



AP Computer Science AB 1999 Sample Student Responses

The materials included in these files are intended for non-commercial use by AP teachers for course and exam preparation; 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 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.

Copyright © 2001 by College Entrance Examination Board. All rights reserved. College Board, Advanced Placement Program, AP, and the acorn logo are registered trademarks of the College Entrance Examination Board.

- (a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor"; 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member "Taylor" was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

Club1

clubName: German

memberList →

Taylor	9
--------	---

 →

Roberts	10
---------	----

 →

Schwab	11
--------	----

 →

Shaw	9
------	---

 →

Hunt	10
------	----

 →

Rodger	10
--------	----

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition: anyClub contains zero or more members, name does not
//               appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//               has been added to anyClub
{
```

```
    Member* newMem;
```

```
    newMem = new Member(name, level, anyClub.memberList);
```

```
    anyClub.memberList = newMem;    // new head assigned.
```

```
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE

- (b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition: level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
{
    Member* list = anyClub.memberList;
    int sum = 0;

    while (list != NULL)
    {
        if (list->level == level)    // is member in the same grade?
        {
            sum++;
        }
        list = list->next;
    }

    return sum;
}
```

In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition: clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
{
    int numMembers, mostMembers = 0, k;

    for (k = 0; k < clubsArray.length(); k++)
    {
        numMembers = CountLevel (clubsArray[k], level);
        if (numMembers > mostMembers) // determine what the highest number of
        {                               // members are in the array of clubs.
            mostMembers = numMembers;
        }
    }

    for (k = 0; k < clubsArray.length(); k++)
    {
        if (CountLevel (clubsArray[k], level) == mostMembers) // using the highest
        {                                                         // number of members,
            cout << clubsArray[k].clubName << endl;           // find which clubs
        }                                                         // have this number
    }                                                         // of members.
}
}
```

- (a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member "Taylor" was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

Club1

`clubName: German`

`memberList` →

Taylor	9
--------	---

 →

Roberts	10
---------	----

 →

Schwab	11
--------	----

 →

Shaw	9
------	---

 →

Hunt	10
------	----

 →

Rodger	10
--------	----

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition: anyClub contains zero or more members, name does not
//               appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//               has been added to anyClub
```

```
    while (anyClub.memberList != NULL)
        anyClub.memberList = anyClub.memberList->next;
    if (anyClub.memberList == NULL) // I realize this if
                                    // statement isn't needed
    {
        anyClub.memberList.name = name;
        anyClub.memberList.level = level;
    }
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE

- (b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition: level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
//                of that level
```

```
{
    int sum = 0;
    while (anyClub.memberlist != NULL)
    {
        if (anyClub.memberlist.level == level)
            sum++;
        anyClub.memberlist = anyClub.memberlist->next;
    }
    return sum;
}
```

In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,  
                               int level)  
// precondition: clubsArray contains clubsArray.length() clubs  
// postcondition: prints the name of the club or clubs in clubsArray  
//                that contain the largest number of members in a given  
//                level in high school (9 - 12), one club per line.
```

```
{  
    int highest = 0, temp  
  
    for (int i = 0; i < clubsArray.length(); i++)  
    {  
        temp = countlevel(clubsArray[i], level)  
        if (temp > highest)  
            highest = temp;  
    }  
  
    for (i = 0; i < clubsArray.length(); i++)  
    {  
        temp = countlevel(clubsArray[i], level)  
        if (temp == highest)  
        {  
            cout << clubsArray[i].clubname << endl;  
        }  
    }  
}  
}
```

- (a) Write function `InsertMember`, as started below. `InsertMember` adds a student with the given name and respective level in high school to the given club.

For example, after the call `InsertMember("Taylor", 9, Club1)`, variable `Club1` might be as shown below. The diagram shows that the new member "Taylor" was inserted at the beginning of the list of members, but the student could have been inserted anywhere in the list.

Club1

clubName: German

memberList →

Taylor	9
--------	---

 →

Roberts	10
---------	----

 →

Schwab	11
--------	----

 →

Shaw	9
------	---

 →

Hunt	10
------	----

 →

Rodger	10
--------	----

Complete function `InsertMember` below. Assume that `InsertMember` is called only with parameters that satisfy its precondition.

```
void InsertMember(const apstring & name, int level, Club & anyClub)
// precondition: anyClub contains zero or more members, name does not
//               appear in anyClub, and level is 9, 10, 11, or 12
// postcondition: a new member with the given name and respective level
//               has been added to anyClub
```

```
newNode * member;
cin >> member, name, member, level;

member->next = memberList;
ptr * member = memberList->next;
}
```

Part (b) begins on page 14.

GO ON TO THE NEXT PAGE

- (b) Write function `CountLevel`, as started below. `CountLevel` counts and returns the number of club members of the specified level in `anyClub`.

For example, the call `CountLevel(Club1, 10)` returns 3, since there are 3 tenth graders in the German club. The call `CountLevel(Club2, 10)` returns 0 since there are no tenth graders in the Computer club.

Complete function `CountLevel` below. Assume that `CountLevel` is called only with parameters that satisfy its precondition.

```
int CountLevel(const Club & anyClub, int level)
// precondition: level is 9, 10, 11, or 12
// postcondition: returns the number of members in anyClub
// of that level
```

```
    member, L = anyClub.members[j];
    int counter counter = 0;
    while (L != null) {
        if (member.L == level)
            counter++;
        L++;
    }
    return counter;
}
```

In writing `PrintClubsWithMostInLevel`, you may call function `CountLevel` specified in part (b). Assume `CountLevel` works as specified, regardless of what you wrote in part (b).

Complete function `PrintClubsWithMostInLevel` below. Assume that `PrintClubsWithMostInLevel` is called only with parameters that satisfy its precondition.

```
void PrintClubsWithMostInLevel(const apvector<Club> & clubsArray,
                               int level)
// precondition: clubsArray contains clubsArray.length() clubs
// postcondition: prints the name of the club or clubs in clubsArray
//                that contain the largest number of members in a given
//                level in high school (9 - 12), one club per line.
```

```
    int k, j; int i = 0; i < clubsArray.length(); i++)
    for (i = 0; i < clubsArray.length(); i++)
        for (k = 0; k < clubsArray.length() - 1; k++)
            if (CountLevel(clubsArray[i], level) > CountLevel(clubsArray[k], level))
                int winner = k;
    return winner;
```

```
return winner;
```

W