**APCS, Spring, 2002**

**Assignment 3:Working with partII of MBCS**

**Files:** [**environ.h**](http://docs.google.com/cs102labs/lab4/environ.h)**,** [**environ.cpp**](http://docs.google.com/cs102labs/lab4/environ.cpp)**,** [**fish.dat**](http://docs.google.com/cs102labs/lab4/fish.dat)**,** [**fish.h**](http://docs.google.com/cs102labs/lab4/fish.h)**,** [**fish.cpp**](http://docs.google.com/cs102labs/lab4/fish.cpp)**,** [**FishDisplyG.h**](http://docs.google.com/cs102labs/lab4/FishDisplayG.h)**,**[**FishDisplayG.cpp**](http://docs.google.com/cs102labs/lab4/FishDisplayG.cpp)**,** [**nbrhood.h**](http://docs.google.com/cs102labs/lab4/nbrhood.h)**,** [**nbrhood.cpp**](http://docs.google.com/cs102labs/lab4/nbrhood.cpp)**,**

[**position.h**](http://docs.google.com/cs102labs/lab4/position.h)**,** [**position.cpp**](http://docs.google.com/cs102labs/lab4/position.cpp)**,** [**randomgen.h**](http://docs.google.com/cs102labs/lab4/randomgen.h)**,** [**randomgen.cpp**](http://docs.google.com/cs102labs/lab4/randomgen.cpp)**,** [**simulate.h**](http://docs.google.com/cs102labs/lab4/simulate.h)**,** [**simulate.cpp**](http://docs.google.com/cs102labs/lab4/simulate.cpp)**,** [**utils.h**](http://docs.google.com/cs102labs/lab4/utils.h)**,** [**utils.cpp**](http://docs.google.com/cs102labs/lab4/utils.cpp)**,** [**fishsim.cpp**](http://docs.google.com/cs102labs/lab4/fishsim.cpp)

**Purpose**

In this programming assignment, we learn how place the fish in a different geometrical relationship.  We also learn how to adapt the two dimensional rectangular grid to represent a hexagonal grid.  Finally, we investigate changes to the way in which the fish move.

**Motivation**

The biologists ask if we can modify the simulation program so that the fish have more freedom of movement.  We propose a hexagonal grid, which gives each fish six neighboring cells into which they can move, or the original rectangular grid with movement along the diagonals added, giving eight directions of movement.  The biologists opt for the hexagonal grid.  They believe that this will better reflect the variety of fish movement and they like keeping the relationships among the neighboring cells symmetric, which would not be the case for a rectangular grid with diagonal movement.

In addition, the biologists want to study a species of fish that does not move backwards at all.  In the hexagonal representation, this means that the fish will move forward or 60 degrees to the right or left with equal probabilities.  It will not move directly back nor at an angle toward the back.

**Part 1: The Hexagonal Grid**

In order to change the behavior of each fish as requested by the biologists, we need to modify the relationships among the positions.  We also need to determine how we can compactly represent a hexagonal grid, where each position has six neighbors.  It turns out that the idea of the representation is easy.  We store the positions in a two dimensional array as we did before, but we interpret each position in the array differently.  If we orient the hexagonal grid as shown below, then each position has one neighbor to the right and one to the left in its own row, but two neighbors in the row above and two neighbors in the row below in directions naturally designated Northeast, Northwest, Southeast and Southwest.  For example, the 5 x 4 grid shown below has the odd numbered columns shaded for clarity.  Notice how the indexing of the Northest, Northwest, Southeast and Southwest neighbors of a position depends on whether that position is in an even or an odd numbered row.  The diagram has the position (2, 2) indicated by an X and its six neighbors designated by NE, E, SE, SW, W, NW.



If we examine the classes making up the fish simulation program, we see the the relationship between neighbors in the two dimensional grid is effectively defined by the four Position member functions North, East, South, West.  These functions are called in the Fish class member function EmptyNeighbors, which constructs a Neighborhood containing the positions to which the fish can move.  Consequently, we can change the internal representation of the fish grid to a hexagonal grid by simply changing these functions appropriately.  In the course of doing this we note that a small change must be made in the Neighborhood class implementation to accommodate the larger number of neighbors now possible and an adjustment to the Display class is needed to represent the new grid on screen.

Before writing code you should plan your changes.

1. Prepare a plan for your modifications, as outlined below.
2. Replace the four direction functions in the Position class with six functions representing the six hexagonal grid directions.
3. Modify the Fish member function EmptyNeighbors to create a Neighborhood containing all the empty neighbors of this Fish, according to the six directions specified by the Position functions.
4. Change the maximum number of Positions that can be stored in a Neighborhood so that all the neighbors can be accommodated.
   * The current implementation of Neighborhood uses an explicit constant (4) in the Neighborhood constructor.  In general, this is a poor design.  It is better to pass the value as a parameter to the Neighborhood constructor.
5. The graphical display can be done using an offset of half a square's width for every other line of a rectangular display.

**Part 2:  Restricted Movement**

A biologist wants to adapt the fish simulation to study a species of fish which moves differently from those modeled by the hexagonal program developed above.  This species cannot move backwards, but they are equally likely to move forward, to the right sixty degrees, or to the left sixty degrees on a given step.  The fish cannot move directly backwards nor at an angle backwards.  This biologist asks us to modify the simulation to change the simulation so that the fish move in this way, but so that other characteristics of the simulation are unchanged.

**Assignment**

In order to change the behavior of each fish as requested by the biologist, we need to know the direction in which the fish is moving.  We will need another data member for the Fish class to record this additional aspect of a fish's state.  We will also need to modify the Move member function in the Fish class to reflect the pattern of movement described above.  There are several ways in which the direction of the fish can be recorded.

Before writing code you should plan your changes.  You might find it useful to use a copy of the Class Diagrams available on the External Links page to do your planning.  In fact, you should do a modified version of the Fish class diagram to reflect the changes that you make.

1. Prepare a plan for your modifications, as outlined below.  You can modify the Fish class diagram from the Class Diagrams on the External Links page to reflect the new design.
2. Select a method for recording a fish's direction of movement and add the appropriate declaration for the data member to the Fish class.
   * int myDir;  representing directions by degrees of the compass (30 Northeast, 90 East, 150 Southeast, 210 Southwest, 270 West, 330 Northwest)
   * apstring myDir;  "Northeast", "Northwest", "East", "Southeast", "Southwest", "West"
   * Position myPrev; the previous position of this Fish
   * your own method
3. Modify the Fish member function ToString so that it reports the direction part of the state of this fish in an appropriate fashion.
4. Write a new Fish private member function LegalMoves.  LegalMoves should return a neighborhood of the legal moves for this Fish according to the pattern of movement described above.  The function LegalMoves should be similar to the existing function EmptyNeighbors.
5. Write a new implementation for the Fish member function Move.
   * Use the function LegalMoves to get the possible moves for this Fish and select from among them at random. The resulting pattern of movement should match the pattern specified above.
   * Update the Fish data member recording the direction appropriately.  It should not be changed if the fish does not move.
6. Devise a plan for testing your modified simulation to see whether the fish move as specified by the biologist.
   * Create the fish data files needed for your tests.
   * Run your tests and record the results.  Determine whether the simulation meets the new specification.