3. This question involves reasoning about the code from the Marine Biology Case Study. A copy of the code is provided as part of this exam.

   Consider modifying fish so they have a direction based on their last move, and move only in a forward direction: either straight ahead or diagonally ahead to the right or left. You will be asked to write three functions for this question:

   (a) a `Position` member function that returns the position to the northeast of the current position,

   (b) a `Fish` member function that returns a neighborhood of positions representing the forward moves that a fish can make, and

   (c) a `Position` member function that returns the direction from this `Position` to an adjacent `Position`.

   For this question, Potential Movement Locations are positions that are:

   (1) adjacent to the current fish position,
   (2) in the direction the fish is moving, `myDir`, or 45 degrees to the right or left of `myDir`, and
   (3) empty and in the environment `env`.

   The diagram below shows three fish, represented by arrows showing their current directions. The Potential Movement Locations for each fish are shaded. For example, for the fish at position (2,1) in the diagram with `myDir` equal `"E"` (as indicated by the arrow), Potential Movement Locations would be those positions to the northeast, east, and southeast that are empty. For the fish at position (2,9) with `myDir` equal `"SW"`, Potential Movement Locations would be those positions to the west, southwest and south that are empty.

***Potential Movement Locations***

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | NW | ▨ N | ▨ NE | | | | | |
| **1** | NW | N | ▨ NE | | W | ➚ | ▨ E | | NW | N | NE | |
| **2** | W | ➔ | ▨ E | | SW | S | SE | | ▨ W | ↙ | E | |
| **3** | SW | S | ▨ SE | | | | | | ▨ SW | ▨ S | SE | |
| **4** | | | | | | | | | | | | |

GO ON TO THE NEXT PAGE.

The `Fish` class is modified to include an additional private data member to hold the direction in which the fish is moving and a private member function to determine the positions of legal forward moves for the fish. The modifications to the `Fish` class are shown below.

```
class Fish
{
  public:
    // ... member functions as in the original version

  private:
    // ... private data as in the original version

    apstring myDir; // "N" for north, "E" for east,
                    // "S" for south, "W" for west,
                    // "NE" for northeast, "SE" for southeast,
                    // "NW" for northwest, "SW" for southwest

    Neighborhood ForwardNbrs(const Environment & env) const;
    // postcondition: returns empty neighbors in a forward direction
    //                from myDir - straight ahead and diagonally ahead
    //                to the right or left

};
```

Four new public member functions are added to the `Position` class: `Northeast()`, `Southeast()`, `Northwest()`, and `Southwest()`, each of which returns the neighboring `Position` in the specified direction. Functions `North`, `East`, `South`, and `West` are unchanged. The modifications to the `Position` class are shown below.

```
class Position
{
  public:
    // ... member functions as in the original version

    Position Northeast() const;
      // postcondition: returns Position northeast of this position
    Position Southeast() const;
      // postcondition: returns Position southeast of this position
    Position Northwest() const;
      // postcondition: returns Position northwest of this position
    Position Southwest() const;
      // postcondition: returns Position southwest of this position

    apstring DirectionTo(const Position & other) const;
    // precondition:  other is adjacent to this Position
    // postcondition: returns the string representation of the direction
    //                from this Position to other

  private:
    // ... private data as in the original version

};
```

GO ON TO THE NEXT PAGE.

(a) You will write the `Position` member function `Northeast`, which is described as follows. `Northeast` should return the position in the environment to the northeast of the current position. In the diagram shown above, if `pos1` is the position (2, 1), the call `pos1.Northeast()` returns the position (1, 2), and if `pos2` is the position (2, 9), the call `pos2.Northeast()` returns the position (1, 10).

Complete function `Northeast` below.

```
Position Position::Northeast() const
// postcondition: returns Position northeast of this position
```

(b) You will write the `Fish` member function `ForwardNbrs`, which is described as follows. `ForwardNbrs` should return a neighborhood consisting of those positions that meet the requirements for Potential Movement Locations.

In writing `ForwardNbrs`, you may use any of the `Fish` and `Position` member functions. Assume that these functions, including `Position::Northeast`, work as specified, regardless of what you wrote in part (a).

An implementation of this function distinguishes among multiple cases based on direction. In writing your code, you must show the code for the two specific cases, north and northeast. You may write `" ... "` to indicate where the remaining cases should be. All statements other than these remaining cases must be shown.

Complete function `ForwardNbrs` below.

```
Neighborhood Fish::ForwardNbrs(const Environment & env) const
// postcondition: returns empty neighbors in a forward direction from
//                myDir - straight ahead and diagonally ahead to the
//                right or left
```

(c) You will write the `Position` member function `DirectionTo`, which returns the direction from this `Position` to `Position other`.

An implementation of this function distinguishes among multiple cases based on direction. In writing your code, you must show the code for the two specific cases, north and northeast. You may write `" ... "` to indicate where the remaining cases should be. All statements other than these remaining cases must be shown.

Complete function `DirectionTo` below.

```
apstring Position::DirectionTo(const Position & other) const
// precondition:  other is adjacent to this Position
// postcondition: returns the string representation of the direction
//                from this Position to other
```

GO ON TO THE NEXT PAGE.