AP Computer Science A
1999 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.

AI C

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function `ComputeGPA`, as started below. `ComputeGPA` should fill in the `GPA` data member for the first `numStudents` records in its `apvector` parameter `roster`. A student's GPA (grade point average) is computed by dividing `gradePoints` by `creditHours`. The GPA for a student with 0 credit hours should be set to 0.

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

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
{
  int i;
    for (i=0; i< numStudents ;i++)
    {
    if (roster[i].creditHours == 0)
    {
      roster[i].GPA = 0;
    }
    else
       roster[i].GPA= (roster[i].gradePoints/roster[i].creditHours);
    }
}
```

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
{
   if ((student.creditHours <= 125) || (student.GPA < 2.0))
      return false;
   return true;
}
```

Part (c) begins on page 6.

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

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

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

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
{
  int i;
  numSeniors = 0;
  for (i=0; i< numStudents; i++)
  {
    if (IsSenior(roster[i])
    {
      numSeniors++;
      seniors [numSeniors-1] = roster[i];
    }
  }
}
```

AIE

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function ComputeGPA, as started below. ComputeGPA should fill in the GPA data member for the first numStudents records in its apvector parameter roster. A student's GPA (grade point average) is computed by dividing gradePoints by creditHours. The GPA for a student with 0 credit hours should be set to 0.

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

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.

int lp;

for (lp=1; lp<=roster.length(); lp++)
{
   if (roster[lp].creditHours == 0)
   {
     roster[lp].GPA = 0.0;
   }
   else
   {
     roster[lp].GPA = roster[lp].gradePoints / roster[lp].creditHours
   }
}
```

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
```

```
if((student.creditHours >= 125) && (student.GPA >= 2.0))
{
  return true;
}
else
{
  return false;
}
```

Part (c) begins on page 6.

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

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

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

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
int lp;
numSeniors = 0;

for (lp = 1; lp <= roster.length(); lp++)
{
  if (IsSenior())
  {
    numSeniors += 1;
    strcpy(seniors[numSeniors].name, roster[lp].name);
    seniors[numSeniors].creditHours = roster[lp].creditHours;
    seniors[numSeniors].gradePoints = roster[lp].gradePoints;
    seniors[numSeniors].GPA = roster[lp].GPA;
  }
}
```

1. Assume that student records are implemented using the following declaration.

```
struct StudentInfo
{
   apstring name;
   int creditHours;
   double gradePoints;
   double GPA;
};
```

(a) Write function `ComputeGPA`, as started below. `ComputeGPA` should fill in the `GPA` data member for the first `numStudents` records in its `apvector` parameter `roster`. A student's GPA (grade point average) is computed by dividing `gradePoints` by `creditHours`. The GPA for a student with 0 credit hours should be set to 0.

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

```
void ComputeGPA(apvector<StudentInfo> & roster, int numStudents)
// precondition:  roster contains numStudents records,
//                0 < numStudents ≤ roster.length(), in which the
//                name, creditHours and gradePoints data members
//                have been initialized.
// postcondition: The GPA data member for the first numStudents records
//                in roster has been calculated.
{
  for (int a=0; a <roster.length(); a++)
  { cin >> StudentInfo.name[a];
    cin >> StudentInfo.creditHours[a];
    cin >> StudentInfo.gradePoints[a];

    if (StudentInfo.creditHours[a]==0)
        StudentInfo.GPA[a]= 0;
    else
        StudentInfo.GPA[a]= StudentInfo.creditHours[a]/StudentInfo.gradePoints[a];
  }
}
```

GO ON TO THE NEXT PAGE

(b) Write function `IsSenior`, as started below. `IsSenior` should return `true` if the given student has at least 125 credit hours and has a GPA of at least 2.0; otherwise, `IsSenior` should return `false`.

For example:

student				Result of the call `IsSenior(student)`
name	creditHours	gradePoints	GPA	
King	45	171	3.8	`false` (not enough credit hours)
Norton	128	448	3.5	`true`
Solo	125	350	2.8	`true`
Kramden	150	150	1.0	`false` (GPA too low)

Complete function `IsSenior` below.

```
bool IsSenior(const StudentInfo & student)
// postcondition: returns true if this student's credit hours ≥ 125
//                and GPA ≥ 2.0; otherwise, returns false
   {
    for (int a=0;a< roster.length();a++){
      if (StudentInfo.creditHours[a]>=125 && StudentInfo.GPA[a]>=2.0)
          return true;
      else
          return false;
    }
   }
```

Part (c) begins on page 6.

(c) Write function `FillSeniorList`, as started below. `FillSeniorList` determines which students in the array `roster` are seniors and copies those students' records to the array `seniors`. It should also set the value of parameter `numSeniors` to be the number of seniors in the array `seniors`.

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

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

```
void FillSeniorList(const apvector<StudentInfo> & roster,
                    int numStudents, apvector<StudentInfo> & seniors,
                    int & numSeniors)
// precondition: roster contains numStudents records,
//               0 < numStudents ≤ roster.length(),
//               and seniors is large enough to hold all of
//               the seniors' records
      {
      for (int a=0;a < roster.length(); a++)
        {
          IsSenior[a];
          if (IsSenior[a]=="true")
            { NumSeniors++;
              Seniors++;
            }
        }
```