

## Базовые понятия

Программа состоит из последовательности инструкций, оформленных в строгом соответствии с правилами, составляющими *синтаксис данного языка*.

При создании программ могут быть допущены синтаксические или логические ошибки.

Синтаксические ошибки – это нарушение правил написания программы.

Логические ошибки разделяются на ошибки алгоритма и семантические ошибки.

Ошибка алгоритма – это несоответствие построенного алгоритма ходу получения результата поставленной задачи.

Семантическая ошибка – неправильное понимание смысла (семантики) операторов выбранного языка программирования.

Алфавит языка Си:

- прописные и строчные буквы латинского алфавита и знак подчеркивания (код 95);
- арабские цифры от 0 до 9;
- специальные символы, смысл и правила использования которых будем рассматривать по тексту;
- разделительные символы: пробел, символы табуляции (`\t`) и новой строки (`\n`).

Каждому из множества значений, определяемых одним байтом, в таблице кодов ставится в соответствие символ.

Символы с кодами от 0 до 127 (первая половина таблицы) одинаковы для всех компьютеров, коды 128 – 255 (вторая половина) могут отличаться и обычно используются для национального алфавита, коды 176 – 223 - символы псевдографики, а коды 240 – 255 – специальные знаки (можно посмотреть в приложении 1 [1]).

# *Лексемы*

Из символов алфавита формируются *лексемы* (элементарные конструкции) языка – минимальные значимые единицы текста в программе:

- идентификаторы (имена объектов);
- ключевые (зарезервированные) слова;
- знаки операций;
- константы;
- разделители (скобки, точка, запятая, точка с запятой, пробельные символы).

Границы лексем определяются другими лексемами, такими как разделители или знаки операций, а также комментариями.

Идентификатор (ID) – это имя программного объекта (константы, переменной, метки, типа, функции и т.д.).

В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания; первый символ ID – не цифра; пробелы и другие специальные символы внутри ID не допускаются.

В Си прописные и строчные буквы – различные символы. Идентификаторы Name, NAME, name – различные объекты.

Ключевые (резервированные) слова не могут быть использованы в качестве идентификаторов.

При именовании объектов следует придерживаться общепринятых соглашений:

- имена переменных и функций обычно пишутся строчными (малыми) буквами;
- имена типов пишутся с большой буквы;
- имена констант – большими буквами;
- идентификатор должен нести смысл, поясняющий назначение объекта в программе, например, `birth_date` – день рождения, `sum` – сумма;
- если ID состоит из нескольких слов, как, например, `birth_date`, то принято либо разделять слова символом подчеркивания, либо писать каждое следующее слово с большой буквы – `BirthDate`.

## *Комментарии*

Базовый элемент языка программирования – *комментарий* – не является лексемой.

Внутри комментария можно использовать любые допустимые на данном компьютере символы, т.к. компилятор их игнорирует.

В Си комментарии ограничиваются парами символов `/*` и `*/`, а в C++ введен вариант комментария, который начинается символами `//` и заканчивается символом перехода на новую строку (до конца текущей строки).

# Общая структура программы на языке Си

1. Директивы препроцессора

2. Область глобальных описаний:

- определение типов пользователя

- описание прототипов функций

- определение глобальных переменных

3. Функции

## ***Простейшая программа***

Рассмотрим кратко основные части структуры программ.

Перед компиляцией программа обрабатывается ***препроцессором***, который работает под управлением директив.

Препроцессорные ***директивы*** начинаются символом ***#***.

Препроцессор выполняет предварительную обработку программы, в основном это подключение (***include***) заголовочных файлов (обычных текстов) с объявлением используемых стандартных библиотечных функций.

Общий формат:

***#include <Имя\_файла.h >***

где ***h (header)*** – расширение заголовочных файлов, в средах *Microsoft Visual C – 20XX* годов некоторые файлы используются без расширения.

Если имя файла заключено в угловые скобки (< >), то поиск данного файла производится в стандартной папке, если в двойные кавычки (" "), то в текущей (рабочей) папке.

К наиболее часто используемым библиотекам относятся:

***stdio.h*** – содержит стандартные функции ввода-вывода данных;

***math.h* (*cmath*)** – математические функции;

***iostream.h* (*istream*)** – ввод-вывод в потоке;

***conio.h*** – функции для работы с консолью (клавиатура, дисплей).

Второе основное назначение препроцессора – обработка макроопределений (замещений).

Макроподстановка ***определить*** (***define***) имеет общий вид:

***#define ID строка***

Например:

***#define PI 3.1415927***

– в ходе препроцессорной обработки программы идентификатор-константа *PI* везде будет заменяться значением 3.1415927

Пример простейшей программы:

```
#include <stdio.h>
```

```
void main(void)          // Заголовок функции
```

```
{                          // Начало функции
```

```
printf (" 10 is the best mark ! ");
```

```
}                          // Конец функции
```

Отличительный признак функции – скобки ( ) после ее имени, в которых заключается список параметров.

Если параметров нет, указывают атрибут **void** (пустой, отсутствующий), атрибут **void** можно не писать.

Перед функцией указывается тип возвращаемого результата, если результата нет – **void**.

В фигурных скобках записываются выполняемые инструкции (код функции), каждая из которых оканчивается символом «;». В нашем примере только стандартная функция **printf** – вывод указанной фразы на экран (из **stdio.h**).

Используя потоковый вывод, этот пример можно записать следующим образом:

```
#include <iostream.h>    //для iostream  
// используем using namespace std;  
void main ()  
{  
    cout << " 10 is the best mark ! " << endl;  
}
```

***cout* (class *output*)** – вывод данных в потоке с помощью операции побитового сдвига **<<**;

***endl* (*end line*)** – перевод курсора на новую строку.

При создании проекта в консольном режиме формируется шаблон основного файла, который может иметь следующий вид

```
int main ( int argc, char* argv[] )  
{  
    return 0;  
}
```

Перед функцией указан тип (**int** – целочисленный) возвращаемого оператором **return 0**; результата (0 – выход из функции без ошибки) и стандартные параметры: **int argc, char\* argv[]**

## ***Типы данных***

Данные разделяются на две категории: простые (скалярные) и сложные (составные) типы данных.

Тип данных определяет:

- внутреннее представление в памяти;
- диапазон допустимых значений;
- набор допустимых операций.

Базовые типы данных: символьный – ***char*** (*character*), целый – ***int*** (*integer*), вещественный обычной точности – ***float***, вещественный удвоенной точности – ***double***.

Данные целого типа могут быть короткими – ***short***, длинными – ***long***, со знаком – ***signed*** и беззнаковыми – ***unsigned***.

Атрибут ***unsigned*** может использоваться для типа ***char***.

Атрибут ***long*** так же может использоваться для типа ***double***.

Тип ***void*** указывает отсутствие типа.

Сложные типы данных: структуры – ***struct***, объединения – ***union***, перечисления – ***enum***.

**Массивы** – составные данные базовых, или объявленных ранее пользовательских типов.

## Диапазон и объем памяти данных

Тип	Объем памяти (байт)	Диапазон значений
<i>char</i>	1	-128 ... 127
<i>int</i>	4	-32768 ... 32767
<i>short (int)</i>	2	-32768 ... 32767
<i>long (int)</i>	4	-2147483648 ... 2147483647
<i>unsigned int</i>	4	0 ... 65535
<i>unsigned long</i>	4	0 ... 4294967295
<i>float</i>	4	$3,14 \cdot 10^{-38}$ ... $3,14 \cdot 10^{38}$
<i>double</i>	8	$1,7 \cdot 10^{-308}$ ... $1,7 \cdot 10^{308}$

## ***Декларация объектов***

Все объекты программы (кроме самоопределенных констант) необходимо декларировать, т.е. объявить компилятору об их свойствах.

Общий формат объявления:

***Атрибуты Список;***

Элементы ***Списка*** разделяются запятыми, а ***Атрибуты*** – разделителями (хотя бы одним пробелом), например:

*long int i, j, k;*

или (атрибут ***int*** можно не писать)

*long i, j, k;*

Атрибуты могут быть следующими:

**Класс памяти** – определяет способ размещения в памяти (статическая, динамическая), область видимости и время жизни (по умолчанию – **auto**), данные атрибуты будут рассмотрены позже;

**Тип** – базовый тип, или созданный ранее тип Пользователя (по умолчанию – тип **int**).

Класс памяти и тип – атрибуты необязательные и при отсутствии одного из них (но не обеих) устанавливаются по умолчанию.

Примеры декларации простых объектов:

***char ss; int i, j, k; double a, b, x;***

## ***Данные целого типа (integer)***

Тип ***int*** – целое число, соответствующее обычному виду целых чисел.

Квалификаторы ***short*** и ***long*** указывают на различные размеры и определяют объем памяти, выделяемый под них, например:

***short*** *x*;

***long*** *x*;

***unsigned*** *x* = 8;

– декларация с инициализацией числом 8;

атрибут ***int*** в этих случаях может быть опущен.

Для определения константных значений используется атрибут ***const***, указывающий запрет изменения введенной величины в программе, например

**const N = 20;**      или      **const double PI = 3.1415926;**

Атрибуты ***signed*** и ***unsigned*** показывают, как интерпретируется старший бит – как знак или как часть числа:

<b><i>int</i></b>	Знак	Значение числа					
	31	30	29	...	2	1	0

– Номера бит

<b><i>unsigned</i></b>	Значение числа						
	31	30	...	2	1	0	

## ***Данные символьного типа (char)***

Любой символ в памяти занимает один байт и соответствует конкретному коду.

Для персональных компьютеров (ПК) наиболее распространена *ASCII (American Standard Code for Information Interchange)* таблица кодов.

Данные типа ***char*** рассматриваются компилятором как целые, поэтому можно использовать величины со знаком *signed char* (по умолчанию) – символы с кодами от  $-128$  до  $+127$  и *unsigned char* – беззнаковые символы с кодами от  $0$  до  $255$ .

Примеры:

```
char res, simv1, simv2;
```

```
char sim = 's';
```

– декларация с инициализацией символом *s*.

## ***Данные вещественного типа (float, double)***

Внутреннее представление этих данных состоит из мантиссы и порядка, т.е.

$$\langle \text{Мантисса} \rangle * 10^{\langle \text{Порядок} \rangle}$$

Характеристика данных:

Тип	Мантисса	Порядок
<b><i>float</i></b> (4 байта)	7 цифр после запятой	± 38
<b><i>double</i></b> (8 байт)	15	± 308
<b><i>long double</i></b> (10 байт)	19	± 4932

Переменная типа ***double*** формально соответствует типу ***long float***.

# КОНСТАНТЫ

Константами называют величины, которые не изменяют значений во время выполнения программы.

***Константа – это неадресуемая величина*** и, хотя она хранится в памяти, определить ее адрес невозможно!

Константы нельзя использовать в левой части операции присваивания.

В языке Си константами являются:

- самоопределенные константы;
- имена (идентификаторы) массивов и функций;
- элементы перечислений.

## Целочисленные константы

**Десятичные константы** – это набор цифр 0...9, **первая из которых не 0** (со знаком или без него).

Для длинных целых констант указывается признак  $L(l)$  –  $273L$  ( $273l$ ). Константа, которая слишком длинна для типа *int*, рассматривается как *long int*.

**Восьмеричные константы** – это набор цифр от 0 до 7, **первая из которых 0**, например:  $020 = 16$  – десятичное.

**Шестнадцатеричные константы** – набор цифр от 0 до 9 и букв от *A* до *F* (*a...f*), начинающаяся символами  $0X$  ( $0x$ ), например:  $0X1F$  ( $0x1f$ ) = 31 – десятичное.

Восьмеричные и шестнадцатеричные константы также могут быть *long*, например,  $020L$  или  $0X20L$ .

### Примеры целочисленных констант:

1992	777	1000L	– десятичные;
0777	00033	01L	– восьмеричные;
0x123	0X00ff	0xb8000L	– шестнадцатеричные.

## **Константы вещественного типа**

Данные константы могут иметь две формы:

1) с фиксированной точкой:

**$\pm n.m$**  ( $n, m$  – целая и дробная части числа);

2) с плавающей точкой (экспоненциальная форма)  
представляется в виде мантииссы и порядка:

**$\pm n.mE\pm p$**

где  **$\pm n.m$**  – мантиисса ( $n, m$  – целая и дробная части числа),  $E$  (или  $e$ ) – знак экспоненты,  $p$  – порядок. Например, число  $1,25 \cdot 10^{-8}$  можно записать  $12.5E-9$ ,  $1.25E-8$  или  $0.125E-7$

Примеры:

1.0    -3.125    100E-10    -0.12537e+5

Пробелы внутри чисел не допускаются. Для разделения целой и дробной части используется точка. Дробную или целую часть можно опустить, но не обе сразу, например,

1. (или 1.0)    .5 (или 0.5)

## **Символьные константы**

*Символьная константа* – это любой символ, заключенный в одинарные кавычки, например: 'a' .

Так же используются специальные управляющие символы (первый символ обратный слеш), например:

**\n** – новая строка;

**\t** – горизонтальная табуляция;

**\0** – нулевой символ.

Простые символы могут вводиться с клавиатуры, а специальные – представляются в исходном тексте парами символов, например: \\ – обратный слеш; \' – апостроф; \" – кавычки.

Примеры символьных констант:

**'A'**    **'9'**    **'\$'**    **'\n'**    **'\t'**

## **Строковые константы**

*Строковая константа* – набор символов, заключенных в кавычки ("). Кавычки не являются частью строки, а служат для ее ограничения. Строка в языке Си представляет собой массив символов. Внутреннее представление константы "1234ABC":

'1' '2' '3' '4' 'A' 'B' 'C' '\0'

В конец строковой константы автоматически добавляется нулевой символ '\0', называемый **нуль-терминатор**, который на печать не выводится и является **признаком окончания строки**.

Примеры строковых констант:

"Summa"      "\n \t Result = \n"      " \" EXIT \" "

Длинную строковую константу можно разбить на несколько с помощью обратного слеша (\). Например:

"Вы учитесь в Белорусском государственном \n университетe информатики и радиоэлектроники"

Такая запись будет представлена как единое целое.

## ***Операции, выражения***

Выражения используются для вычисления значений определенного типа и состоят из операндов, операций и скобок. Операнд может быть, в свою очередь, выражением (константой или переменной).

Операции задают выполняемые действия.

В языке Си используются четыре первичных операции с наивысшим приоритетом:

- операция «( )» при обращении к функции;
- операция индексации «[ ]» при обращении к элементам массива;
- операция доступа к полям структур и объединений при помощи идентификаторов «.» (точка);
- операция доступа к полям структур и объединений при помощи указателей «->» (стрелка).

Операции делятся на **унарные, бинарные и тернарные** – по количеству операндов, и выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки.

Унарные операции имеют больший приоритет над бинарными.

Большинство операций выполняются слева направо. Унарные операции, операции присваивания и условная операция (?:) выполняются справа налево.

## ***Арифметические операции***

Бинарные арифметические операции:

+ (сложение); – (вычитание); / (деление, **для *int* операндов – с отбрасыванием остатка**); \* (умножение); % (остаток от деления **целочисленных операндов** со знаком первого операнда – деление «по модулю»).

Операндами арифметических операций могут быть любые объекты, имеющие допустимые типы (константы, переменные, выражения, функции, элементы массивов).

Унарные операции  $+$ ,  $-$  (знак) определены только для числовых операндов.

Порядок выполнения операций:

- 1) выражения в круглых скобках;
- 2) вычисление функций (стандартные функции и функции пользователя);
- 3) операции  $*$ ,  $/$ ,  $\%$ ;
- 4) операции  $-$ ,  $+$ .

При записи сложных выражений нужно использовать общепринятые математические правила:

$$x + y \cdot z - \frac{a}{b + c} \longleftrightarrow x + y * z - a / (b + c)$$

Т.е. использовать круглые скобки.

Единственной исключительной ситуацией при выполнении арифметических операций является деление на ноль, другие ситуации (переполнение, исчезновение порядка или потеря значимости) компилятором игнорируются.

## ***Операция присваивания***

Общий формат:

***Операнд\_1 = Операнд\_2 ;***

*Операндом\_1 (Левое–значение – Left-Value)* может быть только *адресное значение*, т.е. именованная, либо косвенно адресуемая указателем переменная.

*Операндом\_2 (Правое–значение – Right-Value)* может быть константа, переменная или любое выражение, составленное в соответствии с синтаксисом.

Операция выполняется ***справа налево***.

Тип результата определяется типом левого операнда.

***Приведите примеры «хитрых» ситуаций!***

Присваивание может включать несколько операций, изменяя значения нескольких операндов, например:

$$i = j = k = 0; \quad \leftrightarrow \quad k = 0, j = k, i = j;$$

$$x = i + (y = 3) - (z = 0); \quad \leftrightarrow \quad y = 3, z = 0, x = i + 3 - 0;$$

### ***Примеры недопустимых выражений:***

– присваивание константе:  $2 = x + y;$

– присваивание функции:  $getch() = i;$

– присваивание результату операции:  $(i + 1) = 2 + y;$

## ***Сокращенные формы операции присваивания***

В языке Си используются два вида сокращенной записи операции присваивания:

1) вместо записи  $v = v \# e$ ;

где  $\#$  – любая арифметическая операция, используется запись  $v \# = e$ ;

Например,  $s = s + 2$ ;  $\leftrightarrow s += 2$ ;

***знаки операций записываются без пробелов;***

2) вместо записи  $x = x \# 1$ ;

где  $\#$  – символы операций инкремента (+1), либо декремента (-1), используются записи:

префиксная  $##x$ ;  $++x$ ;  $--x$ ;

или

постфиксная  $x##$ ;  $x++$ ;  $x--$ ;

Операции инкремента (++) и декремента (--) – **унарные**.

Если эти операции используются отдельно, то различий между постфиксной и префиксной формами нет.

Если же они используются в выражении, то

1) в префиксной форме (##x) сначала значение **x** изменится на **1**, а затем **x** будет использовано в выражении;

2) в постфиксной форме (x##) сначала значение **x** используется в выражении, а затем изменяется на **1**.

Например,

для значение `int x, a = 2, b = 5;`

вариант 1: `x = ++a * --b;`

1) `++a = 3`, 2) `--b = 4`, 3) `3 * 4 = 12`, 4) `x = 12;`

вариант 2: `x = a++ * b--;`

1) `2 * 5 = 10`, 2) `x = 10`, 3) `a++ = 3`, 4) `b-- = 4;`

## **Преобразование типов**

Если операнды арифметических операций имеют один тип, то результат операции будет иметь такой же тип.

Если в операциях участвуют операнды различных типов, то они преобразуются к «**большему**» типу (в смысле объема памяти), т.е. неявные преобразования идут от «**меньших**» объектов к «**большим**»:

- значения *char* и *short* преобразуются в *int*;
- если один операнд *double*, то и другой преобразуется в *double*;
- если один операнд *long*, то и другой *long*.

**Внимание!** Результат операции  $1 / 3$  имеет значение **НОЛЬ**, т.к. и **1** и **3** имеют тип *int* !!!

Чтобы избежать такого рода ошибок необходимо явно изменить тип хотя бы одного операнда, т.е. записывать, например:  $1. / 3$ , т.к. **1.** **вещественная константа** !!!

Типы ***char*** и ***int*** могут свободно использоваться в арифметических выражениях.

При присваивании значение правой части преобразуется к типу левой, который и является типом результата. Поэтому необходимо быть внимательным, т.к. при некорректной записи могут возникнуть ошибки.

При преобразовании ***int*** в ***char*** старший байт просто отбрасывается.

Преобразование ***double (float)*** в ***int*** приведет к отбрасыванию дробной части.

Тип ***double*** преобразуется во ***float*** округлением.

Длинное целое преобразуется в ***int*** и ***char*** посредством отбрасывания лишних битов более высокого порядка.

## ***Операция явного приведения типа***

Формат операции:

***(Тип) Выражение;***

ее результат – значение ***Выражения***, преобразованное к заданному ***Типу***.

Рекомендуется использовать эту операцию в исключительных случаях, например:

*double* x;

*int* n = 6, k = 4, m = 3;

$x = (n + k) / m; \quad \rightarrow \quad x = 3;$

$x = (\textit{double})(n + k) / m; \rightarrow x = 3.333333.$

## **Стандартные библиотечные файлы**

В любой программе кроме инструкций используются стандартные функции, входящие в библиотеку языка Си, которые облегчают создание программ.

В стандартных библиотечных файлах описаны прототипы функций, макросы, глобальные константы. Это заголовочные файлы, которые хранятся обычно в папке ***include*** и подключаются на этапе предпроцессорной обработки.

Математические функции находятся в файле ***math.h*** (***cmath***). В файле ***math.h*** среды ***C++ Builder*** описаны макроконстанты, такие как, например  $\pi$ , это ***M\_PI*** (и другие).

У большинства математических функций аргументы и возвращаемый результат имеют тип ***double***. Аргументы тригонометрических функций должны быть заданы в радианах ( $2\pi$  радиан =  $360^\circ$ ).

Математическая функция	Имя функции
$\sqrt{x}$	sqrt(x)
x	fabs(x)      или      abs(x)
$e^x$	exp(x)
$x^y$	pow(x,y)
ln(x)	log(x)
$\lg_{10}(x)$	log10(x)
sin(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsin(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
arctg(x / y)	atan2(x, y)
sh(x)=0.5 (e <sup>x</sup> -e <sup>-x</sup> )	sinh(x)
ch(x)=0.5 (e <sup>x</sup> +e <sup>-x</sup> )	cosh(x)
tgh(x)	tanh(x)
Остаток от деления x на y	fmod(x,y)
Наименьшее целое >=x	ceil(x)
Наибольшее целое <=x	floor(x)

Из библиотеки ***conio.h*** используется функция

***getch( );*** или ***\_getch( );***

Которая выполняет ожидание нажатия любой клавиши, ее результат – **int** код, или **char** символ нажатой клавиши.

Вместо нее можно пользоваться функцией

***system ("pause");***

(может понадобится библиотека ***windows.h***).

## ***Потоковый ввод-вывод***

Для ввода-вывода в C++ используются два класса: ***cin*** (класс *ввода*), ***cout*** (класс *вывода*). Для их работы необходимо подключить файл ***iostream.h*** (***iostream***).

Стандартный поток вывода ***cout*** по умолчанию связан со стандартным потоком ***stdout*** (дисплей монитора), а ввода ***cin*** – со стандартным потоком ***stdin*** (клавиатура).

Вывод на экран (*помещение в поток <<*):

***cout << Имя-Объекта-Вывода;***

Ввод с клавиатуры (*извлечение из потока >>*):

***cin >> Имя-Переменной;***

Пример:

```
#include < iostream.h >          (iostream)
```

```
// using namespace std;
```

```
void main ()
```

```
{
```

```
    int i, j, k;
```

```
    cout << " Input i, j ";
```

```
    cin >> i >> j ;
```

```
    k = i + j ;
```

```
    cout << " Sum i + j = " << k << endl;
```

```
    /* end line – переход на новую строку и  
    очистка буферов ввода-вывода */
```

```
}
```

## ***Использование манипуляторов***

Манипуляторы – специальные функции, возвращающие модифицированные данные потока. Для их использования необходимо подключить заголовочный файл ***iomanip.h***.

Рассмотрим некоторые из них.

***setw (n)*** – устанавливает ширину поля, т.е. ***n*** позиций для вывода следующего за ним объекта.

***setprecision (n)*** – устанавливает ***n*** значащих цифр после запятой с учетом точки или без нее (в зависимости от системы программирования).

## ***Функции вывода данных на экран***

Стандартные функции ввода/вывода описаны в файле ***stdio.h***.

Для вывода на экран чаще всего используются: ***printf*** (***printf\_s***) и ***puts***.

Формат функции форматного вывода на экран:

***printf ( Управляющая Строка , Список Вывода );***

В *Управляющей Строке*, заключенной в кавычки, записывают:

- Поясняющий текст (комментарии);
- Список модификаторов форматов, определяющих способ вывода объектов (признак – символ %);
- Специальные управляющие символы.

В **Списке Вывода** указываются выводимые объекты: константы, переменные, выражения (вычисляемые перед выводом).

Количество и порядок форматов должен совпадать с количеством и порядком следования печатаемых объектов.

Так как функция ***printf*** (***printf\_s***) выполняет вывод данных в соответствии с указанными форматами, формат может использоваться для преобразования типов выводимых объектов.

Если признака модификации (%) нет, то вся информация выводится как комментарии (текст).

Основные модификаторы форматов:

**%d** – десятичное целое число (*int*), можно **%i**

**%c** – один символ (*char*)

**%s** – строка символов (*string*)

**%f** – вещественное типа *float*

**%ld** – длинное целое (*long int*)

**%lf** – вещественное типа *double* (*long float*)

При выводе используются специальные символы:

`\n` – переход на новую строку;

`\t` – горизонтальная табуляция;

`\l` – вывод обратного слеша;

`\'` – вывод апострофа;

`\"` – вывод кавычки;

`\b` – шаг назад;

`\r` – возврат каретки;

`\v` – вертикальная табуляция.

В модификаторах формата функции ***printf*** после символа % можно указывать параметры поля вывода, например,

***%5d*** – ширина поля 5 символов для ***int***,

***%8.4lf*** – ширина поля 8 символов с точкой и знаком и 4 цифры после запятой для ***double***.

Если указанных в ширине позиций для вывода числа не хватает, то происходит автоматическое расширение.

Можно использовать функцию ***printf*** для нахождения кода ASCII любого символа, например:

```
printf ( " %c – %d \n" , 'a', 'a');
```

Функция

```
puts ( Имя-Строки );
```

выводит на экран **ОДНУ** строку, автоматически добавляя к ней символ перехода на начало новой строки (\n).

Аналогом такой функции будет:

```
printf ( " %s \n " , Имя-Строки);
```

## **Функции ввода информации**

Форматированный ввод с клавиатуры:

***scanf*** (*Управляющая Строка* , *Список Ввода*);

в ***Управляющей строке*** указываются ***только*** модификаторы форматов, количество и порядок которых должны совпадать с количеством и порядком вводимых объектов, тип преобразуется в соответствии с модификаторами.

***Список Ввода*** – адреса переменных (через запятую), т.е. для ввода перед именем **СКАЛЯРНОЙ** переменной указывается символ **&** – унарная операция «***взять адрес***».

При вводе строковой переменной операцию **&** не используем, т.к. строка – это массив символов, а ***имя массива*** – это адрес его первого элемента. Например:

```
int k;                // Курс
double st;           // Стипендия
char name[20];       // Имя
printf (" Input курс, stipendia, name \n ");
scanf ("%d%lf%s", &k, &st, name);
```

Вводить данные с клавиатуры можно как в одну строку, разделяя данные хотя бы одним пробелом, так и в столбец, нажимая после каждого значения клавишу *Enter*.

В функции ***scanf*** (***scanf\_s***) используется тот же набор модифи-каторов форматов, что и в ***printf***.

***Внимание!*** Функцией ***scanf*** по формату ***%s*** строка вводится ТОЛЬКО до первого пробела (аналогично и в ***cin***).

Для ввода текста, состоящего из слов, разделенных пробелами, используется функция:

***gets*** (*Имя-Строковой-Переменной*);

При запуске программы автоматически открываются стандартные потоки ввода – ***stdin*** (по умолчанию связан с клавиатурой) и вывода – ***stdout*** (экран монитора).

***Внимание!*** Ввод данных функцией ***gets*** выполняется с использованием потока ***stdin***. Если указанная функция не выполняет своих действий (проскакивает), перед ее использованием необходимо очистить поток (буфер) ввода с помощью функции (библиотека ***stdlib.h***)

***fflush (stdin);***

Пример использования функции ***getch***:

```
char s;
```

```
s = getch();
```

```
cout << "Character = " << s << endl;
```

```
cout << "Code = " << (int) s << endl;
```

переменная `s` – символ нажатой клавиши, а `(int)s` – код этого символа.