Problem 7 - By Hugo Barahona

Solve a Maze

```
class Node {
public:
    Node* right;
    Node* left;
    Node* up;
    Node* down;
    string name;


    Node(){
        right = left = up = down = nullptr;
        name = "end";
    }

    void path(int r, int l, int u, int d, string n){
        name = n;
        if(r == 1){
            right = new Node;
            right->left = this;
        }
        if(l == 1){
            left = new Node;
            left->right = this;
        }
        if(u == 1){
            up = new Node;
            up->down = this;
        }
        if(d == 1){
            down = new Node;
            down->up = this;
        }
    }
```

To represent a maze, I decided to do it as a linked list structure, I could have done it in different ways, one of the ways that probably the best approach was an adjacency matrix, but I was curious to do it in this way so that's what I chose.

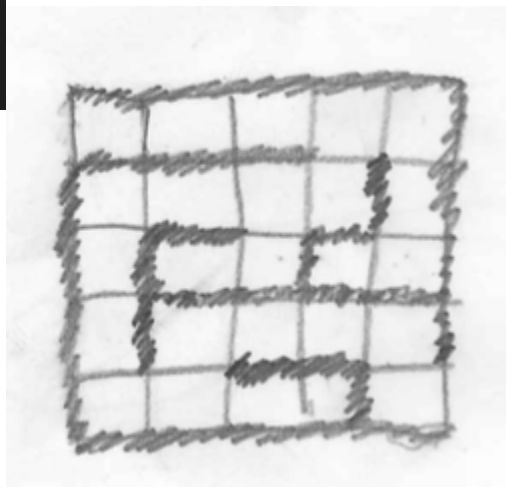
Each node has 4 pointers: right, left, up and down. And also each node has 3 types of names:

Path: means it's just an inner block of the maze.

End: means it's a deadend.

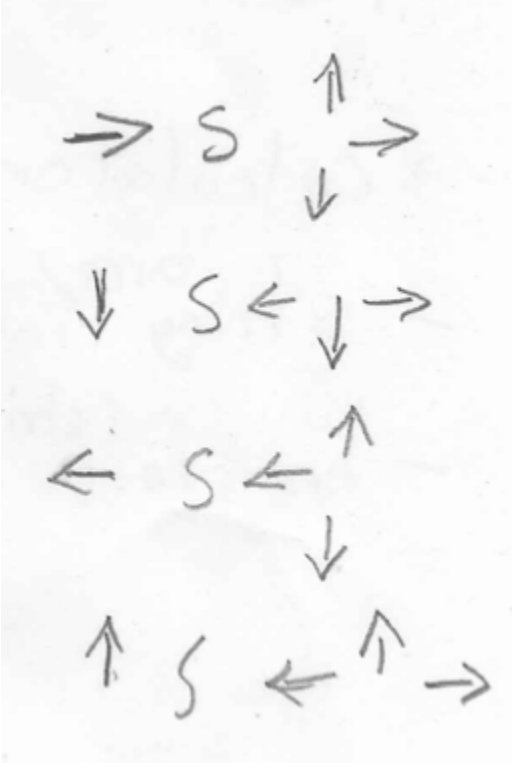
Exit: means that node is the exit.

In main, I start connecting all the nodes. Initially I wanted to create a function that would generate a square maze with no loops using nodes in this kind of linked list structure, but due to time constraints, I decided to create the maze manually, and it represents the same maze as seen to the right of this paragraph.



```
void solveMaze(Node& myMaze){
    bool exitFound = 0;
    Node* trav = myMaze.right; //node pointer to help traverse the maze.
    stack<string> stkDirection; // stack to record the direction taken within the maze.
    string comingFrom = "left";// name of the direction the node trav comes from.
    bool noPathFound = 0;// boolean to determine whether I reach a deadend.
```

I created a void function to solve the maze, that takes the reference of a Node. The main variables used are a boolean to determine whether an exit was found, a Node to traverse, only one stack that stores the direction I've taken each step(However, I believe it can be done with two stacks in an effective way.), a string that tells me which direction I come from and a boolean that tells me if a path was found in the current node (aside from the path I come from). I will go more deep in the explanation of the code, but I will share a few pictures to explain some of the thought process it went into it.



In the left we can see a set of rules I made, the single arrows of the left represent the direction I come from whenever I enter a new node, and the group of arrows on the right are the directions I would have to check. That way I would never check the direction I come from.

Bellow, we have a graphic representation of the maze, and how I saw it, the last node is the one that has the variable name set to "exit".