ЛЕКЦИЯ 7.
Тема: Функции

Вопросы:

- **Подпрограммы.**
- **Объявление и определение функций.**
- **Вызов функции.**
- **Формальные и фактические параметры.**
- **Передача параметров по значению.**
- **Область действия имен.**
- **Понятие указателя и ссылки.**
- **Передача параметров по адресу.**

При решении задач довольно часто встречается ситуация, когда приходится многократно повторять одни и те же группы операторов, или требуется выполнить одни и те же вычисления, но с различными данными. Чтобы облегчить процесс программирования таких задач, в языки программирования было введено понятие ***подпрограмм*.**

Подпрограмма – это именованная группа операторов, оформленная специальным образом, которая может вызываться по имени.

При вызове подпрограммы управление вычислительным процессом передается подпрограмме.

После выполнения подпрограммы осуществляется возврат на оператор основной (вызывающей) программы, следующий за вызовом подпрограммы.

Подпрограммы бывают двух типов: процедуры и функции.

В С и С ++ есть только функции.

Любая программа на С++ состоит из функций, одна из которых должна иметь имя **`main`** (с нее начинается выполнение программы).

Описание функции имеет следующий вид:

```
[модификаторы] тип_возвр_значения
имя_фукции([формальн_параметры])
{тело_функции}
```

Функцию можно рассматривать как операцию, определенную пользователем.

Операнды функции, или ***формальные параметры***, задаются в ***списке параметров*** через запятую. Список параметров заключается в круглые скобки.

Через параметры в функцию можно передавать значения. Параметр описывает тип значения, которое будет передано в функцию при ее вызове.

Функция выполняет определенные действия и возвращает в качестве результата значение в вызывающую программу. Оно называется ***возвращаемым значением***. Тип этого значения объявляется в заголовке функции. Об отсутствии возвращаемого значения сообщают ключевым словом **`void`**.

Действия, которые выполняет функция, составляют ее ***тело***;

тело заключается в фигурные скобки.

Пример 7.1. ***Функция нахождения максимального числа среди трех целых чисел.***

```
int max3(int a,int b,int c)//заголовок ф-ии
{                    //начало тела функции
   int max;
   if (a > b)
     max = a;
   else
     max = b;
   if (c > max)
     max = c;
   return max;
}                       //конец тела функции
```

Вызов этой функции в головной программе:

```
int  m, x=67, y=98, z=56;
cout << max3(x,y,z)<< endl; //98
cout << max3(9,7,111)<< endl; //111
```

Объявление и определение функций

Различают ***объявление*** и ***определение*** функции.

Объявление (прототип) функции совпадает с ее заголовком, отличие лишь в том, что оно заканчивается точкой с запятой:

```
int max3(int a,int b,int c);
```

В объявлении можно не указывать имена формальных параметров:

```
int max3(int,int,int);
```

Определение функции описывает, как она работает, т.е. какие действия надо выполнить, чтобы получить искомый результат.

Определение функции состоит из заголовка и тела.

Определение функции задает:

- тип возвращаемого значения,
- имя функции,
- типы и число формальных параметров,
- объявления локальных переменных,
- операторы (тело функции), определяющие действие функции.

```
int max3(int x,int n)
 //это заголовок функции
```

Имя функции **max3**– это идентификатор.

В скобках – формальные параметры.

Аргументы **x, n** с называются ***формальными*** параметрами. Это переменные, которые определены в теле функции (т.е. к ним можно обращаться только в теле функции). При определении функции программа не знает их значения.

После заголовка записывается тело функции в виде блока.

Специальная инструкция

```
return [ (значение) ];
```

позволяет завершить работу функции и вернуться в вызвавшую функцию. Может иметь различный вид.

```
return r;
return 0;
return (tmp<0 ? -tmp : tmp);
return (a<=10);
```

Функция может содержать несколько инструкций **return.** Выражение, стоящее в **return** преобразуется к типу возвращаемого функцией значения.

```
int f1() {return 1;} //ok
void f2() {return 1;} //error
double f3() {return 1;} //ok –//выполнится преобразование
```

Можно написать функцию, у которой будет только инструкция **return.**

Пример 7.2. ***Функция, вычисляющая максимум из двух целых чисел типа int.***

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

//вызов
int m=max(x,y);
```

Вызов функции

Для вызова функции используется конструкция:

имя_функции ([фактические_параметры])

При вызове функции вместо **формальных** подставляются **фактические** параметры – значения, с которыми функция будет работать.

Вызов функции может находиться в любом месте программы, где по

синтаксису допустимо выражение того типа, который формирует функция.

В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать.

```
int max3 (int a, int b, int c);
…
int max3 (int a, int b, int c)
{…}
…
int x=4,y=88,d=100;
cout << max3(x,y,d) << endl;
```

Фактический параметр может быть **константой, переменной или более сложным выражением.**

Независимо от типа фактического параметр он вначале вычисляется, а затем его величина передается функции.

Фактический параметр - это конкретное значение, которое присваивается перемен-ной, называемой формальным параметром.

```
int a=6,b=6000;
y=max3(a,4*25,b/a); //1000
```

Пример 7.3. ***Функция нахождение НОД двух натуральных чисел.*** (Еще одна реализация алгоритма)

```
int Nod(int x, int y)
{
 // возвращает НОД
      while ( y )
    {
        int tmp = y;
        y = x % y;
        x = tmp;
    }
    return x;
}
```

Пример 7.4. ***Функция нахождение модуля целого числа.***

```
int my_abs(int tmp)
{
 return(tmp<0 ? -tmp : tmp);
}
```

Пример 7.5. ***Функция возведения целого числа в неотрицательную степень.***

```
int step1(int, int);//объявление функции
//далее идет определение функции
int step(int x, int n)
{int r=1;
 for (int i=1; i<=n; i++)
  {
    r=r*x;
  }
 return r;
}
```

Второй способ реализации этой функции:

```
int step2(int x,int n)
{int r=1;
 while  (n!=0)
  {
    if (n%2==1) r=r*x;
    x=x*x;
    n=n/2;
  }
 return r;
}
int main ( )
{ int x=10,n=7,a,b,c,y;
  cout << "Enter a, b";
  cin >> a >> b;
  c=Step2(a,b);
  y=Step2(x,n);
  cout << "c=" << c << endl;
  cout << "y=" << y << endl;
  cout << "15 в 4-ой=" << Step2(15,4)<< endl;
  return 0;
}
```

Функция может возвращать значения любого типа, **кроме массива и функции**.

Но может вернуть **указатель** на область памяти, в которой хранится массив и может вернуть **указатель** на функцию (об этом позже).

Если в качестве типа возвращаемого значения задан тип **void,** то не требуется возвращать значение.

Модификаторы, если есть, определяют класс памяти, что ограничивает видимость функции и других внешних определений.

Функция с классом памяти **static** невидима вне содержащего ее файла. По умолчанию, предполагается **extern.**

Есть функции, у которых отсутствует список формальных параметров, в этом случае записывают пустые скобки, также допускается указывать тип **void.**

Например:

```
  void Code()
```

или

```
  void Code(void)
```

Пример 7.6. ***Функция печати кодировочной таблицы.***

```
void Code ()
{
     printf("Current table: \n");
     for ( int c=0;c<=255;c++)
     {
          printf("%3d-%c  ",c,c);
          if((c+1)%11==0)
          printf("\n");
     }
}
```

Вызов этой функции:

```
Code();
```

Обратите внимание, что скобки при вызове функции необходимо указывать даже тогда, когда список фактических параметров пуст.

В примере используется форматированный вывод, поэтому нужно подключить файл **stdio.h** :

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

Локальные переменные

В функции можно использовать не только переданные ей значения, но и объявлять собственные переменные внутри тела функции. Такие переменные называются ***локальными*** и существуют только внутри функции. После выхода из функции они удаляются из памяти.

Аргументы, переданные функции, так же являются локальными переменными данной функции.

Передача параметров

Все параметры в функции C++ передаются ***по значению***, т. е. при вызове функции в стек заносится копия значения фактического параметра, и операторы в теле функции работают с этой копией.

Поэтому значения фактических параметров, которые переданы в функцию, не изменяются.

Передача параметров по значению предусматривает следующие действия:

1. При компиляции функции выделяются участки памяти для формальных параметров.

2. Формальные параметры - это внут-ренние объекты функции. Для параметров типа **float** формируются объекты типа **double**. Для параметров типа **char, short int** создаются объекты типа **int**.

3. Если параметром является массив, то формируется указатель на начало этого массива и он служит представлением массива-параметра в теле функции.

4. Вычисляются значения выражений, использованных в качестве фактических параметров при вызове функции.

5. Значения выражений-фактических параметров заносятся в участки памяти, выделенные для формальных параметров функции.

6. В теле функции выполняется обработка с использованием значений внутренних объектов-параметров, и результат передается в точку вызова функции как возвращаемое ею значение.

7. Никакого влияния на фактические параметры функция не оказывает.

Пример 7.7.

```
// Вычисление n!
#include <stdio.h>
long fact(int);
int main()
{
 int n;
 printf("введите числа\n");
 scanf("%d",&n);
 printf("Факториал %d=%d\n",n,fact(n));
 return 0;
}
long fact(int num)
{
  int res=1;
  for (;num>1;num--)
      res*=num;
  return res;
}
```

Пример 7.8. ***Функция преобразования числа.***

По заданному натуральному числу (4 байта) получить новое число, переставив младшую цифру исходного числа на место впереди старшей. (Например, из входного значения 789 должно получиться 978.)

Алгоритм см. в предыдущей лекции.

```
int Rev (int n)
{int tmp,digit,n_dig;
 tmp = n;   n_dig = 0;
 while  (tmp > 0)
     {
       tmp = tmp / 10;
       n_dig++;
     }
 tmp = n / 10;
 digit = n % 10;
 int  res =  digit * pow(10,n_dig - 1)+tmp;
 return res;
}
```

Пример 7.9. ***Функция вычисления числа сочетаний из n по k, не используя вычисление факториала.***

```
int Cnk(int n, int k)
{ int i,t,S;
    if ((k==0)||(k==n))
     return 1;
   else
    {if (n-k > k)
         t = k
   else
         t = n-k;
     s=1;
     for (i = 1; i <= t;i++)
         S = S * (n - i + 1) / i;
     return S;
    }
}
```

Пример 7.10. ***Функция перевода числа из 10-ой системы счисления в систему счисления с основанием p.***

Алгоритм см. в предыдущей лекции.

```
int System2_9(int n, int p)
{
 int a=0, r=1;
 while (n >= 1)
 { a=a+(n%p)*r;
```

```
   r=r*10;
   n=n/p;
 }
   return a;
}
```

Пример 7.11. **Вычисление совершенных чисел**

```
#include <iostream>
using namespace std;
int step(int x,int n)
{int r=1;
 while  (n != 0)
 {  if (n % 2 == 1) r=r*x;
    x=x*x;
    n=n / 2;
 }
 return r;}

int Prime(int p)
{ int flag,d;
  if ((p==2) ||(p==3))flag=1;
  else
  {d=2;
   flag=1;
   while ((d<=p/2) && (flag))
        if ( !(p%d)) flag=0;//нет
        else ++d;
  }
 return flag;
}

int main()
{ int p=2,i=1,n,ch;
  cin>>n;
  while (i<=n)
  { if (Prime(p))
       {ch=step(2,p-1)*(step(2,p)-1);
        cout <<ch<<endl;
        i++;
       }
      p++;
  }
  return 0;
```

```
}
```

Пример 7.12. ***Поменять местами значения двух переменных a и b.***

```
void Swap (int a, int b)
{
  tmp = a;
  a = b;
  b = tmp;
}
……  int x=10, y=100;
    Swap(x,y);
```

Обмена нет!!!

Для того чтобы обеспечить изменение передаваемых в функцию фактических параметров, необходимо явно передать ***адрес*** изменяемого параметра. Этого можно достичь двумя способами.

Во-первых, можно в качестве формального параметра описать ***указатель*** на тип, с которым будем работать.

Во-вторых, можно использовать ***ссылки***.

Переменные-указатели

В памяти могут находится данные разных типов, переменные и константы.

Указатель – это переменная, которая содержит адрес памяти. Этот адрес как правило является адресом некоторой другой переменной.

Тип данных переменных, на который указывает указатель называется ***базовым типом данных.*** Стандартный вид обьявления указателя:

```
тип * имя
int c = 100;
int *p =&c;
```

Для объявления указателя нужно, перед именем переменной поставить “*”.

```
int *ip, *iq;      // указатели на целые числа
double *fp;        // указатель на вещественные
char *cp;          // указатель на символьные
int x;
```

Операторы для работы с указателями

Унарные операторы ***** и **&** имеют более высокий приоритет, чем арифметические операторы,

Оператор **&** - этот оператор возвращает адрес объекта, так что инструкция

```
ip = &x;
```

присваивает указателю **ip** адрес переменной **x.**

Унарный оператор ***** возвращает значение переменной, находящейся по данному адресу.

Для получения значение переменной **x**, адрес которой присвоен **ip**, нужно записать ***ip**.

Этот оператор называют оператором ***разыменования или косвенного доступа.***

Пример 7.13. ***Поменять местами значения двух переменных a и b.***

```
void Swap (int *px, int *py)
{
   int tmp = *px;
   *px = *py;
   *py = tmp;
}
…… int a=10, b=100;
   Swap(&a,&b);
```

Ссылки (reference)

Ссылка в программировании — это объект, указывающий на определенные данные, но не хранящий их.

Ссылку можно рассматривать как синоним имени, указанного при ее инициализации, или как указатель, который автоматически разыменовывается.

Ссылки в C++ должны быть связаны с каким-либо объектом. Таким образом, «нулевые ссылки» в C++ отсутствуют.

Ссылочный тип появился в языке C++ и используется главным образом при работе с модифицируемыми параметрами функций.

Ссылочный тип, как и указатель, основан на некотором базовом типе, его описание имеет вид: **базовый_тип &**

```
int c = 100;
int &p = c;
int &r;  //Ошибка
p++;     //c=101;
```

Пример 7.14. ***Поменять местами значения двух переменных a и b.***

```
void Swap(int &a, int &b)
{
   tmp = a;
   a = b;
   b = tmp;
}
…… int a=10, b=100;
Swap(a,b);
```

КОНЕЦ ЛЕКЦИИ