4. This question involves reasoning about the code from the Marine Biology Case Study. A copy of the code is provided in the Appendix.

   The marine biologists want to study a species of fish that eats algae. Any position in the environment grid can contain zero or more units of algae. If there is any algae at a fish's location, the fish eats one unit of the algae and does not move; otherwise, the fish does not eat. If this is the third consecutive step in which the fish has not eaten, then the fish dies and is removed from the environment. If the fish does not eat and does not die, it moves to a position among its empty neighbors that contains the most algae.

   We represent the algae by adding a matrix of integers to the private data of the `Environment` class. This matrix is the same size as `myWorld`, and each entry represents the number of units of algae at that location. We add three public member functions to the `Environment` class, as well as modifying the `Environment` constructor to initialize `myAlgae`.

```
//  Added to the public section of class Environment

    void RemoveFish(const Position & pos);
    // precondition:  there is a fish at pos
    // postcondition: there is no fish at pos

    int NumAlgaeAt(const Position & pos) const;
    // precondition:  pos is a valid position in the environment
    // postcondition: returns the number of units of algae at pos

    void RemoveAlgae(const Position & pos, int numUnits);
    // precondition:  algae at position pos exceeds numUnits
    // postcondition: algae at position pos has been reduced by numUnits



//  Added to the private section of class Environment

    apmatrix<int> myAlgae;  // number of units of algae at each position
```

GO ON TO THE NEXT PAGE.

We modify the `Fish` class by adding a private data member to keep track of how long since the fish ate any algae. We also add public member function `Act` that encapsulates all the actions of a fish for one step of the simulation and we modify the `Move` function so that the fish moves to the position among its empty neighbors that has the most algae. In order to do this we add private member function `MostAlgae` to the `Fish` class.

```
//  Added to the public section of class Fish

    void Act(Environment & env);
    // precondition:  this Fish is stored in env at Location()
    // postcondition: if there was algae at Location(), this Fish ate
    //                and one unit of algae has been removed from
    //                Location(); otherwise, if this was the third
    //                consecutive step that this Fish did not eat,
    //                then this Fish has been removed from env;
    //                otherwise, this Fish moved.
    //                myStepsSinceFed has been updated.



//  Modified and moved to the private section of class Fish

    void Move(Environment & env);



//  Added to the private section of class Fish

    Position MostAlgae(const Environment & env,
                       const Neighborhood & nbrs) const;
    // precondition:  nbrs.Size() > 0
    // postcondition: returns a Position from nbrs that contains
    //                the most algae.

    int myStepsSinceFed; // steps since this fish last ate
```

(a) Write the `Environment` member function `NumAlgaeAt`, which is described as follows. `NumAlgaeAt` should return the number of units of algae at `pos`.

Complete function `NumAlgaeAt` below.

```
int Environment::NumAlgaeAt(const Position & pos) const
// precondition:  pos is a valid position in the environment
// postcondition: returns the number of units of algae at pos
```

GO ON TO THE NEXT PAGE.

(b) Write the `Fish` member function `MostAlgae`, which is described as follows. `MostAlgae` should return a `Position` from `nbrs` that contains the most algae. If more than one position contains the maximum amount, any of those positions may be returned.

In writing `MostAlgae`, you may use any of the `Environment` public member functions, including `NumAlgaeAt`. Assume that `NumAlgaeAt` works as specified, regardless of what you wrote in part (a).

Complete function `MostAlgae` below.

```
Position Fish::MostAlgae(const Environment & env,
                         const Neighborhood & nbrs) const
// precondition:  nbrs.Size() > 0
// postcondition: returns a Position from nbrs that contains
//                the most algae
```

(c) Write the `Fish` member function `Act`, which is described as follows. If there is algae at the fish's current position, the fish should eat one unit of algae and not move. If there is no algae and this is the third consecutive step in which the fish has not eaten, `Act` will cause the fish to die by calling `env.RemoveFish`. If the fish does not eat and does not die, then the fish should move to a neighboring position with the most algae. `Act` should update the state of the environment and the state of the fish appropriately.

In writing `Act`, you may use any member function from the Marine Biology Case Study, including those added at the beginning of the question. Assume that `Fish::Move` has been modified to work correctly and that `Environment::NumAlgaeAt` and `Fish::MostAlgae` work as specified, regardless of what you wrote in parts (a) and (b).

Complete function `Act` below.

```
void Fish::Act(Environment & env)
// precondition:  this Fish is stored in env at Location()
// postcondition: if there was algae at Location(), this Fish ate
//                and one unit of algae has been removed from
//                Location(); otherwise, if this was the third
//                consecutive step that this Fish did not eat,
//                then this Fish has been removed from env;
//                otherwise, this Fish moved.
//                myStepsSinceFed has been updated.
```

END OF EXAMINATION