AP® Computer Science A
2002 Sample Student Responses

The materials included in these files are intended for use by AP teachers for course and exam preparation in the classroom; permission for any other use must be sought from the Advanced Placement Program®. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.

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 4,200 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®, 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.

A3 A

(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
{
  return Position(myRow-1, myCol+1);
}
```

GO ON TO THE NEXT PAGE.

(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
{
  Neighborhood nbrs;
  if (myDir == "N")
  {
    AddIfEmpty(env, nbrs, myPos.North());
    AddIfEmpty(env, nbrs, myPos.Northwest());
    AddIfEmpty(env, nbrs, myPos.Northeast());
  }
  if (myDir == "NE")
  {
    AddIfEmpty(env, nbrs, myPos.Northeast());
    AddIfEmpty(env, nbrs, myPos.North());
    AddIfEmpty(env, nbrs, myPos.East());
  }
  . . .
  return nbrs;
}
```

GO ON TO THE NEXT PAGE.

(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
//
{
   apstring direct;
   if ( North() == other)
      direct = "N";
   if ( Northeast() == other)
      direct = "NE";
   . . .
   return direct;
}
```

GO ON TO THE NEXT PAGE.

A3 B

(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
```

```
Position Position::Northeast() const
{
  int row = myRow, col = myCol;

  return Position(myRow - 1, myCol + 1);
}
```

(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 <u>two</u> specific cases, <u>north</u> and <u>northeast</u>. 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
```

```
Neighborhood Fish::ForwardNbrs(const Environment & env) const
{
  Neighborhood nbrs;
  if (myDir == "N")
  {
    AddIfEmpty(env, nbrs, myPos.North());
    AddIfEmpty(env, nbrs, myPos.Northeast());
    AddIfEmpty(env, nbrs, myPos.Northwest());
    return nbrs;
  }
  if (myDir == "NE")
  {
    AddIfEmpty(env, nbrs, myPos.Northeast());
    AddIfEmpty(env, nbrs, myPos.North());
    AddIfEmpty(env, nbrs, myPos.East());
    return nbrs;
  }
  ... // other cases of myDir
}
```

GO ON TO THE NEXT PAGE.

(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
```

```
apstring Position::DirectionTo(const Position & other) const
{
  if (myCol == other.Col()  &&
      myRow == other.Row() + 1)
    return "N";
  if (myCol == other.Col() - 1  &&
      myRow == other.Row() + 1)
    return "NE";
  ... // other directions
}
```

GO ON TO THE NEXT PAGE.

A3 C

(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
{
    return Position(myRow + 1, myCol + 1);
}
```

(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 <u>two</u> specific cases, <u>north</u> and <u>northeast</u>. 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
{
    Neighborhood nbrs;

    if(myDir == "N") {
        AddIfEmpty(env, nbrs, pos.North());
        AddIfEmpty(env, nbrs, pos.Northeast());
        AddIfEmpty(env, nbrs, pos.Northwest());
    }
    if(myDir == "NE") {
        AddIfEmpty(env, nbrs, pos.North());
        AddIfEmpty(env, nbrs, pos.Northeast());
        AddIfEmpty(env, nbrs, pos.East());
    }
    // ...
    // ...

    return nbrs;
}
```

GO ON TO THE NEXT PAGE.

(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
//
{
```

```
int row, col;
int temp1, temp2;
row = other.Row();
col = other.Col();

if((Row()-row) == 1 && (Col()-col) == 1)
    return("Northeast");
if((Row()-row == 0 && (Col() - col) = 1)
    return("North");

" "
. . .

}
```