CS 302 – Assignment #4

Purpose: Learn to about stacks and queues using linked lists
Due: Tuesday (2/14)
Points: 125 Part A → 75 pts, Part B → 50 pts

Assignment

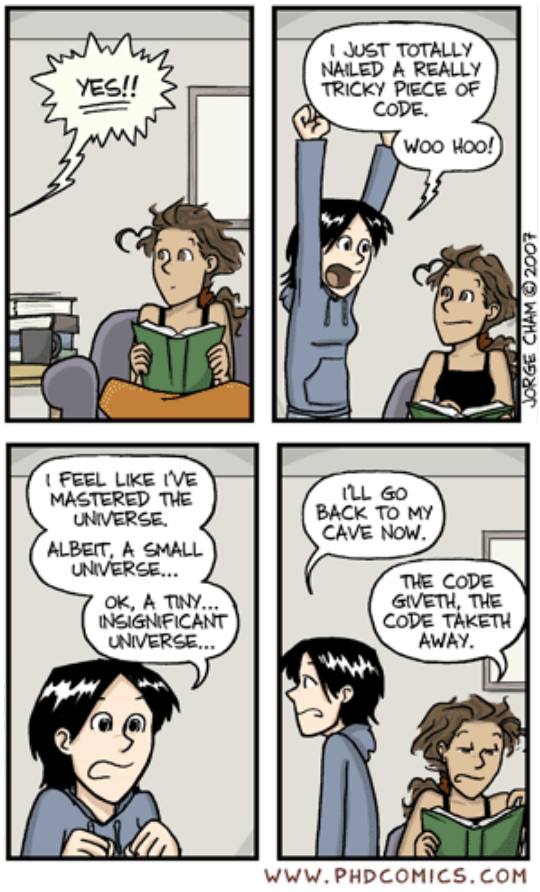
Part A:

A maze[1] is a path or collection of paths, typically from an entrance to a goal. For this assignment we will use a depth first technique to create a two-dimensional maze. Implement a series of C++ classes to create maze generator program using a stack as follows:

The classes are as follows:

- ***linkedStack*** to implement the stack using a linked list
- ***mazeGenerator*** to implement the stack-based maze generation algorithm.

The final output will be a list of maze walls that will be input to a utility program to provide an image of the maze. A main will be provided that can be used to test the classes.

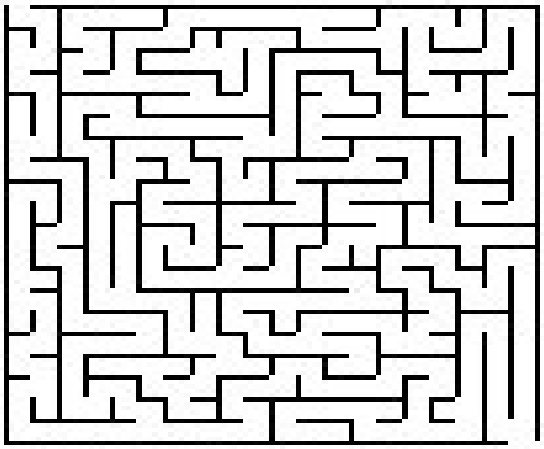


Part B:

When completed, create and submit a write-up (PDF format) not to exceed ~500 words including the following:

- Name, Assignment, Section
- Summary of the implemented linked stack data structures.
- Big-O for each of the stack operations (push, front, pop).
- Comparison of stack data structure to the queue data structure.

It should be noted that there are many implementation variations on these data structures and algorithms. These are the data structures and algorithms that must be implemented. Copying code from the net will result in a zero for the assignment and referral to the Office of Student Conduct.



Example Maze (JPG Format)

1 For more information, refer to: https://en.wikipedia.org/wiki/Maze

Submission:

- Part A → Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.
- Part B → A copy of the write-up including the chart. Must use PDF format.

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

Maze Generation Algorithm

The following algorithm will implement a depth first approach for maze generation. For this assignment, the start cell will always be upper left and the end cell will always be lower right.

- Create an empty stack
- Initialize stack with the start cell (0)
- Loop until the stack is empty
  - pop current cell off the stack and mark it as visited
  - If current index has any unvisited neighbors (order → up, down, left, right)
    - choose a random unvisited neighbor cell
      - directionChoice = rand()%unvisitedNeighborsCount
    - remove the wall between the current cell and the new cell
    - push the current cell on the stack
    - push the new neighbor cell on the stack

Refer to the following sections for a more detailed explanation of the applicable data structures.

Maze Data Structures

There are many ways in which to represent a two-dimensional maze. This assignment will simultaneously use two different ways.

A standard two-dimensional array is used to track the visited and unvisited cells.

In addition, as shown below, a 5x5 maze can be logically represented as a basic grid, with each cell numbered.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

A maze ***wall*** can be represented as the boundary between two cells in the grid. For example, **`WALL 5 6`** is highlighted in the grid. It should be obvious that a wall can only exist between adjacent cells. As such, **`WALL 11 17`** would not be valid for this 5x5 maze. We assume all external walls are solid, excluding the start and exit walls.

The final output of the program will be a list of walls which will be input to a utility program that will create an image file as output. To represent the walls, a simple two-dimension array will be used. The array will be dynamically allocated to the exact number of possible interior walls for the given size. The number of interior walls and resulting wall indexes will be based on the maze size (rows and columns). For the 5x5 maze, there are exactly 40 interior walls. The walls for the 5x5 maze are listed before for reference. *Note*, this list includes all possible walls, which is the initial condition.

first wall	second wall
0	1
1	2
2	3
3	4
5	6
6	7
7	8
8	9
10	11
11	12
12	13
13	14
15	16
16	17
17	18
18	19
20	21
21	22
22	23
23	24

first wall	second wall
0	5
1	6
2	7
3	8
4	9
5	10
6	11
7	12
8	13
9	14
10	15
11	16
12	17
13	18
14	19
15	20
16	21
17	22
18	23
19	24

Walls that are removed, can be marked with invalid values (such as -1).

Maze Data File Conversion

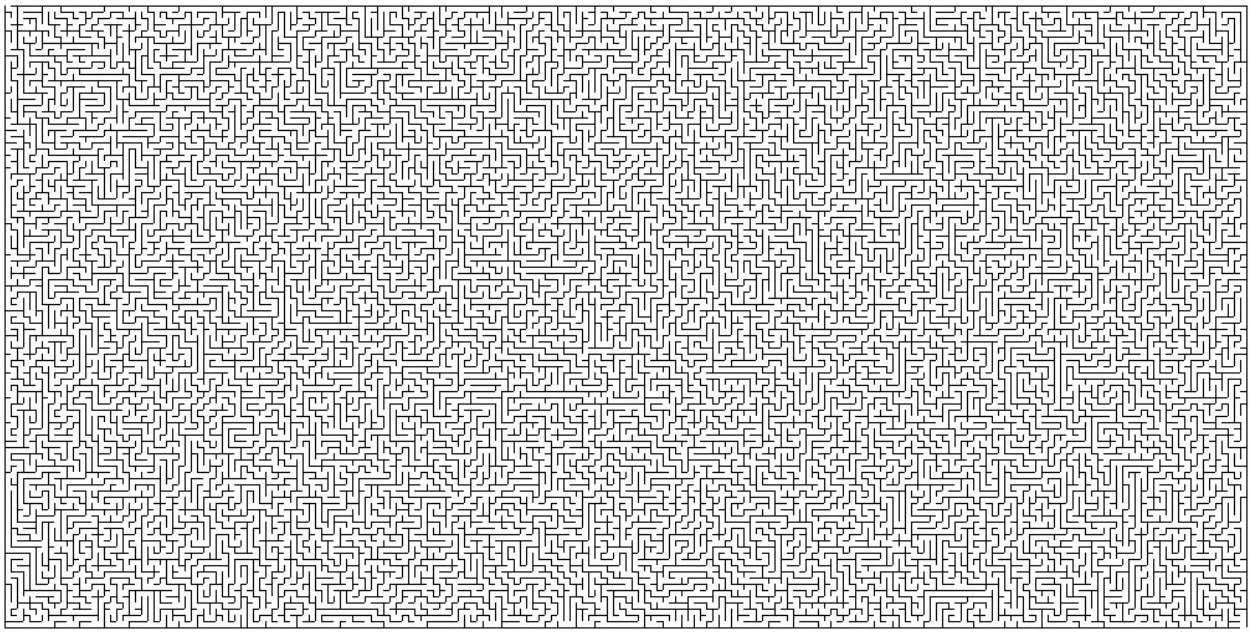
A provided program, *make_ppm*, will read the maze data file and convert to a image (PPM) format. The *ImageMagick* program suite utility program, *convert*, can convert the PPM file into a JPG or GIF image. First, install the *imageMagick* suite via the software center.

The provided program and the ImageMagick *convert* utility can be used as follows:

```
ed-vm% ./makeMaze -h 100 -w 200 -f mazeFile3.txt
ed-vm% cat mazeFile3.txt | ./maze_ppm 5 | convert - maze3.jpg
```

Which will read the **mazeFile3.txt** file generated by your program, pipe it to the maze conversion program (which output PPM data, and then pipe that to the *imageMagick* conversion utility which outputs in JPG format.

For example, the JPG file from the previous commands would be as follows:



Class Descriptions

- Linked List Stack Class
  The linked stack class will implement a stack with a linked list including the specified functions. We will use the following node structure definition.

```
template <class myType>
struct nodeType {
      myType item;
      nodeType<myType> *link;
};
```

linkedStack<myType>
-nodeType<myType> *stackTop
-itemCount: int
+linkedStack()
+~linkedStack()
+isEmpty() const: bool
+getStackCount(): int
+push(const myType &newItem): void
+pop(): myType

Function Descriptions

- The *linkedStack()* constructor should initialize the stack to an empty state (*stackTop*=NULL, etc.).
- The *~linkedStack()* destructor should delete the stack (including releasing the allocated memory).
- The *isEmptyStack()* function should determine whether the stack is empty, returning *true* if the stack is empty and *false* if not.
- The *getStackCount()* function should return the count of elements on the stack.
- The *push(const Type& newItem)* function will add the passed item to the top of the stack.
- The *pop()* function will remove the top item from the stack and return return it. If the stack is empty, nothing should happen and it should return 0.

- Maze Generator Class
  The maze generator class will implement the maze generator algorithm including the functions:

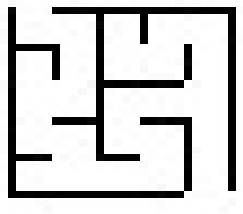
mazeGenerator
-height: int
-width: int
-maze: **char
-walls: **int
-MIN_SIZE=5: static const int
-MIN_SIZE=1000: static const int
+mazeGenerator(int, int)
+~mazeGenerator()
+createMaze() const: void
+printWalls(const string): bool
+printMaze() const: void

Function Descriptions

- The *mazeGenerator(int,int)* constructor create the data structures for a maze based on the passed height and width (in that order). First, the passed height and width must be verified as between between the MIN_SIZE and MAX_SIZE (inclusive). If either is not valid, all class variables should be initialized to 0 or NULL as appropriate. If the parameters are valid, class variables should be set and the maze (height * width) and walls array ((width-1)*height + (height-1)*width) should be created and initialized as appropriate.
- The *~mazeGenerator()* should delete the dynamically allocated data structures and reset all class variables (0 or NULL as appropriate).
- The *createMaze()* function should generate a maze based on the provided algorithm updating the data structures as required.

- The *printWalls()* function should print the maze wall data to the passed file name. The file must start with the size (ROWS ***r*** COLS ***c***) and then the data about which walls are to be included (e.g., WALL 1 2, etc.). If the file can be successfully created, the function should return true, and false otherwise. For example, for the 5 by 5 test maze, the file would contain the following:

```
ROWS 5 COLS 5
WALL 1 2
WALL 2 3
WALL 5 6
WALL 6 7
WALL 8 9
WALL 11 12
WALL 16 17
WALL 18 19
WALL 23 24
WALL 0 5
WALL 7 12
WALL 8 13
WALL 11 16
WALL 13 18
WALL 15 20
WALL 17 22
```



  If there is a problem opening or writing to the file, the function should return false. Otherwise, the function should return true. *Note*, the maze image file generated from this data (but not this function) is shown on the right for reference.
- The *printMazeText()* function should print maze in textual format. For example, for the 5 by 15 maze, the output would look like the following:

```
Generated Maze:

+  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|     |           |              |           |
+--+  +  +--+--+--+  +  +--+--+  +  +--+--+  +
|  |  |              |  |  |     |  |     |  |
+  +  +--+--+--+--+--+  +  +  +--+--+  +  +  +
|     |        |        |  |  |        |     |
+  +--+  +--+  +  +--+--+  +  +  +--+--+--+  +
|     |     |  |  |        |  |     |        |
+--+  +--+  +  +  +  +--+  +  +--+  +  +--+--+
|           |     |     |           |        |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  +
```

  The display of the text version of the maze is selected based on a command line option.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

Example Execution – Testing Program:

Below are example program executions.

```
ed-vm%
ed-vm% ./testDS
-----------------------------------------------------------------
CS 302 - Assignment #4
Basic Testing for Linked Stack Data Structure

*****************************************************************
Test Stack Operations - Reversing:

Original List:     2 4 6 8 10 12 14 16 18 20
Reverse List:      20 18 16 14 12 10 8 6 4 2
Copy A (original): 2 4 6 8 10 12 14 16 18 20

-----------------------------------------------------------------
Test Stack Operations - Doubles:

Original List:     1.1 3.3 5.5 7.7 9.9 11.11 13.13 15.15
Reverse List:      15.15 13.13 11.11 9.9 7.7 5.5 3.3 1.1
Copy A (original): 1.1 3.3 5.5 7.7 9.9 11.11 13.13 15.15

-----------------------------------------------------------------
Test Stack Operations - Multiple Links Test:

Stack Element Count: 303
Stack Link Count: 4

Multiple items, test passed.

-----------------------------------------------------------------
Test Stack Operations - Many Items:

Stack Element Count: 400000
Stack Link Count: 4000

Many items, test passed.

-----------------------------------------------------------------
Game Over, thank you for playing.
ed-vm%

ed-vm%
ed-vm%  ./makeMaze -h 5 -w 15 -f mazeFile.txt -p
***********************************************************
CS 302 - Assignment #4
Maze Generator.


Generated Maze:

+  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|     |           |              |           |
+--+  +  +--+--+--+  +  +--+--+  +  +--+--+  +
|  |  |              |  |  |     |  |     |  |  |
+  +  +--+--+--+--+--+  +  +  +--+--+  +  +  +
|     |        |        |  |  |        |     |
+  +--+  +--+  +  +--+--+  +  +  +--+--+--+  +
|     |     |  |  |        |  |     |        |
+--+  +--+  +  +  +  +--+  +  +--+  +  +--+--+
|           |     |     |           |        |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  +

Game over, thanks for playing.
ed-vm%
ed-vm%
```