/\*

Author: Stewart Moon

Class: CSC 223-100

Professor: Jeffrey Howard

Due Date: 12/7/2017

Page & Problem #: Moduel 13

Description: This program will read from a text file that contains an maze. It will store it into an array and naviage the maze.

\*/

#include <iostream>

#include <fstream>

using namespace std;

class maze {

public:

char runMaze();

void fillArray();

void printArray();

//Deconstructor

~maze();

private:

char CheckUp();

char CheckDown();

char CheckLeft();

char CheckRight();

void NavigateUp();

void NavigateDown();

void NavigateRight();

void NavigateLeft();

char\*\* myArray;

int s1, s2, c1, c2;

};

//Pre-condition: No parameters are passed

//Post-condiotion: Variable of type char is returned

//Description: This Method controls what functions to call.

int main()

{

maze maze;

maze.fillArray();

maze.runMaze();

return 0;

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: When function is called ir prints out the Array of the Maze

void maze::printArray()

{

//Print Array

for (int i = 0; i<s1; i++)

{

for (int j = 0; j<s2; j++)

{

cout << myArray[i][j];

}

cout << endl;

}

}

//Pre-condition: No parameters are passed

//Post-condiotion: No parameters are returned

//Description: This Method will read from a text file and fill the 2D array with the infromation from the text

void maze::fillArray()

{

ifstream inStream;

inStream.open("Maze.txt");

//Check to verify file is open.

if (inStream.fail()) {

cout << "File could not be opened" << endl;

exit(1);

}

//Creating the sizes for the dynamic array (s for rows, c for columns)

inStream >> s1 >> s2 >> c1 >> c2;

//Create Dynmaic Array

myArray = new char\*[s1];

for (int i = 0; i != s1; i++)

{

myArray[i] = new char[s2];

}

//Fill Array

for (int i = 0; i < s1; i++)

for (int j = 0; j < s2; j++)

{

inStream >> myArray[i][j];

}

inStream.close();

}

//Pre-condition: No parameters are passed

//Post-condiotion: Variable of type char is returned

//Description: This Method controls how the Player Navigates the Maze

char maze::runMaze()

{

//Logic to Check for E

if (CheckRight() == 'E')

{

myArray[c1][c2] = ' ';

cout << "End!" << endl;

printArray();

return 'E';

}

if (CheckLeft() == 'E')

{

myArray[c1][c2] = ' ';

printArray();

return 'E';

}

if (CheckDown() == 'E')

{

myArray[c1][c2] = ' ';

printArray();

return 'E';

}

if (CheckUp() == 'E')

{

myArray[c1][c2] = ' ';

printArray();

return 'E';

}

//Logic to Check for O

if (CheckRight() == 'O')

{

myArray[c1][c2] = ' ';

NavigateRight();

runMaze();

}

if (CheckLeft() == 'O')

{

myArray[c1][c2] = ' ';

NavigateLeft();

runMaze();

}

if (CheckDown() == 'O')

{

myArray[c1][c2] = ' ';

NavigateDown();

runMaze();

}

if (CheckUp() == 'O')

{

myArray[c1][c2] = ' ';

NavigateUp();

runMaze();

}

//Logic to Check for ' '

if (CheckRight() == ' ')

{

myArray[c1][c2] = 'X';

NavigateRight();

runMaze();

}

if (CheckLeft() == ' ')

{

myArray[c1][c2] = 'X';

NavigateLeft();

runMaze();

}

if (CheckDown() == ' ')

{

myArray[c1][c2] = 'X';

NavigateDown();

runMaze();

}

if (CheckUp() == ' ')

{

myArray[c1][c2] = 'X';

NavigateUp();

runMaze();

}

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Moves the cursor right

void maze::NavigateRight()

{

c2 = c2 + 1;

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Moves the cursor Navigateleft

void maze::NavigateLeft()

{

c2 = c2 - 1;

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Moves the cursor CheckUp

void maze::NavigateUp()

{

c1 = c1 - 1;

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Moves the cursor CheckDown

void maze::NavigateDown()

{

c1 = c1 + 1;

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Checks what is above the cursor

char maze::CheckUp()

{

return myArray[c1 - 1][c2];

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Checks what is below the cursor

char maze::CheckDown()

{

return myArray[c1 + 1][c2];

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Checks what is CheckLeft to the cursor

char maze::CheckLeft()

{

return myArray[c1][c2 - 1];

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Checks what is right to the cursor

char maze::CheckRight()

{

return myArray[c1][c2 + 1];

}

//Pre-condition: No parameters are passed

//Post-condiotion: No variables are returned

//Description: Destructor for maze

maze::~maze()

{

delete myArray;

myArray = NULL;

}