AP® Computer Science A
2001 Sample Student Responses

These materials were produced by Educational Testing Service (ETS), which develops and administers the examinations of the Advanced Placement Program for the College Board. The College Board and Educational Testing Service (ETS) are dedicated to the principle of equal opportunity, and their programs, services, and employment policies are guided by that principle.

The College Board is a national nonprofit membership association dedicated to preparing, inspiring, and connecting students to college and opportunity. Founded in 1900, the association is composed of more than 3,900 schools, colleges, universities, and other educational organizations. Each year, the College Board serves over three million students and their parents, 22,000 high schools, and 3,500 colleges, through major programs and services in college admission, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT™, the Advanced Placement Program® (AP®), and Pacesetter®. The College Board is committed to the principles of equity and excellence, and that commitment is embodied in all of its programs, services, activities, and concerns.

(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.

```
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.

```
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
```

```
{
  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.

```
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.

```
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.

```
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
```

{

position pos = location();

env.AddFish(pos.North(), 0, myprobdie)
env.AddFish(pos.South(), 0, myProbDie)
env.AddFish(pos.East(), 0, myProbDie)
env.AddFish(pos.West(), 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,

```
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.

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

```
if (r.RandReal() < myProbDie)
  { RemoveFish(Location()) }
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.

```
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) = emptFish;

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.

```
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, nbrs, pos, North());
AddIfEmpty(env, nb
Neighborhood nbrs
nbrs = EmptyNeighbors(env, pos).

```
for (int x=0; x < nbrs.numrows()-1; x++)
{ for (int y=0; y < nbrs.numcolms()-1; y++)
  {
    nbrs[x][y].AddIfEmpty();
    nbrs[x][y].myAge = 0;
    nbrs[x][y].myProbDie = 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,

```
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.

```
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
    move;
```