



AP[®] Computer Science AB 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.

Copyright © 2002 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, SAT, and the acorn logo are registered trademarks of the College Entrance Examination Board. APIEL is a trademark owned by the College Entrance Examination Board. PSAT/NMSQT is a registered trademark jointly owned by the College Entrance Examination Board and the National Merit Scholarship Corporation. Educational Testing Service and ETS are registered trademarks of Educational Testing Service.

- (a) Modify the Environment member function AllFish to use the revised data structure. In writing AllFish, you may use any other Environment member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified.

Complete function AllFish below. Note the changes shown in bold.

```

apvector<Fish> Environment::AllFish() const
// postcondition: returned vector (call it fishList) contains all
//                fish in top-down, left-right order:
//                top-left fish in fishList[0],
//                bottom-right fish in fishList[fishList.length()-1];
//                # fish in environment is fishList.length()
{
    apvector<Fish> fishList(myFishCount);
    int r, k; // c from original not needed
    int count = 0;
    apstring s = "";
    ListNode * tempPtr;

    // look at all grid positions, store fish found in vector fishList

    // insert code here
    for (r = 0; r < myWorld.length(); r++)
    {
        for (tempPtr = myWorld[r]; tempPtr != NULL; tempPtr = tempPtr->next)
        {
            fishList[count] = tempPtr->theFish;
            count++;
        }
    }

    // end of inserted code

    for (k = 0; k < count; k++)
    {
        s += fishList[k].Location().ToString() + " ";
    }
    DebugPrint(5, "Fish vector = " + s);
    return fishList;
}

```

GO ON TO THE NEXT PAGE.

- (b) Modify the Environment member function AddFish to use the revised data structure. The new fish should be inserted into the correct row's linked list, maintaining the order of the list sorted by column index.

In writing AddFish, you may use any other Environment member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified.

Complete function AddFish below.

```

void Environment::AddFish(const Position & pos)
// precondition: no fish already at pos, i.e., IsEmpty(pos)
// postcondition: fish created at pos
{
    if (! IsEmpty(pos))
    {
        cerr << "error, attempt to create a fish at non-empty: "
              << pos << endl;
    }

    // insert code here
    myFishCount++;
    ListNode *temp = myWorld[pos.Row()];
    if (temp == NULL || temp->columnIndex > pos.Col())
    {
        myWorld[pos.Row()] = new ListNode(Fish(myFishCount, pos), pos.Col(), temp);
    }
    else
    {
        while ((temp->next != NULL) && (temp->next->columnIndex < pos.Col()))
        {
            temp = temp->next;
        }
        ListNode *aaa = new ListNode(Fish(myFishCount, pos), pos.Col(), temp->next);
        temp->next = aaa;
    }
    myFishCount++;
}

```

GO ON TO THE NEXT PAGE.

- (a) Modify the Environment member function AllFish to use the revised data structure. In writing AllFish, you may use any other Environment member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified.

Complete function AllFish below. Note the changes shown in bold.

```

apvector<Fish> Environment::AllFish() const
// postcondition: returned vector (call it fishList) contains all
//                fish in top-down, left-right order:
//                top-left fish in fishList[0],
//                bottom-right fish in fishList[fishList.length()-1];
//                # fish in environment is fishList.length()
{
    apvector<Fish> fishList(myFishCount);
    int r, k;    // c from original not needed
    int count = 0;
    apstring s = "";
    ListNode * tempPtr;

    // look at all grid positions, store fish found in vector fishList

    // insert code here
    for(r=0; r < myWorld.length(); r++){
        tempPtr = myWorld[r];
        while(tempPtr){
            fishList[count] = tempPtr->theFish;
            count++;
            tempPtr = tempPtr->next;
        }
    }
    // end of inserted code

    for (k = 0; k < count; k++)
    {
        s += fishList[k].Location().ToString() + " ";
    }
    DebugPrint(5, "Fish vector = " + s);
    return fishList;
}

```

GO ON TO THE NEXT PAGE.

- (b) Modify the Environment member function AddFish to use the revised data structure. The new fish should be inserted into the correct row's linked list, maintaining the order of the list sorted by column index.

In writing AddFish, you may use any other Environment member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified.

Complete function AddFish below.

```
void Environment::AddFish(const Position & pos)
// precondition: no fish already at pos, i.e., IsEmpty(pos)
// postcondition: fish created at pos
{
    if (!IsEmpty(pos))
    {
        cerr << "error, attempt to create a fish at non-empty: "
              << pos << endl;
    }

    // insert code here
    int col = pos.Col();
    myFishCreated++; myFishCount++;
    ListNode * temp = myWorld[pos.Row()], last = NULL;
    while (temp && temp->columnIndex < col) {
        last = temp;
        temp = temp->next;
    }
}
```

```
temp = new ListNode(Fish(myFishCreated, pos), col, temp);
temp = new ListNode(Fish(myFishCreated, pos), col, temp);
if (!last)
    myWorld[pos.Row()] = temp;
else
    last->next = temp;
```

```
}
```

GO ON TO THE NEXT PAGE.

- (a) Modify the Environment member function AllFish to use the revised data structure. In writing AllFish, you may use any other Environment member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified.

Complete function AllFish below. Note the changes shown in bold.

```

apvector<Fish> Environment::AllFish() const
// postcondition: returned vector (call it fishList) contains all
//                fish in top-down, left-right order:
//                top-left fish in fishList[0],
//                bottom-right fish in fishList[fishList.length()-1];
//                # fish in environment is fishList.length()
{
    apvector<Fish> fishList(myFishCount);
    int r, k;    // c from original not needed
    int count = 0;
    apstring s = "";
    ListNode * tempPtr;

    // look at all grid positions, store fish found in vector fishList

    // insert code here
    for (r=0; r < myWorld.length(); r++)
    {
        tempPtr = myWorld[r];
        while (tempPtr != NULL)
        {
            fishList[count] = tempPtr->theFish;
            count++;
            tempPtr = tempPtr->next;
        }
    }
    // end of inserted code

    for (k = 0; k < count; k++)
    {
        s += fishList[k].Location().ToString() + " ";
    }
    DebugPrint(5, "Fish vector = " + s);
    return fishList;
}

```

GO ON TO THE NEXT PAGE.

- (b) Modify the `Environment` member function `AddFish` to use the revised data structure. The new fish should be inserted into the correct row's linked list, maintaining the order of the list sorted by column index. In writing `AddFish`, you may use any other `Environment` member functions or the public member functions of any other class used in this case study. Assume that all member functions work as specified. Complete function `AddFish` below.

```
void Environment::AddFish(const Position & pos)
// precondition: no fish already at pos, i.e., IsEmpty(pos)
// postcondition: fish created at pos
{
    if (! IsEmpty(pos))
    {
        cerr << "error, attempt to create a fish at non-empty: "
              << pos << endl;
    }

    // insert code here

    myFishCreated++;

    int R = pos.Row(), C = pos.Col();
    ListNode *Temp = myWorld[R], *Last = Temp;
    while (Temp->columnIndex < C)
    {
        Last = Temp;
        Temp = Temp->next;
    }
    Last->next = new ListNode(Fish(myFishCreated, pos), C, Temp);
    myFishCount++;
}
```

GO ON TO THE NEXT PAGE.