CSCI 2170 Spring 2006
OLA 4 **Name _________________**
Part 1 (30 pts) due Tuesday, Feb 28th,
Part 2 (20 pts) due Thursday March 2nd,
Final program due Tuesday March 14th)

This program requires you to solve the maze problem using recursion plus backtracking. Detailed description of the program is given in the book on pages 273-276 (chapter 5).

You can download the 3 datafiles MyMaze1.dat, MyMaze2.dat, and MyMaze3.dat from directory ~cli/data/. You can test your program using these or your own data. To turn in your assignment, you need to include the results generated from all three data files.

The data file is formatted as the following:

```
20 7      ← size of the maze   number of columns  number of rows
0 18      ← the coordinates of the entrance point
6 12      ← the coordinates of the exit point
****************** *
*     *      ***** *
* ***** ***        *
* ***** *****   ** *
* *             *  *
* *******  *       *
************ *******
```

Your program needs to :

1. (1) Use dynamic memory allocation for the 2D array Maze
2. (2) Define two classes: **CreatureClass** and **MazeClass** (for MazeClass, you have to define your own destructor to free memory space dynamically allocated for the Maze).
3. (3) Implement recursive GoNorth(), GoSouth(), GoEast(), GoWest() client program functions as described in book.
4. (4) If a path exists between entrance and exit, print out the entire maze, including
    a. the path from the entrance point to the exit point marked by characters 'p',
    b. spaces explored by the creature by character ‘v’, like the following:

```
******************p*
*     *pppppp*****p*
* *****p***vppppppp*
* *****p*****   ** *
* *vvvvpppppp   *  *
* *******  *p      *
************p*******
```

If no path exists for the problem, display an appropriate message and the entire maze after the exploration process.

Part A: (30 pts)
For this part of the assignment, implement the MazeClass. A Maze object consists of the following data:

- a maze (2D array of character),

- the number of rows in maze,
- the number of columns in maze,
- the entrance location of the maze, and
- the exit location of the maze.

The methods in this class include the following:

- ReadMaze that reads a maze from a file,
- Display that display the maze,
- GetEntrace that returns the location of the entrance location,
- GetExit that returns the location of the exit location,
- MarkVisited that marks one maze location being visited,
- MarchPath that marks one maze location is part of the path being explored by a creature,
- IsWall that determines whether a particular location in the maze is wall,
- IsClear that determines whether a particular location in the maze is clear of objects,
- IsPath that determines whether a particular location is part of the path being explored,
- IsVisited that determines whether a particular location has been visited and does not lead to exit,
- IsExit that determines whether a particular location in the maze is the exit point, and
- IsInMaze that determines whether a location provided is part of the maze.

You are required to use dynamic allocation of memory to create the maze array. As a result, you need to explicitly define the destructor for the MazeClass to release memory space allocated when the maze object exits its scope.

Write the header file and implementation file for the MazeClass, and then write a simple client program to test your MazeClass. This client program should:

- create and initialize a maze object (including reading the maze from a data file),
- display message that report the entrance and exit locations of the maze, and
- display message for 3 different locations of the maze whether it is wall, is clear, or is in maze. Select the 3 locations to be such that one location is wall, one location is clear, and one location is outside of the maze.

Turn in your program with a script file containing the following:

```
frank % pr –n MazeClass.h
frank% pr –n MazeClass.cpp
frank% pr –n ola3a.cc
frank% aCC MazeClass.cpp ola3a.cc –o run
frank% run
```

<u>Part B: (20 points)</u>

For this part, you are required to implement the **CreatureClass.** A creature can more up, down, left or right, one step at a time. It can be assigned to, or put at, a location, and can report its current location. The only data it needs to keep up with is its location, or coordinates(x, y), at any given time. First, write the header file and the implementation file of CreatureClass. Then, write a simple client program to test this class. The client program should create a creature, assign it to a location of your choice, make it move x steps left, y steps down, z steps right, and q steps up, and then ask it to report its current location.

Turn in your program with a script file containing the following:

```
frank % pr –n creatureClass.h
frank% pr –n creatureClass.cpp
frank% pr –n ola3b.cc
frank% aCC creatureClass.cpp ola3b.cc –o run
frank% run
```

<u>Final Program (50 pts)</u>

Implement the final client program that solve the maze. Implement recursive GoNorth(), GoSouth(), GoEast(), GoWest() client program functions as described in the text book and discussed in class. This program will include both the mazeClass and the creatureClass header files.

Your program should be flexible – the entrance and exit points in a maze can be on any of the four sides.

Turn in your program with a script file containing the following:

```
frank % pr –n creatureClass.h
frank% pr –n creatureClass.cpp
frank % pr –n MazeClass.h
frank% pr –n MazeClass.cpp
frank% pr –n ola3.cc
frank% aCC creatureClass.cpp MazeClass.cpp ola3.cc –o run
frark% run   3 times with the three maze data.
```