**Lab Experience Nine**

**Objectives:**

1. Introduction to modular programming with functions.
2. Understanding parameter passing between functions.

**Background**

It is a common processto solve problems by breaking the problem up into smaller tasks that can be easily solved. This process in Computer Science is called **modularity** and is implemented in C++ with functions. A function is a segment of code that solves one task and one task only. For example the function **pow** only performs exponential operations and nothing else. In addition, it is the programmer’s responsibility to provide the correct information to the **pow** function, otherwise unpredictable results might occur, i.e. **pow(0,0)** should not work and either an error message will be displayed or a garbage value will be returned.

**Functions**

A function is used in C++ to accomplish one and only one task. It is common to have more than one function in a C++ program to solve the problem.

Example:

Write a program that asks the user to enter an item’s wholesale cost and its markup percentage and displays the items retail price.

There are 3 sub problems in the above example:

1. Getting wholesale cost and markup percentage from the user.
2. Calculating the retail price of the item.
3. Displaying the retail price.

The solution of this problem can be written with 3 programmer-defined functions. The above problem will be completed by the end of this lab.

**Creating Programmer-defined Functions**

When creating a function, the programmer must write the definition. The definition consists of four parts:

1. Return-type: The function can return any valid C++ data type or return nothing. If the function returns nothing the return type will use the keyword **void**.
2. Name: A unique identifier that is descriptive of what the function will accomplish.
3. Parameter line: Used to pass data to the function. If the parameter list is **void** means the function is self-contained and does not need any external information to accomplish its task.
4. Body: The C++ statements necessary to accomplish the task specified for the function.

A function has the following syntax:

return-valuefunctionName(datatype param-1, datatype param-2, …,datatypeparam-n)

Programmer defined identifier following the same rules as naming variables.

Specifies the type of information returned by the function via a return statement.

Parameters are variables defined by the programmer that will be used within the function to solve the specified task. Data will be passed to the function via the parameter list. Each parameter used must be preceded by the data type of the parameter. This is used by the function for syntax checking. The number of parameters used by the function can range from zero to n.

**Examples:**

int main(){ // is a function with a return value of int and zero parameters.

intstrcmp(char \*array1, char \*array2) // is a function with a return value of int and two parameters.

When a function has a return type of void, a return statement is not needed. If a return statement is desired, place **return;** before the right curly brace**}.**

**voiddisplayRetail(double retailPrice){// header**

**// Some C++ statements;**

**}**

**Calling a function**

Before a function is called it must be defined. It was a common practice to define all functions before main, but this is now discouraged. Instead a prototype of the function is placed before the function main with the same characteristics of the function header and the function definition is placed after main. The purpose of the prototype is define the function name and to provide syntax checking between the calling statement and the function definition.

**Example:**

**voiddisplayRetail(double); // prototype note semicolon and only data types**

**// in the parameter list**

**int main(){**

**double price;**

**// some more C++ statements**

**displayRetail(price); // call to the function, note no return type or**

**// data type in front of price.**

**return 0;**

**}// end main**

**voiddisplayRetail(double retailPrice){ // function header and definition.**

**cout<< fixed <<showpoint<<setprecision(2);**

**cout<< “The retail price of the item is $ <<retailPrice<<endl;**

**}**

What happens when the program executes? The value contained in **price** is copied to the variable **retailPrice** and execution is transferred to the function **displayRetail**. Execution is transferred back to the statement following **displayRetail** when the function terminates. The function will terminate when a return statement or a right curly brace is reached.

The variable **price** is commonly referred to as the **actual parameter** (also called argument) to the function and **retailPrice** is called the **formal parameter** since **retailPrice** is used within the body of the function.

The process of copying the contents of the actual parameter to a formal parameter is called **pass by value** or **call by value**. Since **retailPrice** contains a copy of **price**, any changes made to **retailPrice** will not change the value contained in price.

When using function there are essentially three parts that the programmer needs to complete.

1. The heading must have both data type and variable names for all its formal parameters.
2. The prototype must be placed before main and must have the data types. The parameters names can be included within the prototype, but they are ignored.
3. The call must have the name of the function (not preceded by the return type), but must not have the data type for its actual parameters.

**Pass by value**

When a function utilizes **pass by value** the following steps occur when the function is invoked or called:

1. The formal parameter is allocated memory with a unique address. This means a formal parameter and an actual parameter can have the same name, but each will have its own memory address.
2. Type coercion will occur if the data types of the actual parameter and the formal parameter are not the same. I.e. an int converted to double and vice versa.
3. The contents of the actual parameter are copied into the formal parameter thus preventing any accidental changes to the contents of the actual parameter.
4. When the function terminates, the formal parameters memory location is reallocated back to the operating system and is no longer available to the program.

**Pass by reference**

A function can only return a single value with the return statement. What method is used when more than one value needs to be returned to the calling function? pass by reference.

**Example:**

**voiddisplayRetail(double); // prototype note semicolon and only data types**

**// in the parameter list.**

**voidgetData(double &, double &); // prototype Note the &**

**int main(){**

**doublewholesaleCost, pctMarkup;**

**double price;**

**getData(wholesaleCost, pctMarkup); call the function**

**// more C++ statements**

**displayRetail(price); // call to the function, note no return type or**

**// data type in front of price.**

**return 0;**

**}// end main**

**voiddisplayRetail(double retailPrice){ // function header and definition.**

**cout<< fixed <<showpoint<<setprecision(2);**

**cout<< “The retail price of the item is $ <<retailPrice<<endl;**

**}// end displayRetail**

**voidgetData(double &wholesale, double &markup){**

**cout<< “Enter the wholesale cost of the item ==>”;**

**cin>> wholesale;**

**cout<< “\nEnter the percent markup of the item ==>”;**

**cin>> markup;**

**}// end getData**

The & (ampersand) character is used in C++ to differentiate between call by value and call by reference. The formal parameter uses the same memory location as the actual parameter whenever call by reference is used. This means any changes to the formal parameter in the function will change the value of the actual parameter.

When a function utilizes **pass by reference** the following steps occur when the function is invoked or called:

1. The **address** of the actual parameter’s memory location is passed to the formal parameter. This means any changes made within the function to the formal parameter will change the contents of the actual parameter.
2. Type coercion will **NOT**occur if the data types of the actual parameter and the formal parameter are not the same. **The data types of the formal and actual parameters must be identical**.

For example consider the following code snippets from the program on the previous page. The address of **wholesaleCost** is passed to the formal parameter **wholesale** and any changes made to **wholesale** within the function **getData** will change the contents of **wholesaleCost.** The same is true for**pctMarkup**and **markup**.

**getData(wholesaleCost, pctMarkup); call the function**

**voidgetData(double &wholesale, double &markup){**

**cout<< “Enter the wholesale cost of the item ==>”;**

**cin>> wholesale;**

**cout<< “\nEnter the percent markup of the item ==>”;**

**cin>> markup;**

**}// end getData**

**Returning a Value From a Function**

When a function returns a value the function call must be used in an expression, appear on the right hand side of an assignment statement, or in a **cout** statement. If none of these are used the returned value is discarded and execution will continue.

**voiddisplayRetail(double); // prototype note semicolon and only data types**

**// in the parameter list.**

**voidgetData(double &, double &); // prototype Note the &**

**doublecalculateRetailPrice(double, double); prototype note the return value is double**

**int main(){**

**doublewholesaleCost, pctMarkup;**

**double price;**

**getData(wholesaleCost, pctMarkup); call the function**

**price = calculateRetailPrice(wholesaleCost, pctMarkup); // place the returned value**

**// in the variable price**

**displayRetail(price); // call to the function, note no return type or**

**// data type in front of price.**

**return 0;**

**}// end main**

**voiddisplayRetail(double retailPrice){ // function header and definition.**

**cout<< fixed <<showpoint<<setprecision(2);**

**cout<< “The retail price of the item is $ <<retailPrice<<endl;**

**}// end displayRetail**

**voidgetData(double &wholesale, double &markup){**

**cout<< “Enter the wholesale cost of the item ==>”;**

**cin>> wholesale;**

**cout<< “\nEnter the percent markup of the item ==>”;**

**cin>> markup;**

**}// end getData**

**doublecalculateRetailPrice(double wholesale, double markup){**

**double retail = wholesale \* (1 + markup/100.0); // the variable retail**

**return retail; // is called a local variable**

**// and is only known within this**

**// function**

**}// end calculateRetailPrice**

The above program is the completed problem posed at the beginning of the functions section. Make sure you understand how information is passed between functions and how a problem can be reduced to a series of sub problems.

**Lab Exercises**

**Directions:**

Start Microsoft word and record the questions and answers to all of the exercises in your word document   
Answer the following questions based on material presented in lecture and found in chapters 1-6 of the textbook.

**Fill in the blank**

1. The word \_\_\_**Void**\_\_\_\_ precedes the name of the function prototype and heading indicating the function does not return a value.
2. Pass by \_\_**Value**\_\_\_\_\_\_ copies the contents of the actual parameter to the formal parameter.
3. Pass by \_\_**Reference**\_\_\_\_\_\_ passes the address of the actual parameter to the formal parameter.
4. When pass by value is used any changes made to the formal parameters contents will change the actual parameters contents.Is this statement True or is it False.**False**
5. The for loop initialization expression is executed only \_\_**Once**\_\_\_\_\_\_\_.

**Exercises:**

1. The following if statement determines whether choice is equal to ‘Y’ or ‘y’.

if (choice == 'Y' || choice == 'y')

Simplify this statement by using either the toupper or tolower function.

**If (toupper(choice) == ‘Y’)**

1. Assume the input is a string variable called ***partNumber***. Write the code that counts the number of alphabetic characters in the string variable.

**int alphaCount = 0; // keeps track of the number of letters**

**for (int i = 0; i < partNumber.length(); i++)**

**if (isalpha(partNumber.at(i)))**

**alphaCount++ ;**

1. Examine the following statements.

stringstr = "237.89";

double value;

Write a statement that converts the string in str to a double and stores the result in value.

**value = atof(str.c\_str());**

**Exercise 9**

1. #include <iostream>

using namespace std;

1. int multiply(int, int);
2. int main(){
3. intmultiplier,multiplicand,product;
4. cout<< "Enter the first number ===> " ;
5. cin>> multiplier; << === **Arguments aka ACTUAL paramters**
6. cout<< "Enter the second number ===> " ;
7. cin>> multiplicand;
8. product = multiply(multiplier, multiplicand);
9. cout<< "Product of " << multiplier << " & " << multiplicand

<<" is " << product;

1. return 0;
2. }// end main
3. //
4. // Multiply two numbers and return their product
5. // Pre: num1 and num2 contain the values to be multiplied.
6. // Post: The product of num1 and num2.
7. int multiply(int num1, int num2){ << === **parameters aka FORMAL parameters**
8. return (num1 \* num2);
9. }

Questions 1 through 7 refer to the program above.

1. If the user enters 17 and 21, respectively, what will be the output of the program?

**Product of 17 & 21 is 357**

1. What is statement 2 called in the program?

**Function Prototype**

1. Using complete sentences explain how the values in the actual parameters are sent to the formal parameters. Explain what happens to memory once the function is complete.

**The actual parameters will have their values copied from their memory location to the memory location of the formal parameter. The values are passed by value.**

**After the function has completed, the formal parameter values are terminated, and the memory is reallocated.**

1. What are statements 17 through 19 called in the program?

**The function definition**

1. Write down the formal parameters of the function multiply.

**num1 num2**

1. Write down the actual parameters (arguments) of the function multiply.

**multiplier multiplicand**

1. Does the function multiply change the actual parameters when the function is called or invoked? Why or Why not?

**No, because the values are passed by value, which means the actual parameters are copied to the memory location of the formal paramters.**

**Due Dates:**

As specified on the D2L Drop box folder for lab 9.

**What to hand in:**

1. Place the word document into the lab 9 drop box folder using your name and the lab number as the file name. For example: timwrennlab9.docx.