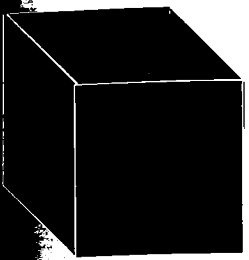
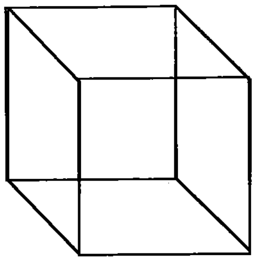
Course Info: ______________ ***Name:*** ______________________________________

Lab 1.1 Black-box and White-box Program Testing



Objective: Lab Exercise 1.1 is related to Chapter 1: "Software Development" in the text *ADTs, Data Structures, and Problem Solving in C++*, 2E and is intended to help you understand two methods of testing a program as described in Section 1.4 and to provide a review of some basic statements and arrays in C++.



In the Text: Read Section 1.4: "Testing, Execution, and Debugging," starting on page 30 of the text. The terms *black-box* testing and *white-box* testing are introduced on page 32.

Description: This exercise demonstrates the difference between *black-box* and *white-box* testing. In black-box testing, one simply executes the program with test data without any knowledge of the program's contents—the program is said to be a black box because only the externals are available. In white-box testing, one analyzes the program's structure by looking inside the program and tracing its execution for various data sets, selecting them in such a way as to exercise the various paths of program execution.

Black-box Testing

Your instructor will give you directions on how to access or prepare a binary executable program for searching a matrix with 3 rows and 3 columns for some number. (A listing of the source program `search.cpp` used to prepare this binary executable is on the last page of this lab exercise.) If you are working on some other system, you will need to generate this binary executable yourself, using the source code on the last page. The program searches a 3×3 matrix for some number. You are to carry out black-box testing of this program beginning with the matrix

$$\begin{bmatrix} 45 & 77 & 93 \\ 78 & 79 & 85 \\ 72 & 96 & 77 \end{bmatrix}$$

☐ $\boxed{1}$ Following the output prompt, enter this test matrix into the program.

☐ $\boxed{2}$ Search for value 77. What output is produced?

________________________ Is this expected? Y|N ____

☐ $\boxed{3}$ Search for value 99. What output is produced?

________________________ Is this expected? Y|N ____

At this point do you feel confident that the program is correct? Y|N ____

Why? or Why not?

____

$\boxed{4}$ Try searching for other values and/or with another matrix. Try enough values to show that your testing has either turned up an error or given you confidence that the program is correct. Record what inputs you used and the resulting output.

Number entered	Found or not found

Is the program correct or incorrect at this point in your estimation?

Correct ____ Incorrect ____

White-box Testing

We are now ready to explore white-box testing. The function being used to search during our black-box testing is given below (with some additional comments):

```
bool matrixSearch(Matrix & mat, int n, int item)
/*--------------------------------------------------------------------
 Search the n X n matrix mat in rowwise order for item.

  Precondition: Matrix mat is an n X n matrix of integers with n > 0.
  Postcondition: True is returned if item is found in mat, else false.

  NOTE: mat[row][col] denotes the entry of the matrix in the
        (horizontal) row numbered row (counting from 0) and the
        (vertical) column numbered col.
 -------------------------------------------------------------------*/
{
  bool found;                              // 1
  for (int row = 0; row < n; row++)        // 2
    for (int col = 0; col < n; col++)      // 3
      if (mat[row][col] == item)           // 4
        found = true;                      // 5
      else                                 // 6
        found = false;                     // 7
  return found;                            // 8
}
```

$\boxed{5}$ Read the code over carefully. Does it appear to you to be correct? Y|N ____
If you think it isn't and know why it is incorrect, give the reason in the space below:

✍ $\boxed{6}$ There are two general ways to trace code: 1) using a debugger or debugging statements inserted in the code, and 2) manually (desk checking). Complete the following manual trace using

$$\texttt{mat} = \begin{bmatrix} 45 & 77 & 93 \\ 78 & 79 & 85 \\ 72 & 96 & 77 \end{bmatrix} \text{ and } \texttt{item} = 77$$

The numbers in the column labeled "Statement number" are taken from the inline comments in the source code for `matrixSearch()` on the preceding page.

Statement number	`row`	`col`	`mat[row][col]`	`found`
1	?	?	?	? (? = undefined)
2	?	?	?	?
3	0	0	45	?
4,7	0	0	45	false
3	0	1	77	false
4,5	0	1	77	true
3	0	2		

✍ $\boxed{7}$ Complete the following trace using `mat` = $\begin{bmatrix} 45 & 77 & 93 \\ 78 & 79 & 85 \\ 72 & 96 & 77 \end{bmatrix}$ and

`item` = one of the values used earlier to show that the program was incorrect

Statement number	`row`	`col`	`mat[row][col]`	`found`
1	?	?	?	? (? = undefined)
2	?	?	?	?
3	0	0	45	?
4,____	0	0	45	
3	0	1	77	
4,____	0	1	77	
3	0	2		

8 We will now try some automated tracing. Using the program `search.cpp` on the last page of this handout (downloadable from the website whose URL is given in the preface), add the output statement highlighted below that displays the state of the loop on each iteration and execute the program with the values used in your trace tables.

```
{
  if (mat[row][col] == item)
    found = true;
  else
    found = false;
  cerr << row <<" " << col << " "<< mat[row][col] <<" "
       << boolalpha << found << endl;
}
```

After examining the traces produced by the output, it should be clear why the program fails to produce the correct output. Describe the reason clearly below:

---

9 Once the error is identified, it needs to be corrected. If we replace the function `matrixSearch()` with the following, then the program is correct.

```
bool matrixSearch(Matrix & mat, int n, int item)
{
  bool found=false;
  for (int row = 0; row < n; row++)
    for (int col = 0; col < n; col++)
     if (mat[row][col] == item)
           found = true;
  return found;
}
```

Run this corrected version of `matrixSearch()` and confirm that is correct.

Check here when finished _______

10 The preceding function `matrixSearch()` is correct but is not very efficient. It can be *easily* modified, however, to produce a better, more efficient, solution to the problem of searching a matrix. Make this change to the code and explain below why it is more efficient than the earlier version.

You have finished! Hand in: 1) this lab exercise with answers filled in, and 2) printouts of (i) your final program, (ii) evidence that it compiles correctly, and (iii) an execution using the values you used to test the original function in the first part of this lab.

```
/*--- search.cpp -------------------------------------------------------
  Program to read a 3 X 3 matrix of integers mat and an integer item,
  and search mat to see if it contains item.

  Add your name here and other info requested by your instructor.
 ---------------------------------------------------------------------*/

#include <iostream>
using namespace std;

const int SIZE = 3;   //Set Matrix size
typedef int Matrix[SIZE][SIZE];

bool matrixSearch(Matrix & mat, int n, int item);
/*----------------------------------------------------------------------
  Search the n X n matrix mat in rowwise order for item.

  Precondition:  Matrix mat is an n X n matrix of integers with n > 0.
  Postcondition: True is returned if item is found in mat, else false.
 ---------------------------------------------------------------------*/

int main()
{
 // Enter the matrix
  Matrix mat;
  cout << "Enter the elements of the " << SIZE << " X " << SIZE
       << " matrix rowwise:\n";
  for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
      cin >> mat[i][j];

 // Search mat for various items
  int itemToFind;
  char response;
  do
  {
    cout << "Enter integer to search for: ";
    cin >> itemToFind;
    if (matrixSearch(mat, SIZE, itemToFind))
      cout << "item found\n";
    else
      cout << "item not found\n";
    cout << "\nMore items to search for (Y or N)? ";
    cin >> response;
  }
  while (response == 'Y' || response == 'y');
}

//-- (-- Incorrect --) Definition of matrixSearch()
bool matrixSearch(Matrix & mat, int n, int item)
/*---------------------------------------------------------------------
  Search the n X n matrix mat in rowwise order for item

  Precondition:  Matrix mat is an n X n matrix of integers with n > 0.
  Postcondition: True is returned if item is found in mat, else false.

  NOTE: mat[row][col] denotes the entry of the matrix in the
        (horizontal) row numbered row (counting from 0) and the
        (vertical) column numbered col.
 --------------------------------------------------------------------*/
{
  bool found;
  for (int row = 0; row < n; row++)
    for (int col = 0; col < n; col++)
      if (mat[row][col] == item)
        found = true;
      else
        found = false;
  return found;
}
```