Assignment 3: Maze
(updated 4/18 to simplify and add hints)

Goals: Understanding recursion, solving mazes, working with multiple ADTs

For this assignment, you will write a program that can solve a maze. The problem description is taken from Carrano Chapter 5 Problem 9 (Carrano Chapter 5, Problem 4 in 6th edition) which is linked to this assignment.

The maze is provided by a text file in the following format:

```
20    7
0     18
xxxxxxxxxxxxxxxxx x
x      x       xxxx x
x xxxxx   xxxxx   xx x
x xxxxx xxxxxxx xx x
x x            xx xx x
x xxxxxxxxxx xx     x
xxxxxxxxxxxxxxxxxxxxx
```

The first 2 numbers are:	`width-of-maze    height-of-maze`
The next 2 numbers are:	`row-exit         column-exit`
`x`	`represents wall`
`space`	`represents movable space`

<u>Unlike the textbook version, the entrance to the Maze is not specified as part of the maze.txt file but will be provided by Creature's location</u>

When maze is printed, you should also add

`*`	`part of the path to exit`
`+`	`visited square not part of the path to exit`

The following public functions need to be implemented:

```cpp
// follows the format provided above printing the maze
ostream &operator<<(ostream &out, const Maze &maze);

// prints current location of creature, for example C(7,3)
ostream &operator<<(ostream &out, const Creature &creature);

bool Maze::IsClear(int row, int column) const;
bool Maze::IsWall(int row, int column) const;
bool Maze::IsPath(int row, int column) const;
bool Maze::IsVisited(int row, int column) const;
```

```
// mark the maze with *
void Maze::MarkAsPath(int row, int column);

// mark the maze with +
void Maze::MarkAsVisited(int row, int column);

// Maze constructor requiring a valid file name
explicit Maze::Maze(string mazeFile);


// returns a string in the form of NNEEN
// (where N means North, E means East, etc)
// that will let the creature get to the exit
// if creature cannot get to the exit, returns "X"
string Creature::Solve(Maze &maze);
```

You may choose to have additional public or private functions as needed. For example, you can pass the path-so-far by reference to a function like `Creature::goNorth` or you can return a path from that

You can assume that mazes will have less than 100 rows and 100 columns.

Under unix, compile your code using

g++ -g -Wall -Wextra assignment3.cpp maze.cpp creature.cpp -o assignment3

You need to submit `assignment3.zip` with the following files in it. See course assignments page for instructions on how to create it

`maze.h` - the prototypes of `Maze` class functions
`maze.cpp` - the implementation of `Maze` class functions
`creature.h` - the prototypes of `Creature` class functions
`creature.cpp` - the implementation of `Creature` class functions
`maze1.txt` - a test maze file
`maze2.txt` - another test maze file
`assignment3.cpp` - tests demonstrating solving different mazes
`output.txt` - See course assignments page for instructions on how to create it
`selfassessment.txt` - See course assignments page for the template

Tips & Hints
- Implement the Maze constructor and the `operator<<` for Maze first. You will need them when debugging your code.
- You can use `ifstream` to read from a text file. The text file has to in the same directory as the executable. In the case of CLion, you will need to put it in cmake-build-debug directory.
- You can use `>>` to read integers, but to read blank spaces you need to use `get` function. To read the invisible end-of-line characters, you can use `getline` and discard what is read.
- Confirm that your maze.txt file has spaces rather than tabs
- You can use `char field[MAX_SIZE][MAX_SIZE]` or `int field[MAX_SIZE][MAX_SIZE]` to store your maze. `char` might be easier since you do not have to convert back and forth between different representations.
- Calling `Creature::goNorth` and passing the path-so-far by reference is one way to keep track of the path. There are other ways depending on how you implement it.
- You should write the algorithm to solve the maze first and then work on adding path string.
- Start with simple mazes. Mazes where creature is already at exit or where creature only has to go North to get to exit before moving to more complex ones.