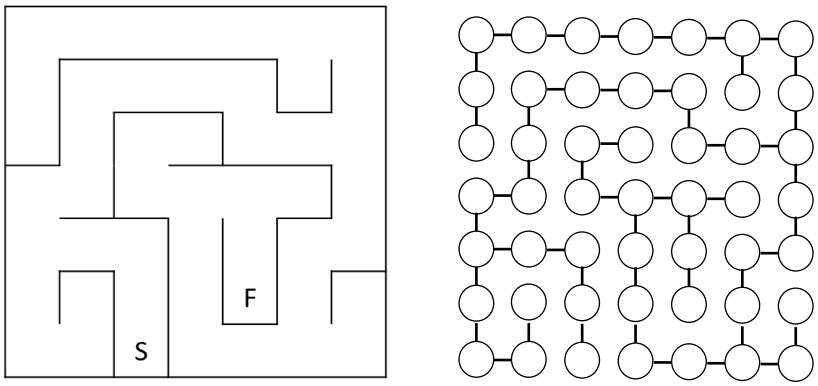
CX 4010 / CSE 6010
Assignment 5
Total Amazement

Due Date: 11:59pm on Thursday, October 15
Submit a single zipfile as described herein to Canvas

In this assignment, you will work in small groups to develop a program to generate and then solve a maze.

A maze can be represented as a connected graph on a grid, where the graph nodes are arranged in rows and columns and edges can only exist between up-down and left-right neighbors. Here is an example of a graph and the corresponding maze, where S designates the start of the maze and F the finish.



A wall in the maze corresponds to the lack of an edge; similarly, the lack of a wall in the maze corresponds to an edge in the graph.

In this assignment you should represent your graph as an adjacency list.

Maze generation

The maze can be generated in different ways. For this assignment, you will use a randomized depth-first search.

- Choose a random position to start; mark it as visited.
- Create a link to a randomly selected unvisited neighbor node; mark it as visited and store its predecessor (in this case, the start node).
- Continuing from the newly selected node, create a link to a randomly selected unvisited neighbor node; mark it as visited and store its predecessor.
- Continue until you reach a node that does not have a neighbor that is an unvisited node.
- When that happens, backtrack (using the stored predecessor information) until you reach a node that has an unvisited neighbor. Continue from that node by choosing a new unvisited neighbor and creating a new link to it.
- Repeat until all nodes have been visited. The last node visited will be the finish node.

The maze generation code should take three command-line arguments: two positive integer arguments representing the number of rows in the maze grid and the number of columns in the maze grid followed by a string argument representing the name of the text file to which the maze information should be written. Once the maze has been created, it should be written out to a file with the name as given in the command-line argument. The format of the file should be as follows.

```
nrows ncols
start finish
<list of indices of nodes in adjacency list of node 0>
<list of indices of nodes in adjacency list of node 1>
…
<list of indices of nodes in adjacency list of node nrows*ncols-1)>
```

Here the nodes should be indexed using a standard conversion from 2d indexing to 1d indexing as

$$index1d = index_{row} * (ncols) + index_{col},$$

where $ncols$ is the number of columns and the indices for the rows and columns run from 0 to $nrows - 1$ and from 0 to $ncols - 1$, respectively. Overall, $index1d$ will take on values from 0 to $nrows * ncols - 1$.

As an example, consider a 2×2 maze grid where the maze begins in the upper left and finishes in the lower left (forming a backward "C" shape). Then the maze file would be written as follows.

```
2 2
0 2
1
0 3
3
1 2
```

It is expected that you should be able to generate a maze using a grid (graph) with up to 100 squares (nodes). This could include something like a symmetric 100×100 grid or an asymmetric 9×12 grid, as examples.

Maze solution

A generated maze can be solved using a number of approaches. For this assignment, you will use breadth first search.

- Start from the designated start position.
- Perform breadth-first search as usual by adding all nodes connected to the start node to a queue. Store the predecessor node (the start node) as well when adding a node to the queue.
- Process the next node in the queue (in first-in, first-out fashion) by adding all its neighbors to the queue along with the current node as the predecessor.
- Eventually, the algorithm will encounter the finish node, and at this point the search can stop.
- Use the predecessor nodes to construct a path from the start to the finish. This will be a sequence of vertices where every consecutive pair of vertices forms an edge in the graph.

The maze generation code should take a single command-line string argument that is the name of the text file from which the maze information should be read using the format described above. After the solution has been found using breadth-first search, you should print to the screen the sequence of vertices (as identified by their 1d indices) forming the path from the start to the finish.

It is expected that you should be able to solve a maze generated using a grid (graph) with up to 100 squares (nodes). This could include something like a symmetric 100×100 grid or an asymmetric 9×12 grid.

Division of labor

Most groups will have two members. One should write the maze generation code and the other should write the maze solution code. You most likely will work together on how to represent the graph within your program and it is ok for you to use the same support code (e.g., how to represent a graph node), but the search algorithms should be written separately, each by one member of the group. If you are assigned to a group of three, two of you should develop separate implementations for one of the tasks (your choice of maze generation or maze solution) such that each can be combined with the work of the third group member. Overall, each team of three then will have two implementations to solve the problem, each with one portion developed by a single student and combined with the other portion developed by one of the other two students.

Group assignments will be available in Canvas.

Submission information

You should submit to Canvas **a single zipfile** that is named according to the Georgia Tech login of someone in your group—the part that precedes @gatech.edu in your GT email address. To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

The zipfile should include the following files:

(1) your code (all .c and .h files), with the main code that handles arguments, function calls, error handling, etc. named **main_generate.c** and **main_solve.c** for the maze generation and solution codes, respectively, and any helper functions in files named **generate.h/generate.c** and **solve.h/solve.c**. If you are using linux or Mac OS, we recommend you use a **makefile** to compile and run your program, and you should include it if so.

(2) a **README** text file (not formatted in a word processor, for example) that includes the compiler and operating system you used for compiling and running your code (both generation and solution) along with instructions on how to compile and run your program. If you are in a group of three, you should include instructions for both complete implementations in the same README.

(3) a series of slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named **slides.pptx** or **slides.pdf**, in the following order:

- 1 slide: your names and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document

but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any structs you may have used, the purpose of any functions you created, etc.).

- 1 slide: a brief explanation of your approach for generating a maze, how well you think your approach works, and any ideas you may have for future improvements.
- 1 slide: a brief explanation of your approach for solving a maze, how well you think your approach works, and any ideas you may have for future improvements.
- 1 slide: evidence of correct operation of your code (e.g., perhaps for a small, but not trivially small, maze).
- 1 slide: a description of the division of labor across the team for this assignment (who did what) with specific references to the code.