Lab # 11: Functions - I

EC-102 – Computer Systems and Programming

Usman Ayub Sheikh

School of Mechanical and Manufacturing Engineering (SMME),
National University of Sciences and Technology (NUST)

November 2, 2015

Outline

What are Functions?

- Groups a number of program statements into a unit and gives it a name
- This unit can then be invoked from other parts of the program
- So far, we have studied only one function i.e. `main()` function

Why Use Functions?

- To aid in the conceptual organization of a program
- To reduce program size
- The function's code is stored in only one place in memory

Why Use Functions?

Write a C++ program that displays the following table to the console

```
1 ****************************************
2 Data-type  Range
3 ****************************************
4 char       -128 to 127
5 short      -32,768 to 32,767
6 int        System dependent
7 long       -2,147,483,648 to 2,147,483,647
8 ****************************************
```

A Simple Function

```
1  // demonstrates a simple function
2  #include <iostream>
3  using namespace std;
4
5  void starline(); // function declaration
6
7  int main()
8  {
9      starline();
10     cout << "Data-type  Range" << endl;
11     starline(); // call to function
12
13     cout << "char     -128 to 127" << endl;
14     cout << "short    -32,768 to 32,767" << endl;
15     cout << "int      System dependent" << endl;
16     cout << "long     -2,147,483,648 to 2,147,483,647" << endl;
17     starline();
18
19     return 0;
20 }
```

A Simple Function

```
21 // function definition
22 void starline() // function declarator
23 {                // function body starts here
24     for(int j = 0; j < 45; j++)
25     {
26         cout << "*";
27     }
28     cout << endl;
29 }
```

The Function Declaration

- Also known as a prototype
- Just as a variable can't be used without telling the compiler what it is, functions also need to be declared before they are called

```
1 void starline();
```

- Tells the compiler that at some later point we plan to present a function called `starline`
- The keyword `void` specifies that the function has no return value and the empty parentheses indicate that it takes no arguments
- Is terminated by a semicolon
- The information in the declaration is also sometimes referred to as the function **signature**

Calling the Function

- The function is called/invoked as follows:

```
1 starline();
```

- The function name followed by parentheses
- The syntax is very similar to that of the declaration except that the return type is not used
- Terminated by a semicolon
- Executing the call statement causes the function to be executed i.e. control is transferred to the function

The Function Definition

- The definition contains the **actual code** for the function
- Here goes the definition for `starline()`:

```
1 void starline()
2 {
3     for(int j = 0; j < 45; j++)
4     {
5         cout << "*";
6     }
7     cout << endl;
8 }
```

- The definition consists of a line called the **declarator**, followed by the function **body**
- The function body is composed of the statements that make up the function and is delimited by braces
- The declarator must agree with the declaration and is **not** terminated by a semicolon

Comparison with Library Functions

- For a library function, we don't need to write the declaration or definition

```
1 pow(2, 4);
```

- The declaration is in the header file specified at the beginning of the program e.g. `cmath` for `pow` function
- The definition is in a library file that's linked automatically to our program when we build it

Passing Arguments to Functions

- An argument is a piece of data passed from a program to the function
- Arguments allow a function to operate with different values
- By using arguments, we can create a function that prints any character any number of times

Passing Constants as Arguments

```
1  // demonstrates function arguments
2  #include <iostream>
3  using namespace std;
4
5  void rep_char(char, int); // function declaration
6
7  int main()
8  {
9      rep_char('-', 45);
10     cout << "Data-type  Range" << endl;
11     rep_char('-', 45); // call to function
12
13     cout << "char     -128 to 127" << endl;
14     cout << "short    -32,768 to 32,767" << endl;
15     cout << "int      System dependent" << endl;
16     cout << "long     -2,147,483,648 to 2,147,483,647" << endl;
17     rep_char('=', 45);
18
19     return 0;
20 }
```

Passing Constants as Arguments

```
21 void rep_char(char ch, int n)
22 {
23     for(int j = 0; j < n; j++)
24     {
25         cout << ch;
26     }
27     cout << endl;
28 }
```

Passing Variables as Arguments

```
1  // demonstrates function arguments
2  #include <iostream>
3  using namespace std;
4
5  void rep_char(char, int); // function declaration
6
7  int main()
8  {
9      char chin;
10     int nin;
11
12     cout << "Enter a character: ";
13     cin >> chin;
14     cout << "Enter number of times to repeat it: ";
15     cin >> nin;
16
17     rep_char(chin, nin);
18
19     return 0;
20 }
```

Passing Variables as Arguments

```
21 void rep_char(char ch, int n)
22 {
23     for(int j = 0; j < n; j++)
24     {
25         cout << ch;
26     }
27     cout << endl;
28 }
```

Passing Structures as Arguments

```
1  // demonstrates passing structure as argument
2  #include <iostream>
3  using namespace std;
4  struct Distance
5  {
6      int feet;
7      float inches;
8  };
9
10 void engldisp(Distance);
11
12 int main()
13 {
14     Distance d1, d2;
15     cout << "Enter feet: "; cin >> d1.feet;
16     cout << "Enter inches: "; cin >> d1.inches;
17     cout << "\nEnter feet: ";
18     cin >> d2.feet;
19     cout << "Enter inches: ";
20     cin >> d2.inches;
```

Passing Structures as Arguments

```
21      cout << "\nd1 = ";
22      engldisp(d1);
23      cout << "\nd2 = ";
24      engldisp(d2);
25      cout << endl;
26      return 0;
27  }
28
29  void engldisp(Distance dd)
30  {
31      cout << dd.feet << "\'-" << dd.inches << "\"";
32  }
```