Lab 10: Functions

Objectives:
The objective of this lab is to understand the concept of function and its use.

Recall that the concept of a FUNCTION in programming is similar to the same concept in algebra. Like an algebra function, a program function has a name and takes a certain type and number of data values as inputs. The function then processes these inputs in some way to produce a single output data value. In algebra, a function often has only one input, a real number data value. And the way an algebra function processes the inputs to produce the output is usually defined by a mathematical expression or formula. In programming, of course, the way a function processes the inputs is defined by a block of statements, called the BODY or IMPLEMENTATION of the function. And it is often the case that a program function will have more than one input data value. These inputs are also called ARGUMENTS or PARAMETERS of the function. The use of functions plays a key role in C++ programming. Functions provide an elegant mechanism for breaking complex programs into simpler, easier to understand SUB-PROGRAMS. They also provide a means for the same code instructions to be used over and over again in the same program or in other programs, which can save time and money. It is easy to create a function in a program. The compiler needs to know the following information, in this order:

1. The type of the output value (no identifier necessary).
2. The identifier of the function.
3. The types and identifiers of the function parameters (may be several).
4. The body of the function (a list of statements inside of braces).

The general syntax for DEFINING (creating) a program function is:

```
return_type function_identifier( list of parameters )
{
    list of statements;
    return ( return_value );
}
```

The last statement in the function body begins with the reserved word "return" followed by an expression inside of parentheses (paren's are optional). This is the value that is output by the function when the function is CALLED or INVOKED in another part of the program. The return value must be of the data type indicated on the first line of the function definition. Also shown on the first line is the identifier for the function and then, inside of parentheses, the list of function parameters. Each parameter is indicated by both its data type and an identifier, and
parameters in the list are separated by commas. Here, for instance, is a C++ implementation of the algebra function $f(x) = (x - 2)^2$:

```cpp
double f (double x)
{
    x = x - 2;
    return (x*x);
}
```

Therefore, this function is named "f" and has one input, a double called "x". It outputs a double value as well (note the expression $x^2$ evaluates as double). Also note that the input $x$ can be used just like a normal variable.

**Task 1: DO-WHILE loop statement**

The purpose of this task is to trace the execution of a FUNCTION.

The body of a function does not execute until the function is used in another part of the program. The only exception to this rule is the main body of the program. You can now see that the main body of a program is really the body of a function called "main". Every C++ program must have a function called "main". Note that the main function outputs an integer, which is why the last line of the main body of all programs we have seen so far is " return 0; " Every C++ program begins executing by invoking the main function, thus the main function is called the ENTRY POINT for the program. The main function has no inputs, which is indicated by using the reserved word "void" in place of the parameter list.

Other function definitions are often placed after the main function in the source code file for a program. The body of a function can contain any type of statement, such as input/output statements, selection statements, and loops. This includes declaration statements for variables and named constants. A variable that is defined within a block of statements, such as the body of a function, is called a LOCAL variable. This means that the variable is only in existence while the given block of code is executing. If the block of code where a local variable is declared terminates, the local variable goes out of existence and is considered to be undeclared. Thus it cannot be used outside of that block. Here is simplified example of some function definitions and local variables:

```cpp
int main(void)
{
    int a, b=5;
    double c=10.0;
    ...
    a = P( b, c );
    ...
    return 0;
}

int P(int m, double n)
{
    int x, y, c;
    ...
    return x;
}
```
The local variables of function P( ) in the above example are x, y, and c. The parameters m and n in the function P( ) definition are called FORMAL PARAMETERS and also count as local variables. The statement a = P( b, c ); in the main function is a FUNCTION CALL that invokes (starts) the execution of the body of the called function P. The parameters b and c are called ACTUAL PARAMETERS.

When the function P( ) is called from the main function, the formal parameters are ALLOCATED (set up in memory) and are automatically initialized from the values of the actual parameters, in order. So in this example, m is initialized as 5 and n is initialized as 10.0. Then the other local variables are allocated and the function body executes to the RETURN statement. The value stored in the local variable x is returned to the main function and assigned to the variable a. All the local variables in the function P( ) are DEALLOCATED (deleted from memory) when the function terminates.

Activity 1.1: Study the following program carefully.

In the following program, the two functions PrintStuff( ) and NumStuff( ) are called from the main program (function). Since the body of the main function is placed before the bodies of the two functions in the source code file, the function identifiers need to be declared before the main function body, otherwise the compiler will give "undeclared identifier" syntax errors when the function call statements are reached in the main function. The mechanism used to declare a function identifier without actually defining the body of the function is called a FUNCTION PROTOTYPE or DECLARATION. This consists of the first line of the function definition, except that the identifiers for the formal parameters can be left out (but the data types of the formal parameters must remain). Note also that function prototypes must terminate with a semicolon. The return data type of the function PrintStuff( ) is "void". This means that PrintStuff( ) does not return a data value. This is why there is no RETURN statement in the body of PrintStuff( ). Another interesting feature of this program is the fact that there are variables named "n", one in the main function and one in the function PrintStuff( ). It is important to understand that these variables are NOT the same and do not represent the same memory location. Different values may be stored in these two variables.

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

//Program name: Stars_and_Bars
void PrintStuff(char, int);
int NumStuff(int);
int main(void)
{
    int counter, n;
    for (counter = 0; counter <= 4; counter++)
    {
        n = NumStuff(counter);
        if (counter%2 == 0)
        {
            cout<n;
            PrintStuff('*', n);
            cout<n;
        }
        else
        {
            cout<n;
            PrintStuff('-', n);
            cout<n;
        }
    }
}```
Activity 1.2: Predict the output from the program Stars_and_Bars.

Predicted Output:

Activity 1.3: Start MS Visual Studio and create a C++ project called Lab10. Then add a new file Lab10Tsk1.cpp to the project. Copy the program in Activity 1.1 to Lab10Tsk1.cpp. Then run the program Stars_and_Bars and observe the output. Explain any discrepancies with your predicted output.

Observed Output:

Activity 1.4: List all the identifiers for variables used as FORMAL PARAMETERS in the program Stars_and_Bars. Then list the identifiers for all variables used as ACTUAL PARAMETERS.

<table>
<thead>
<tr>
<th>Formal parameter:</th>
<th>Actual parameters:</th>
</tr>
</thead>
</table>

Activity 1.5: Now modify the program Stars_and_Bars by deleting the declaration statement "int n;" from the function PrintStuff(). Predict the syntax error that this causes in the program. Then try compiling the modified program to confirm your prediction.
Task 2: For-Loop
The purpose of this task is to illustrate how to IMPLEMENT functions.

One advantage to using functions is to simplify the main function. Sometimes key segments or processes in the main function can be broken out as functions that are called from the main function. This can make the main function easier to design, code, trace, and debug.

Activity 2.1: Study the following program carefully. This program is designed to compute the monthly payment on an amount borrowed with a given interest rate and pay-back period.

Activity 2.2: Add a new cpp file to Lab10 project called Lab10Tsk2.cpp and copy the program in Activity 2.1 to Lab10Tsk2.cpp. Then run the Amortize program with the following values for the input and observe the output.

<table>
<thead>
<tr>
<th>Amount:</th>
<th>Rate:</th>
<th>Year:</th>
<th>Payment:</th>
</tr>
</thead>
<tbody>
<tr>
<td>9500</td>
<td>8%</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>9%</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>7.5%</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Activity 2.3: Modify the program Amortize by inserting a function FindPayment( ) that will compute the monthly payment. The modified program should appear as follows:

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

// Program name: Amortize
int main(void)
{
    double Amount, Rate, Payment, Z;
    int Years, NoPayment;
    cout<<"Enter the borrowed amount --->";
    cin>>Amount;
    cout<<endl<<"Enter the annual interest rate (%) --->";
    cin>>Rate;
    cout<<endl<<"Enter the pay-back period (in years) --->";
    cin>>Years;
    //These statements compute the monthly payment******
    Payment = FindPayment (Amount, Rate, Year);
    //****************************************************************************
    cout<<endl;
    cout<<setprecision(2)<<fixed<<showpoint;
    cout<<"The monthly payment is $"<<Payment<<endl;
    cin.get(); cin.get();
    return 0;
} // Amortize

// *****Insert function definition for FindPayment( ) here*****
```

Activity 2.4: Test your modified program with the same input as in Activity Two. Print and turn in a copy of the modified program.

<table>
<thead>
<tr>
<th>Amount</th>
<th>Rate</th>
<th>Year</th>
<th>Payment</th>
</tr>
</thead>
<tbody>
<tr>
<td>9500</td>
<td>8%</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>9%</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>20000</td>
<td>7.5%</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Task 3: For-Loop
The purpose of this task is to gain experience developing functions based on PRECONDITIONS and POSTCONDITIONS.

It is important to document the purpose of a function and provide instructions about how to use and call a function. This is so the function can be used by programmers other than the original programmer, or can be used again in the future. Using functions created by other programmers can be a tremendous way of saving time and therefore money (this process is called CODE REUSE). From your own experience, think of how convenient it has been to use the functions declared in the header file cmath, such as pow( ) and sqrt( ). One means of documenting how
functions work are PRECONDITIONS and POSTCONDITIONS. Function preconditions document the conditions that are necessary for the function to execute properly and produce correct output. Preconditions often describe how the parameters to a function must be initialized when the function is called and begins executing. Postconditions, on the other hand, give information about the status of a program when the function terminates. They typically describe the output value returned by the function (if any), and other sorts of actions performed by the function, such as input or output to a stream. Remember that pre- and post-conditions are merely comments, so they don't really affect the way a function executes. But one of your goals as a programmer should be to make sure that any function you write conforms to its pre- and post-conditions. Moreover, any function you write should be documented with enough information in the pre- and post-conditions that another programmer can understand the nature and purpose of the function and how it can be used correctly.

Activity 3.1: Study the following program carefully. This program is intended to take a text file and convert upper-case characters to lower-case and vice versa. Punctuation should not be affected. In the input text, a '-' (dash) indicates a space and a '/ ' (slash) indicates the end of the line. The character ' * ' is the sentinel value used to indicate end-of-file. This program will use a function called SwitchCase( ). The declaration for SwitchCase and pre- and post-conditions are given in the program.

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

char SwitchCase(char);

//Program name: ViceVersa
char c;
fstream Infile;
int main(void)
{
    Infile.open("a:\inlab83.txt", ios::in);
    Infile>>c;
    while (c != ' * ')
    {
        while (c != '/ ')
        {
            if (c == '- ')
            cout<<' ';
            else
            if (((c>='A')&&(c<='Z'))||((c>='a')&&(c<='z')))
                cout<<SwitchCase(c);
            else
                cout<<c;
        } // not end of line
        cout<<endl;
        Infile>>c;
    } // not end of file
    Infile.close();
    cin.get(); cin.get();
    return 0;
} // ViceVersa

char SwitchCase (char Target)
/* Pre-condition: Target is a letter.
Post-condition: The value returned is Target switched from upper-case to lower-case or vice versa. */
{
}
} // SwitchCase
Activity 3.2: Add a new cpp file to Lab10 project called Lab10Tsk3.cpp and copy the program ViceVersa in Activity 3.1 to Lab10Tsk3.cpp. Then modify the program ViceVersa by completing the code body for SwitchCase(). Be careful that SwitchCase() satisfies the pre- and post-conditions. As you write the function body, remember you can assume that the preconditions will be true as the function executes. Any program that uses this function should be careful to only use the function in the way described by the pre- and post-conditions. Create a text file called InLab103.txt and test the modified program with the input file that you just created. Print and turn in a copy of your modified program.

InLab103.txt

This-file's-just-a/
test-to-see-how-your-program-converts-lowercase-letters/
to-UPPERCASE-letters-and-vice-versa.--After-running-the-program-all/
the-letters-should-switch-cases./

Good-Job./*

Activity 3.3: What provisions are made in the main function of the program ViceVersa to guarantee that when the function SwitchCase() is called, the pre-condition for this function will be true?.

Answer: