Lab 8
Functions

When we build a house why do we make rooms? In our daily life why do we use so many different people who specialize in different professions, such as physicians, lawyers, accountants, etc? Same reasoning can be applied to programming. When we write a good program, it must be divided into modules; each module is designed to do a particular task. In C++ there are two types of modules, functions and classes. In this chapter we will deal with functions. You can write functions that can be used over and over again in a program or in several programs. In fact, the C++ standard library comes with many functions for mathematical, string, and character manipulations.

A large programming job needs to be broken down to manageable smaller modules; this breaking down process is called top down design. Here is a popular poem written using functions. I have broken down the poem into four functions, four verses and one chorus. The main function simply calls the other functions. Even though the program is given in two text boxes for clarity, please combine them into one program.

Program 8-1

```
/*****************************
 Program: Function Calls
Written by: Dr. John P. Abraham
Instructional objective: Functions
*****************************/

#include <iostream>
using namespace std;

void chorus();
void verse1();
void verse2();
void verse3();
void verse4();

int main()
{
  verse1();
  chorus();
  verse2();
  chorus();
  verse3();
  chorus();
  verse4();
  chorus();
```

```
return(0);
}
```

Program 2-1 continued:

```
void chorus()
  {
       cout <<"\nTwinkle, twinkle, little star,";
       cout <<"\nHow I wonder what you are! ";
       cout << "\nUp above the world so high, ";
       cout << "\nLike a diamond in the sky!\n";
  }

void verse1()
{
cout <<"\nWhen the blazing sun is gone," ;
cout <<"\nWhen he nothing shines upon,";
cout <<"\nThen you show your little light,";
cout <<"\nTwinkle, twinkle, all the night. \n";

}

void verse2()
{
cout <<"\nThen the traveler in the dark, ";
cout <<"\nThanks you for your tiny spark, ";
cout <<"\nHe could not see which way to go,";
cout <<"\nIf you did not twinkle so. \n";

}

void verse3()
{
cout <<"\nIn the dark blue sky you keep, ";
cout <<"\nAnd often through my curtains peep, ";
cout <<"\nFor you never shut your eye, ";
cout <<"\nTill the sun is in the sky.\n ";

}
void verse4()
{
cout <<"\nAs your bright and tiny spark, ";
cout <<"\nLights the traveler in the dark,";
cout <<"\nThough I know not what you are, ";
cout <<"\nTwinkle, twinkle, little star. \n";
}
```

Program Run 8-1

```
When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
```

```
Like a diamond in the sky!

Then the traveler in the dark,
Thanks you for your tiny spark,
He could not see which way to go,
If you did not twinkle so.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky!

In the dark blue sky you keep,
And often through my curtains peep,
For you never shut your eye,
Till the sun is in the sky.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky!

As your bright and tiny spark,
Lights the traveler in the dark,
Though I know not what you are,
Twinkle, twinkle, little star.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky!
```

When we run this program, the poem is displayed verse1, chorus, verse2, chorus, and so on. Look at the main: it is not cluttered. In the main, the functions were **called**. The function *chorus* was called four times; all other functions were called only once. The main function **returned** a zero to the operating system indicating all is well; all other procedures returned nothing. Before calling the functions, the **prototypes** of the functions must be declared. Please underline the prototypes just above the main in order for you to remember to do it when you write your programs. The function prototype contains the following information: (1) name of the function, (2) the type of data returned by the function, and (3) number, type, and order of parameters passed into the function. Let us write another program to find centigrade given Fahrenheit.

Program 8-2

```
/************************************
 Program: Function Calls
```

```
Written by: Dr. John P. Abraham
Instructional objective: Passing Parameters
************************************/


#include <iostream>
using namespace std;

float celsius(float);

int main ()
{
  float c, f;  //Centigrade and Fahrenheit
  cout << "Enter today's temperature in degrees of Fahrenheit ";
  cin >> f;
  c = celsius(f);
  cout << "That equals " << c << " degrees of celsius!\n" ;
  return(0);
  cout << "Press the Enter key to stop the Program!";
 getchar();
}

float celsius (float f)

{
 return ((f-32)*5/9);
}
```

Program Run 8-2

```
Enter today's temperature in degrees of fahrenheit 92
That equals 33.3333 degrees of celcius!
```

The function prototype is given immediately after the include statement. The prototype indicates that the function-celsius will receive one float parameter and will return one float value back. In the main variable c is assigned the result of the function-celsius. It is very important that you understand this concept of passing parameters and returning results. Please type this program in as many times as required until you can do it without referring to the notes.

Next we need to concentrate on the **scope of variables**. This world has lots of people called John. So when someone calls John, which John should answer? In a program, you may use the same identifier name in different functions. There needs to be some rules about the scope of the variables. *The scope of variables concept is very important, please pay attention to every sentence in the following few paragraphs*. When a parameter is passed in, as in the example in program 8-2, the contents of that memory location

(referred to as f in this example) is sent to the called function (function-celsius). This variable exists in that function only during the execution of that function. If that function is called again, the old value is gone and whatever is passed is placed there. Run the following program to understand this concept.

Program 8-3

```
/*****************************
 Program: Function Calls
Written by: Dr. John P. Abraham
Instructional objective: Scope of variables
*****************************/
#include <iostream>
using namespace std;

void changex (int);
int main ()
{
  int x;
  x = 5;
  cout << "In the main, value of x before the first call ----> " << x <<"\n";
  changex(x);
  cout << "In the main, value of x after the first call -----> " << x <<"\n";

  x = 1;
  cout << "In the main, value of x before the second call ----> " << x <<"\n";
  changex(x);
  cout << "In the main, value of x after the second call -----> " << x <<"\n";
 return (0);
}

void changex (int x)
{
 cout << "    in the function, value of x as it came in----> " << x << "\n";
 x = x+10;
 cout << "    in the function, value of x after adding 10---> " << x << "\n";
}
```

Program Run 8-3

```
In the main, value of x before the first call ----> 5
    in the function, value of x as it came in----> 5
    in the function, value of x after adding 10---> 15
In the main, value of x after the first call -----> 5
In the main, value of x before the second call ----> 1
    in the function, value of x as it came in----> 1
    in the function, value of x after adding 10---> 11
In the main, value of x after the second call -----> 1
```

When running this program please notice that x is first assigned value of 5 and this value is passed into the function as a copy (now x has a copy). The x is changed in the function to 15 by adding a 10. However after the function is executed, everything in the function including the x in the function, is erased. The original x in the main still has the value of 5. There is a way to keep the values of variables in a function. This is done by declaring the variables as **static**. This will not be discussed further here.

When a function is called and copies of the parameters are passed in as discussed in the previous paragraph, it is known as **call-by-value**. Another way to pass a parameter is to call-by-reference. When a parameter is passed **call-by-reference** the address of the original memory location is passed to the function. The function may change the contents of that memory location. The advantage here is that by not duplicating large blocks of data, memory can be saved. A disadvantage is accidentally changing (**side effect**) the value of a variable.

Now let us study the concept of **global variables**. A variable that is visible to a lower level module is called a global variable. In the program 8-4, x is declared as a global variable. This variable is visible by all modules and any module can change its value.

Program 8-4

```
/*****************************************
 Program: Function Calls
Written by: Dr. John P. Abraham
Instructional objective: Global variables
*****************************************/

#include <iostream>
using namespace std;

int x;
void getvalue ();
int main ()
{
  x = 1;
  getvalue();
  cout << "x contains a value of " << x <<"\n";
  return(0);
}

void getvalue()
 {
  cout << "enter a value for x ";
  cin >> x;
 }
```

Program Run 8-4

```
enter a value for x 8
x contains a value of 8
```

Whatever you entered for the x in the function was displayed in the main; the function placed a value in x and the main was able to access that value. You can see the problems global variables can cause if you are not extremely careful. Therefore, use of global variables should be avoided unless you are a seasoned programmer.

It is time for us to examine the concept of **local variables**. Suppose a global variable x exists, and you declared another x in a function, and now you want to assign a value to x. Which x should get the value? Let us modify program 2-4 to include a local variable.

Program 8-5

```
/*****************************
 Program: Function Calls
Written by: Dr. John P. Abraham
Instructional objective: local variables
*****************************/

#include <iostream>
using namespace std;

int x;
void getvalue ();
int main ()
{
  x =1;
  getvalue();
  cout << "x contains a value of " << x <<"\n";
  return(0);
}

void getvalue()
 {
  int x;
  cout << "enter a value for x ";
  cin >> x;
 }
```

Program Run 8-5

```
enter a value for x 15
x contains a value of 1
```

The x in the function getvalue is a local variable. Now you have an x global and an x local. When you assign a value to x, the local gets it and the global is not changed. The global was assigned a value of 1 in the main and the program displays a one.

Let us write a program to illustrate call-by-reference mentioned earlier. When a parameter is passed call-by-reference, any changes made to that variable in the function, actually is changing the original variable. In the following example (Program 8-6), length and width are passed by reference. In order to pass by reference we use an ampersand & in front of the variable name. Note how the prototype is declared; here also we use an ampersand. Another interesting feature of functions is that you do not have to use the same identifier in the called and calling modules. For example, in the calling module the identifiers were length and width; in the called module the identifiers were l and w. The names do not have to match, but their order has to.

Program 8-6

```
/*******************************
   Program to Find square feet
   Instructional objective: Passed-by-reference.
   By Dr. John Abraham
   Created for 1380 students
*******************************/


#include <iostream.>
using namespace std;

void getmeasurements (int &, int &);
float calc_sqyards(int, int);
int main()
 {
        int length, width;
        float sqyards;
        getmeasurements(length, width);
        sqyards = calc_sqyards(length, width);
        cout << "The area in square yards is " << sqyards;
        return (0);
 }

void getmeasurements(int &l, int &w)
 {
  cout << "Enter length of the room in feet ";
  cin >> l;
  cout << "Enter width of the room in feet ";
  cin >> w;
 }
```

```
float calc_sqyards (int length, int width)

{
 float area;
 area = length * width /9;
 return(area);
}
```

Program Run 8-6

```
Enter length of the room in feet 18
Enter width of the room in feet 15
The area in square yards is 30
```

Assignment

Write a program with two functions, one function to input two integers and the other to exchange these values if the first integer is bigger than the second. See the program runs given below.

Please enter two integers 90 80
values input for a and be respectively--> 90 80
after exchange values for a and b--> 80 90

Please enter two integers 40 50
values input for a and be respectively--> 40 50
Exchange was not necessary.