

## Лабораторная работа 2

### «Организация ввода/вывода данных средствами языка C и C++»

#### Язык C

**printf( )** – функция форматного вывода в стандартный поток stdout

**scanf( )** – функция форматного ввода

Данные функции определены в библиотеке stdio.h.

Функции используют управляющую строку и список аргументов.

#### **Обращение к функции вывода**

**printf(форматная строка[,аргумент, ...]);**

Здесь квадратные скобки означают, что функция допускает переменное число необязательных аргументов, но параметр форматная строка должен присутствовать обязательно. Если список аргументов присутствует, то в форматной строке для каждого из них должна быть указана спецификация формата вывода.

**аргумент 1, аргумент 2 и т.д.** - это печатаемые параметры, которые могут быть переменными, константами или даже выражениями, вычисляемыми перед выводом на печать.

Строка описания формата может состоять из обычных символов, специальных символов (escape-последовательностей), спецификаций полей формата вывода аргументов и прочитывается слева направо. Когда встречается первая спецификация формата, значение первого аргумента после форматной строки преобразуется и выводится согласно спецификации формата. Подобным образом обработка продолжается до конца форматной строки. Если задано больше аргументов, чем спецификаций формата, лишние аргументы игнорируются.

**Форматная строка** - строка символов, показывающая, как должны быть напечатаны параметры.

**Например**, в операторе

```
printf("%d студентов получили оценку %f.\n", number, z);
```

управляющей строкой служит фраза в кавычках, а number и z - аргументы или в данном случае значения двух переменных.

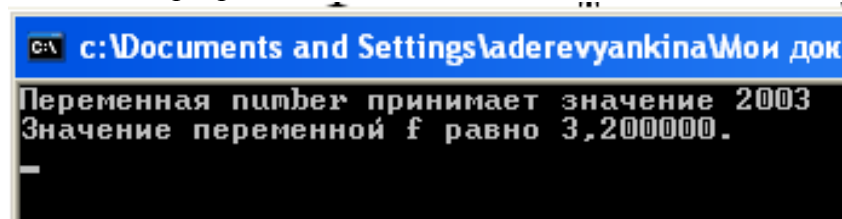
Инструкции, передаваемые функции printf( ), когда необходимо напечатать некоторую переменную, зависят от того, какого типа эта переменная. Например, при выводе на печать целого числа применяется формат %d, а при выводе символа - %c. Форматы перечислены ниже:

<b>Формат</b>	<b>Тип выводимой информации</b>
%d	десятичное целое число
%c	один символ
%s	строка символов
%e	экспоненциальная запись
%f	число с плавающей точкой, десятичная запись
%g	используется вместо записи %f или %e
%u	десятичное целое число без знака
%o	восьмеричное целое число без знака
%x	шестнадцатеричное целое число без знака

### Пример 1

```
#include <stdio.h>
int main( )
{
    int number = 2003;
    float f = 3.2;
    printf("Переменная number принимает значение %d \n", number);
    printf("Значение переменной f равно %f.\n", f);
    return 0;
}
```

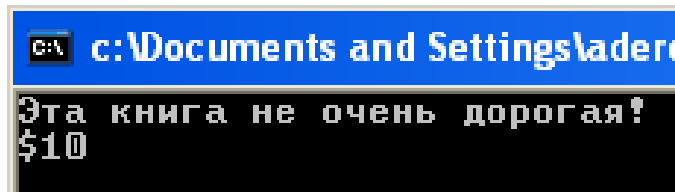
Результат выполнения программы:

A screenshot of a Windows command prompt window. The title bar shows the path 'c:\Documents and Settings\aderevyankina\Мои доку...'. The command prompt shows the output of the first program: 'Переменная number принимает значение 2003' followed by a new line, and 'Значение переменной f равно 3.200000.' followed by a new line.

### Пример 2

```
#include <stdio.h>
int main( )
{
    int cost = 10;
    printf("Эта книга не очень дорогая!\n");
    printf("%c%d\n", '$', cost);
}
```

Результат выполнения программы

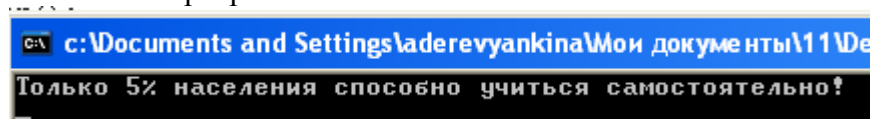
A screenshot of a Windows command prompt window. The title bar shows the path 'c:\Documents and Settings\aderevyankina\Мои доку...'. The command prompt shows the output of the second program: 'Эта книга не очень дорогая!' followed by a new line, and '\$10'.

Если нужно напечатать сам символ %, то компилятор примет его за ошибочную спецификацию преобразования. Выходом из создавшейся ситуации служит довольно простое решение - писать два символа % подряд.

### Пример 3

```
int i=2+3;
printf("Только %d%% населения способно учиться самостоятельно! \n",i);
```

Результат выполнения программы

A screenshot of a Windows command prompt window. The title bar shows the path 'c:\Documents and Settings\aderevyankina\Мои документы\11\De...'. The command prompt shows the output of the third program: 'Только 5% населения способно учиться самостоятельно!' followed by a new line.

Можно расширить основное определение спецификации преобразования, поместив модификаторы между знаком % и символом, определяющим тип преобразования. При

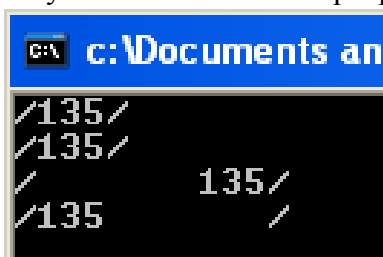
использовании одновременно нескольких модификаторов они должны быть указаны в том порядке, в котором перечислены ниже

Модификаторы	Значение
-	Аргумент будет печататься с левой позиции поля заданной ширины. Обычно печать аргумента оканчивается в самой правой позиции поля. Пример: %-10
строка цифр	Задаёт минимальную ширину поля. Большее поле будет использоваться, если печатаемое число или строка не помещается в исходном поле <b>Пример: %4d</b>
строка цифр	Определяет точность: для типов данных с плавающей точкой число печатаемых цифр справа от десятичной точки; для символьных строк - максимальное число печатаемых символов <b>Пример: %4.2f (две десятичные цифры для поля шириной в четыре символа)</b>
l	Соответствующий элемент данных имеет тип long, а не int <b>Пример: %ld</b>

#### Пример 4

```
int main( )
{
    printf("%d\n",135);
    printf("%2d\n",135);
    printf("%10d\n",135);
    printf("%-10d\n",135);
}
```

Результат выполнения программы:



Первая спецификация преобразования %d не содержит модификаторов. Это так называемый выбор по умолчанию, т. е. результат действия компилятора в случае, если вы не дали ему никаких дополнительных инструкций. Вторая спецификация преобразования - %2d. Она указывает, что ширина поля должна равняться 2, но, поскольку число состоит из трех цифр, поле автоматически расширяется до необходимого размера. Следующая спецификация %10d показывает, что ширина поля равна 10. Последняя спецификация %-10d также указывает ширину поля, равную 10, а знак - приводит к сдвигу всего числа к левому краю.

Функция printf возвращает количество позиций в потоке, которое займут все выводимые данные, а в случае ошибки значение -1 (EOF).

## Обращение к функции ввода

**scanf(форматная строка[,адрес аргумента, ...]);**

Строка описания формата ввода может содержать:

- пробельные символы, тогда при вводе будут игнорироваться все пробельные символы предшествующие первому непробельному;
- печатные символы - все ASCII-символы, кроме %, тогда в потоке ввода должны содержаться такие же символы, которые извлекаются из него но не помещаются в аргументы ("ввод с паролем");
- спецификация формата, которая определяется по знаку %. Спецификация формата заставляет scanf прочитать и преобразовать символы на входе (входное поле) в значение определяемого типа. Значение присваивается переменной, адрес которой указан в списке аргументов.

**Функция scanf( ) использует тот же набор символов спецификации преобразования, что и функция printf( ).**

Спецификации %f, %g и %e эквивалентны. Эти спецификации допускают наличие или отсутствие знака строки цифр с десятичной точкой или без нее и поля показателя степени.

**Функция printf( ) использует имена переменных, константы и выражения, а функция scanf( ) - только адреса переменных.** При использовании scanf необходимо соблюдать два правила:

- Если нужно ввести некоторое значение и присвоить его переменной одного из основных типов, то перед именем переменной требуется писать символ &.
- Если необходимо ввести значение строковой переменной, использовать символ %s не нужно.

Как и printf(), функция scanf() дает возможность модифицировать некоторое число своих спецификаторов формата. В спецификаторах формата можно указать модификатор максимальной длины поля (**width**). Это целое число, расположенное между % и спецификатором формата; оно ограничивает число символов, считываемых из этого поля. Например, чтобы считывать в переменную str не более 20 символов:

```
scanf("%20s", str);
```

Если поток ввода содержит больше 20 символов, то при следующем вызове функций ввода считывание начнется после того места, где оно закончилось при предыдущем вызове. Например, если в ответ на вызов scanf() из этого примера вводится

```
ABCDEFGHIJKLMNORSTUVWXYZ
```

то в str из-за спецификатора максимальной ширины поля будет помещено только 20 символов, то есть символы вплоть до T. Это значит, что оставшиеся символы UVWXYZ пока еще не прочитаны. При следующем вызове scanf(), например при выполнении оператора scanf("%s", str);

в str будут помещены буквы UVWXYZ. Ввод из поля может завершиться и до того, как будет достигнута максимальная длина поля — если встретится разделитель. В таком случае scanf() переходит к следующему полю.

Таким образом, входное поле, выделяемое из потока ввода, включает все символы до первого пробельного символа, или до первого символа, который не может быть преобразован согласно спецификации формата, или до длины поля width, если она определена.

Функция `scanf` сканирует (просматривает) каждое входное поле, символ за символом. Чтение может быть остановлено раньше, чем достигнут пробельный символ по различным причинам:

- определенное `width` количество символов введено;
- следующий символ не может быть преобразован, как определено;
- следующий символ конфликтует с печатным символом в управляющей строке, в которой подразумевается соответствие;
- следующий символ не должен встретиться (или не встретился) в заданном наборе символов.

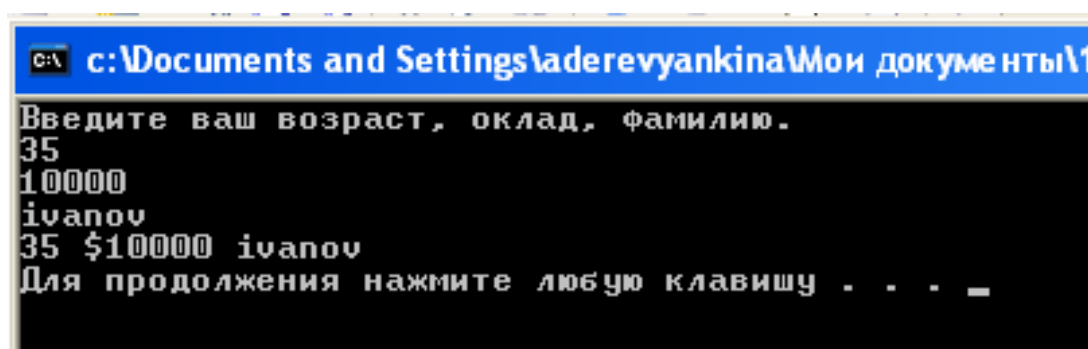
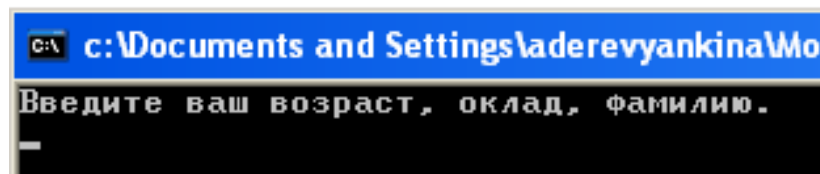
Когда это произошло, следующий невведенный символ рассматривается как первый символ следующего входного поля.

Функция `scanf` возвращает целое значение: число полей, которые были успешно преобразованы и присвоены (не включает поля, которые были прочитаны, но не присвоены), число ноль, если не было присвоенных полей.

### Пример 5

```
int main()
{
    int age;
    float assets;
    char fio[50];
    printf("Введите ваш возраст, оклад, фамилию. \n");
    scanf("%d %f", &age, &assets);
    scanf("%s", fio); /* & отсутствует при указании
                        массива символов */
    printf("%d $%.0f %s\n", age, assets, fio);
}
```

Результат выполнения программы



`scanf()` может прочитать поле, но не присваивать прочитанное значение никакой переменной; для этого надо перед литерой-спецификатором формата поля поставить звездочку, \*. Например, когда выполняется оператор `scanf("%d%*c%d", &x, &y);`

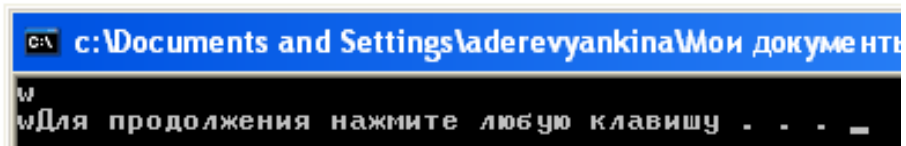
можно ввести пару координат 10,10. Запятая будет прочитана правильно, но ничему не будет присвоена. **Подавление присвоения особенно полезно тогда, когда нужно обработать только часть того, что вводится.**

### Функции ввода (getchar()) – вывода (putchar()) одного символа

Функция getchar( ) получает один символ, поступающий от клавиатуры, и передает его выполняющейся в данный момент программе. Функция putchar( ) получает один символ, поступающий из программы, и пересылает его для вывода на экран.

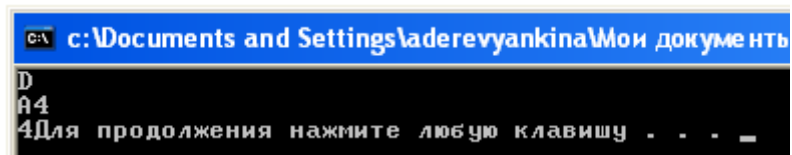
#### Пример 6

```
#include <stdio.h>
int main( )
{
    char ch;
    ch=getchar( );
    putchar(ch);
}
```



#### Пример 7

```
int main( )
{
    char ch='A';
    putchar('D');
    putchar('\n');
    putchar('\007');//звуковой сигнал
    putchar(ch);
    putchar(getchar());
    return 0;
}
```

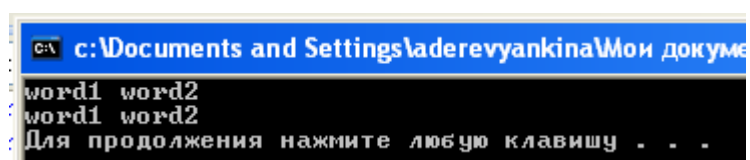
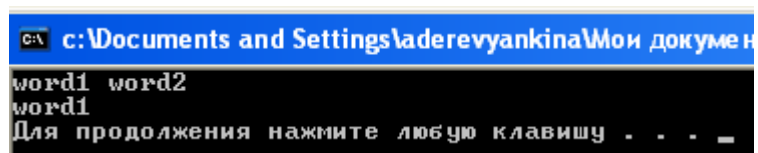


Для ввода строки символов игнорируя первый пробельный символ используется функция gets().

#### Пример 8

```
int main( )
{
    char str[50];
    scanf("%s",str);
    printf("%s\n", str);
    return 0;
}

int main( )
{
    char str[50];
    gets(str);
    printf("%s\n", str);
    return 0;
}
```



Для вывода широких символов (unicode) используется аналог оператора printf – **wprintf**. Например:

```
wchar_t wc=L'q';  
wprintf(L"Character = %c\n",wc);
```

## **Язык C++**

Возможность управлять вводом-выводом в C++, обеспечивают форматирующие функции-члены, флаги и манипуляторы. Флаги, функции и манипуляторы выполняют одну и ту же задачу — задают определённый формат ввода/вывода информации в потоках. **Ввод/вывод на экран/с экрана в C++ осуществляется с помощью объектов классов `istream` и `ostream` `cin` и `cout` соответственно.** Описания `cin` и `cout` содержатся в файле `iostream`.

Различие между функциями флагами и манипуляторами форматирования состоит в способе их применения.

```
cout.fill('/*symbol*/'); // устанавливает символ заполнитель  
// где symbol - символ заполнитель, символ передаётся в одинарных кавычках  
  
cout.width(/*width_field*/); // задает ширину поля  
// где width_field - количество позиций(одна позиция вмещает один символ)  
  
cout.precision(/*number*/); // задает количество знаков после десятичной точки  
// где number - количество знаков после десятичной точки
```

Доступ к функциям осуществляется через операцию точка, а в круглых скобках передаётся аргумент. Аргумент функции `fill()` может передаваться в виде символа, обрамленного одинарными кавычками, или в виде числа(код символа). Одних функций не достаточно для форматирования потоков ввода/вывода, поэтому в C++ предусмотрен ещё один способ форматирования — флаги.

Флаги форматирования позволяют включить или выключить один из параметров ввода/вывода. Чтобы установить флаг ввода/вывода, необходимо вызвать функцию `setf()`, если необходимо отключить флаг вывода, то используется функция `unsetf()`. Далее показаны конструкции установки и снятия флагов вывода.

```
// установка флага вывода  
cout.setf( ios::/*name_flag*/ );  
// где name_flag - это имя флага
```

Доступ к функциям оператора вывода выполняется через операцию точка. Метод `setf()` принимает один аргумент — имя флага. Флаги вывода объявлены в классе `ios`, поэтому, перед тем, как обратиться к флагу, необходимо написать имя класса - `ios`, после которого, с помощью операции разрешения области действия, вызвать нужный флаг.

```
// снятие флага вывода  
cout.unsetf( ios::/*name_flag*/ );  
// где name_flag - это имя флага
```

Если при вводе/выводе необходимо установить(снять) несколько флагов, то можно воспользоваться поразрядной логической операцией ИЛИ |. В этом случае конструкция языка C++ будет такой:1

```
// установка нескольких флагов
cout.setf( ios::*name_flag1* | ios::*name_flag2* | ios::*name_flag_n* );
// снятие нескольких флагов
cout.unsetf( ios::*name_flag1* | ios::*name_flag2* | ios::*name_flag_n* );
```

### Основные флаги форматирования

**boolalpha** Вывод логических величин в текстовом виде (true, false)

*Пример:*

```
cout.setf(ios::boolalpha);
bool log_false = 0,
log_true = 1;
cout << log_false << endl << log_true << endl;
```

*Результат:*

false  
true

**oct (hex)** Ввод/вывод величин в восьмеричной (шестнадцатеричной) системе счисления (сначала снимаем флаг dec, затем устанавливаем флаг oct)

*Пример:*

```
cout.unsetf(ios::dec);
cout.setf(ios::oct);
int value;
cin >> value;
cout << value << endl;
```

*Результат:*

Ввод: 99  
Вывод: 63

**scientific** Вывод чисел с плавающей точкой в экспоненциальной форме

*Пример:*

```
cout.setf(ios::scientific);
double value = 1024.165;
cout << value << endl;
```

*Результат:*

1.024165e+003

**fixed** Вывод чисел с плавающей точкой в фиксированной форме(по умолчанию)

*Пример:*

```
double value = 1024.165;
cout << value << endl;
```

*Результат:*

1024.165

Ещё один способ форматирования — форматирование с помощью манипуляторов.



Манипуляторы не дают ничего сверх того, что обеспечивают методы классов потоков, они введены для удобства использования в переопределенной операции << (или >>).

Для использования манипуляторов с одним параметром необходимо подключить файл **iomanip.h**, а для манипуляторов без параметров файл **iostream**.

### Основные манипуляторы

**endl**      Переход на новую строку при выводе

*Пример:*

```
cout << «Пример1» << endl << «Пример2»;
```

*Результат:*

Пример1

Пример2

**boolalpha**      Вывод логических величин в текстовом виде (true, false)

*Пример:*

```
bool log_true = 1;
```

```
cout << boolalpha << log_true << endl;
```

*Результат:*

True

**oct (hex, dec)**      Вывод величин в восьмеричной (шестнадцатеричной, десятичной) системе счисления

*Пример:*

```
int value = 64;
```

```
cout << oct << value << endl;
```

*Результат:*

100

**scientific**      Вывод чисел с плавающей точкой в экспоненциальной форме

*Пример:*

```
double value = 1024.165;
```

```
cout << scientific << value << endl;
```

*Результат:*

1.024165e+003

**fixed**      Вывод чисел с плавающей точкой в фиксированной форме (по умолчанию).

*Пример:*

```
double value = 1024.165;
```

```
cout << fixed << value << endl;
```

*Результат:*

1024.165

**setprecision(int count)**      Задаёт количество знаков после запятой, где count - количество знаков после десятичной точки

Для работы данного манипулятора необходимо подключить заголовочный файл `#include <iomanip>`

*Пример:*

```
cout << fixed << setprecision(3) << (13.5 / 2) << endl;
```

*Результат:*

6.750

**setw(int number)** Установить ширину поля, где number - количество позиций, символов (выравнивание по умолчанию по правой границе).

*Пример:*

```
cout << setw(40) << «Привет» << endl;
```

*Результат:*

Привет

**setfill(int symbol)** Установить символ заполнитель. Если ширина поля больше, чем выводимая величина, то свободные места поля будут наполняться символом symbol - символ заполнитель.

Символ-заполнитель – это символ, используемый для заполнения промежутков между отформатированным представлением величины и позицией, отмечающей минимальную ширину поля. По умолчанию заполнителем является пробел.

*Пример:*

```
cout << setfill('0') << setw(4) << 15 << endl;
```

*Результат:*

0015

**right** Выравнивание по правой границе(по умолчанию). Сначала необходимо установить ширину поля(ширина поля должна быть заведомо большей чем, длинна выводимой строки).

*Пример:*

```
cout << setw(40) << right << «Привет» << endl;
```

*Результат:*

Привет

**left** Выравнивание по левой границе. Сначала необходимо установить ширину поля (ширина поля должна быть заведомо большей чем, длинна выводимой строки).

*Пример:*

```
cout << setw(40) << left << «Привет» << endl;
```

*Результат:*

Привет\_\_\_\_\_

Для вывода одного символа за один раз используется функция **cout.put**.

### Пример 9

```
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "Russian");
    char c='a';
    cout.put(toupper(c));
    system("pause");
    return 0;
}
```

toupper – преобразует символ в верхний регистр

Точно так же как выходной поток **cout** позволяет записать вывод на экран, входной поток **cin** позволяет читать ввод с клавиатуры.

### Пример 10

```
#include <iostream.h>
using namespace std;
int main()
{
    int number;
    cout << "Введите ваше любимое число и нажмите Enter: ";
    cin >> number;
    cout << "Ваше любимое число равно " << number << endl;
    return 0;
}
```

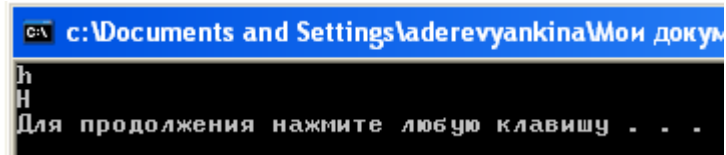
### Пример 11

```
#include <iostream.h>
using namespace std;
int main()
{
    int first, second;
    cout << "Введите два числа и нажмите Enter: ";
    cin >> first >> second;
    cout << "Были введены числа " << first << " и " << second << endl;
}
```

Для ввода одного символа используется **cin.get**.

### Пример 12

```
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "Russian");
    char c;
    cin.get(c);
    cout.put(toupper(c));
    cout<<endl;
    system("pause");
    return 0;
}
```



**cin** использует пустые символы, такие как пробел, табуляция или возврат каретки, для определения, где заканчивается одно значение и начинается другое. Для ввода строки данных, состоящую из слов разделенных пробелами, используется функцию **cin.getline**. Для использования **cin.getline** указывается символьная строка, в которую будут помещаться символы, а также размер строки, как показано ниже:

**cin.getline(string, 64);**

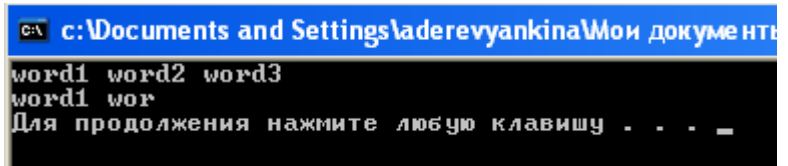
Когда **cin.get** читает символы с клавиатуры, она не будет читать символов больше, чем может вместить строка. Удобным способом определить размер строки является использование оператора C++ **sizeof**, как показано ниже:

**cin.getline(string, sizeof(string));**

### Пример 13

```
#include <iostream>
using namespace std;
```

```
int main( )
{
    setlocale(LC_ALL, "Russian");
    char st[10];
    cin.getline(st, sizeof(st));
    cout<<st<<endl;
    system("pause");
    return 0;
}
```



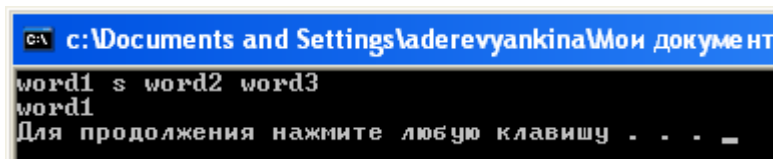
Для организации ввода символов до и включая определенный символ в `cin.getline` передается это символ.

Например, следующий вызов заставляет функцию `cin.getline` читать строку текста, пока не встретится возврат каретки, или пока не будут прочитаны 64 символа, или пока не встретится буква Я:

**`cin.getline(string, 64, 'Я');`**

#### Пример 14

```
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "Russian");
    char st[10];
    cin.getline(st, sizeof(st), 's');
    cout<<st<<endl;
    system("pause");
    return 0;
}
```



**`cin.clear()` - сброс битов ошибок входного стандартного потока**

**`cin.sync()` - очищение буфера стандартного ввода**

Для вывода широких символов (unicode)

```
wchar_t wc=L'q';
wcout<<wc<<endl;
```

## Задания

**Каждое задание реализовать на С и на С++**

**I** Наберите, оттранслируйте и выполните программу нахождения суммы трех переменных целого типа (int), значения которых вводятся с клавиатуры. Измените тип переменных на double. Добейтесь правильной работы программы.

**II** Объявить переменные x,y,z целого типа. Организовать их ввод с клавиатуры. Объявить три строковые переменные st1, st2, st3. Организовать их ввод с клавиатуры. Реализовать вывод переменных в следующем виде:

```
st1.....X
st2.....Y
st3.....Z
```

**III** Объявить переменную x целого типа и y вещественного типа. Организовать ввод значений переменных с клавиатуры. Вывести значение переменной x в шестнадцатеричном и восьмеричном виде. Значение переменной y вывести в экспоненциальной форме, с точностью один, два и три знака после запятой.

**IV** Организовать следующий вывод переменной x вещественного типа (например x равно 1,3456)

```
1,3
1,35
1,346
1,3456
```

**V** Ввести следующую программу

```
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "Russian");
    char str1[5];
    char str2[3];

    cin.getline(str1,5);
    cin.getline(str2,3);

    cout<<endl<<"str1 = "<<str1;
    cout<<endl<<"str2 = "<<str2<<endl;;

    system("pause");
    return 0;
}
```

Запустить программу на исполнение

Ввести с клавиатуры 4444 33

Посмотреть, что будет на экране.

Снова запустить программу на исполнение.

Ввести с клавиатуры 4444 (нажать Enter и далее ввести) 33

Посмотреть, что будет на экране

Преобразовать программу к следующему виду:

```
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "Russian");
    char str1[5];
    char str2[3];

    cin.getline(str1,5);
    cin.clear();
    cin.getline(str2,3);

    cout<<endl<<"str1 = "<<str1;
    cout<<endl<<"str2 = "<<str2<<endl;;

    system("pause");
    return 0;
}
```

Повторить предыдущие запуски и посмотреть, что будет на экране.

Объяснить полученные результаты.

**VI** Написать программу, которая вводит координаты начала и конца отрезка, сохраняющиеся в четырех переменных целого типа. При этом пользователь вначале вводит координаты начала отрезка в виде (5,4), затем вводится координаты конца отрезка.

**VII** Пользователь вводит строку. Написать программу, которая помещает в переменную str часть строки длиной не более 8 символов и оканчивающую буквой g.

**VIII** С клавиатуры вводится строка. Первые два символа должны считываться с переменные типа char, а оставшиеся часть строки в строковую переменную.

**IX** Реализовать программу, проводящую опрос пользователя. По очереди пользователю задается пять вопросов, он на них вводит ответы, далее экран очищается выводится результат опроса в следующем виде:

```
|-----|
1. Формулировка вопроса                                ответ
2. Формулировка вопроса                                ответ
3. Формулировка вопроса                                Ответ
....
|-----|
```

system("cls"); - очистка экрана

**X** Реализовать программу которая задает символьную переменную и символьную переменную широкую. Вывести размер одной и другой переменной. Вывести их значения.

## Контрольные вопросы

1. Какие средства Си позволяют выполнять форматированный ввод/вывод на консоль?
2. Для чего нужна форматная строка функций printf/scanf?
3. Сколько аргументов можно передать функции printf/scanf?
4. Какие спецификации вывода можно использовать для ввода/вывода целых значений? Вещественных значений? Символов? Строк?
5. Какую спецификацию нужно задать в функции scanf для ввода шестнадцатеричного целого значения?
6. Почему функции scanf передаются адреса, а функции printf значения?
7. Что будет, если в форматной строке функции printf встретятся символы x=?
8. Что будет, если в форматной строке функции scanf встретятся символы x=?
9. Что возвращает функция printf? scanf?
10. Какие функции ввода/вывода отдельных символов вы знаете в Си?
11. Какие функции в Си вы знаете для ввода строк?
11. Какие средства C++ позволяют выполнять форматированный ввод/вывод на консоль?
12. Для чего нужны манипуляторы и флаги форматирования?
13. Какие функции вы знаете для ввода символов и строк в C++?
14. Что делает операция sizeof()?
15. Средства языка Си и C++ для вывода широких символов?