**Submission of Your Work**

You need to prepare and submit ONE SINGLE MS Word document to Canvas (in your lab section) as LastName\_FirstName\_Labxy.doc. It must contain:

* Your NAME only on page 1
* For ***each*** question:
  + Specify the question number.
  + Source code. Copy/Paste your final source code. You must include standard “comment header” even if code is provided. *Do Not* paste a snippet of your source code, it must be copy/pasted.
  + Initial test plan. After reading the question requirements, but **before** beginning any coding, create the test case table, below, completed through column Expected Output. Include in your report.
  + Final test plan. Write your program then complete the **test table** with actual output results and include in your report *AFTER* your source code.
  + Output results. Paste in a snippet of output showing results for **every listed test case in your final test plan**, labeled with test case #

Test Table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test # | Valid / Invalid Data | Description of test | Input Value | Expected Output | Actual Output | Test Pass / Fail |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

* Add / delete rows from Test Table as necessary
* Modify column widths as necessary
* Test both valid and invalid input
* Test for every output expected
* If failure is an expected output and it happens then that test Passes
* Any test that fails means the program must be fixed so that it passes the test
  + Failing tests need a new test row, ie 1a, 1b, etc, showing corrections from original

**Header File Definition – override C++ vector definition**

#ifndef VECTOR\_H

#define VECTOR\_H

class Vector

{

public:

Vector (); // default constructor

Vector (int s); // makes size = s,

//allocates s space

// e.g. entries = new int[size],

// makes all entries 0

Vector (const Vector & other);

// copy constructor

// makes a deep copy

~Vector (); // default destructor

void print (); // Prints out the vector

void set( int val, int pos); // if 0 <=pos<size

// stores val at pos in entries

// otherwise

// error message

private:

int size; // sets the # of elements used

int \*entries; // point to array of integers with size entries

// e.g. entries = new int[size]

};

#endif

**Print ( ) Specification**:

Print will put out the elements surrounded by []’s as shown below

[1 2 3]

**Write methods defined in header file and use the following “stub” Main to test header file**

#include “Vector.h”

#include <iostream>

int main()

{

// REQUIRED CODE

Vector a, b(3), c(3) ;

a.print(); // outputs []

b.print(); // outputs [ 0 0 0 ]

c.set(0,-1); // output error message

c.set(1,0);

c.set(2,1);

c.set(3,2);

c.set(4,3); // outputs error message

c.print(); // outputs [ 1 2 3 ]

Vector d(c);

d.print(); // outputs [ 1 2 3 ]

d.set(0,1);

d.print(); // outputs [ 1 0 3 ]

c.print(); // outputs [ 1 2 3 ] proves deep copy

// ADDITIONAL TEST CASES

[ Insert your code for YOUR test cases here ]

return 0;

}

**Program 2.**

Write a function to calculate the square root of a float number with the following interface:

double squareRoot( double x)

{

// assert that x is not negative

...

}

The function should return an approximation to using series approximation.

Let x0 = x/2. Then x­n+1 = (xn + x/xn)/2. Keep computing terms until the difference between xn and xn+1 is less than 0.0001. In addition, if the input variable x is negative, your function should stop the execution via the assert( ) function, which is a built-in C++ function. You need to add a function call to assert( ) at the beginning of the above function to guarantee the precondition of this function is correct.

In your main function, design it to request the user to enter as many numbers as they want (continue to enter values? “y” or “n”). Test cases must include values 3,0, and -3, as well as others you think are appropriate.

The specifics of the assert message shown depends on the specific implementation in the compiler, but it should include: the <*bool exp*> whose assertion failed, the name of the source file, and the line number where it happened. A usual expression format is:  
  
Assertion failed: *<bool exp>* file *filename*, line *line number*

#include <cassert>

void print\_number(int myInt) {

assert (myInt > 5);

This sample snippet will abort when myInt has value of 5 or less because myInt > 5 evaluates to false.

**Program 3.**

Redo program 2 using try/catch instead of assert, using multiple catches [negative, default, others?]