь краткая таблица математических функций языка С.

Математические функции (#**include** <math.h>)

int abs (**int** i)	| i | но: abs(-32768) = -32768
double fabs (**double** x)	| x |
double ceil (**double** x)	наименьшее целое . x (округление x в сторону увеличения)
double floor (**double** x)	наибольшее целое , x (округление x в сторону уменьшения)
double sin (**double** x)	sin x
double cos (**double** x)	cos x
double tan (**double** x)	tg x
double asin (**double** x)	arcsin x
double acos (**double** x)	arccos x
double atan (**double** x)	arctg x
double sqrt (**double** x)	корень квадратный из x
double exp (**double** x)	e^x
double log (**double** x)	ln x
double log10 (**double** x)	log10 x по основанию 10
double pow (**double** x, **double** y)	x^y $x^y = e^{y * \ln x} = \exp(y*\log(x))$