(a) Write the Environment member function RemoveFish, as started below. RemoveFish checks
its precondition and prints an error message if the precondition is not met. Otherwise, RemoveFish
removes the fish in position pos from the environment and updates myFishCount.

In writing RemoveFish, you do not need to include calls to DebugPrint.

Complete function RemoveFish below.

```cpp
void Environment::RemoveFish(const Position & pos)
// precondition: there is a fish at pos (IsEmpty(pos) is false)
// postcondition: fish removed from pos; IsEmpty(pos) is true
{
    if (IsEmpty(pos))
    {
        cerr << "error - attempt to remove nonexistent fish at:"
             << pos << endl;
        return;
    }
    myFishCount--;
    myWorld[pos.Row()][pos.Col()] = Fish();
```
(b) Write the Fish member function Breed, as started below. Breed asks the environment, env, to add a new fish in every one of the fish's empty neighboring positions, each with age 0 and with the same probability of dying as this fish.

In writing Breed, you do not need to include calls to DebugPrint. Assume that all member functions of the Environment class work as specified above.

Complete function Breed below.

```cpp
void Fish::Breed(Planet & env) {
    // precondition: this fish is stored in env at Location()
    // this fish is old enough to breed
    // postcondition: the neighboring empty positions of this fish have been filled with new fish, each with age 0 and the same probability of dying as this fish

    Position N = myPos.North(), S = myPos.South(), E = myPos.East(), W = myPos.West();
    if (env.IsEmpty(N))
        env.AddFish(N, 0, myProbDie);
    if (env.IsEmpty(S))
        env.AddFish(S, 0, myProbDie);
    if (env.IsEmpty(E))
        env.AddFish(E, 0, myProbDie);
    if (env.IsEmpty(W))
        env.AddFish(W, 0, myProbDie);
}```
(c) Write the Fish member function Act, as started below. Act will, with probability myProbDie, cause the fish to die by calling env.RemoveFish. If the fish does not die, it should increment its age. If its new age is three, it should breed; otherwise, it should attempt to move. You will not receive full credit if you reimplement Move and Breed within function Act.

In writing Act, you do not need to include calls to DebugPrint. Assume that all member functions of the Environment and Fish classes work as specified above. You may also assume that Environment member function RemoveFish and the Fish member function Breed work as specified, regardless of what you wrote in parts (a) and (b).

Complete function Act below.

```c
void Fish::Act(Environment & env)
// precondition: this fish is stored in env at Location()
// postcondition: this fish has moved, bred, or died
{
  RandGen r;
  double die = r.RandReal();
  if (die <= myProbDie)
    env.RemoveFish(myPos);
  else
  {
    myAge++;
    if (myAge == 3)
      Breed(env);
    else
      Move(env);
  }
```
(a) Write the `Environment` member function `RemoveFish`, as started below. `RemoveFish` checks its precondition and prints an error message if the precondition is not met. Otherwise, `RemoveFish` removes the fish in position `pos` from the environment and updates `myFishCount`.

In writing `RemoveFish`, you do not need to include calls to `DebugPrint`.

Complete function `RemoveFish` below.

```cpp
void Environment::RemoveFish(const Position & pos)
    // precondition: there is a fish at pos (IsEmpty(pos) is false)
    // postcondition: fish removed from pos; isEmpty(pos) is true
{
    if (IsEmpty(pos))
    {
        cerr << "error - attempt to remove nonexistent fish at:"
            << pos << endl;
        return;
    }

    // myWorld [pos.Row][pos.Col].amIdefined = false;
    myFishCount --;  
```
(b) Write the Fish member function Breed, as started below. Breed asks the environment, env, to add a new fish in every one of the fish's empty neighboring positions, each with age 0 and with the same probability of dying as this fish.

In writing Breed, you do not need to include calls to DebugPrint. Assume that all member functions of the Environment class work as specified above.

Complete function Breed below.

```c
void Fish::Breed(Environment & env)
// precondition: this fish is stored in env at Location();
//               this fish is old enough to breed
// postcondition: the neighboring empty positions of this fish have
//                been filled with new fish, each with age 0 and
//                the same probability of dying as this fish
```

```c
for (Position pos = location();
```

```c
    (env.Empty(pos));
```

```c
    env.AddFish(pos.North(), 0, myProbDie);
```
(c) Write the Fish member function Act, as started below. Act will, with probability myProbDie, cause the fish to die by calling env. RemoveFish. If the fish does not die, it should increment its age. If its new age is three, it should breed; otherwise, it should attempt to move. You will not receive full credit if you reimplement Move and Breed within function Act.

Note: If \( r \) is defined as follows,

\[
\text{RandGen } r;
\]

then the expression \( (r.\text{RandReal()} < \text{myProbDie}) \) will evaluate to true with probability \( \text{myProbDie} \).

In writing Act, you do not need to include calls to DebugPrint. Assume that all member functions of the Environment and Fish classes work as specified above. You may also assume that Environment member function RemoveFish and the Fish member function Breed work as specified, regardless of what you wrote in parts (a) and (b).

Complete function Act below.

```cpp
void Fish::Act(Environment & env)
// precondition: this fish is stored in env at Location()
// postcondition: this fish has moved, bred, or died

if (r.\text{RandReal()} < \text{myProbDie})
    \text{RemoveFish(Location())};
else
    myAge++;
    if (myAge == 3)
        \text{Breed(env)};
    else
        \text{move()};
```
(a) Write the Environment member function RemoveFish, as started below. RemoveFish checks its precondition and prints an error message if the precondition is not met. Otherwise, RemoveFish removes the fish in position pos from the environment and updates myFishCount.

In writing RemoveFish, you do not need to include calls to DebugPrint.

Complete function RemoveFish below.

```cpp
void Environment::RemoveFish(const Position & pos)
// precondition: there is a fish at pos (IsEmpty(pos) is false)
// postcondition: fish removed from pos; IsEmpty(pos) is true
{
    if (IsEmpty(pos))
    {
        cerr << "error - attempt to remove nonexistent fish at:"
 << pos << endl;
        return;
    }
    myWorld (pos) = emptyFish
    myFishCount --;
```
(b) Write the Fish member function Breed, as started below. Breed asks the environment, env, to add a new fish in every one of the fish's empty neighboring positions, each with age 0 and with the same probability of dying as this fish.

In writing Breed, you do not need to include calls to DebugPrint. Assume that all member functions of the Environment class work as specified above.

Complete function Breed below.

```cpp
void Fish::Breed(Environment & env)
// precondition: this fish is stored in env at Location();
// this fish is old enough to breed
// postcondition: the neighboring empty positions of this fish have
// been filled with new fish, each with age 0 and
// the same probability of dying as this fish

// Neighborhood nbrs
AddIfEmpty(env, n brs, pos, North(t));
AddIfEmpty(env, n brs, pos, North(e));

n brs = Empty Neighbors(env, pos);

for (int x=0; x < n brs, numRows()-1; x++)
    for (int y=0; y < n brs, numCols()-1; y++)
        n brs[x][y].AddIfEmpty(1);
        n brs[x][y].my Age = 0;
        n brs[x][y].my ProbDie = env[x][y];
```
(c) Write the Fish member function `Act`, as started below. `Act` will, with probability `myProbDie`, cause the fish to die by calling `env.RemoveFish`. If the fish does not die, it should increment its age. If its new age is three, it should breed; otherwise, it should attempt to move. You will not receive full credit if you reimplement `Move` and `Breed` within function `Act`.

Note: If `r` is defined as follows,

```cpp
RandGen r;
```

then the expression `r.RandReal() < myProbDie` will evaluate to `true` with probability `myProbDie`.

In writing `Act`, you do not need to include calls to `DebugPrint`. Assume that all member functions of the Environment and Fish classes work as specified above. You may also assume that Environment member function `RemoveFish` and the Fish member function `Breed` work as specified, regardless of what you wrote in parts (a) and (b).

Complete function `Act` below.

```cpp
void Fish::Act(Environment & env)
// precondition: this fish is stored in env at Location()
// postcondition: this fish has moved, bred, or died
    RandGen r;
    if (r.RandReal() == myProbDie)
        env.RemoveFish();
    else if (myAge == 3)
        Fish.Breed(env);
    else Act;
    move;
```