Lecture 10
Functions (Subprograms) in C Language

As we considered before any text of C/C++ program represents a collection of functions having at least one function named function **main( ).**

A **function** (common name used in C/C++ languages for a **subprogram**) is a self-contained unit of a program that evaluates a particular algorithm for sub-problem of a given problem. The place of a function evaluation in the program is named the **function call**. A function can be called from some place of the **main( )** function or another function of the program. But function **main( ),** existing in any C/C++ program, can be called only by operating system when a given program starts running. A function which calls another function is named a **calling function** and a function which is called by another function is named a **called function.**

After calling a function control of running program goes to that function and the code within the body of the called function is executed, and when the function has finished executing, control returns to the point of the program at which that function was called. When execution of function **main( )** is terminated control returns to the operating system.

Function fundamentals

In C/C++ languages are used three main notations related to a function:

1. function definition (function code, text of function);

2. function declaration (prototype of function or **interface of function)** and

3. function call.

1. When you create a function by writing the **code of function (function definition)** you need to specify:

a) the **function header** as the first line of the function definition.

b) the **function body** following the function header which represents an executable code of the function, enclosed between braces.

The **function header** defines: a) the **type** for the value that the function determines and returns back to the calling function (is shorter called type of the function). Some functions do not return value and in this case they have type **void**; b) the **name** of the function; c) the **list of function parameters** (defines, what types and values can be passed to the function when it's called), enclosed in round brackets (the list of function parameters is also named the **function signature**). Some functions can have an empty list of parameters.

The **function body** determines what operations the function performs on the values that are passed to it.

Any function can have one or more return statements in its body but only one of them will be performed.

A function of void type can have return statements without return value.

Code of a function (function definition) can be written at any place of program which represents a collection of codes of functions and, if necessary, also declarations of functions (see p.2) or/and **global variables** (which are variables defined out of function definitions and valid for all program).

2. When using functions we make a distinction between a function **definition** and a function **declaration**.

A **function declaration** is a statement that defines the essential characteristics of a function. It defines its return value type, its name and the type of each of its parameters. A function declaration is also called a **function prototype** or a **function interface,** because it provides all the external specifications for the function. Usually, a function prototype is used when the call of that function is situated in the program before the text of function.

Ex.: Prototype of function **mult( ):**

int mult (int x, int y); or **int mult (int , int);**

3. **A function call** can be written at the corresponding place of the program where it is necessary to evaluate that function and consists of two parts: a) **function name** and b) **list of arguments** enclosed in round brackets following the function name. An argument of a function is any possible expression or constant value. Note that types, order and number of actual arguments have to be strictly the same as types, order and number of formal parameters of function. If the parameter list of a function is empty than the list of arguments of that function is also empty, still the name of function is followed by opening and closing round brackets.

There are two kinds of the function calls: a) function call as expression and b) function call as statement.

A function call as expression has a value which represents the return value of that function and can be used in program as any another value is used, for example, can be assigned to a variable or can be an operand of another expression.

So, a call of a function which has **void** type can not be used as an expression , it can be used only as a function call statement, but a call of a function which type is **not void** can be used as an expression and as a function call statement.

Ex.: Function calls as expressions: **product = mult(a,b); var = d*sin(2*x); y = 5 – 6*pow(3, 3);**

Function call as statements: **printf("%d\n", a);**

Example of program which determines the product of two integer numbers introduced from keyboard:

```
#include <stdio.h>
int  mult( int x, int y);
 int  main( )
 {
   int a, b, product;
   clrscr( );
   printf(" Enter two integer numbers :   ");
   scanf("%d%d", &a, &b);
   product = mult(a, b);
   printf("\n Result:   %d \n" , product);
   return 0;
  }
  int  mult( int x, int y)
   {
      int pr;
      pr= x*y;
      return pr;
   }
```

Lecturer: Mihail Kulev, associate professor