AP® Computer Science A
2003 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 whose mission is to prepare, inspire, and connect students to college and opportunity. Founded in 1900, the association is composed of more than 4,300 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 admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). 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.

For further information, visit www.collegeboard.com

For the College Board's online home for AP professionals, visit AP Central at apcentral.collegeboard.com.

A4 A

(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
{

return (myAlgae[pos.Row()][pos.Col()]);

}
```

(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
{
int i;
pos max = nbrs.Select(0);
for(i=1; i<nbrs.Size(); i++)
    if(env.NumAlgaeAt(max) < env.NumAlgaeAt(nbrs.Select(i)))
        Max = Nbrs.Select(i);
return max;
}
```

Part (c) begins on page 22.

(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.
{
}
```

```
if (env.NumAlgaeAt(Location()) > 0)
{
env.RemoveAlgae(Location(), 1);
mySteps Since Fed = 0;
}
else
   if(mySteps Since Fed == 2)
       env.RemoveFish(location());
   else {
       Move(env);
       mySteps Since Fed ++;
       }
}
```

A4 B

(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

{
   return  myAlgae[pos.RowO][pos.colO];
}
```

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

{
  int  algaecount = 0;
  Position  pos;

  for (int i = 0; i < nbrs.Size(); i++)
  {
    Position current = nbrs.Select(i);

    if (env.NumAlgaeAt(current) > algaecount)
    {
      algaecount = env.NumAlgaeAt(current);
      pos = current;
    }
  }

  return pos;
}
```

Part (c) begins on page 22.

(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.
{
   int algcount = env.NumAlgaeAt(Location());
   if (algcount > 0)
   { env.RemoveAlgae(Location(), algcount - 1);
     myStepsSinceFed = 0;
     env.Update(Location(), *this);          // Update private var
   }
   else if (myStepsSinceFed == 3)
   {
     env.RemoveFish(Location());
   }
   else
   {
     myStepsSinceFed++;
     env.Update(Location(), *this);
     Move(env);
   }

}
```

GO ON TO THE NEXT PAGE.

A4 C

(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
{

  return myAlgae[pos.Row()][pos.Col()];

}
```

(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
{
  Position P, Most;
  Most = nbrs.Select(0);
  for(int k=0; k<nbrs.Size(); k++)
  {
    P = nbrs.Select(k);
    if(env.NumAlgaeAt(P) > env.NumAlgaeAt(Most))
      Most = P;
  }

  return Most;
}
```

Part (c) begins on page 22.

(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.
{
  if(mySteps Sinceted >= 3)
    env. Remove Fish (Location());
  else if(env. NumAlgae At(Location()) > 0)
  {
    env. Remove Algae (Location(), 1);
    mySteps Since Fed = 0;
  }
  else
  {
    Move(env);
    my Steps Since Fed++;
  }
}
```

GO ON TO THE NEXT PAGE.