Functions returning integers

In the Introduction we mentioned the notion of modularity, and how, especially in the case of large programs, can benefit from breaking them down into modules. Among the several ways to modularize programs, we introduced one example of **functions**.

For this section, our program will calculate the quotient of two integers. Admittedly, this is not a programming breakthrough, but it affords us the opportunity to learn a little bit more about functions.

The structure of the program can be described as follows:

1. The main function.
2. A function named quotient, written by us.

    a) The input to the function quotient will be an integer value

    b) The output will also be an integer value, the result of dividing the fiorst integer by the second.

The code for the function is:

```
1 int divide (int x, int y)

2 {

3 int result;

4 result = x / y;

5 return result;

}
```

The code above was written in the simplest possible way, and therefore is not the most efficient. In addition, line numbers were added to facilitate the explanation. Line numbers are not used when writing actual programs.

Line number 1 is the **header** of the function. The first word specifies the type of value the function will return to the calling program. It is followed by the name we gave to the function, and then, in parenthesis, the inputs, and their types, that our function expects. These are called **parameters.** In our case, the function expects to be passed two integers when it is called.

One observation worth making is that we have designed function divide without any knowledge of the values that will be divided. We simply write functions in general, using named parameters instead of actual values, and we describe (code) what operations to perform with those values to achieve the result.

Our function proper starts after the header. The body of the function is enclosed in { }. That's where the code goes. In our case, line 3 is the beginning of the code. We start by defining a variable result to hold, you guessed it, the result of our operation. The name *result* was chosen by us, and could have been a different one.

The next line (4) is where the action takes place: the two numbers sent to the function are divided and their result is assigned to *result*. Inside the function, the names *x* and *y* stand for the values sent to our function; these names were chosen by

us when writing the header (line 1), and could have been different without affecting the behavior of the program.

Now we will see how to use our function in a full program. Our main function (mandatory in any program) will be:

```
#include <iostream>
```

using namespace std;

```
1 int divide (int, int);


2 int main (void) {


3 cout << divide(7, 2);


4 return 0;


5 }
```

This part of our program contains an **include directive** used to, of course, include the module definitions provided by C++ to provide input/output capabilities. This is a perfect example the use of tools and modules programmed by others.

The name iostream is the name of a file containing the code needed for us to be able to use *cout* << to produce output. This file is standard in C++, and will have to be included any time we intend to send output to the screen or get input form the keyboard.

Line 1 is called a function prototype. A prototype is used to describe the "characteristics" of the functions we intend to use in our programs. In this case, since function *divide* will be used, we include its prototype, which is very similar to the function header. The only real distinction is that the prototype does not give names to the parameters, it's simply a description which could be verbalized as: *I intend to use a function called divide which returns an integer and expects to integers to be passed to it.*

Line 2 contains the definition of main, as we presented it in previous sections. Line 3 calls our function divide and sends the returned value to the screen.

Of course, for our program to work, we'll have to put these pieces together, so that when line 3 is executed, our program can jump to function *divide*, calculate its result and return it to line 3 so it can be sent to the screen (cout) by the << operation.

Now we will see the complete program.

The Whole Picture

The full code follows.

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

1 int quotient (int, int);

2 int main (void) {

3 cout << divide(7, 2);

4 return 0;

5 }

//---------------------------------
```

```
6 int divide (int x, int y)

7 {

8 int result;

9 result = x / y;

10 return result;

11 }
```

As before, line numbers are included only for reference purposes. Notice the use of // preceding a **comment**, that is, a part of the program that is not supposed to be compiled, and therefore can be used to place notes, explanations or anything we want: in our case, a dividing line to separate visually (no programming effect) our main function from our divide function.

Now we can see the structure of the program as described in the outline of the preceding section.

Following this structure we can write programs that use several functions written by us. Once we have identified the parts of our programs we want to modularize as functions, and once those functions have been written, we just write in their prototypes (usually before the main function) and then the function definitions after the code for main.

We'll see the logic behind this in the next section.