Programming assignment 1

Due: Feb 25, 2:59 pm EST

Submit through canvas

In this programming assignment, you are asked to find a path out of a given maze. Bonus credits: generate a random maze.

A maze of size m-by-n is represented by a 2-d array of cells. Each cell has at most 4 walls in its north (N), south (S), west (W) and east (E). The (0, 0) cell is always the entrance and the (m-1, n-1) cell is always the exit. The boundary of the maze has all walls. At each cell in the maze, you can only move to its neighboring cells on its N, S, W or E when there are no walls in between (you cannot visit the cells on its NE, SE, SW or NW). A path from the entrance to the exit is a sequence of cells in the maze such that the first cell is the entrance, the last cell is the exit, and there are no walls between the consecutive cells along the path.

You are asked to

1. Implement a **cell** class in C++. The class cell should have a member variable called **walls**, which is a binary array of size 4 indicating whether there are walls to the N, S, W and E of the cell. The class cell should also have other functions to initialize a cell with 4 walls, break a wall of a cell, etc. Submit a **cell.h** file (10 points) with the implementation.
2. Implement a **maze** class in C++, in which a maze of size m-by-n is implemented using a 2-d array of **cells**. Submit a **maze.h** file with the implementation.
    a. The maze class should have a member function **readMaze (20 point)** which can read in from an input file a maze of a certain size and save the maze in the 2-d array. Such an input file will have the following format

        2 3

        0 0 1 0 1 1

        0 1 1 1 1 1

        0 2 1 0 1 0

        ….

        That is, the first line has 2 numbers (m and n) indicating the size of the maze. In the above case, the maze will have m = 2 rows and n = 3 columns. There will be another m-times-n lines. Each line is for a cell in the maze and has the format

        i j N S W E

        that is, the first two number are the position (i,j) of the cell, the next 4 numbers indicate whether the cell has a wall on its N, S, W and E, respectively (1 means there is a wall and 0 means there is no wall).

    b. (bonus credits) The maze class should have a member function **generateRandomMaze** that generates a random maze of a certain size. This function should provide **a user interface** (2 points) so that the user of your program can input the desired size of a maze. The function should then generate a random maze of the provided size **using a stack** that

has a unique valid path from the entrance to the exit (30 points). The pseudo-code is as follows:

```
generateRandomMaze()
        (m, n) = get the size of the maze from the user
        initiate an empty maze of size m by n, where all the cells have 4 walls and all the
cells are unvisited
        initiate an empty stack
        mark maze entrance as visited
        push entrance onto the stack
        while stack is not empty do
                pop current cell off stack
                if current cell has unvisited neighbors then
                        choose a random unvisited neighbor
                        remove walls between the cell and its neighbor
                        mark the neighbor as visited
                        push the current cell back on stack
                        push the neighbor on stack
                end
        end
        print out the generated maze into a file
```

Note that when you remove walls between a cell and its neighbor, you need to remove 2 walls, one of the cell and one of its neighbor. There is a function like random() in C++ which generates random numbers.

The maze class should have a member function named **printMaze(m, n, output_file_name) (5 points)** and the generateMaze function should call this function to output the generated maze into an output file named maze_m_n.txt where m and n are the size of the maze (e.g., maze_5_2.txt) and the output file should follow the format as the input file as described above.

c. The maze class should have a member function called **DFS (30 points)** which finds a path from the maze. You are asked to implement this function using a stack (you will not get any points if you do it using a recursive function). The basic idea is you always keep the previous path up to the current cell in a stack. Once you find nowhere to go from the current cell (e.g., either all the neighboring cells have been visited already or there are walls between the cell and its neighbors), you trace back to the previous cell on the current path and restart from there. You may want to take a look at the pseudo-code of generateRandomMaze to get the idea of using a stack.
   You need to implement the stack and its operations (**pop, push, etc**) in a **stack.h** file (20 points). Submit the stack.h file.

d. The maze class should have a member variable called **path** which should be implemented from **a class of linkedList.** Implement the linkedList class and submit a linkedList.h file which includes the construction of a linked list, its traversals and deconstruction (20 points). The output of the DFS function should a path saved in a linkedList.

e. The maze class should have a member function called **printPath (5 points)** which prints out a path found by DFS into an output file name path.txt. This function should call a member function from the class linkedList.

3. Your main.cpp should implement the following:

Read in a maze
Find a path from the maze
Print the path into an output file
Destroy the maze

(bonus credits)
Generate a random maze
Print the maze into an output file
Destroy the maze

4. What to submit
    a. Source code in a fold called mazeCode. You need to submit all the necessary codes, including the required files/classes and anything else (e.g., external libraries, other classes) that is required to correctly compile or run your code.
    b. A README file describe how you implement DFS and how to compile your code. If you do bonus credits, please indicate that in the README file. Your code should be developed on Linux and compile using g++. Please give a comment line you used to compile the code. For example, if you compile your code using
    g++ you_code
    then in the README file, please explicitly state this command. Please provide all necessary information in the README file. It is expected that the TA/graders can successfully compile and run your code just based on the README file.
5. Grading:
    a. Please strictly follow the naming scheme and use the names specified as in this description for the functions and files. If you are not using these names, 20 points will be taken off (non-negotiable!)
    b. Your code has to be able to compile on machine pegasus.cs.iupui.edu. If the compilation is not successful by copy-paste the compilation comment provided in your README file, one third of the total points will be taken off (non-negotiable!)