# ЛЕКЦИЯ 8.
## Тема: Функции(продолжение)

**Вопросы:**

1. Константные указатели
2. Указатели на константу
3. Константные ссылки.
4. Параметры со значениями по умолчанию.
5. Перегрузка функций.
6. Стандартные функции.
7. Форматированный ввод-вывод.
8. Функции общего назначения.

***Константные указатели*** - это указатели, значение которых не изменяется в ходе выполнения программы. Они описываются так:

```
указательный_тип const имя инициализатор
```

Пример:

```
int a;
int * const p=&a;
```

Изменить значение константного указателя нельзя, а изменить значение памяти, адрес которой находится в указателе – можно:

```
p++;  // нельзя!
(*p)++; // можно!]
```

***Указатели на константу*** - Это указатели, содержащие адрес константы. Они описываются так:

```
const указательный_тип имя [инициализатор]
```

Пример:

```
int a;
const int *p=&a;
```

Изменить значение такого указателя можно, а изменить значение памяти, адрес которой находится в указателе – нельзя:

```
 p++;  // можно!
(*p)++; // нельзя!
 a++; // можно!
```

## *8.1.Константные ссылки*

Если требуется запретить изменение параметров во время выполнения функции и выдавать соответствующие сообщения на этапе компиляции, формальные параметры описываются с модификатором **const**.

```
void my_func(const double &d)
```

## *8.2.Параметры со значениями по умолчанию*

Для упрощения вызова функции, в ее заголовке можно указать значения параметров по умолчанию. Такие параметры должны быть последними в списке параметров, например:

```
int func1 (int a,int b=0,char c=' '){…}
int func2(int n, int m = 3,k){…} // Ошибка
```

При вызове функции можно опускать фактические параметры, если их значения совпадают со значениями по умолчанию. Однако, если один из параметров опущен, то должны быть опущены и все последующие.

***Пример 8.2.***

```
#include <iostream.h>
 void print ( int num_k=1, int num_d=1,
              char* name="Красивая ")
 { cout << name << " дом " << num_d << " кв. "
       << num_k << endl;}
int main()
{    print();
     print(15);
     print(15,32);
     print(3,5,"Новая ");
     return 0;
}
```

## *8.3.Перегрузка функций*

Часто удобно для функций, выполняющих один и тот же алгоритм для данных разных типов, иметь одинаковые имена.

Создание нескольких различных функций с одинаковым именем, но с различными спецификациями (количеством и/или типом аргументов, но не типом возвращаемого значения) называется ***перегрузкой функций***.

При вызове функции компилятор сам подбирает наиболее подходящий, по его мнению, вариант.

Если точного соответствия не найдено, то выполняется преобразования типов.

Функция max

```
int max(int a, int b)
{ return (a<b ? b : a);}
```

может быть переопределена (перегружена), с тем чтобы она, находила максимум среди данных типа **double** и возвращала результат этого типа.

```
double max(double a,double b)
     { return (a<b ? b : a);}
```

Отсутствие перегрузки приведет к тому, что **max(3.5, 7.1)** будет = 7.

Если при вызове компилятор не сможет подобрать нужную функцию – будет ошибка компиляции.

***Пример 8.3.***

```
double sum(double a,double b,double c)
     { return (a+b+c);}
int sum(int x,int y,int z)
     { return (x+y+z);}
int sum(int x,int y)
     { return (x+y);}
…
cout << sum(1,2,3) << endl;//2-ая
cout << sum(1,2) << endl;//3-я
cout << sum(1.0/25,2.5,3.7); //1-ая
```

## *8.4.Стандартные функции*

### *Математические функции*

В файле <math.h> (или <cmath>) описываются стандартные математические функции и определяются макросы.

При вызове функций могут возникать синтаксические ошибки при несоответствии типов.

***Ошибки выполнения***

***Ошибка области*** возникает, если аргумент выходит за область значений, для которой определена функция.

***Ошибка диапазона*** возникает если результат функции не может быть представлен в виде **double**, происходит переполнение или потеря значимости.

В таблице приведены математические функции. Переменные x и y имеют тип **double**, n - тип **int**, и все функции возвращают значения типа **double**.

Углы в тригонометрических функциях задаются в радианах.

**x=-123.123;**

```
a=350;
b=24;
y=ceil(x) ;  cout<<y<<endl;//-123
y=floor(x);  cout<<y<<endl;//-124
y=fabs(x);   cout<<y<<endl;//123.123;23
y=ldexp(x, n); cout<<y<<endl; //-15759.7
y=frexp(x, &n);
 cout<<y<< " "<<n<<endl;//0.961898  7
y=modf(x,&z);
 cout<<y<<" "<<z<<endl; //-0.123  123
y=fmod(a,b); cout<<y<<endl;   //14
```

### *8.5.Форматированный ввод-вывод*

В языке C++ нет встроенных средств ввода-вывода – он осуществляется с помощью функций, типов и объектов, содержащихся в стандартных библиотеках. Используются два способа:

- функции, унаследованные из языка C;
- объекты C++.

Основные функции ввода-вывода в стиле C:

для ввода:

```
int scanf(const char* format[,arg,...]);
```

для вывода:

```
int printf(const char* format[,arg,...]);
```

Функции в стиле C представлены в файле **<stdio.h>**, поэтому в программе необходима директива

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

#### *8.5.1.Функция printf*

```
int printf(const char* format[,arg,...]);
```

выводит на стандартное устройство вывода значения перечисленных в списке аргументов в формате , определенном строкой **format**.

Строка формата содержит два типа объектов – простые символы, которые при выводе копируются в поток вывода (на экран в нашем случае) и спецификации преобразования, каждая из которых приводит к преобразованию и выводу следующего аргумента. Перед каждой спецификацией пишется %.

Поэтому для включения % в вывод можно использовать %% в форматируемой строке.

***Спецификации формата*** имеют следующий вид:

**%[флаги] [ширина] [.точность] [модификатор] [тип]**

- **флаги -** любые символы, уточняющие формат вывода;
- **ширина -** минимальное число выводимых символов;
- **очность -** максимум цифр после запятой;
- **модификатор** префикс - уточняет тип.

Поведение `printf` не определено в случае нехватки аргументов для форматирования.

`printf` заканчивает работу, если встречает конец форматируемой строки. Если аргументов больше, чем требуется, то лишние аргументы игнорируются.

***Спецификаторы типа:***

**%** - выводится символ % , никакие аргументы не используются;

**% c** - аргумент выводится как отдельный символ;

**% s** - аргумент является строкой: символы строки выводятся до тех пор, пока не будет достигнут нулевой символ или не будет выведено количество знаков, указанное в спецификаторе точности;

**% d** - целочисленный аргумент выводится как десятичное целое со знаком (считывается целое int - тоже что и i);

**% i** - выводится десятичное целое со знаком (считывается целое **`int`** - тоже что и d);

**% o** - выводится восьмеричное целое со знаком (считывается целое **`int`**);

**% u** - выводится десятичное целое без знака (считывается **`int`**);

**% x** - выводится шестнадцатичное целое без знака, используются цифры **abcdef** после 9 (считывается **`int`**);

**% X** - выводится шестнадцатичное целое без знака, используются цифры **ABCDEF** после 9 (считывается **`int`**);

**% f** - выводится значение со знаком в виде [-]9999.9999 (считывается число с плавающей точкой);

**% e** - выводится значение со знаком в виде [-]9.9999e[+|-]999 (считывается число с плавающей точкой);

**% E** - выводится тоже самое, что и при **e**, но используется E для записи экспоненты (считывается число с плавающей точкой);

**% g** - выводится значение со знаком как в случае **f** или **e**, в зависимости от заданного значения и точности - нули на конце и десятичные точки печатаются только в случая необходимости (считывается число с плавающей точкой);

**% G** - выводится тоже самое, что и при **g**, но используется **E** для записи экспоненты (считывается число с плавающей точкой);

**% p** - выводит указатель в формате данной реализации;

***Формат вывода символов***

`%[-][0][ширина][h|l][c]`

**-** - выравнивание влево (по умолчанию вправо);

**0** - символ заполнитель поля вывода

**ширина:** - минимальное число выводимых символов;

**h** - однобайтовый символ;

**l** - широкий символ;

**c** - спецификация для вывода символов.

```
char c='*';
printf("%010c\n",c);     //   000000000*
printf("%10c\n",c);      //             *
printf("%-10c\n",c);     //  *
printf("%-10hc\n",c);    //  *
```

***Формат вывода строки***

**%[-] [0] [ширина] [.точность] [h|l] [s]**

**-** - выравнивание влево (по умолчанию - вправо);

**0** - символ заполнитель поля вывода;

**ширина** - минимальное число выводимых символов;

**[.точность]** - максимальное число выводимых символов;

**h** - однобайтовый символ;

**i** - широкий символ;

**s** - спецификация для вывода строки.

```
char* st="hello";
printf("%010s\n",st);   // 00000hello
printf("%10s\n",st);    //     hello
printf("%-10s\n",st);   // hello
printf("%-10hs\n",st);  // hello
printf("%10.3s\n",st);  //       hel
```

***Форматы вывода числовых данных со знаком***

Для вывода целых чисел со знаком:

**%[-] [+] [пробел] [0] [ширина] [.точность] [h|l] {d|i}**

**-** - выравнивание влево (по умолчанию - вправо);

**пробел** - выводит пробел в позицию знака;

**ширина** - минимальное число выводимых символов;

**[.точность]** - минимальное количество цифр, которые должны быть выведены;

**h** - модификатор **`short`**;

**l** - модификатор **`long`**;

**d, i** - спецификации для вывода чисел.

```
int x=1201;
printf("%10d\n",x);   //        1201
printf("%-10d\n",x);  // 1201
printf("%+10d\n",x);  //       +1201
printf("%-10ld\n",x);  // 1201
printf("%d\n",x);     //  1201
printf("%010d\n",x); //  0000001201
printf("% 10d\n",x);  //            1201
```

***Форматы вывода числовых данных без знака***

Для вывода целых чисел без знака:

**%[-] [#] [0] [ширина] [.точность] [h|l] {u|o |x | X}**

**-** - выравнивание влево (по умолчанию - вправо);

**#** - выводит начальный **0** для чисел в 8с/с и **x** для 16 с/с;

**ширина -** минимальное число выводимых символов;

**[.точность] -** минимальное количество цифр, которые должны быть выведены;

**h** - модификатор **`short;`**

**l** - модификатор **`long;`**

**u** - число в 10 с/с;

**o** - число в 8 с/с;

**x(X)** - число в 16 с/с.

***Форматы вывода действительных чисел***

Для вывода действительных чисел со знаком:

**%[-] [#] [+|пробел] [0] [ширина] [.точность] [L|l] {f|e |E |g|G}**

**-** - выравнивание влево (по умолчанию - вправо);

**#** - выводит точку, для типов g|G выводит завершающие нули;

**ширина -** минимальное число выводимых символов;

**[.точность]** - количество цифр после десятичной точки для f|e |E и количество значащих цифр для g|G. Числа округляются отбрасыванием. По умолчанию выводится 6 цифр;

**L,l** - модификатор **`long;`**

**f** - число типа **`double;`**

**e** - число типа **`double`** в экспоненциальной форме, экпонента обозначается e ;

**E** - число типа **`double`** в экспоненциальной форме, экпонента обозначается E ;

**G,g** - вывод числа в наиболее коротком формате f или e (f или E).

```
printf("%10.3f\n",y); //   1234.560
printf("%-10.6f\n",y);// 1234.560000
printf("%+15f\n",y);  //    +1234.560000
printf("%-10lf\n",y);  // 1234.560000
printf("%f\n",y);      //  1234.560000
printf("%10.3g\n",y); //   1.23e+003
printf("%-10.6g\n",y);// 1234.56
printf("%+15e\n",y);   //  +1.234560e+003
printf("%+15E\n",y);  //   +1.234560E+003
printf("%-10G\n",y);  // 1234.56
printf("%f\n",y);      // 1234.560000
```

***Формат вывода указателей***

Формат вывода:

**%[-] [+| пробел] [ширина] [p]**

**+ или | пробел** - выравнивание вправо;

**ширина** - минимальное число выводимых символов;

**p** - данные типа **void.**

```
char c1='*';
char *p=&c1;
printf("%p\n",p); //  0012FF27
```

### *8.5.2.Функция scanf*

Функция **scanf** вводит аргументы, применяя к каждому определитель формата из *format. Заголовок функции имеет следующий вид:

```
int scanf(const char* format[,arg,...]);
```

Аргументы должны задавать **адреса переменных (&)**

Поведение **scanf** не определено в случае нехватки аргументов для форматирования.

**scanf** заканчивает работу, если встречает конец форматируемой строки.

Если аргументов больше, чем требуется, то лишние аргументы игнорируются.

**format** - указатель на строку знаков, содержащую два типа объектов: спецификации преобразования, начинающиеся с % и управляющие символы.

Спецификации преобразования имеют следующую форму:

**%[*] [ширина] [модификатор] [тип]**

**ширина -** мин число выводимых символов.

**модификатор префикс** - уточняет тип.

***** - означает пропуск при вводе поля, определенного данной спецификацией.

```
int x,b;
char c, char str[]=”HELLO”;
scanf("%10d",&x);
scanf("%d",&x);
scanf("%c",&c);
scanf("%x",&x);
scanf("%s",&str);
scanf("%*d%d",&x,&b); // пропуск поля ввода, значение b не изменится
```

### *8.6. Функции общего назначения*

В заголовке **<stdlib.h> (<cstlib>)** объявляется набор функций, служащих для преобразования данных, генерации случайных чисел, получения и установки переменных среды, управления выполнением программ и выполнения команд.

**abs –** модуль целого числа

**bsearch -** двоичный поиск

**qsort** - сортировка массива

Для получения псевдослучайных чисел используют функции

```
int rand(void);
```

**rand** возвращает целое число из последовательности псевдослучайных чисел в интервале между 0 и значением RAND_MAX (по умолчанию 32767).

```
void srand(unsigned int );
```

Эту функцию можно применить для получения другой последовательности псевдослучайных чисел при разных запусках программы, задавая различные стартовые точки.

Обычно функция **srand** вызывается до первого вызова функкции **rand.**

```
int a,b,c;
srand(NULL);
a = rand();
b = rand();
c = rand()%100;
```

**УСР**

**1. Изучить материалы файла “Форматированный ввод-вывод.doc”.**

**2. Функции printf и scanf из файла "Стандартные Функции stdio.doc ”.**

**3. Прочитать информацию "Стандартные библиотека математических функций math.doc“ и"Стандартные Функции stdlib.doc ”.**