# Chapter 10. Functions in C++. Function parameters. Links and indexes. Function and variable modifiers

## 10.1 Functions in C++

Functions are named blocks of code that are designed to perform specific operations. Functions serve as a key tool for structuring program code. Functions work in such a way that input data is placed in them, processed, and the output returned from the function is derived from it.

A function declaration, which can also be referred to as a prototype, header, or signature, specifies its name, return type, and list of parameters to be passed. A function definition contains, in addition to the declaration, the function body, which is a sequence of operators and descriptions in curly braces. Figure 10.1- The syntax of the function definition is presented.

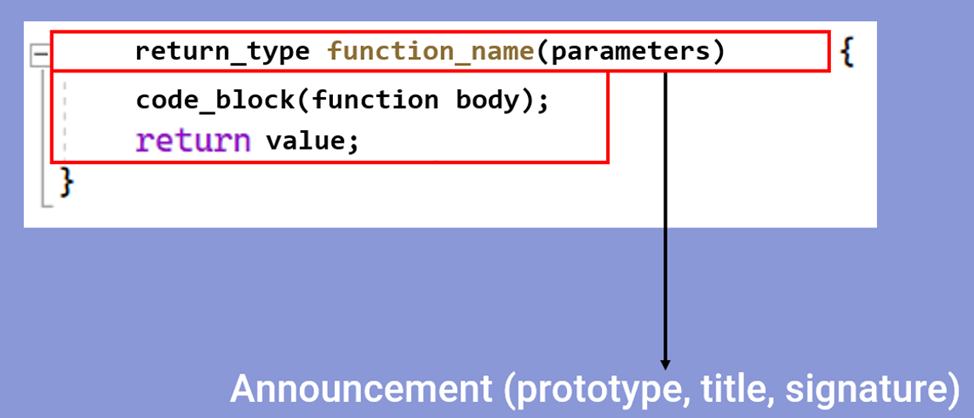


Figure 10.1 – Structure of the declaration and function definitions

## 10.2 Function main()

The necessary function that is written first is the "main()" function. This function marks the starting point of the program execution. The return value must be of an integer type. The standard provides for two function formats: with parameters and without parameters (figure 10.2). The parameter names in the program can be anything, but it is common to use argc and argv. Argc takes a number of command-line arguments, and argv takes an array of strings that contains command-line arguments [17].

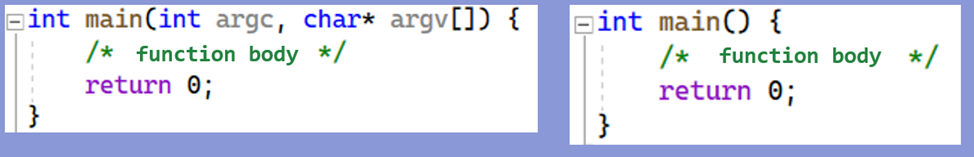


Figure 10.2 – Types of the main() function

## 10.3 Function Return Type

The return type of a function determines the type of data that the function returns after it is executed. This type can be any built-in or user-defined data type. When the function exits, it returns the value of a particular data type back to the calling part of the program. The mechanism for returning from a function is implemented using the return operator. In the example shown in figure 10.3, the function calculates the sum of two integers, accordingly, its return value will be of the int type.

You cannot return a pointer to a local variable from a function, because the memory allocated by local variables when you enter the function is freed after returning from it.

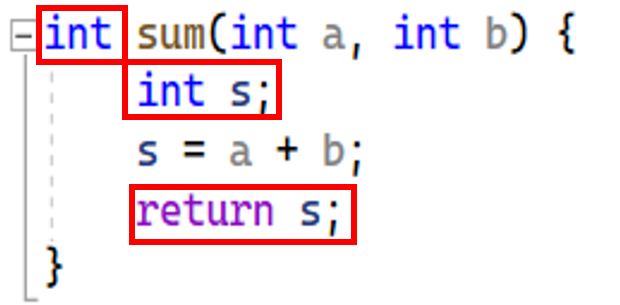
****

Figure 10.3 – The sum of integer parameters of type a and b is returned by the sum of integer parameters of type a, b.

A function of type void is used in the declaration to indicate that the function does not return a value. Such functions perform certain actions or operations, but do not provide a result in the form of a value. In such functions, the return statement can be omitted if the return occurs before the closing curly brace. Figure 10.4 shows that the function calculates the sum of two numbers, but does not return a value, but immediately outputs the result to the console.

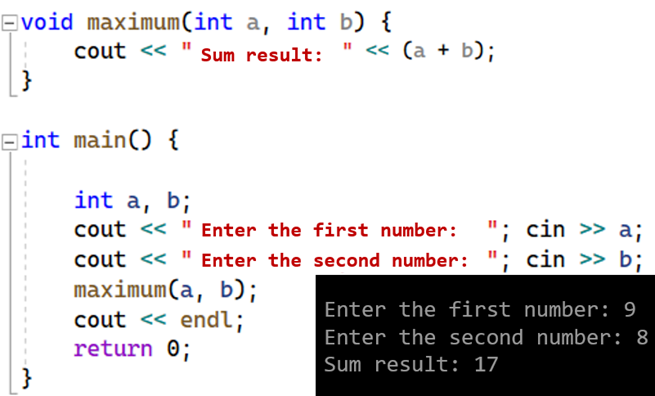


Figure 10.4 – Example of a void function

## 10.4 Function Parameters

Function parameters are variables that are passed to the function when it is called. Function parameters pass data to the function that the function works with. Parameters are defined in the function header and can be used inside the function body.

The parameters listed in the header are called formal parameters, or simply parameters, and those written in the function call statement are called actual parameters, or arguments.

The names of formal and actual parameters may not match, but their number and types must match [33]. In figure 10.5, the actual parameters are first and second, and when passed to the function, these parameters are substituted for the parameters a and b. The value of first takes the place of a, the value of second takes the place of b.

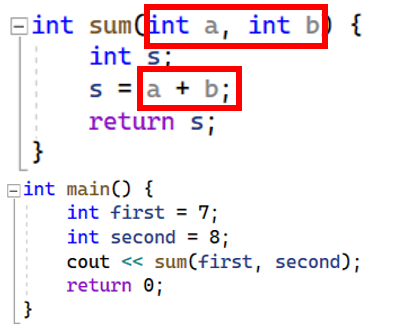


Figure 10.5 – Example of using function parameters

## 10.5 Passing Parameters to a Function

There are two ways to pass parameters to a function: by value and by address (passing to an address is done using references or pointers).

When passing *by value,* function operators work with *copies of arguments*. The function does not have access to the original values of the parameters, and, therefore, there is no way to change them. Therefore, after the function is completed, the values of the arguments in the computer's memory remain original.

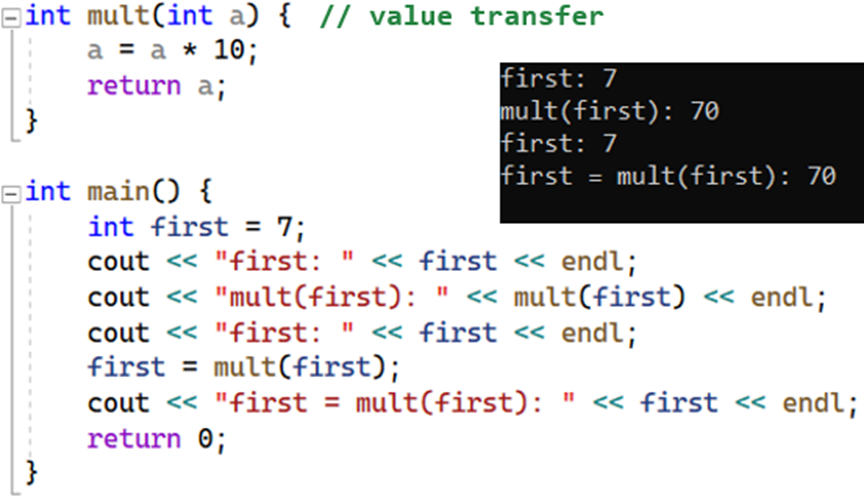


Figure 10.6 – Example of passing parameters to a function by value

When passed *to an address*, the function accesses *memory locations* at those addresses and can change the original values of the arguments. When passed by reference, the address of the parameter specified when the parameter was called is passed to the function, and within the function, all accesses to the parameter implicitly access or dereference the parameter value [32]. Dereference. Examples of passing parameters to an address are shown in figures 10.7 and 10.8.

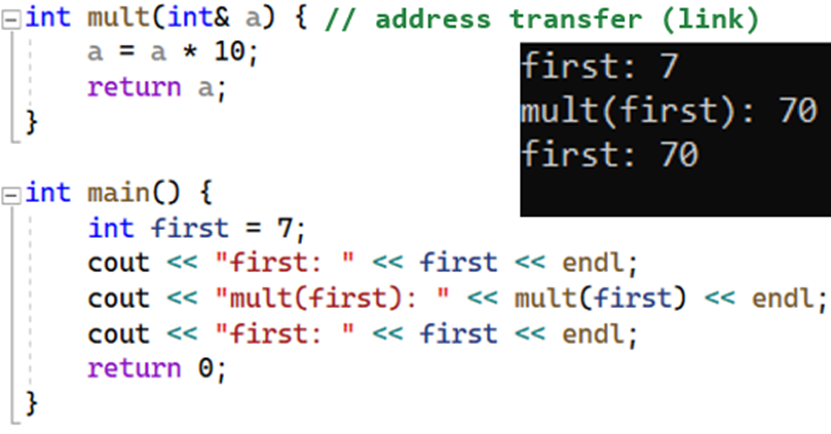


Figure 10.7 – Example of passing parameters to a function at (link)

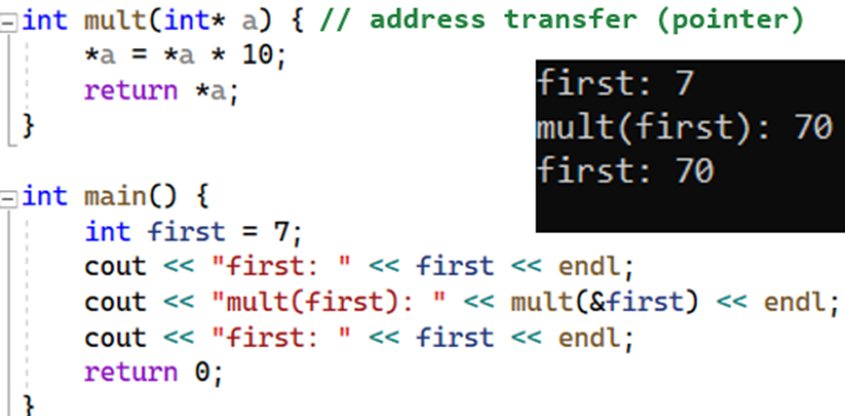


Figure 10.8 – Example of passing parameters to a function by address (pointer)

When an array is used as a parameter, a pointer to its first element is passed to the function. In this case, information about the number of elements of the array is lost, and its dimension should be passed through a separate parameter (figure 10.9).

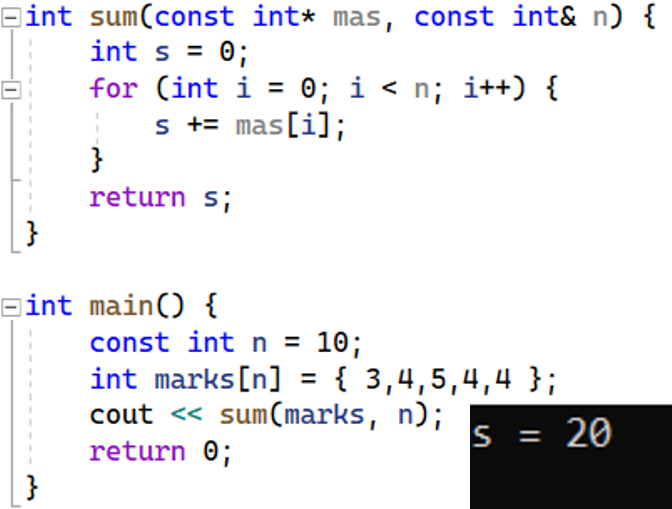


Figure 10.9 – Example of passing an array to a function

## 10.6 Function modifiers

Function modifiers are special keywords that change the behavior of functions and how they interact with data and calling code. Modifiers allow you to further customize the functions according to the requirements of the program.

The "inline" modifier provides a recommendation to the compiler to embed the function code directly at the point of the call, instead of the normal function call. This can improve performance, reduce the overhead of calling a function, and make your code more efficient. The "inline" modifier is often used for small and simple functions (figure 10.10).

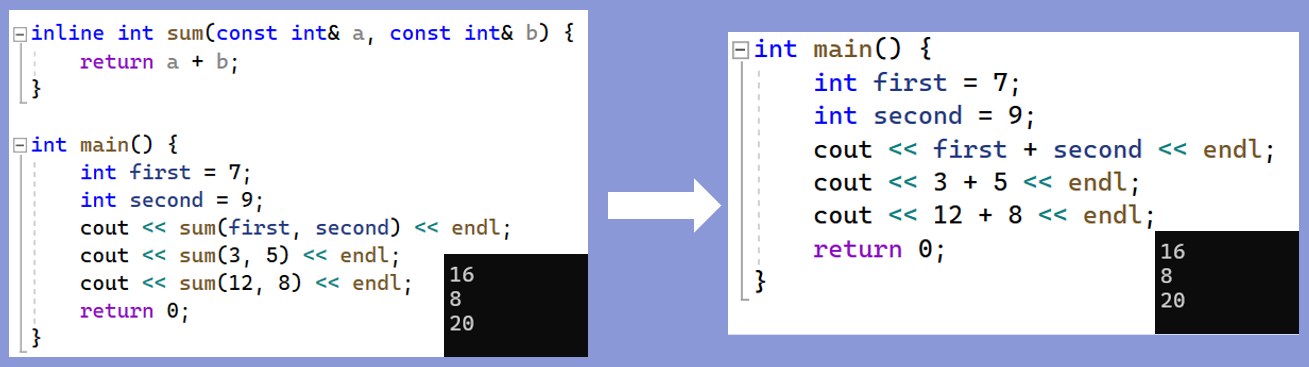


Figure 10.10 – Example of a function with the "inline" modifier

The static modifier limits the scope of a feature to only the current file in which it is defined. Such a function is called a "file function" or "local function". A file function with the "static" modifier is available only within the current file and cannot be used from other files (Figure 10.11). If the same function with the "static" modifier is defined in different files, each of these files will have its own copy of the function, and they will not communicate with each other.

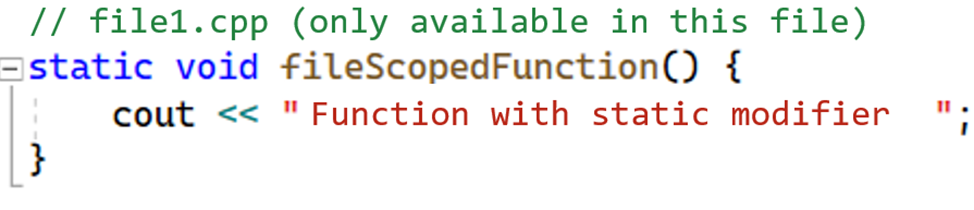


Figure 10.11 – Example of a function with the "static" modifier

The "extern" modifier allows you to declare functions in one file and use them in other files (figure 10.12). This is useful for separating code and creating more organized and scalable projects. However, it should be borne in mind that the use of "extern" may require attention to the correct order of compilation and linking of files. The "extern" modifier can also be used when the function prototype is in one file and the declaration is in another.

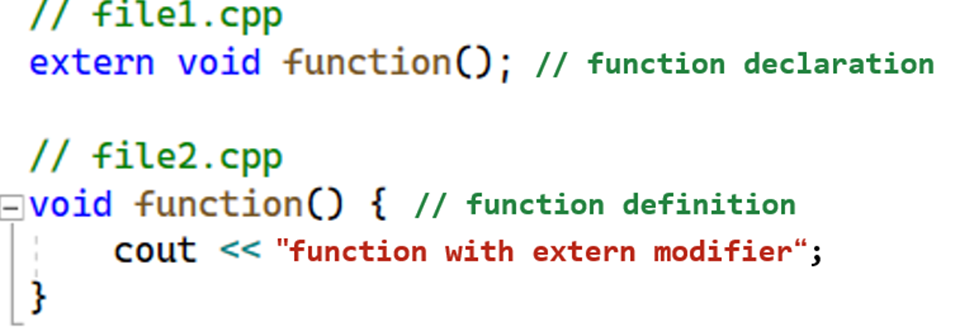


Figure 10.12 – Example of a function with the "extern" modifier

## 10.7 Variable modifiers. Static modifier

Also, in addition to functions, some of the modifiers can be used for variables. All quantities described within the function, as well as its parameters, are local. Their scope is a function. The values of local variables between calls to the same function are not persisted. If this is to be avoided, the "static" modifier is used when declaring local variables. In figure 10.13, the static variable n is placed in a data segment and initialized once the first time a statement containing its definition is executed. An automatic variable or a variable with the "auto" modifier m is initialized each time a function is entered. The automatic variable is initialized each time it enters a loop block. The "auto" modifier is set by default when a variable is declared, and means that the variable will be of a compiler-defined type based on the initializer, making the code cleaner and more flexible.

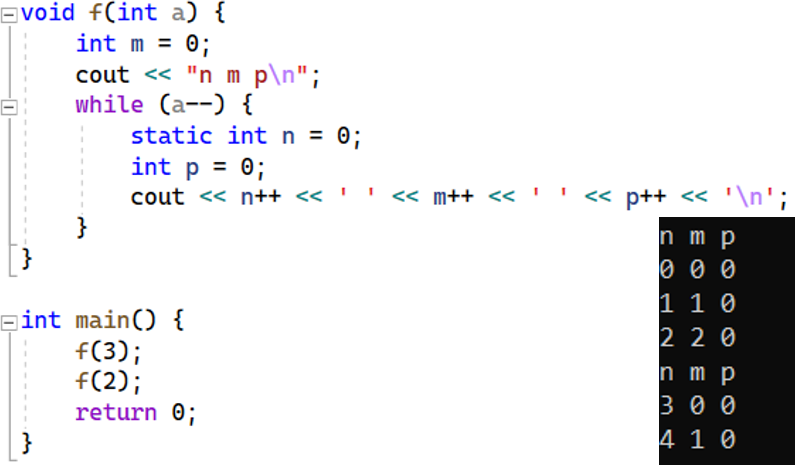


Figure 10.13 – Example of a variable working with the "static" modifier