National Yang Ming Chiao Tung University
Department of Electrical and Computer Engineering
Data Structures, Fall 2023

November 7, 2023
Instructor: H.-M. Chen
TA: S.-P. Wang

Programming Assignment #3

Stacks & Queues

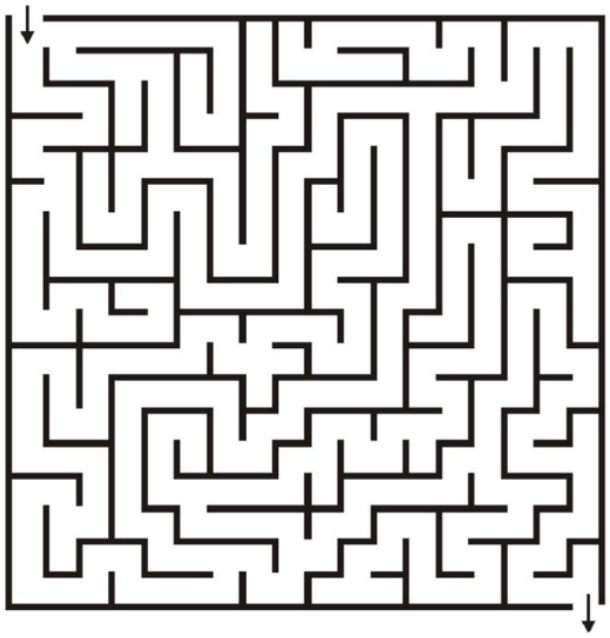
Be sure to design your own "pointer-based Stack" in this programming assignment (array-based stacks will not be graded). You are not allowed to use any container in the Standard Template Library (STL), or any other open-source library, such as vector, list, map, set, and all the other container classes. The submitted source codes will NOT be graded if those open-source container classes are found in your source codes.

1. Problem Description

In this assignment, you'll be given a **nxn perfect maze** , which means the maze has only **1 entrance (0,1)** and **1 exit (n-1, n-2)** , the positions of entrance and exit will be fixed and will not vary between different cases. you're asked to use **stack** to solve this maze, and print out the stack operations during the process and the maze graph after solved.

The priority order of the directions while discovering the maze will be :

right > down > left > up



2. Input Format

All input data comes from standard input (**cin**)

The 1st line in **case.txt** has a single **integer n** , (**5 ≤ n ≤ 101,** n is an **odd number**)
The following **n lines** represent a **n x n maze**.

means **wall** **.** means **passage**

3. Output Format

All output data should go to standard output (**cout**)

The **first K lines** represent total K stack operations in the format:

Stack_op + direction + current_x + current_t

separated by single space character, with an end-line character at the end of line.
The **following n lines** represent the maze image after solved :

means **wall** **.** means **passage** **+** means **right path**

4. Sample Input / Output

Sample Input 1

```
5
#.###
#.#.#
#.#.#
#...#
###.#
```

Sample Output 1

```
Push down 0 1
Push down 1 1
Push down 2 1
Push down 3 1
Push right 3 2
Push right 3 3
Push down 4 3
#+###
#+#.#
#+#.#
#+++#
###+#
```

Sample Input 2

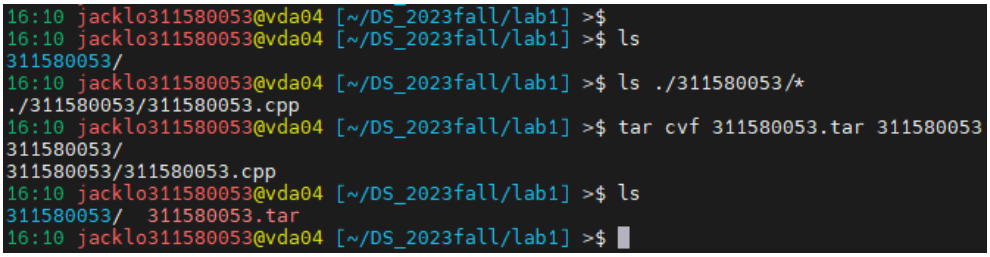
```
11
#.#########
#.........#
#.#########
#.........#
#.#.#######
#.#.#.....#
#.#.#####.#
#.#.......#
#####.#.#.#
#.....#.#.#
#########.#
```

Sample Output 2

```
Push down 0 1
Push down 1 1
Push right 1 2
Push right 1 3
Push right 1 4
Push right 1 5
Push right 1 6
Push right 1 7
Push right 1 8
Push right 1 9
Pop right 1 8
Pop right 1 7
Pop right 1 6
Pop right 1 5
Pop right 1 4
Pop right 1 3
Pop right 1 2
Pop right 1 1
Push down 2 1
Push down 3 1
Push right 3 2
Push right 3 3
Push right 3 4
Push right 3 5
Push right 3 6
Push right 3 7
Push right 3 8
Push right 3 9
Pop right 3 8
Pop right 3 7
Pop right 3 6
Pop right 3 5
Pop right 3 4
Pop right 3 3
Push down 4 3
Push down 5 3
Push down 6 3
Push down 7 3
Push right 7 4
Push right 7 5
Push right 7 6
Push right 7 7
Push right 7 8
Push right 7 9
Push down 8 9
Push down 9 9
Push down 10 9
#+#########
#+.........#
#+#########
#+++......#
#.#+#######
#.#+#.....#
#.#+#####.#
#.#++++++++#
#####.#.#+#
#.....#.#+#
#########+#
```

5. Submission Information

1. Your program must be written in C/C++ language and can be compiled on the Linux platform.
2. Please put the required files in a folder named with your Student_ID and the required files should also be named with your Student_ID (.cpp, .c).
3. To submit your program, please use the command below to compress the folder named with "[Student_ID].tar" in the Linux environment and upload it to E3.
   tar cvf Student_ID.tar Student_ID



File hierarchy example:

```
311555008.zip
|  311555008                  # folder
|  |  311555008.cpp           # source file
```

6. Due Date

Be sure to upload the tar file by "November 21, 2023". There will be a 10% penalty per day for the first four days (weekend included) and will not be accepted afterwards.

7. Grading Policy

The programming assignment will be graded based on the following rules:

- Pass the open cases with compilable source code (60%)
- Pass the hidden cases with compilable source code (40%)
- -10% of your total score if any file occurs naming error or not compress
- You need to use your **self-made stack structure** during the implementation, **If you directly solve the problem by using ex. Recursion, you would get any score !!!**

- ***No credits for plagiarism***

8. How to use create more test data

1. Generate new maze using **maze_generator**

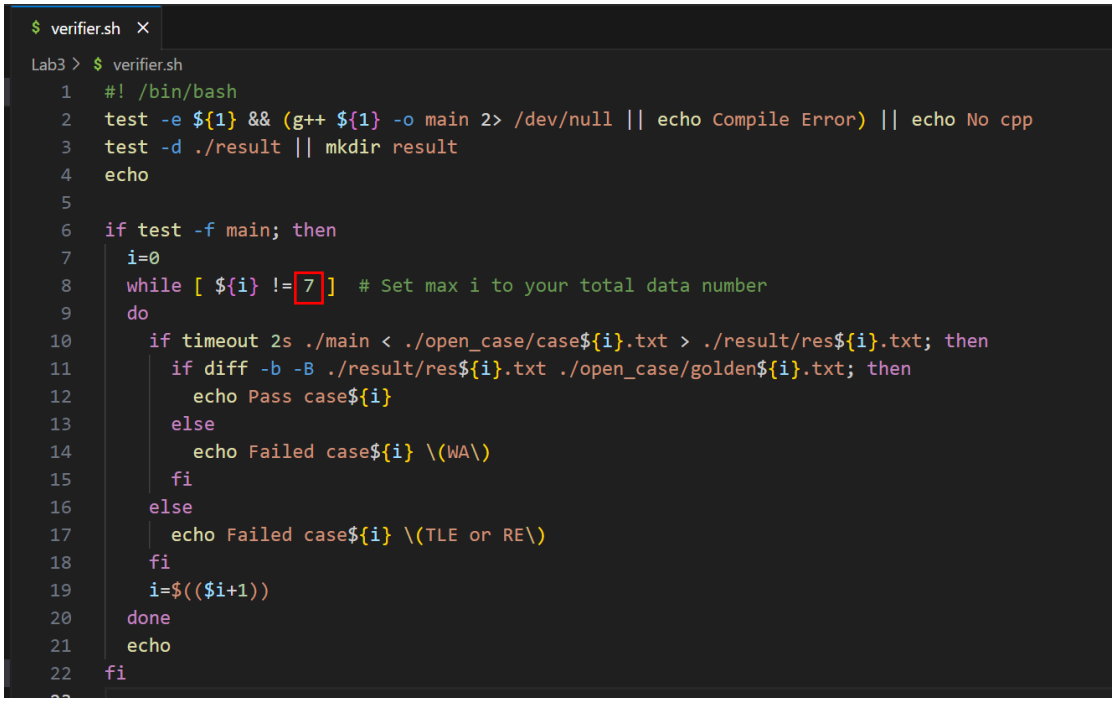
```
10:19 tommy25582143@vda04 [~/DS_TA/Test] >$ tar xvf Lab3.tar
Lab3/
Lab3/result/
Lab3/maze_generator
Lab3/open_case/
Lab3/open_case/golden3.txt
Lab3/open_case/case4.txt
Lab3/open_case/case0.txt
Lab3/open_case/golden5.txt
Lab3/open_case/golden1.txt
Lab3/open_case/case1.txt
Lab3/open_case/case2.txt
Lab3/open_case/case5.txt
Lab3/open_case/case3.txt
Lab3/open_case/golden4.txt
Lab3/open_case/golden2.txt
Lab3/open_case/golden0.txt
Lab3/golden_generator
Lab3/verifier.sh
10:19 tommy25582143@vda04 [~/DS_TA/Test] >$ cd Lab3
10:19 tommy25582143@vda04 [~/DS_TA/Test/Lab3] >$ chmod 755 maze_generator
```

```
10:22 tommy25582143@vda04 [~/DS_TA/Test/Lab3] >$ ./maze_generator open_case/case6.txt

--------------------------------
   Maze generation algorithms
https://github.com/ferenc-nemeth
--------------------------------

Width of the maze (must be odd number): 9
Height of the maze (must be odd number): 9

Aldous-Broder ------------ 1
Binary tree -------------- 2
Kruskal's ---------------- 3
Prim's ------------------- 4
Recursive backtracking --- 5
Recursive division ------- 6
Select an algorithm: 4

Maze generated and saved!
```

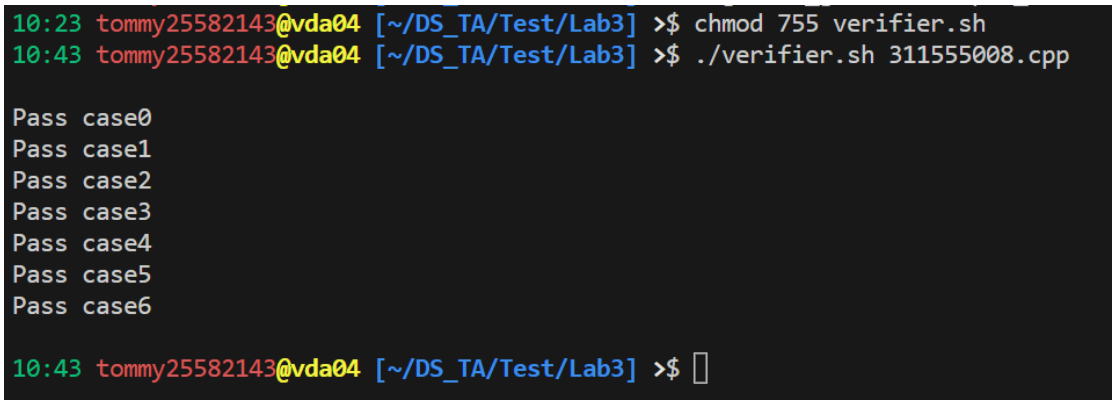
File number must be **different** and **consecutive** numbers, or else the original file would be overwritten, and the verifier would crash.

2. Generate golden answer (**golden6.txt**) by your own.
   (You can compare the golden you generated with your classmates)

3. Modify **verifier.sh**, increase the upper bound of i .



```
$ verifier.sh
Lab3 > $ verifier.sh
#! /bin/bash
test -e ${1} && (g++ ${1} -o main 2> /dev/null || echo Compile Error) || echo No cpp
test -d ./result || mkdir result
echo

if test -f main; then
  i=0
  while [ ${i} != 7 ]  # Set max i to your total data number
  do
    if timeout 2s ./main < ./open_case/case${i}.txt > ./result/res${i}.txt; then
      if diff -b -B ./result/res${i}.txt ./open_case/golden${i}.txt; then
        echo Pass case${i}
      else
        echo Failed case${i} \(WA\)
      fi
    else
      echo Failed case${i} \(TLE or RE\)
    fi
    i=$(($i+1))
  done
  echo
fi
```



```
10:23 tommy25582143@vda04 [~/DS_TA/Test/Lab3] >$ chmod 755 verifier.sh
10:43 tommy25582143@vda04 [~/DS_TA/Test/Lab3] >$ ./verifier.sh 311555008.cpp

Pass case0
Pass case1
Pass case2
Pass case3
Pass case4
Pass case5
Pass case6

10:43 tommy25582143@vda04 [~/DS_TA/Test/Lab3] >$
```

9. Hint

Algorithm: Maze Solving Algorithm with Stack

Input: maze **M**, start_pnt **s**, destination **d**, current_pnt **c**

```
1: Initialize an empty stack S
2: Initialize c ← s
3: S.push(s);
4: while c != d    do
5:     if S.empty() then
6:         return path not found;
7:     end if
8:
9:     for c's all connected points n do
10:        if exists n that can be visited then
11:            S.push(n);
12:            break;
13:        end if
14:        else
15:            S.pop()
16:        end else
17:    end for
18:    c ← S.top()
15: end while
16: return path found;
```

- the idea is quite simple – keep going until you hit a dead end, then go back to the previous intersection point and try a different direction, repeat these steps until you finally get to the destination.
- Remember to mark the points when you visit them.
  So you wouldn't visit the same point twice.

- **You have to use stack during the implementation.**
  If you directly solve the problem by using ex. Recursion,
  you would get 0 score !!!

Contact

王淞平 tommy25582143.cs11@nycu.edu.tw