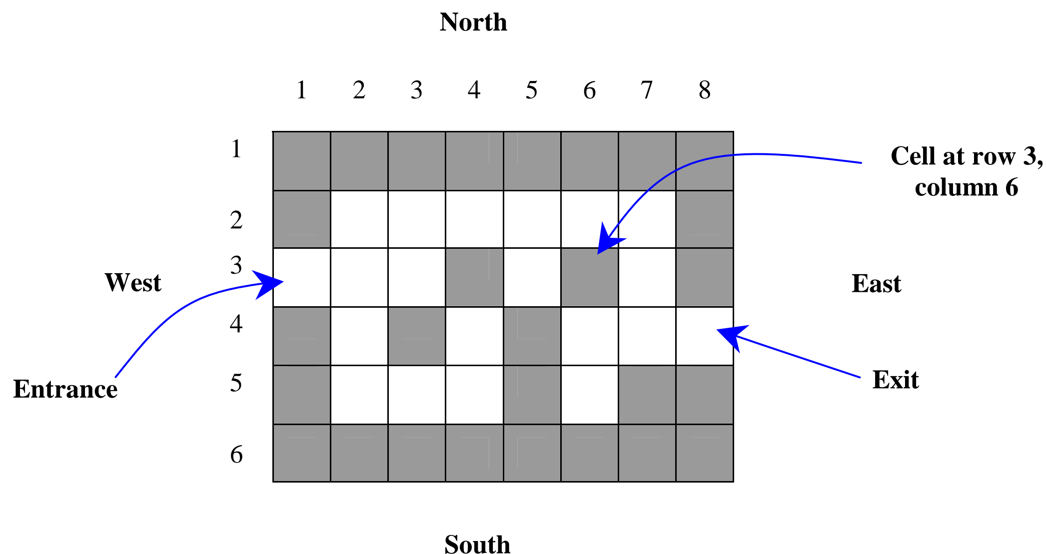
Maze Crawler

For this project, you will be designing and then implementing a prototype for a simple game. The "moves" in the game will be specified by a list of commands given in a text input file. There is no graphical interface, but feedback will be provided in an output file.

Maze: a rectangular grid of cells, each identified by a pair of integers specifying its location (row and column) in the grid. Initially, each grid cell may be empty or be a wall. The locations of walls will be specified in an input file and will not change during the course of a game. Grid locations are labeled as shown below, with the indicated conventions regarding directions:



The maze will always have two cells designated as the Entrance and Exit.

Dramatis personae: two types of creatures, MazeMonsters and Humans, may inhabit the maze. Each creature will always have a particular location (cell) and may move in any of the four cardinal directions (East, North, West, or South) but not diagonally. Neither type of creature can move through a wall. Also, neither type of creature knows whether an adjacent cell is open or a wall cell; it must discover that fact by attempting to move into that cell.

Each creature will also have an energy level, represented by a nonnegative integer. Moving from one cell to an adjacent cell uses one unit of energy. If a creature's energy level drops to zero, it dies (and disappears from the game).

A creature may also attempt to move out of its current cell in a direction that is blocked by a wall. In that case, it remains in its current cell and loses one unit of energy.

A MazeMonster is familiar with the maze and always knows its current location, not that a MazeMonster is smart enough to actually take advantage of that information. A Human never knows his/her current location within the maze.

A Human will not enter a cell containing a MazeMonster. If a MazeMonster enters a cell containing a Human, it immediately eats the Human, increasing its energy level by the amount of energy the Human possessed. If a MazeMonster enters a cell containing another MazeMonster the result depends on their relative energy levels. If both have the same energy level, then both die immediately. If they have different energy levels, then the one with more energy eats the one with less energy; in this case, the winner's energy level remains unchanged.

At any given time, there may be one to three MazeMonsters in the maze and zero or one Human.

Victor: MazeMonsters are created in Victor's laboratory (on command from the input file). Once a MazeMonster is created, it and Victor's laboratory have no further interaction. Humans are not created in Victor's laboratory.

Driver file description:

Your program **must** read its input from a file named `Maze.in` — use of another input file name will annoy the person conducting your project demonstration. The first two lines of the input file specify the dimensions of the maze for the current game. The next two lines specify the Entrance and Exit cells for the maze. Each value will be preceded by a label ending with a colon (`':'`) and a number of spaces (not a tab). These lines will always be logically and syntactically correct. For example:

```
Rows:      6
Columns:   8
Enter:    ( 3, 1)
Exit:     ( 4, 8)
```

Assuming there are R rows and C columns in the maze, the next R lines of the input file will specify which cells are open and which are walls. Each of these rows will contain exactly C characters, followed by a newline. The characters correspond, in order, to the cells in that row of the maze and are to be interpreted as follows:

`W`	cell is a wall
`' '` (space)	cell is open

These lines of input will also always be logically and syntactically correct. For example:

```
WWWWWWWW
W      W
   W W W
W W W   
W   W WW
WWWWWWWW
```

Each remaining line of the input file will specify one of the commands described below. Each line consists of a sequence of "tokens" which will be separated by single tab characters. A newline character will immediately follow the final "token" on each line.

`make` **`monster`** `<name>` `<location>` `<energy>`

If the specified name is not in use, and the location is valid (within the maze boundaries and not a wall) and the energy value is positive, then Victor's laboratory will create a MazeMonster as specified. Otherwise, the command does not alter the state of the game.

`make` **`human`** `<name>` `<energy>`

If a Human is already in the maze, the command does not alter the state of the game. Otherwise, if the specified name is not in use, and the energy value is positive, then a Human will be created in the Entrance cell for the maze. In the unfortunate event that there is a MazeMonster in that cell, the Human is immediately eaten.

`<name>` **`step`** [**`East`** | **`North`** | **`West`** | **`South`**]

If the name is not the name of an existing MazeMonster or Human, the command does not alter the state of the game.

If the name is that of an existing MazeMonster or Human, the named creature will attempt to move to the adjacent cell in the specified direction (the target cell). If the target cell is a wall or lies outside the maze (see special case below), the creature will lose one unit of energy and remain in its current cell. If the cell is empty, the creature will move into that cell and lose one unit of energy.

If the named creature is a Human, and a MazeMonster occupies the target cell, the Human will lose one unit of energy and remain in his/her current cell. If the named creature is a MazeMonster, and the target cell is occupied

by a Human or another MazeMonster, then the creature will move into that cell and the rules stated earlier apply (someone's going to die).

Special case: if the creature is in either an Entrance or Exit cell, and the specified direction would take it out of the maze, then it will make the move and disappear from the game.

`<name>` **`trek`** `[`**`East`** | **`North`** | **`West`** | **`South`**`]`

This is simply an iteration of the `step` command. The named creature will move as far as possible, given the rules for the `step` command, in the specified direction. Warning: be careful with your calculation of the energy consumption.

`<name>` **`see`** **`monster?`**

If the named creature is an existing Human, then the Human will "look" (not move) as far as possible in each direction from its current location, attempting to spot a MazeMonster. (This will evidently require that the Human have some sort of communication with the game entity that keeps track of current locations of creatures and the maze layout.) If the Human spots any MazeMonsters, he/she should produce output signaling that and specifying the direction(s). This command does not use any energy.

This command applies only to Humans, so if the named creature is a MazeMonster, the command should be treated as an error.

`<name>` **`status`**

If the name is that of an existing creature, the name, location and energy level of the named creature will be displayed to standard output (`cout`).

`display status`

For every existing creature, display its name, location and energy level to standard output (`cout`).

`display maze`

Display to standard output (`cout`) a map of the maze. The map should indicate clearly which cells are empty, which cells are walls, which cells contain a Human and which cells contain a MazeMonster. This map should be relatively compact. This is one situation where adding whitespace does not increase clarity.

Legend: in the commands above:

`<name>`	a string, which may contain any character except a newline or tab, and which may not be one of the keywords (in boldface) used in the command descriptions above
`<location>`	a pair of integers formatted as `(r, c)`
`<energy>`	a positive integer

You may assume that the input file will conform to the given syntax. It is certainly possible that commands may name nonexistent creatures, request the creation of an excessive number of Humans and/or MazeMonsters, or specify locations that lie outside the maze boundaries. Your program should be prepared to deal gracefully with any of those situations.

A sample input file is shown on pages 7–8.

Output description and sample:

All output for the program is to be written to standard output (`cout`), but this should be easily reconfigurable. At minimum, all specified output should be produced, and any command that cannot be successfully processed should produce a useful error message. Ideally, each command will produce some output (whether specified above or not). One good approach would be to echo each command that's read from the input file to the output, followed by an error message or the specified output for that command, or by a success message.

Sample output is shown on pages 9–10. The format does not have to be exactly identical, but should be neat, and readable.

Design Homework Problem:

For this project you will produce an interim design, which you will submit as a homework assignment.

You should identify a reasonable set of classes, document each with a class form, and produce a class diagram that shows how you will use association and/or aggregation in the program. There is no specified minimum number of classes; however, it is better to have too many than to have too few. Your design should largely isolate file accesses (input/output) from computation. You should, of course, attempt to identify classes that correspond to useful clear abstractions, provide public interfaces that are both useful and coherent, and achieve a reasonable balance of responsibilities among the classes.

You must submit this design by 5:00 pm on Friday, March 10. Your design will be graded, and a discussion of reasonable design alternatives will be held in class during the week of March 20. Submit your design either as a PDF file or an MS Word document. The class diagram should be represented using the notations presented in the course notes and earlier project specifications. The class forms should follow the template given below.

Name:	
Purpose:	
Constructors:	
Operations: *Mutators*:	
Reporters:	
Data Members:	

Submit your file (not zipped) to the Curator System as P3Design at: http://spasm.cs.vt.edu:8080/curator/

Project Deliverables:

Your final project submission must include the following:

- All source code (`*.cpp` and `*.h` files) comprising your project.
- MS Visual C++ project files (`dsp` and `dsw`) or Unix `makefile`, as appropriate. (**VC++ users:** do not submit unnecessary files, such as `ncb`, `opt`, `ilk`, `obj`, `pch`, or `pdb` files).
- Your updated final design. This should be an MS Word document or PDF file as described in the preceding section.
- One input file and the corresponding output from your testing.
- A brief ASCII text readme file, named `readme.txt`, containing:
  - Your name and e-mail address.
  - Your instructor's name, and the section of the course you're enrolled in (example: 8:00 MWF).
  - The project name.
  - The platform and compiler you're using (example: MSVC++ 6.0 under NT).
  - Any special execution instructions. (If your program is not fully functional, be sure to mention that here and to follow the directions in the Evaluation section above.)
- An ASCII text file, named `pledge.txt`, containing the Honor Code Pledge listed at the end of this file.

Your project submission will consist of a zipped archive file containing all of the items specified above.

You are allowed to submit your solution up to five times, in case you detect (or solve) problems after your first submission. Your last submission will be the only one tested and graded. Note that, due to late penalties, it is possible that a fixed but late submission may receive a lower score than a faulty but on-time submission.

You will submit this zip file to the Curator System (as Project3), and you will demonstrate its operation to the GTA of your section. In order to receive any credit for this assignment, you are required to meet with the GTA, even if your program does not compile and run.

The submission client can be found at: http://spasm.cs.vt.edu:8080/curator/

Evaluation:

Your score on this project may be based on several factors:

- Runtime testing of your program, and correctness and completeness of output.
- Quality of your final design, and conformance of your implementation to the design representation.
- Adherence to good software engineering practice.
- Quality of internal documentation.

The relative weighting of these factors will be decided later, but proper execution will be worth approximately 50% and the quality of internal documentation will be considered. To receive partial credit for programs that are non-working, or are not fully functional, a brief one or two paragraph description of the problem(s) must be included in the assignment submission, in an ASCII text file named `problems.txt`. The location of the problem, minimally identified to a specific function or functions, must also be specified along with possible corrections that need to be made.

Programming Standards:

We will be evaluating your source code on this assignment for the quality of internal documentation and programming style, so you should observe good practice. The specifications from projects 1 and 2 apply. In addition:

Coding:

- Use classes where appropriate.
- Avoid the use of public data members in your classes.
- Divide each of your class implementations into a pair of matching header (`.h`) and source (`.cpp`) files, named to correspond to the class.
- Use explicit association where appropriate, managed by the passing and/or storage of class pointers.
- Use aggregation where appropriate.
- Design your classes so that each has a well-defined and reasonable set of responsibilities.
- Since there is no constraint on its dimensions, use a dynamic data structure to represent the maze grid.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header preceding your `main()` function:

```
//     On my honor:
//
//     - I have not discussed the C++ language code in my program with
//       anyone other than my instructor or the teaching assistants
//       assigned to this course.
//
//     - I have not used C++ language code obtained from another student,
//       or any other unauthorized source, either modified or unmodified.
//
//     - If any C++ language code or documentation used in my program
//       was obtained from another source, such as a text book or course
//       notes, that has been clearly noted with a proper citation in
//       the comments of my program.
//
//     - I have not designed this program in such a way as to defeat or
//       interfere with the normal operation of the Automated Grader.
```

Failure to include this pledge in a submission is a violation of the Honor Code.

Sample Input File:

```
Rows:      6
Columns:   8
Enter:    ( 3, 1)
Exit:     ( 4, 8)
WWWWWWWW
W      W
   W W W
W W W
W   W WW
WWWWWWWW
make        human       Carter      15
make        monster     Khafre      ( 3, 5)     25
make        monster     Ramesses    ( 5, 6)     25
display     maze
Carter      step        East
Carter      see         monster?
Carter      step        North
Khafre      step        North
display     maze
Carter      see         monster?
Carter      step        South
Khafre      trek        West
Khafre      status
Carter      step        East
Carter      step        North
Carter      trek        East
Carter      status
Ramesses    step        North
Carter      step        South
Khafre      trek        East
Carter      see         monster?
Carter      step        South
Khafre      step        South
display     status
display     maze
Carter      step        East
Carter      see         monster?
Khafre      step        South
Ramesses    step        East
display     maze
Khafre      status
Ramesses    status
Carter      status
Carter      step        East
Carter      status
make        human       Peabody     20
Peabody     trek        East
Peabody     step        North
Ramesses    trek        North
Peabody     see         monster?
display     status
display     maze
Peabody     step        South
Ramesses    trek        West
Peabody     status
Ramesses    status
Peabody     step        East
Ramesses    step        South
Peabody     step        North
Ramesses    step        East
```

```
Peabody     trek       East
Ramesses    step       North
Peabody     trek       South
display     maze
Ramesses    trek       East
Peabody     step       West
Ramesses    trek       South
Peabody     step       South
Ramesses    step       West
Peabody     step       South
display     maze
Peabody     status
Ramesses    status
Ramesses    step       South
display     status
```

Sample Output:

```
Maze initialized:       8 x 6       Entrance ( 3, 1)  Exit ( 4, 8)
Human created:          Carter
Monster created:        Khafre
Monster created:        Ramesses

Maze Map: 12345678
         1WWWWWWWWW
         2W       W
         3H   WMW W
         4W W W
         5W    WMWW
         6WWWWWWWWW

Carter does not see monster.

Maze Map: 12345678
         1WWWWWWWWW
         2WH   M  W
         3    W W W
         4W W W
         5W    WMWW
         6WWWWWWWWW

Carter sees monster East.
Khafre is at ( 2, 2) with energy level 20.
Carter is at ( 2, 7) with energy level  5.
Carter sees monster North.

Status:
    Khafre         ( 3, 7)      13
    Ramesses       ( 4, 6)      24
    Carter         ( 4, 7)       3

Maze Map: 12345678
         1WWWWWWWWW
         2W       W
         3    W WMW
         4W W WMH
         5W    W WW
         6WWWWWWWWW

Carter sees monster West.

Khafre not found.
Ramesses is at ( 4, 7) with energy level 23.
Carter is at ( 4, 8) with energy level 2.

Maze Map: 12345678
         1WWWWWWWWW
         2W       W
         3    W W W
         4W W W MH
         5W    W WW
         6WWWWWWWWW

Carter not found.

Human created:          Peabody

Peabody sees monster East.
```

These energy levels ARE correct. If you're off by one, go back and read page one more carefully.

```
Status:
    Ramesses      ( 2, 7)     20
    Peabody       ( 2, 3)     16

Maze Map: 12345678
         1WWWWWWWWW
         2W H    MW
         3   W W W
         4W W W
         5W   W WW
         6WWWWWWWWW

Status:
    Ramesses            ( 2, 2)     14
    Peabody             ( 3, 3)     15

Peabody can't step East from ( 3, 3).

Maze Map: 12345678
         1WWWWWWWWW
         2W M     W
         3   W W W
         4W W W H
         5W   W WW
         6WWWWWWWWW

Peabody can't step South from ( 5, 6).

Maze Map: 12345678
         1WWWWWWWWW
         2W       W
         3   W W W
         4W W WM
         5W   WHWW
         6WWWWWWWWW

Peabody is at ( 5, 6) with energy level 2.
Ramesses is at ( 4, 6) with energy level 2.

Ramesses eats Peabody.

Status:
    Ramesses            (5, 6)      3
```