Lab Manual
to Accompany

ADTs, Data Structures,
and
Problem Solving
with **C++**

SECOND EDITION

LARRY
NYHOFF

PEARSON
Prentice
Hall

Upper Saddle River, New Jersey 07458

Vice President and Editorial Director, ECS: *Marcia J. Horton*
Acquisitions Editor: *Tracy Dunkelberger*
Assistant Editor: *Sarah E. Parker*
Editorial Assistant: *Christianna Lee*

Executive Managing Editor: *Vince O'Brien*
Managing Editor: *Camille Trentacoste*
Production Editor: *Donna M. Crilly*
Director of Creative Services: *Paul Belfanti*
Cover Designer: *Daniel Sandin*

Manufacturing Manager, ESM: *Alexis Heydt-Long*
Manufacturing Buyer: *Lisa McDowell*
Executive Marketing Manager: *Pamela Shaffer*
Marketing Assistant: *Barrie Reinhold*

Pearson Prentice Hall
Pearson Education, Inc.
Upper Saddle River, NJ 07458

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Printed in the United States of America

ISBN: 0-13-148758-2

Pearson Education Ltd., *London*
Pearson Education Australia Pty. Ltd., *Sydney*
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.. *Hong Kong*
Pearson Education Canada, Inc., *Toronto*
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education—Japan, *Tokyo*
Pearson Education Malaysia, Pte. Ltd.
Pearson Education, Inc., *Upper Saddle River, New Jersey*

Preface

This lab manual contains a collection of lab exercises and projects that have been used successfully in my CS2 course for the past several years. They are coordinated with the material in the first 14 chapters of my text *ADTs, Data Structures, and Problem Solving with C++*, 2E. (For more information about the text see the website at `cs.calvin.edu/books/c++/ds/2e`.)

For most of the lab exercises, there is a corresponding project that builds on the lab. In a few cases there are two projects. It is not intended that both projects be assigned but rather that one be selected that best fits the content and nature of your course. Also, in two cases, there are two labs and two projects for a single chapter (Chapter 2 and Chapter 10). Here, too, one can pick and choose a combination of them.

Some of the labs and projects may have more material than is appropriate for your course. Almost all of them are designed, however, in such a way that parts of them may be omitted or replaced.

Typically, each lab involves a file that students use as they work through the lab exercise, modifying it, compiling and executing it, and reporting what happens. These files can be downloaded from the website for this lab manual:

`http://cs.calvin.edu/books/c++/ds/2e/labmanual`

An errata list is also available there.

The lab exercises and projects have all been used in a course in which some students worked in Unix-based gnu C++ and others used Microsoft's Visual C++ or Metrowerks' CodeWarrior C++. They should all compile, link, and execute in any environment that supports ANSI standard C++. Any problems or necessary modifications that are reported to me will be posted at the website given above.

I thank Professor Raymond Schneider of Bridgewater College for his hard work on updating and improving the first edition of the lab manual. Much of what he did has been incorporated into this new edition. Any responsibility for errors, however, belongs to me and corrections, comments, and suggestions about the lab exercises and projects should be e-mailed to me at the address below. Also feel free to contact me if there are difficulties with accessing items at the above website or if you have other questions.

Larry Nyhoff
Calvin College
Dept. of Computer Science
nyhl@calvin.edu

Contents

Lab 1.1 Black-box and White-box Program Testing

NOTES TO THE INSTRUCTOR

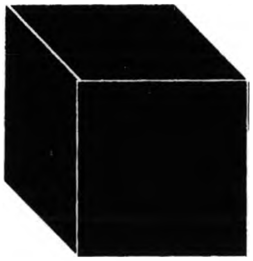
The first three lab exercises (1.1, 2.1, and 2.2) and projects (2.1 and 2.1) are intended for use in the first week(s) of the course. Lab 1.1 is related to Chapter 1: "Software Development" in the text *ADTs, Data Structures, and Problem Solving in C++*, 2E, more specifically, to Section 1.4: "Testing, Execution, and Debugging." In fact, it is a lab version of Programming Problems 2 and 3 at the end of Chapter 1. Programming Problems 1 and 4 can be used as supplementary exercises to tie in issues of code structure. This lab exercise also provides a review of some basic statements and arrays in C++.

Notes:

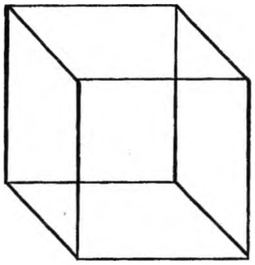
Lab Exercise 1.1 uses the file `search.cpp`, which can be downloaded from the website whose URL is given in the preface. You should prepare a binary executable called `search` (or whatever you prefer) from the source file `search.cpp` and make both files available to the students or provide `search.cpp` to them and have them compile and execute it but instruct them ***not*** to examine the code in the first part of the lab exercise (Black-box Testing).

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?

____

[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
}
```

[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		

✍ □7 Complete the following trace using $\texttt{mat} = \begin{bmatrix}45 & 77 & 93\\ 78 & 79 & 85\\ 72 & 96 & 77\end{bmatrix}$ 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)
else;//continue  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;
}
```