|  |
| --- |
| #include <fstream> |
|  | #include <cstring> |
|  | #include <iostream> |
|  | #include <string> |
|  | #include <algorithm> |
|  | #include <vector> |
|  |  |
|  | using namespace std; |
|  |  |
|  | class Cell { |
|  | public: |
|  | // Default constructor that creates a blank cell (3x3 grid of '.') |
|  | Cell() { |
|  | for (int i = 0; i < 3; i++) { |
|  | for (int j = 0; j < 3; j++) { |
|  | content[i][j] = '.'; |
|  | } |
|  | } |
|  | isAWall, isStart, isFinish, isBarrier, hasBeenVisited = false; |
|  | cellLocX, cellLocY = 0; |
|  | } |
|  |  |
|  | // overloaded constructor that accepts a char to add to cell ID (index 1,1) |
|  | Cell(char cellType) { |
|  | for (int i = 0; i < 3; i++) { |
|  | for (int j = 0; j < 3; j++) { |
|  | // if a wall, 'X' and flag, otherwise '.' |
|  | if (cellType == 'X' || cellType == 'x') { content[i][j] = 'X'; isAWall = true; } |
|  | else { content[i][j] = '.'; } |
|  | } |
|  | } |
|  | // Set flags based on passed cellType |
|  | if (cellType == 'S' || cellType == 's') { content[1][1] = 'S'; isStart = true; } |
|  | else if (cellType == 'F' || cellType == 'f') { content[1][1] = 'F'; isFinish = true; } |
|  |  |
|  | isBarrier, hasBeenVisited = false; |
|  | cellLocX, cellLocY = 0; |
|  | } |
|  |  |
|  | // Update the celltype after creation(same as overloaded constructor) |
|  | void UpdateCellType(char cellType) { |
|  | for (int i = 0; i < 3; i++) { |
|  | for (int j = 0; j < 3; j++) { |
|  | // if a wall, 'X' and flag, otherwise '.' |
|  | if (cellType == 'X' || cellType == 'x') { content[i][j] = 'X'; isAWall = true; } |
|  | else { content[i][j] = '.'; } |
|  | } |
|  | } |
|  | // Set flags based on passed cellType |
|  | if (cellType == 'S' || cellType == 's') { content[1][1] = 'S'; isStart = true; } |
|  | else if (cellType == 'F' || cellType == 'f') { content[1][1] = 'F'; isFinish = true; } |
|  |  |
|  | isBarrier, hasBeenVisited = false; |
|  | cellLocX, cellLocY = 0; |
|  | } |
|  | // future functionality |
|  | bool UpdateCellHasBeenVisited() { |
|  | hasBeenVisited = true; |
|  | } |
|  | // used in printing and writing to file/ future will put those functions in here |
|  | char ReturnCellValue(int row, int col) { |
|  | return content[row][col]; |
|  | } |
|  |  |
|  | protected: |
|  | char content[3][3]; |
|  | int cellLocX, cellLocY; |
|  | bool isAWall, isStart, isFinish, isBarrier, hasBeenVisited; |
|  | }; |
|  |  |
|  | // Print the array of Cell Objects to STD\_OUT |
|  | // Need to work on passing 2D array by ref |
|  | // So we dont have to define the size of the array here |
|  | void PrintArray(Cell \*\* array, int rowSize, int colSize) { |
|  | // Add numerical header to displayed grid |
|  | cout << " "; |
|  | for (int i = 0; i < colSize; i++) { |
|  | if (i < 10) { |
|  | cout << " " << i << " "; |
|  | } |
|  | else { |
|  | cout << " " << i << " "; |
|  | } |
|  | } |
|  | cout << endl; |
|  | // iterating through the array |
|  | for (int i = 0; i < rowSize; i++) { |
|  | for (int m = 0; m < 3; m++) { |
|  | // Add row count to first column |
|  | if (m == 1) { |
|  | if (i < 10) { cout << i << " "; } |
|  | else { cout << i << " "; } |
|  | } |
|  | else { cout << " "; } |
|  | for (int j = 0; j < colSize; j++) { |
|  | for (int n = 0; n < 3; n++) { |
|  | cout << array[i][j].ReturnCellValue(m, n) << " "; |
|  | } |
|  | } |
|  | cout << endl; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Write the Array to the Output File |
|  | void WriteArray(Cell \*\* array, int rowSize, int colSize) { |
|  |  |
|  | ofstream outFile; |
|  | outFile.open("outputFile"); |
|  |  |
|  | // Add numerical header to displayed grid |
|  | cout << " "; |
|  | for (int i = 0; i < colSize; i++) { |
|  | if (i < 10) { |
|  | outFile << " " << i << " "; |
|  | } |
|  | else { |
|  | outFile << " " << i << " "; |
|  | } |
|  | } |
|  | outFile << endl; |
|  | // iterating through the array |
|  | for (int i = 0; i < rowSize; i++) { |
|  | for (int m = 0; m < 3; m++) { |
|  | // Add row count to first column |
|  | if (m == 1) { |
|  | if (i < 10) { outFile << i << " "; } |
|  | else { outFile << i << " "; } |
|  | } |
|  | else { outFile << " "; } |
|  | for (int j = 0; j < colSize; j++) { |
|  | for (int n = 0; n < 3; n++) { |
|  | outFile << array[i][j].ReturnCellValue(m, n) << " "; |
|  | } |
|  | } |
|  | outFile << endl; |
|  | } |
|  | } |
|  | outFile.close(); |
|  | } |
|  |  |
|  | int main() { |
|  |  |
|  | // Dirty read/parse file (must be all ints) |
|  | // want to move to function and parse no matter the content |
|  | int rowSize, colSize; |
|  | int startRow, startCol; |
|  | int finishRow, finishCol; |
|  | int wallRow, wallCol; |
|  |  |
|  | // Open and read size and start/finish positions from inputfile |
|  | ifstream myFile; |
|  | myFile.open("inputFile"); |
|  | myFile >> colSize >> rowSize; |
|  | myFile >> startCol >> startRow >> finishCol >> finishRow; |
|  |  |
|  | // error check the file input |
|  | if(colSize < 0 || rowSize < 0 || startCol < 0 || |
|  | startCol > colSize || startRow < 0 || startRow > rowSize || |
|  | finishCol < 0 || finishCol > colSize|| finishRow < 0 || finishRow > rowSize ){ |
|  | cout << "There was an error! Check file contents." |
|  | << "One of the input numbers is out of range." |
|  | << endl << "File input checked: Size, Start, and Finish Coordinates." |
|  | << endl; |
|  | return 1; |
|  | } |
|  |  |
|  | // Create the array of objects |
|  | Cell\*\* maze; |
|  | maze = new Cell\*[rowSize]; |
|  | for (int i = 0; i < rowSize; i++) { maze[i] = new Cell[colSize]; } |
|  |  |
|  | // UpdateCells to walls based on inputFile data |
|  | for (int h = 0; h < 79; h++) { |
|  | myFile >> wallCol >> wallRow; |
|  |  |
|  | // error check the file input |
|  | if (wallCol < 0 || wallCol > colSize || wallRow < 0 || wallRow > rowSize){ |
|  | cout << "There was an error! Check file contents." |
|  | << "One of the input numbers is out of range." |
|  | << endl << "File input checked: Wall Cordinates." |
|  | << endl; |
|  | return 1; |
|  | } |
|  | maze[wallRow][wallCol].UpdateCellType('X'); |
|  | } |
|  |  |
|  | // UpdateCells to start/finish |
|  | maze[startCol][startRow].UpdateCellType('S'); |
|  | maze[finishCol][finishRow].UpdateCellType('F'); |
|  |  |
|  | // Close File |
|  | myFile.close(); |
|  |  |
|  | // Print to the Screen |
|  | PrintArray(maze, rowSize, colSize); |
|  |  |
|  | // Write to file |
|  | WriteArray(maze, rowSize, colSize); |
|  |  |
|  | // Press a key |
|  | cout << endl << "Press <Enter> to continue..." << endl; |
|  | cin.get(); |
|  |  |
|  | return 0; |
|  | } |

Input file

|  |  |
| --- | --- |
|  | 10 11 |
|  | 0 0 |
|  | 10 3 |
|  | 0 6 |
|  | 0 7 |
|  | 0 9 |
|  | 1 1 |
|  | 1 2 |
|  | 1 4 |
|  | 1 7 |
|  | 2 1 |
|  | 2 7 |
|  | 2 8 |
|  | 2 10 |
|  | 3 1 |
|  | 3 2 |
|  | 3 3 |
|  | 3 4 |
|  | 3 7 |
|  | 4 0 |
|  | 4 6 |
|  | 4 9 |
|  | 4 10 |
|  | 5 2 |
|  | 5 3 |
|  | 5 4 |
|  | 5 5 |
|  | 5 7 |
|  | 6 1 |
|  | 6 8 |
|  | 6 9 |
|  | 7 1 |
|  | 7 2 |
|  | 7 3 |
|  | 7 6 |
|  | 8 1 |
|  | 8 5 |
|  | 8 6 |
|  | 8 8 |
|  | 8 10 |
|  | 9 3 |
|  | 9 7 |