**Lab Experience Ten**

**Objectives:**

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

**Background**

It is a common process to 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-value functionName(datatype param-1, datatype param-2, …,datatype param-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.

int strcmp(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**}.**

**void displayRetail(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. The function header followed by the function definition follows the function header.

The purpose of the prototype is to declare the datatype of the function name and to specify the number of parameters required by the function. This provides the compiler to perform syntax checking of the function call in your program.

**Example:**

**void displayRetail(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**

**void displayRetail(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. All C++ functions 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 called or invoked:

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.*

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**.

**Example:**

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

**// in the parameter list.**

**void getData(double &, double &); // prototype Note the &**

**int main(){**

**double wholesaleCost, 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**

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

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

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

**}// end displayRetail**

**void getData(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**

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**

**void getData(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.

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

**// in the parameter list.**

**void getData(double &, double &); // prototype Note the &**

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

**int main(){**

**double wholesaleCost, 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**

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

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

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

**}// end displayRetail**

**void getData(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**

**double calculateRetailPrice(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. The for loop initialization expression is executed only **once**.

**Exercise 1 Line numbers are used for reference only.**

1. #include <iostream>
2. using namespace std;
3. void getNumbers(int &num2, int &num2);
4. void displayResults(int number1, int number2, int product, double avg);
5. int multiply(int number1, int number2);
6. void average(double number1, double number2, double &avg);
7. int main(){
8. int multiplier,multiplicand,product;
9. double averageNum;

1. getNumbers(multiplier, multiplicand);
2. product = multiply(multiplier, multiplicand);
3. average(multiplier, multiplicand, averageNum);
4. displayResults(multiplier, multiplicand, product, averageNum);
5. return 0;

}// end main

///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*multiply\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Directions: Multiply two numbers and return their product

//

// Pre: num1 and num2 contain the values to be multiplied.

//

// Post: The product of num1 and num2.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. int multiply(int num1, int num2){
2. return (num1 \* num2);

}// end multiply

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*average\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Description: Function to find the average between two numbers

//

// Pre: number1 and number2 must contain values

//

// Post: avg will contain the average of the two numbers

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. void average(double number1, double number2, double &avg){
2. avg = (number1 + number2) / 2;

}// end average

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*getNumbers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Description: Function to prompt the user for two numbers

//

// Pre: Two integer variables must be provided

//

// Post: The calling function argument values will be changed.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. void getNumbers(int &num1, int &num2) {
2. cout << "Enter the first number ===> ";
3. cin >> num1;
4. cout << "Enter the second number ===> ";
5. cin >> num2;

}// end getNumbers

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*displayResults\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Description: Function to display the results of calculations

//

// Pre: All parameters must contain values

//

// Post: The results will be displayed to the screen

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. void displayResults(int number1, int number2, int product, double avg) {

1. cout << "Product of " << number1 << " & " << number2

<< " is " << product;

1. cout << "The average of " << number1 << " & " << number2

<< " is " << avg;

}// end displayResults

Download the above program from D2L and answer the questions below.

1. Comment out lines 3- 6 and describe what happens when you try to execute the program. Remove the comments after you have answered the question.

I get errors saying that I have no identifiers.

1. Change line 11 to the following: **multiply(multiplier, multiplicand);**

Describe what happens? If the assertion window appears, click on the ignore button. Change the statement back to the original once you have answered this question.

I get an error saying that there’s no local variable “product” used.

1. Change line 15 from **int multiply(int num1, int num2)** to **int multiply(int num1, num2)** . Describe what happens when you try to execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that the identifier “num2” is undefined.

1. Comment out line 12 and describe what happens when you execute the program. Remove the comment after you have answered the question.

I get an error saying that “averageNum” is uninitialized.

1. Change line 5 to **void multiply(int, int );** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

The program seems to be working correctly.

1. Change line 5 to **int multiply(int );** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that the “multiply” function does not take two arguments.

1. Change line 5 to **int multiply( );** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that the “multiply” function does not take two arguments.

1. Change line 11 to **product = int multiply(multiplier, multiplicand);** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that the int type name is not allowed.

1. Change line 11 to **product = multiply(multiplicand);** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying I have too few arguments in the function call.

1. Change line 16 to **return(multiplier \* multiplicand);** and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that multiplier and multiplicand is undefined.

1. Change line 8 to double multiplier,multiplicand,product; and describe what happens when you execute the program. Change the statement back to the original once you have answered the question.

I get an error saying that the reference type int cannot be initialized with a value of double

**Due Dates:**

As specified on the D2L Drop box folder for lab 10.

**What to hand in:**

1. Place the word document into the lab 10 drop box folder using your name and the lab number as the file name. For example: timwrennlab10.docx.