

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, новой строки и новой строки.

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 [1,2]).

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 (зарезервированные) 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 С++ введен вариант комментария, который начинается символами `//` и заканчивается символом перехода на новую строку.

Общая структура программы на языке Си

1. Директивы препроцессора

2. Определение типов пользователя

(typedef)

3. Описание прототипов функций

4. Определение глобальных переменных

5. Функции

Простейшая программа

Рассмотрим кратко основные части структуры программ.

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

Препроцессорные ***директивы*** начинаются символом **#**, за которым следует ее наименование, указывающее выполняемое действие.

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

Общий формат:

#include <Имя_файла.h >

где ***h*** – расширение заголовочных файлов.

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

К наиболее часто используемым библиотекам относятся:

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

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

math.h – математические функции;

iostream.h – ввод-вывод в потоке;

Второе основное назначение препроцессора – обработка макроопределений.

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

#define ID строка

Например:

#define PI 3.1415927

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

Пример простейшей программы:

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

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

```
{                          // Начало функции
```

```
printf (“ Высшая оценка знаний – 10 ! ”);
```

```
}                          // Конец функции
```

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

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

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

В фигурных скобках записывается текст (код) функции, т. е. набор выполняемых ею инструкций, каждая из которых оканчивается символом «;». В нашем примере только функция **printf** – вывод указанной фразы на экран.

В предыдущем примере для вывода использована стандартная функция **printf**, описанная в файле **stdio.h**.

Используя потоковый вывод, этот пример можно записать следующим образом:

```
#include <iostream.h>  
void main ()  
{  
    cout << “ 10 is the best mark ! ” << endl;  
}
```

Если параметров нет, атрибут **void** можно не писать.

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

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

Типы данных

Данные в языке Си 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*** может использоваться для типа ***double***.

Тип ***void*** указывает его отсутствие.

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

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 присутствии.

Общий формат объявления:

Атрибуты Список-Объектов;

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

long int 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 одновременно) 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, обычно соответствующее естественному размеру 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 можно использовать 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:

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

– Номера бит

<i>unsigned</i>	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 (см. Приложение 1).

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 символьной переменной с инициализацией СИМВОЛОМ *s*.

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

Характеристика данных:

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

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

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

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

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*.

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 размещаются в памяти по формату *double* и 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, $1,25 \cdot 10^{-8}$ можно записать $1.25E-8$ 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 символ, заключенный в одинарные кавычки (апострофы), например: 'a'.

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

\n – новая строка;

\t – горизонтальная табуляция;

\0 – нулевой символ.

При присваивании символьной переменной они должны быть заключены в апострофы.

Текстовые символы непосредственно вводятся с клавиатуры, а специальные и управляющие – представляются в исходном тексте парами символов, например: \\ – обратный слеш; \' – апостроф; \" – кавычки.

Примеры символьных констант: 'A' '9' '\$' '\n'

Строковые константы

Строковая константа – символы, заключенные в кавычки ("). Кавычки не являются частью строки, а служат только 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 компилятор автоматически добавляет 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\n университете информатики и радиоэлектроники"

Компилятор воспримет такую запись, 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 действия, которые необходимо выполнить.

В языке Си используются четыре первичных операции:

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

Операции делятся на унарные, бинарные и тернарные – по количеству участвующих 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 традиционных арифметических операций (+, −, *, /) могут быть любые объекты, имеющие допустимые типы (константы, переменные, функции, элементы массивов, арифметические выражения).

Унарные операции +,− (знак) определены только для числовых операндов, при этом «+» носит только информационный характер, «−» меняет знак операнда на противоположный (не адресная операция).

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 (L–значение – Left-Value) может быть только *адресное выражение*, т.е. именованная, либо косвенно адресуемая указателем переменная.

Операндом_2 (R–значение – Right-Value) может быть константа, переменная и любое выражение, составленное в соответствии с синтаксисом языка Си.

Операция выполняется справа налево.

Тип результата определяется типом левого операнда.

Приведите примеры «хитрых» ситуаций!

Присваивание значения в языке Си рассматривается как выражение, имеющее значение левого операнда после присваивания.

Поэтому присваивание может включать несколько операций, изменяя значения нескольких операндов, например:

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

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

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; рекомендуется использовать запись $v \# = e$;

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

знаки операций записываются без пробелов;

2) вместо записи $x = x \# 1$;

где $\#$ – символы, обозначающие операцию инкремента (+1), либо декремента (-1), рекомендуется использовать запись:

префиксную $\#\#x$; $++x$; $--x$;

или

постфиксную $x\#\#$; $x++$; $x--$;

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,

```
int x, a = 2, b = 5;
```

```
1) x = ++a * --b;
```

```
    a = 3, b = 4, x = 12;
```

```
2) x = a++ * b--;
```

```
    x = 10, a = 3, 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 «большемому» 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 преобразуется в *long*.

Внимание! Результат операции **1 / 3** 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 смешиваться в арифметических выражениях. Переменные *char* автоматически преобразуются в *int*.

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 неконтролируемые ошибки.

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

Если *double x; int i;*

то *i = x;* приведет к преобразованию *double* в *int* с отбрасыванием дробной части.

Тип *double* преобразуется во *float* округлением.

Длинное целое преобразуется в целое 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 с расширением **.h*, которые хранятся в папке *include* и подключаются на этапе предпроцессорной обработки.

Математические функции языка Си декларированы в файле ***math.h*** (некоторые в ***stdlib.h***). В файле ***math.h*** описаны макроконстанты, такие как, например π , это ***M_PI*** (и другие).

У большинства математических функций аргументы и возвращаемый результат имеют тип *double*. Аргументы тригонометрических функций должны быть заданы в радианах (2π радиан = 360^0).

Математическая функция	Имя функции
\sqrt{x}	sqrt(x)
x	fabs(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)
$\text{sh}(x)=0.5 (e^x - e^{-x})$	sinh(x)
$\text{ch}(x)=0.5 (e^x + e^{-x})$	cosh(x)
tgh(x)	tanh(x)
Остаток от деления x на y	fmod(x,y)
Наименьшее целое $\geq x$	ceil(x)
Наибольшее целое $\leq x$	floor(x)

Из библиотеки ***conio.h*** при создании
КОНСОЛЬНЫХ приложений мы будем
пользоваться только функцией

getch();

Которая выполняет ожидание нажатия
любой клавиши;

ее результат – код нажатой клавиши.

Потоковый ввод-вывод

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

Стандартный поток вывода ***cout*** по умолчанию связан со стандартным устройством вывода ***stdout*** (дисплей монитора), а ввода ***cin*** – со стандартным устройством ввода ***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 >
```

```
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 */
```

```
}
```

Функции вывода данных на дисплей

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

Для вывода на экран чаще всего используются: ***printf*** 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: переменные, константы, выражения (вычисляемые перед выводом).

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

Так как функция ***printf*** 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.

Основные модификаторы формата:

%d – десятичное целое число (*int*);

%c – один символ (*char*);

%s – строка символов;

%f – вещественное типа *float*;

%ld – длинное целое (*long int*);

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

Управляют выводом специальные последовательности символов:

\n – новая строка;

\t – горизонтальная табуляция;

\b – шаг назад;

\r – возврат каретки;

\v – вертикальная табуляция;

**** – обратная косая;

\' – апостроф;

\" – кавычки;

\0 – нулевой символ (пусто).

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

`%5d` – для *int*,

`%8.4lf` – для *double* (4 цифры после запятой для поля, шириной 8 символов).

Если указанных позиций для вывода целой части числа не хватает, то происходит автоматическое расширение.

Можно использовать функцию *printf* для нахождения кода ASCII некоторого символа:

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

Функция

puts (*Имя-Строки*);

выводит на экран 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 переменной указывается символ **&** – операция «***взять адрес***».

Если вводим значение строковой переменной, то символ **&** не используем, т.к. строка – это массив символов, а *имя* массива – это адрес его первого элемента. Например:

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


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

В функции *scanf* используется тот же набор основных модификаторов форматов, что и *printf*.

Внимание! Функцией *scanf* по формату *%s* строка вводится только до первого пробела.

Для ввода фраз, состоящих из слов, разделенных пробелами, используется функция:

gets (*Имя-Строковой-Переменной*);

Символы вводятся при помощи функции ***getch()***.

Простой ее вызов организует задержку выполнения программы до нажатия любой клавиши.

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.

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

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

fflush (*stdin*);