Programming Assignment # 1

Due dates: Part 1: Sept 3; Part 2: Sept 8 ; Part 3: Sept 19

Problem Description

You will write a program that keeps track of locations of "people" (actually integers) in a "grid world." Your program takes as input 3 items:

- `N`: the number of people in the world; this value is fixed for the duration of the program. People are referred to by integers in the range $0..N-1$.
- `nrows`: the number of rows in the grid.
- `ncols`: the number of columns in the grid.

We refer to a particular entry in the grid as a *district*. Indices always begin at 0 (for people, rows and columns).

When your program starts it will take these parameters (as specified on the command line) and, depending on the mode of the program, will either assign each person to a random district (this is the default mode) or assign all N people to district $(0, 0)$ (details on how these modes are specified appears later in this handout).

Once the program has set up the grid, it enters an interactive loop which supports four commands:

- `members i j`: this prints out a list of all of the people currently living in district (i, j). The runtime of this operation must be *linear* in the number of people living in the district (*not* in the number of people in the world – N).
- `move x i j`: this moves person x from wherever they currently are to district (i, j). This operation must take only constant time!
- `whereis x`: this reports the district (i.e., the (i, j) pair) where person x is currently living.
- `quit`: this deallocates all data structures as necessary and terminates the program.

Part 1: Solution design

In this part you will design a solution to the problem, but will not do any implementation. You should think of this as a prelinary design document.

The document will include the following:

(a) An informal description of what it means for an operation to take constant time including examples (not necessarily relating to this project) illustrating constant time and non-constant time operations.

(b) Solution description. This should include "boxes and arrows" diagrams clearly illustrating how you intend to organize the data as well as pseudo-code describing how you will accomplish each operation in the required time. The pseudo-code can include English, but must be very clear and described in a step-by-step fashion. You will need to make a clear argument that your design meets the runtime requirements. Though you will not be writing real code at this point, it will probably help if you use variable names to refer to your main data structures.

Part 2: Class interface design

Now that you have a strategy for how to solve the problem, you will start your program design. You should think of your program in two components (and multiple files): the *application* and the underlying *classes* (ADTs) which the application will use to achieve the desired functionality.

In this part of the project, you will design the class interfaces (as `.h` files) which you will use for your final program. (In theory, once this interface is established, the application and the classes can be implemented independently and then linked together).

You will turn in these `.h` files and explain in comments how they are consistent with the design from Part 1.

Part 3: Implementation

Now you will fill in the blanks and complete the program. This includes the implementation of the classes and the implementation of the application program.

Your program should be well documented and obey standards of good coding style.

Program Behavior

Your executable will be called `gridworld`. A synopsis of how your program will run from the command line is as follows:

```
gridworld [-norand] [-ncols numrows] [-nrows numcols] [-n N]
```

If the program is run without any command line arguments, it uses the following defaults: `nrows=5`; `ncols=5`, `N=5`. Additionally, the program runs initially assigns each person to a random district *unless* the `-norand` flag is specified.

Once the program begins, it enters the interpreter loop executing the commands described above. A sample session is below.

```
% gridworld
> members 3 4
    people: 3 7
> whereis 7
    district (3,4)
> move 3 0 0
    ok
> whereis 3
    district (0,0)
> members 3 4
    people: 7
> members 5 5
    error: value out of range
> wheris 10
    error: value out of range
> quit
    goodbye...
%
```

Grading

Programs will be turned in using the `turnin` command on the department machines. More specific instructions will appear on the webpage.

Programs will be graded both on their accuracy (as evidenced by our executing them), on the readability of the source code and evidence of good programming practices. Programs that fail to compile will receive no credit. Otherwise, partial credit will be given. Therefore, if you cannot complete the entire assignment, do try to get at least some part(s) to work correctly so as to maximize the credit you earn. Late penalties will be enforced per the syllabus.