ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Урок №8

Программирование на языке C

Содержание

1. Введение в мир функций

Необходимость использования. Объявление. Вызов.

Зачастую складывается такая ситуация, когда в нашей программе некоторые отрезки кода повторяются несколько раз и нам приходится много раз подряд набирать один и тот же фрагмент кода. Такая ситуация имеет свои отрицательные стороны. Во-первых, процесс создания программы становится более нудным и длительным, во-вторых, увеличивается объем конечного файла. Как быть в этом случае?! Существует ли какой-нибудь механизм позволяющий автоматизировать действия программиста и сократить код программы?! Да, существует, — отвечаем вам мы. Называется такой механизм — функцией. Итак:

Функция — специальная конструкция, с помощью которой какой-либо фрагмент кода повторяющийся в программе два или более раз выносится за тело программы. Этот фрагмент получает собственное имя и, в дальнейшем, для того, чтобы воспользоваться вынесенным кодом, необходимо будет указать это имя.

Синтаксис объявления

Существует два способа объявления функции:

- Функция объявляется до функции main.
- Функция объявляется с помощью прототипа, а после функции main, описывается тело объявленной функции.

Первый способ: до функции main.

Общий синтаксис объявления функции:

```
возвращаемое_значение имя_функции (параметры)
{
        блок_повторяющегося_кода (тело);
}
```

1. Имя функции подчиняется тем же правилам, что и имя переменной и, естественно, выбирается программистом.

2. Параметры — входные данные, которые необходимы функции для работы над кодом. В качестве параметров используют обычные переменные, указывая для каждого параметра его тип данных. Если функция не нуждается во входных данных, скобки следует оставить пустыми. Второе название параметров — аргументы.

3. Возвращаемое значение — результат работы функции. На место возвращаемого значения подставляется любой из базовых типов. Это тип тех данных, которые функция поставит на место своего вызова в программе. Если функция ничего не возвращает, на место возвращаемого значения подставляется **void(пусто)**. В общем и целом «осмысленное» возвращаемое значение указывается в том случае, если результат работы функции необходим для дальнейших вычислений.

Примечание: Нельзя создавать одну функцию внутри другой.

Примечание: Нельзя вызвать функцию до ее объявления.

Синтаксис вызова функции.

Чтобы воспользоваться функцией на определенном отрезке кода, следует вызвать ее прямо на этом отрезке. Вызов функции является предписанием операционной системе начать выполнение фрагмента кода, который содержится в теле функции. По завершении выполнения функции, программа должна продолжить работу в основном коде с того же места, где была вызвана функция. Вызов функции состоит из указания имени функции, передачи аргументов (если таковые имеются) и получения возвращаемого значения (если есть необходимость его получать):

```
имя_переменной=имя_функции(параметр1, параметр2,...., параметрN);
```

1. Типы данных значений должны соответствовать типам данных аргументов в определении функции. Исключением являются случаи, когда передаваемое значение может быть с легкостью преобразовано к нужному типу.

2. При вызове функции всегда необходимо указывать такое количество параметров, какое было определено при объявлении.

3. Тип переменной, в которую запишется возвращаемое значение, должен совпадать с тем типом, который функция, собственно, возвращает. Это не обязательно, но желательно.

Ключевое слово return

Для возвращения Значения из функции в программу на то место, из которого была вызвана эта функция, используется оператор return. Синтаксис возврата таков:

```
return значение;
```

Если функция не возвращает никаких значений, то оператор return можно использовать просто для остановки функции, для этого просто пишем:

```
return; // в данном случае return отработает для функции, как break для цикла.
```

Запомните важные моменты обращения с return:

1. Операторов возврата может быть несколько (в зависимости от ситуации), но отработает только один из них.

2. Если сработал return (вне зависимости от формы), все что расположено в функции ниже него уже не отработает.

3. Если тип возвращаемый функцией не void, то необходимо ВСЕГДА использовать форму: **return значение**;

Теперь, когда с теоретической частью мы знакомы, переходим к следующему разделу — примеры на создание функций.

2. Примеры на создание и вызов функций

Функция, которая не принимает никаких параметров и не возвращает никакого значения.

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

// создание функции
void Hello(){
        // показ на экран строки текста
        cout<<"Hello, World!!!\n\n";
}
void main(){

        Hello(); // вызов
        Hello(); // вызов
        Hello(); // вызов

}
```

Результат — три раза фраза — Hello, World!!! — на экране.

Функция, которая принимает один параметр, но не возвращает никакого значения.

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

// рисует линию из звездочек длиной count
void Star(int count){
        for(int i=0;i<count;i++)
                cout<<'*';
        cout<<"\n\n";
}

void main(){

        Star(3); // показ линии из трех звездочек
        Star(5); // показ линии из пяти звездочек

}
```

Функция, которая принимает два параметра, но все еще не возвращает никакого значения.

```
#include <iostream>
using namespace std;
// рисует линию из символа - symb, длиной count
 void AnyLine(char symb, int count){
        for(int i=0;i<count;i++)
                cout<<symb;
        cout<<"\n\n";
 }
 void main(){

        AnyLine('+',3); // показ линии из трех плюсов
        AnyLine('=',5); // показ линии из пяти знаков равно

 }
```

Функция, которая принимает два параметра, и возвращает значение.

```
#include <iostream>
using namespace std;
// вычисляет степень (Pow) числа (Digit)
int MyPow(int Digit, int Pow){
        int key=1;
        for(int i=0;i<Pow;i++)
                key*=Digit;
        return key;
}
void main(){
         // запись возвращаемого результата в переменную res
        int res=MyPow(5,3);
        cout<<"Res = "<<res<<"\n\n";

}
```

3. Передача аргументов. Прототипы функций

Передача аргументов по значению.

Поговорим, о том, что происходит в оперативной памяти. Аргументы, которые указываются при определении функции, называются формальными. Это связанно с тем что, они создаются в момент вызова функции в оперативной памяти. При выходе из функции такие параметры будут уничтожены. Поэтому, если в другой функции программы будут параметры с тем же именем, то конфликта не будет. Рассмотрим, один из способов передачи аргументов:

Пример работы формальных параметров при передаче данных по значению.

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

// Должна менять значения переменных местами
void Change(int One, int Two){
        cout<<One<<" "<<Two<<"\n\n";// 1 2
        int temp=One;
        One=Two;
        Two=temp;
        cout<<One<<" "<<Two<<"\n\n";// 2 1
}
void main(){

        int a=1,b=2;
        cout<<a<<" "<<b<<"\n\n"; // 1 2
        // передача по значению
        Change(a,b);
        cout<<a<<" "<<b<<"\n\n"; // 1 2
}
```

1. В функцию передаются не a и b, а их точные копии.

2. Все изменения происходят с копиями (One и Two), при этом сами a и b остаются неизменными.

3. При выходе из функции временные копии уничтожаются.

Исходя из вышеописанного — пока будьте внимательны при обработке значений внутри функции. В последствие, мы научимся решать данную проблему.

Примечание: Кстати, с массивами такого не случается. Все изменения происходящие с массивом в функции — сохраняются при выходе из неё.

Кое-что о массивах...

Некоторую особенность имеет использование массивов в качестве аргументов. Эта особенность заключается в том, что имя массива преобразуется к указателю на его первый элемент, т.е. при передаче массива происходит передача указателя. По этой причине вызываемая функция не может отличить, относится ли передаваемый ей указатель к началу массива или к одному единственному объекту. При передаче одномерного массива достаточно просто указать пустые квадратные скобки:

```
int summa (int array[ ], int size){
     int res=0;
     for (int i = 0; i < size; i++)
        res += array[i];
     return res;
}
```

Если в функцию передаётся двумерный массив, то описание соответствующего аргумента функции должно содержать количество столбцов; количество строк — не-

существенно, поскольку, как говорилось ранее в функцию фактически передаётся указатель.

```
int summa (int array[ ][5], int size_row, int size_col){
     int res=0;
     for (int i = 0; i < size_row; i++)
         for (int j = 0; j < size_col; j++)
                res += array[i][j];
     return res;
}
```

Прототипы функций или второй способ объявления.

При втором способе объявления функции необходимо сообщить компилятору о том, что функция существует. Для этого до main предоставляется имя функции, ее аргументы, а также тип возвращаемого значения. Такую конструкцию называют прототипом функции. Когда компилятор встречает прототип функции он точно знает о том, что функция существует в программе после main — и, она должна там быть.

```
библиотеки
возвращаемое_значение имя_функции(аргументы);
void main(){
        тело main;
}
возвращаемое_значение имя_функции(аргументы){
        тело функции;
}
```

```
#include <iostream>
using namespace std;
// прототипы
void MyFunc();
void MyFuncNext();

void main(){
        MyFunc();       //MyFunc
        MyFuncNext();   //MyFuncNext
}
//описания
void MyFunc(){
        cout<<"MyFunc\n";
}
void MyFuncNext(){
        cout<<"MyFuncNext\n";
}
```

Считается, что такое объявление функции является наиболее красивым и правильным.

4. Область видимости. Глобальные и локальные переменные

Область видимости.

Любые фигурные скобки в программном коде образуют, так называемую область видимости, это означает, что переменные, объявленные внутри этих скобок будут видны только внутри этих скобок. Другими словами, если к переменной, созданной внутри функции, цикла, if и так далее обратится из другого места программы, то произойдет ошибка, так как после выхода из фигурных скобок эта переменная будет уничтожена.

```
int a=5;
if(a==5){
        int b=3;
}
cout<<b; // ошибка! b не существует
```

Глобальные и локальные переменные

Согласно правилам области видимости — переменные делятся на два вида — локальные и глобальные.

Локальные переменные создаются внутри какого — нибудь отрезка кода, что это значит для программы, мы уже знаем.

Глобальные переменные создаются вне всяких областей видимости. Преимущественно до функции main(). Такая переменная видна в любом месте программы. По умолчанию глобальные переменные в отличии от локальных

инициализируются 0. И, главное, те изменения, которые происходят с глобальной переменной внутри функции, при выходе из последней сохраняются.

Примечание: Запомните — если есть, например, глобальная переменная под названием a, а внутри функции определяется переменная под тем же именем, то тогда внутри функции будет использоваться переменная, объявленная внутри этой функции. Поэтому избегайте применения имен переменных, которые незримо уже используются во внешних областях видимости. Этого можно достигнуть, вообще избегая использования в программе одинаковых идентификаторов.

```
int a=23; // глобальная a
void main(){
        int a=7;// локальная a
        cout<<a; // 7, используется локальная
}
```

5. Аргументы (параметры) по умолчанию

Формальному параметру функции может быть задан аргумент по умолчанию. Это означает, что в данный аргумент значение при вызове можно не передавать. В этом случае будет использовано значение по умолчанию.

Общий синтаксис для реализации такого подхода имеет следующий вид:

```
тип_возвращаемого_значения имя_функции(тип_арг имя_арг=значение_по_умолчанию)
```

Здесь **значение_по_умолчанию** и есть значение, присваиваемое аргументу, если он опущен при вызове. Разумеется, аргументов по умолчанию может быть несколько:

```
тип_возвращаемого_значения имя_функции(арг1=значение, арг2=значение)
```

Аргументами по умолчанию могут быть аргументы, начиная с правого конца списка параметров функции и далее последовательно слева направо без перерывов. Например:

```
void foot (int i, int j = 7) ;      //допустимо

void foot (int i, int j = 2, int k) ;        //недопустимо

void foot (int i, int j = 3, int k = 7) ;    //допустимо

void foot (int i = 1, int j = 2, int k = 3); //допустимо

void foot (int i=- 3, int j);                //недопустимо
```

Рассмотрим пример на работу с параметрами по умолчанию.

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

// рисует линию из звездочек длиной count
void Star(int count=20){
        for(int i=0;i<count;i++)
                cout<<'*';
        cout<<"\n\n";
}

void main(){

        Star(); // показ линии из 20 звездочек
        Star(5); // показ линии из пяти звездочек

}
```

Вот и все на сегодня. А теперь — дерзайте!!! Теста сегодня не будет, так что за вами, только домашнее задание. Удачи!!!

6. Домашнее задание

1. Написать функцию, которая получает в качестве аргументов целое положительное число и систему счисления, в которую это число должно переводится (системы счисления от 2 до 36). Например, при переводе числа 27 в систему счисления 16 должно получиться 1B; 13 в 5-ю — 23; 35 в 18-ю — 1H.

2. Игра «кубики». Условие: имеется два игральных кубика со значениями от 1 до 6. Игра происходит с компьютером, кубики бросаются поочередно. Побеждает тот, у кого сумма выпавших очков по итогам пяти бросков больше. Предусмотрите возможность получения первого хода человеком или компьютером. Кубики отображаются с помощью символов. В конце игры необходимо выводить среднюю сумму по броскам для обоих участников.