IE 170 Laboratory 3: Stacks

Dr. T.K. Ralphs

Due February 13, 2006

1 Laboratory Description and Procedures

1.1 Learning Objectives

1. Understand each of the key terms listed below.
2. Understand the concept of a new data type.
3. Understand the role of the interface and the implementation with respect to designing and implementing a new data type.
4. Understand how to implement a new data type in C++ using classes.
5. Understand how to write a client application using the interface to a data type.

1.2 Key Words

1. data type
2. C++ class
3. data member
4. member function
5. interface
6. implementation
7. client

1.3 Scenario

The assignment for the lab is to write a program that could help a robot find its way from its starting point through a series of corridors to a specified destination or conclude that there is no path to the destination. Such a program could possibly be used to guide a robot through a factory or even to find a path through a space too confined for humans, such as a collapsed building. The very simple algorithm that we will use for this lab makes use of trial and error to find a path. We will use a *stack* data structure to keep track of the path followed so far so that we can retrace our steps when we reach a dead end.

In this scenario, we want the robot to be able to make fast decisions and therefore our stack data structure should be as efficient as possible at performing the necessary operations. Your primary task is to design a stack data structure that can be used with the maze program as a *client.* You will also be asked to complete the shell of the maze program itself.

1.4 Program Specifications

1.5 Designing and Analyzing the Algorithm

For this lab, we will not focus on analyzing the algorithm, but rather on designing and analyzing the data structure. Up until now, we have primarily used data types that are native to the C++ language. Here, we will further illustrate the design of new data types, as well as the concept of separating the interface from the implementation and the use of a data type by a client program using an interface. To analyze a data structure, we will look at the running times of each of its operations. In the case of a stack, the basic operations are putting an item on the stack and taking an item off the stack.

1.5.1 Program Function

The program should begin by reading in the maze from a file. The format of the maze is a rectangle formed by X's and O's. The X's denotes the squares that the robot may travel into and the O's denote the squares where the robot may not travel. To begin with, each square of the maze is marked either as a WALL or as UNVISITED. Once the maze is read in, the robot can begin to explore. To move through the maze, the robot may go from the square it currently occupies (always starting from the top left corner) to any of the four squares that may be reached by going right, left, up, or down and is marked UNVISITED. Once the robot chooses to move to a square, it is immediately marked VISITED. Once at the new square, the robot should try each new direction in turn and if no direction is found to be UNVISITED, then it must backtrack using the stack.

The stack stores all moves made to arrive at the current square. After making each move, the move is put on the stack so that it can be undone. Once the robot begins exploring (always from the top left corner), it should continue to follow the above algorithm until either finding a path to the exit (always at the bottom right corner) or concluding that there is no such path (when the stack is empty and the robot has backtracked all the way back to the starting point). If a path to the exit is found, then the stack can be read to determine the path taken. If a path is found, it should be printed to a file in the same format as the input except that a "*" should appear in any square that is part of the path.

1.5.2 Algorithms

The algorithm is described above.

1.5.3 Data Structures

This program should be written in an object-oriented fashion with adherence to the basic principles discussed in lecture regarding the separation of interface from implementation. The class for storing the maze has been defined for you, as well as the interface for the stack class. You should implement the stack class and design a class for a linked list. The maze itself should be stored in a two-dimensional array. The set of moves taken to get to the current position should be stored in a stack. The stack should be implemented using the linked list class.

2 Laboratory Test Files

The test files for this laboratory are in the zip archive `Lab3.zip` available on the course Web site. The archive will unpack into a directory called `Lab3` with subdirectories `data`, `generator`, and `shell`. In the `data` subdirectory, the files named `*.dat` contain graphs that you can use to test your code. The files `*.sol.txt` contain the solutions.

3 Laboratory Assignments

3.1 Programming (50 points)

1. Write a program that behaves in the specified manner. You may use the files in the `shell` subdirectory as a guide.
2. Verify that your program compiles and runs correctly using the test files provided for the lab.

3.2 Analysis (20 points)

1. (5 points) Analyze the running time of each of the operations performed by your stack data structure.
2. (5 points) Compare the linked list implementation of a stack to one using arrays.
3. (10 points) What is the theoretical worst case running time of finding a path through the maze?

3.3 Follow-up Questions (30 points)

- (10 points) CLRS 4.3-2
- (10 points) CLRS 4-1 (a,f,h), 4-4 (c,h)
- (10 points) CLRS 4-3 (a)