Department of Information Technology

HD in Game Software Development (IT114107)
ITP4713 Introduction to Object Oriented Programming
Assignment

Assignment Specification

This assignment is to implement a maze solver using Object-Oriented Programming and Data Structure Techniques. The maze is stored in a 2D character array, where `'s'` represents starting point, `'t'` represents the target, space `' '` represents walkable path, `'#'` represent wall, below is an example

```
  0123456
0 s#
1  ## ##
2  ## ###
3
4 ### ###
5   #   #
6 #   #
7 t # ##
```


Your maze solver will do the following:

1. Find the position of `'s'` and `'t'` from the 2D array
2. Start to solve the maze by using depth first search (select the first available path), starting from `'s'` until `'t'` is found. During the search, if no more possible move, the solver will backtrack to last position which has alternative choice and search again. Failed position will be marked by `'x'` and success position will be marked by `'.'`.

Following is the sample result (Note that it is not a must all cell are filled at the end of the search):

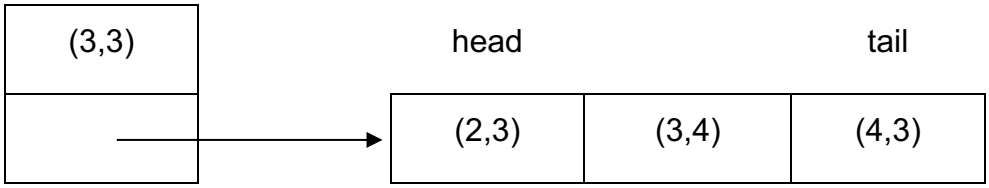
```
  0123456
0 s#xxxxx
1 .##x##x
2 .##x###
3 ....xxx
4 ###.###
5 xx#.xx#
6 #...#xx
7 t.#x##x
```

Requirement

Your program is required to create a 2D array (either statically or dynamically) which store the maze. The maze is hardcoded (you may also read it from a data file, which is the bonus part of this assignment).

Before the solver start, the maze is passed to a function by reference and get the starting and target position by reference.

After that, the solver starts to solve the maze from the starting point. The depth first search can be done by **a stack of data structure** which store the visiting node and its possible moves in the order of up, right, down, left, below is an example of a **single data structure**:



The search starts by visiting the starting point, i.e. pushing a data structure (starting point and its possible moves) of the starting point to stack and keep searching target from the first available move. Below is the suggested procedure:

Visit (Push)

If not destination

    Find possible moves (empty, i.e. not # . x s)

    If there is a possible move

        Mark current visit by '.'.

        Remove the first possible move from list

        Visit the first possible position (i.e. repeat the first step)

    else

        Mark current visit by 'x'.

        Backtrack to last move (Pop)

        Re-visit the last move and visit the next possible move (if any)

else

  Target found

Note about the pop:

If all data structure has been popped, the maze cannot be solved, i.e. no solution.

Following are some sample output of different mazes, you are required to print the **question**, **steps of solving** and also the **final result**:

Sample 1: Simple (success)

```
  0123
0 #s##
1 # ##
2 # ##
3 #t##


Visit (0,1) => { (1,1) }
  Next move (1,1), updated list: { }
Visit (1,1) => { (2,1) }
  Next move (2,1), updated list: { }
Visit (2,1) => { (3,1) }
  Next move (3,1), updated list: { }
Visit (3,1) => Goal!


  0123
0 #s##
1 #.##
2 #.##
3 #t##
```

Sample 2: Simple (backtrack)

```
  0123
0 #s##
1 #
2 # ##
3 #t##


Visit (0,1) => { (1,1) }
  Next move (1,1), updated list: { }
Visit (1,1) => { (1,2), (2,1) }
  Next move (1,2), updated list: { (2,1) }
Visit (1,2) => { (1,3) }
  Next move (1,3), updated list: { }
Visit (1,3) => { }
  No possible move, backtrack to (1,2)
Re-visit (1,2) => { }
  No possible move, backtrack to (1,1)
Re-visit (1,1) => { (2,1) }
  Next move (2,1), updated list: { }
Visit (2,1) => { (3,1) }
  Next move (3,1), updated list: { }
Visit (3,1) => Goal!


  0123
0 #s##
1 #.xx
2 #.##
3 #t##
```

Sample 3: Simple (failed)

```
  0123

0 #s##

1 # ##

2 # ##

3 ###t



Visit (0,1) => { (1,1) }

  Next move (1,1), updated list: { }

Visit (1,1) => { (2,1) }

  Next move (2,1), updated list: { }

Visit (2,1) => { }

  No possible move, backtrack to (1,1)

Re-visit (1,1) => { }

  No possible move, backtrack to (0,1)

Re-visit (0,1) => { }

  No possible move, cannot backtrack anymore



No solution

  0123

0 #s##

1 #x##

2 #x##

3 ###t
```

Sample 4: Another example

```
  0123

0 #s##

1 # #

2 t

3 # #


Visit (0,1) => { (1,1) }

  Next move (1,1), updated list: { }

Visit (1,1) => { (2,1) }

  Next move (2,1), updated list: { }

Visit (2,1) => { (2,2), (3,1), (2,0) }

  Next move (2,2), updated list { (3,1), (2,0) }

Visit (2,2) => { (2,3) }

  Next move (2,3), updated list { }

Visit (2,3) => { (1,3), (3,3) }

  Next move (1,3), updated list { (3,3) }

Visit (1,3)  => { }

  No possible move, backtrack to (2,3)

Re-visit (2,3)  => { (3,3) }

  Next move (3,3), updated list { }

Visit (3,3)  => { }

  No possible move, backtrack to (2,3)

Re-visit (2,3)  => { }

  No possible move, backtrack to (2,2)
```

```
Re-visit (2,2)  => { }

  No possible move, backtrack to (2,1)

Re-visit (2,1) => { (3,1), (2,0) }

  Next move (3,1), updated list { (2,0) }

Visit (3,1)  => { }

  No possible move, backtrack to (2,1)

Re-visit (2,1) => { (2,0) }

  Next move (2,0), updated list { }

Visit (2,0) => Goal!


  0123
0 #s##
1 #.#x
2 t.xx
3 #x#x
```

Bonus

Instead of hardcode the maze, you can implement a bonus part of the assignment, which read the maze from a file with the following format

1. The first row stored the number of row and column of the maze
2. The second row and after stored the maze

Example:

```
8 7
s#
 ## ##
 ## ###

### ###
  #   #
#   #
t # ##
```



```
#include <iostream>
#include <fstream>
using namespace std;


char** arr2D;//maze golobal 2d CHAR ARRAY
int size2D[2];//stor len of x and y
int sPosition[2];//s position
int tPosition[2];//t position


void readDataFromFileLBLInttoCharArray()
{
	ifstream fin("maze.txt");//read file
	const int LINE_LENGTH = 100;
```

```
        char str[LINE_LENGTH];//size 100 char array
        fin.getline(str, LINE_LENGTH);
        size2D[0] = int(str[0]) - 48;
        size2D[1] = int(str[2]) - 48;
        //creat 2D arrary
        arr2D = new char*[size2D[0]];
                for (int i = 0; i < size2D[0]; ++i)
                        arr2D[i] = new char[size2D[1]];
```



```
        int postY = 0;
```



```
        while (fin.getline(str, LINE_LENGTH))
        {
```



```
                for (int i = 0; i < size2D[1]; i++)
                {
                        arr2D[postY][i] = str[i];
                }
                postY++;
        }
}

int main() {
        readDataFromFileLBLInttoCharArray();
        cout << size2D[0] << " "<<size2D[1] << endl;
```

```
        cout << " ";
        for (int i = 0; i < size2D[1]; i++)
        {
                cout << i;
        }
        cout << endl;
        for (int i = 0; i < size2D[0]; i++)
        {
                cout << i << "";
                for (int j = 0; j < size2D[1]; j++)
                {
                        cout << arr2D[i][j];
                }
                cout << endl;
        }
        system("pause");
        return 0;
}
```


Technical Requirement

In this assignment, you are required to use a 2D array to model the maze. Linked List and Stack to solve the maze. Following is the mark distribution:

Linked List:	10 marks
Stack:	10 marks
Maze:	5 marks
Find s and t:	5 marks

Solve the maze:	35 marks
Print the question:	5 marks
Print the steps:	5 marks
Print the result:	5 marks
Documentation:	20 marks
Bonus:	10 marks

Documentation includes evidence of testing (10%) and explanation on how data structure is used (10%)

Instructions to Students

1. This is an End of Module Assessment and the weighting of this assignment is 50% of the Module Mark.
2. This assignment should be done by each individual student. Plagiarism will be treated seriously. All assignments that have been found involved wholly or partly in plagiarism (no matter these assignments are from the original authors or from the plagiarists) will score Zero mark.

3. You must use C++ to develop the programs.
4. You are required to hand in
    - 5.1 Explanation on how data structure is used in your program in MS Word
    - 5.2 Source code of all classes which should be well-commented.
    - 5.3 The evidence of testing. Prepare a word document with several test cases showing different inputs for different situations that your program may encounter and how your program responses to show the capability of your program. For each test case, states the objective of the test case, input data and expected result. You should also include screen dump for each test run as evidence.

6. Submit all your works (in a zip file) to the Moodle website (http://moodle.vtc.edu.hk) by 23:55, 17 July 2020 (Friday). Late submission may score ZERO mark.

7. Folders in the zip file

    Program

        Main.cpp

        LinkedList.h

        …

    Documents

        Data_Structure_Explanation.doc

        Evidence_of_Testing.doc