Lab 9a - CMPS 1044 Computer Science I

Functions Part II

Lesson objectives:

- Demonstrate the use of a simple function definition and call
- Demonstrate the correct use of function prototypes
- Demonstrate functions calling other functions

Recall from Lab 9 that a **function** is an independent program designed to accomplish a task. Functions are reusable blocks of code that can make a programmer more efficient.

Any function can access/call any other function (if programmed appropriately). The most common use is **main()** calling other functions. That will be our emphasis in this lab.

Part 1: Functions in C++

Let us create a function named char_printer. This function will accept two arguments: a char and an int. The char will be printed n times, where n is the int passed in the function call. The function will not return any data, so the return type will be void.

To make a function work, three items are needed:

1. A function **prototype** above main()
2. A function **definition** below main()
3. A function **call** somewhere in the program

1. The Prototype

A function **prototype** is a placeholder. Its job is to let the compiler know information about a function that will appear later in the program. Without a **prototype**, a function must be defined before any calls to the function are made.

The **prototype** is placed after `using namespace std;` but before **main()**.

Anatomy of a prototype: <return type> function_name(<data type>);

Our char_printer function would have the following **prototype**:

```
void char_printer(char, int);
```

2. The Definition

The function **definition** is placed below **main()**. A function **definition** contains statements that make up the function, which includes the return type, function name, parameter list, and the body. Note the absence of the semicolon at the end of the function **header**.

```
void char_printer(char c, int n)
{
	for (int i = 0; i < n; i++)
		cout << c;
	cout << '\n';
}
```

3. The Call

A function is executed when it is **called**. If needed, it is necessary to pass literal values or variables to the function as an argument(s). Keep in mind that some functions do not accept any arguments. The function **call** does not include a return data type such as int, float, double, etc.

The char_printer function can be **called** in several different ways.

```
char_printer('x', 12); // Prints x 12 times
```

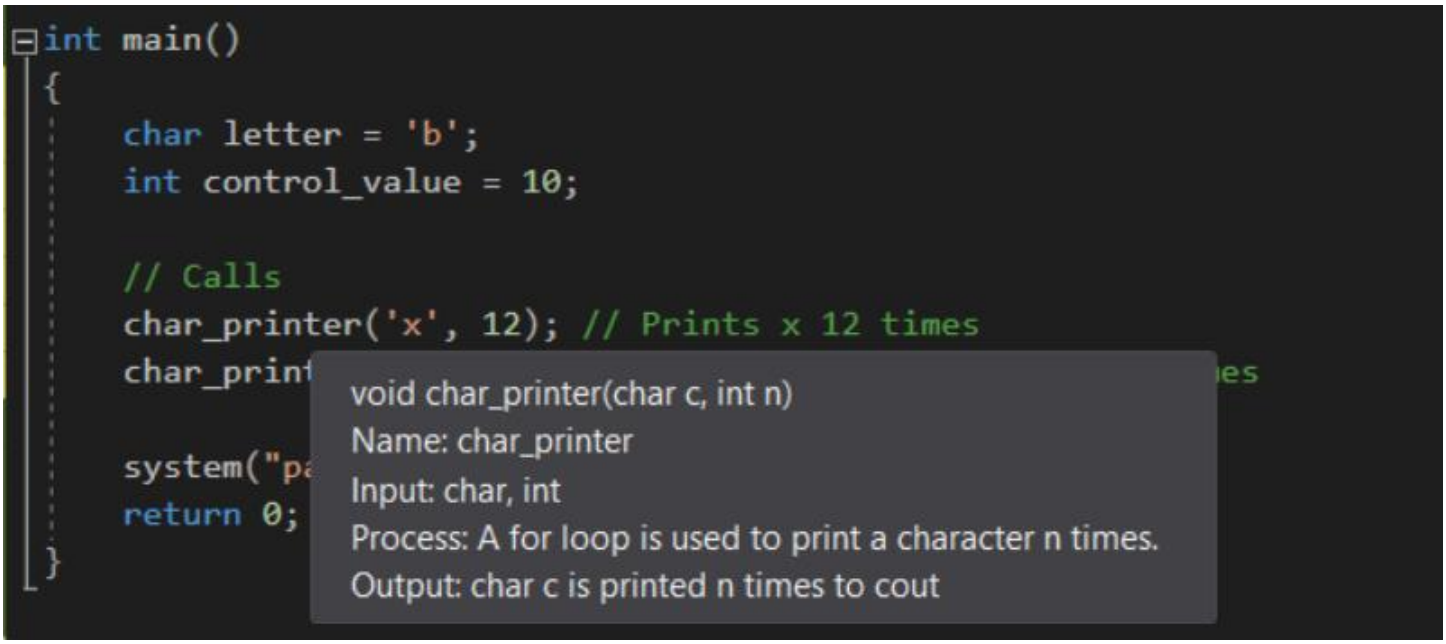
To pass a variable to a function as an argument, the variable must be declared and given some value before the function **call** is made.

```
char letter = 'b';
int control_value = 10;
char_printer(letter, control_value); // Prints b 10 times
```

Don't forget to comment your function using the template from Lab 9.

```
// Name: char_printer
// Input: char, int
// Process: A for loop is used to print a character n times.
// Output: char c is printed n times to cout
void char_printer(char c, int n)
{
	for (int i = 0; i < n; i++)
		cout << c;
	cout << '\n';
}
```

Visual Studio can use the documentation you provide to give helpful information when writing code. For example, scrolling over a function call will reveal documentation written by the programmer.



The following code is the full program that implements the char_printer function we just used:

```
#include <iostream>

using namespace std;

// Prototype
void char_printer(char, int);

int main()
{
	char letter = 'b';
	int control_value = 10;

	// Calls
	char_printer('x', 12); // Prints x 12 times
	char_printer(letter, control_value); // Prints b 10 times

	system("pause");
	return 0;
}

// Name: char_printer
// Input: char, int
// Process: A for loop is used to print a character n times.
// Output: char c is printed n times to cout
void char_printer(char c, int n)
{
	for (int i = 0; i < n; i++)
		cout << c;
	cout << '\n';
}
```

Part 2: Functions calling other functions

Let us explore the idea of functions calling other functions using a simple program that contains functions designed to output **cout** statements. Copy this code into a Visual Studio project and trace the execution path. Note the use of function prototypes above **main()** and the location of the function definitions below **main()**.

```
#include <iostream>

using namespace std;

// Prototypes
void func1();
void func2();

int main()
{
	cout << "Inside main(). Calling func1().\n";
	func1();
	cout << "Back inside main()\n";

	system("pause");
	return 0;
}

void func1()
{
	cout << "Inside func1(). Calling func2()\n";
	func2();
```

```
	cout << "Back inside func1(). We have returned from func2()\n";
}

void func2()
{
	cout << "Inside func2()\n";
}
```

The **cout** statements are used to help aid in following the execution path. Note that when a function is called, it executes its code then returns to the calling function. This program's path is as follows:

main() -> func1() -> func2() -> func1() -> main()

Part 3: Lab Assignment

Write a program that implements a calculator. Include a function that prompts the user for two numbers and uses a switch statement to show a menu of mathematical operations to choose from. Based on the user's selection, call a separate function for addition, subtraction, multiplication, or division. Once the answer is calculated, use a separate function to print the answer to the screen.