**Lab Experience Ten**

**Objectives:**

1. Understanding the differences between pass by reference and pass by value
2. Understanding the scope rules: Global vs. Local variables, what happens when two variables have the same name.
3. To understand scope as applied to functions.
4. Understand function stubbing
5. Creating a program using functions.

**Background**

Scope is a term used to define the area in the program where the variable is known and can be used. The types of scope are block, local, and global. Global scope should be avoided except when passing large amounts of data between functions. I.e. when a two dimensional array is used to represent the palette in a graphics toolbox program.

Function stubbing is used to test each function separately before writing the entire program. A common mistake of most beginning programmers is to write the entire program and then debug it. If a logic error occurred in the program, the programmer debugging the program is literally “looking for the needle in the haystack” because the program could be terminating abnormally in the first function, but the real cause of the error could have been caused in the fourth function not performing the correct task.

Arrays in Computer Science are data structures used to hold large amounts of the same type of data instead of using multiple variables. An array represents consecutive memory locations referenced by the same name with an offset into memory which is represented by the subscript.

**Scope Rules**

If two variables with the same name are used within a program which one does the program use? This is decided by the scoping rules.

Block scope is defined by the block where the variable has been declared. Block scope is also referred to as local scope and local scope takes precedence over global scope.

Global scope is defined from the point of declaration in the program down. Globally defined variables are placed outside any function and can be used by any functions defined below the declaration.

Because of the scoping rules, multiple references to the same name can occur, but each reference could be referring to a different address in memory. Which one is used is dependent on where the declaration of the variable is placed within the program or by the absence of a declaration.

If a function references a variable that has not been defined within the function, either through the parameter list or by a variable declaration, the function looks “outside” its definition to locate the variable. If the variable has been defined globally, the function will use the globally defined variable, if it hasn’t been defined globally, a syntax error will be generated with the message being an undeclared identifier.

It is possible to have two variables with the same name, one defined globally and one defined locally. When this occurs the variable defined locally will be used instead of the globally defined variable.

**Example:**

**#include <iostream>**

**using namespace std;**

The boxes represent the scoping rules. The outside box represents the global scope of the variable x defined above funstuff(). The inside boxes represent the local scope of each function.

Since x is declared in the function main, all references of x in main will use the local declaration.

Since x is not defined in the function funstuff the globally defined variable x will have its value changed. This could be by accident thus the program could produce what is called a side-effect. A side-effect means something has changed and this change could have an adverse effect on the results being generated by the program.

**int x = 5;**

**doublefunstuff();**

**int main(){**

**int x = 3;**

**// more statements**

**return 0;**

**}**

**doublefunstuff(){**

**x = 10;**

**// more statements**

**}**

**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 the lab word document   
Answer the following questions based on material presented in lecture and found in chapters 1-6 of the textbook.

**Exercise 1**

1. Enter the program below into a new Visual Studio.Net project solution.
2. Insert the following comments at the top of the program.

**// Computer Science 1106**

// your name **here**

For your convenience the program is listed below:

// This program illustrates global variables.

**#include <iostream>**

**using namespace std;**

**int x; // a global variable**

**voidprintVariable(); // prototype**

**int main(void){**

**x = 300;**

**cout<< "main before printVariable: The value in the variable x is " << x <<endl;**

**printVariable();**

**cout<< "main after printVariable: The value in the variable x is " << x <<endl;**

**return 0;**

**}**

**//**

**// This function produces a side effect**

**//**

**voidprintVariable(){**

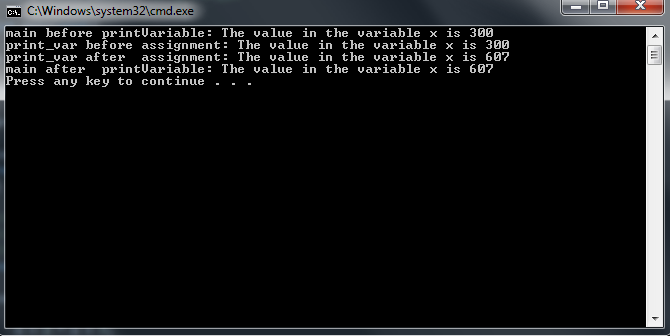
**cout<< "print\_var before assignment: The value in the variable x is " << x <<endl;**

**x = 2 \* x + 7;**

**cout<< "print\_var after assignment: The value in the variable x is " << x <<endl;**

**}**

1. Place the answers to all of the questions below in this word document directly below each question.
   1. Run the program and paste the output into your word document.



* 1. Using complete sentences write a detailed explanation of how the program executes.

**x is declared globally, outside main. It is then initialized to 300. Console then outputs that value for x, which is 300. printVariable runs, console outputs x, which is still 300. x is then multiplied by 2, and adds 7 to make 607. Console outputs 607. printVariable then closes and the console outputs 607 for the value of the global variable x.**

* 1. Move the declaration of **x** into main. Attempt to compile the program and **explain** the compilation problem using complete sentences. Copy and paste the source code and output into your word document. (hint: possible scope problem?)

**When the function printVariable is called, printVariable looks for a variable x that is outside of main. Because of scope, the declared x inside main is not seen by printVariable.**

// Computer Science 1106

// Chris Dang

// This program illustrates global variables.

#include <iostream>

using namespace std;

//int x; // a global variable

void printVariable(); // prototype

int main(void){

int x = 300;

cout<< "main before printVariable: The value in the variable x is " << x <<endl;

printVariable();

cout<< "main after printVariable: The value in the variable x is " << x <<endl;

return 0;

}

//

// This function produces a side effect

//

void printVariable(){

cout<< "print\_var before assignment: The value in the variable x is " << x <<endl;

x = 2 \* x + 7;

cout<< "print\_var after assignment: The value in the variable x is " << x <<endl;

}

* 1. Fix the compilation problem by passing the variable **x** as a parameter to the function **printVariable.**Did it work? Why or why not? Yes, but not in the same way when the variable x was declared as a global variable. This time, the last time the console outputted x, it printed 300.
  2. Change the prototype and function header of **printVariable**to make this work. Copy and paste the source code and output into your word document. Paste the output of your program below the changed source code.

// Computer Science 1106

// Chris Dang

// This program illustrates global variables.

#include <iostream>

using namespace std;

//int x; // a global variable

int printVariable(int x); // prototype

int main(void){

int x = 300;

cout<< "main before printVariable: The value in the variable x is " << x <<endl;

printVariable(x);

cout<< "main after printVariable: The value in the variable x is " << x <<endl;

return 0;

}

//

// This function produces a side effect

//

int printVariable(int x){

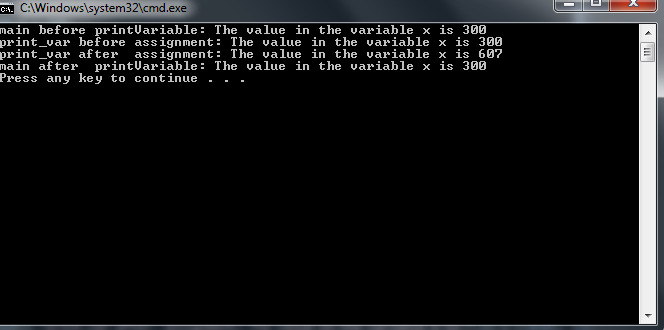
cout<< "print\_var before assignment: The value in the variable x is " << x <<endl;

x = 2 \* x + 7;

cout<< "print\_var after assignment: The value in the variable x is " << x <<endl;

return x;

}



* 1. Suppose we want to return this adjusted value of **x** to main so that the output matches part a’s output (shown below). Make the necessary changes to your program. When done, paste the completed program and the output into your word document.

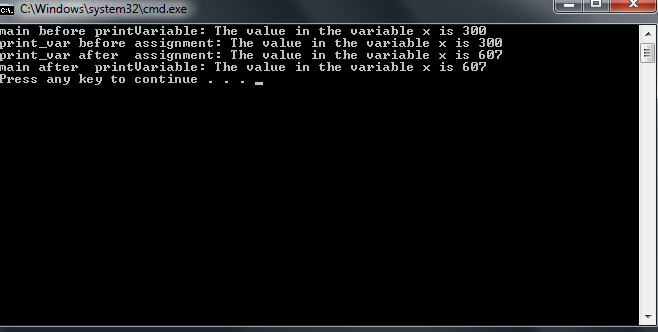
main before **printVariable**: The value in the variable x is 300

print\_var before assignment: The value in the variable x is 300

print\_var after assignment: The value in the variable x is 607

main after **printVariable**: The value in the variable x is 607

Press any key to continue



// Chris Dang

// This program illustrates global variables.

#include <iostream>

using namespace std;

//int x; // a global variable

int printVariable(int x); // prototype

int main(void){

int x = 300;

cout<< "main before printVariable: The value in the variable x is " << x <<endl;

x = printVariable(x);

cout<< "main after printVariable: The value in the variable x is " << x <<endl;

return 0;

}

//

// This function produces a side effect

//

int printVariable(int x){

cout<< "print\_var before assignment: The value in the variable x is " << x <<endl;

x = 2 \* x + 7;

cout<< "print\_var after assignment: The value in the variable x is " << x <<endl;

return x;

}

**Exercise 2 Function Stubbing:**

**Description of the problem.**

The verbal part of the SAT test counts 800 points. Students can take the test several times, and colleges evaluate a prospective student based on the largest score for each part of the test. The program that you will complete will read two SAT verbal scores from the user and display the larger of the two scores.

An example of the output is listed below:

**This program interactively reads two SAT**

**verbal scores and prints the larger**

**Enter an SAT verbal score between 0 and 800 ======> 550**

**Enter an SAT verbal score between 0 and 800 ======> 600**

**The larger score is 600**

**Press any key to continue**

What you must do:

1. Download the file **functionStubbing.cpp** from D2L.
2. Start Visual C++ and load the program **functionStubbing.cpp** into your workspace. Note: If you try to compile the program, you will have two errors concerning the functions **max** and **outOfRange**.
3. Change the comments in the program using your name.
4. Follow these directions explicitly:
5. Replace the comment in the body of **readScore** with statements to convert it to a stub.  **A program Stub consists of an output statement stating that the function was entered correctly and if the function should return a value or change the parameter**. To complete the stub place the following C++ statements in the function definition of **readScore**:

**cout<< “Entering function readScore “ <<endl;**

**score = 500;**

**cout<< “Leaving function readScore “ <<endl;**

1. Replace the comment in the body of **outOfRange** with statements to convert it to a stub.

Have the function body print something similar to part a.

**outOfRange should return false. So add the statement :**return false; **after all of your cout statements.**

1. Replace the comment in the body max of with statements to convert it to a stub. Have the function body print something similar to part a.

**max should return the value contained in the first parameter(which in our case is the variable x).**

1. Build and run your program saving it to the **file lab8.cpp**. The output should match what is listed below exactly. Paste the current source code and the output produced into your word document.

//

// Programmer: Chris Dang

//

// File: FunctionStubbing.cpp Class: Computer Science 1106

//

// Program Description: This program reads two SAT verbal scores(between 0 and 800)

// for a student and prints the larger of the two scores.

//

#include <iostream>

using namespace std;

// Global Constants

const int MAX\_SCORE = 800;

const int LOW\_SCORE = 0;

// Function Prototypes

void instructions(void); // Function to display instructions to the user

void readScore(int &score); // Function get the scores from the user

bool outOfRange(int score); // Boolean valued function will return true or false

int max(int x, int y); // Returns the score that is the largest

int main(void){

int verbalScore1, // SAT verbal score #1

verbalScore2; // SAT verbal score #2

instructions(); // Display instructions to the user

readScore(verbalScore1); // Get the first test score

if(outOfRange(verbalScore1))

cout << verbalScore1 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else{

readScore(verbalScore2); // Get the second test score

if(outOfRange(verbalScore2))

cout << verbalScore2 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else

cout << "The larger score is " << max(verbalScore1, verbalScore2) << endl;

}

return 0;

}

//

// Function to print instructions on what the program does

//

void instructions(void){

cout << "This program interactively reads two SAT\n";

cout << "verbal scores and prints the larger\n\n";

}

//

// Function to read a score interactively

//

void readScore(int &score){

cout << "Entering function readScore " <<endl;

score = 500;

cout << "Leaving function readScore " <<endl;

}

//

// Function to return true if the SAT score is out of

// range, false otherwise. Range: 0-800

//

bool outOfRange(int score){

cout << "Entering function outOfRange " <<endl;

score = 500;

cout << "Leaving function outOfRange " <<endl;

return false;

}

//

// Function to return maximum of two integer parameters

//

int max(int x, int y)

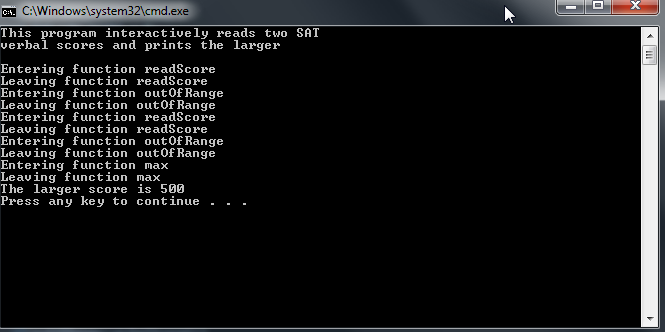
{

cout << "Entering function max " <<endl;

cout << "Leaving function max " <<endl;

return x;

}



**This program interactively reads two SAT**

**verbal scores and prints the larger**

## Entering function readScore

**Leaving function readScore**

**Entering function outOfRange**

**Leaving function outOfRange**

**Entering function readScore**

**Leaving function readScore**

**Entering function outOfRange**

**Leaving function outOfRange**

**Entering function max**

**Leaving function max**

**The larger score is 500**

**Press any key to continue**

1. At this point we have tested the communication between all of our functions. Now we need to test the if statements to make sure there isn’t a logic error.
2. Change the 500 to 900 in the function **readScore** and the word false to true in the function **outOfRange**. Execute the program and your output should match the output below exactly. Paste the source code in your word document and the output into your word document.

//

// Programmer: Chris Dang

//

// File: FunctionStubbing.cpp Class: Computer Science 1106

//

// Program Description: This program reads two SAT verbal scores(between 0 and 800)

// for a student and prints the larger of the two scores.

//

#include <iostream>

using namespace std;

// Global Constants

const int MAX\_SCORE = 800;

const int LOW\_SCORE = 0;

// Function Prototypes

void instructions(void); // Function to display instructions to the user

void readScore(int &score); // Function get the scores from the user

bool outOfRange(int score); // Boolean valued function will return true or false

int max(int x, int y); // Returns the score that is the largest

int main(void){

int verbalScore1, // SAT verbal score #1

verbalScore2; // SAT verbal score #2

instructions(); // Display instructions to the user

readScore(verbalScore1); // Get the first test score

if(outOfRange(verbalScore1))

cout << verbalScore1 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else{

readScore(verbalScore2); // Get the second test score

if(outOfRange(verbalScore2))

cout << verbalScore2 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else

cout << "The larger score is " << max(verbalScore1, verbalScore2) << endl;

}

return 0;

}

//

// Function to print instructions on what the program does

//

void instructions(void){

cout << "This program interactively reads two SAT\n";

cout << "verbal scores and prints the larger\n\n";

}

//

// Function to read a score interactively

//

void readScore(int &score){

cout << "Entering function readScore " <<endl;

score = 900;

cout << "Leaving function readScore " <<endl;

}

//

// Function to return true if the SAT score is out of

// range, false otherwise. Range: 0-800

//

bool outOfRange(int score){

cout << "Entering function outOfRange " <<endl;

score = 500;

cout << "Leaving function outOfRange " <<endl;

return true;

}

//

// Function to return maximum of two integer parameters

//

int max(int x, int y)

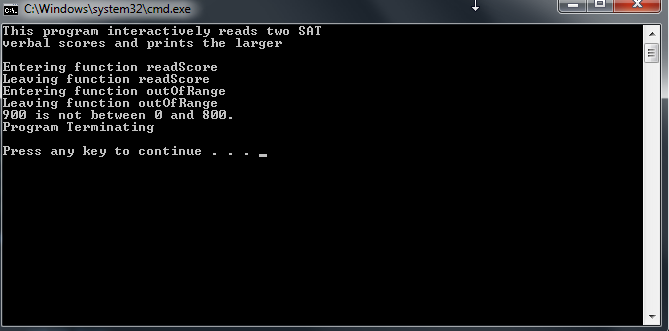
{

cout << "Entering function max " <<endl;

cout << "Leaving function max " <<endl;

return x;

}



**This program interactively reads two SAT**

**verbal scores and prints the larger**

**Entering function readScore**

**Leaving function readScore**

**Entering function outOfRange**

**Leaving function outOfRange**

**900 is not between 0 and 800.**

**Program Terminating**

**Press any key to continue**

1. At this point in the Software development cycle the testing of the if statements and the functions are complete. Everything is working, so now is the time to replace the stub code with actual code to solve the problem.
2. Change the function **readScore** to appear exactly as below, which means deleting all of the stub statements that we used to test communication between the functions.

voidreadScore(int&score){

cout<< "Enter an SAT verbal score between " << LOW\_SCORE << " and "

<< MAX\_SCORE <<" ======> ";

cin>> score;

}

Since we are using call-by-reference, the actual parameters value will be changed whenever this function is called.

1. The program still doesn’t work, so we have to change the functions **outOfRange** and **max** accordingly. Change the functions so that they appear exactly as below:

booloutOfRange(int score){

return (score < LOW\_SCORE || score > MAX\_SCORE);

}

//

// Function to return maximum of two integer parameters

//

int max(int x, int y){

return (x > y ? x : y);

}

1. The ternary operator is being used In the function **max.** The ternary operator’s syntax is:

**condition ?result if condition is true: result if condition is false**

After you have completed all of the above parts, run your program using the test values below:

|  |  |  |
| --- | --- | --- |
| **Verbal SAT score number 1** | **Verbal SAT score number 2** | **Expected Results** |
| **550** | **600** | **600 is the larger** |
| **900** |  | Program terminating |
| **500** | **500** | **500 is the larger** |
| **500** | **835** | **Error Program terminating** |

**Paste the program functionStubbing.cpp into your word document. Paste the output of each run of your program using the test data above.** **//**

// Programmer: Chris Dang

//

// File: FunctionStubbing.cpp Class: Computer Science 1106

//

// Program Description: This program reads two SAT verbal scores(between 0 and 800)

// for a student and prints the larger of the two scores.

//

#include <iostream>

using namespace std;

// Global Constants

const int MAX\_SCORE = 800;

const int LOW\_SCORE = 0;

// Function Prototypes

void instructions(void); // Function to display instructions to the user

void readScore(int &score); // Function get the scores from the user

bool outOfRange(int score); // Boolean valued function will return true or false

int max(int x, int y); // Returns the score that is the largest

int main(void){

int verbalScore1, // SAT verbal score #1

verbalScore2; // SAT verbal score #2

instructions(); // Display instructions to the user

readScore(verbalScore1); // Get the first test score

if(outOfRange(verbalScore1))

cout << verbalScore1 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else{

readScore(verbalScore2); // Get the second test score

if(outOfRange(verbalScore2))

cout << verbalScore2 << " is not between " << LOW\_SCORE << " and "

<< MAX\_SCORE << ". \nProgram Terminating" << endl << endl;

else

cout << "The larger score is " << max(verbalScore1, verbalScore2) << endl;

}

return 0;

}

//

// Function to print instructions on what the program does

//

void instructions(void){

cout << "This program interactively reads two SAT\n";

cout << "verbal scores and prints the larger\n\n";

}

//

// Function to read a score interactively

//

void readScore(int &score){

cout<< "Enter an SAT verbal score between " << LOW\_SCORE << " and "

<< MAX\_SCORE <<" ======> ";

cin>> score;

}

//

// Function to return true if the SAT score is out of

// range, false otherwise. Range: 0-800

//

bool outOfRange(int score){

return (score < LOW\_SCORE || score > MAX\_SCORE);

}

//

// Function to return maximum of two integer parameters

//

int max(int x, int y){

return (x > y ? x : y);

}

Label each run of your program by stating the test data used and which run number it is. **For example:**

Run #1:

Test Data: 550, 600

Results:

This program interactively reads two SAT

verbal scores and prints the larger

Enter an SAT verbal score between 0 and 800 ======> 550

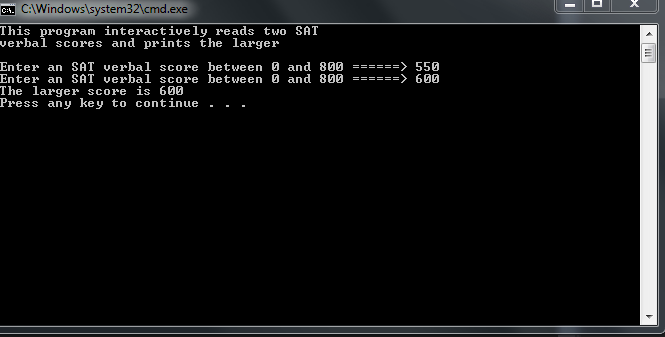
Enter an SAT verbal score between 0 and 800 ======> 600

The larger score is 600

Press any key to continue

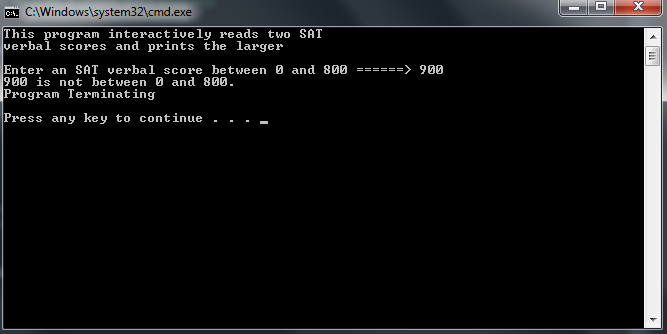
Run #1:

Test Data: 500, 600



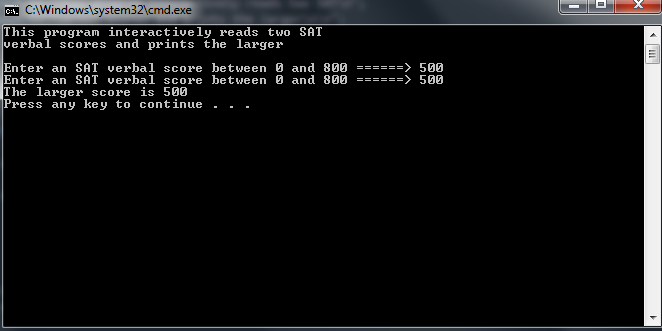
Run #2

Test Data: 900



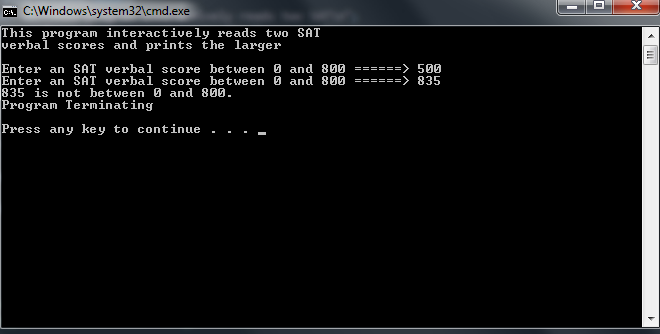
Run #3:

Test Data: 500, 500



Run #4:

Test Data: 500, 835



**Due Dates:**

As indicated on the Lab Ten Dropbox Folder.

**What to hand in:**

1. Save the word document using your name and the lab number as the file name. eg. timwrennlab10.docx.
2. Place the word document into the Lab Experience Ten drop box folder.