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

```c
struct StudentInfo
{
    string 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.

```c
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:

<table>
<thead>
<tr>
<th>student</th>
<th>creditHours</th>
<th>gradePoints</th>
<th>GPA</th>
<th>Result of the call <code>isSenior(student)</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>45</td>
<td>171</td>
<td>3.8</td>
<td>false (not enough credit hours)</td>
</tr>
<tr>
<td>Norton</td>
<td>128</td>
<td>448</td>
<td>3.5</td>
<td>true</td>
</tr>
<tr>
<td>Solo</td>
<td>125</td>
<td>350</td>
<td>2.8</td>
<td>true</td>
</tr>
<tr>
<td>Kramden</td>
<td>150</td>
<td>150</td>
<td>1.0</td>
<td>false (GPA too low)</td>
</tr>
</tbody>
</table>

Complete function `isSenior` below.

```cpp
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.

```c
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, j;
    numSeniors = 0;
    for (i = 0; i < numStudents; i++)
    {
        if (IsSenior(roster[i]))
        {
            numSeniors++;
            seniors[numSeniors-1] = roster[i];
        }
    }
}
```

GO ON TO THE NEXT PAGE
1. Assume that student records are implemented using the following declaration.

```c
struct StudentInfo
{
    string 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.

```c
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:

<table>
<thead>
<tr>
<th>student</th>
<th>creditHours</th>
<th>gradePoints</th>
<th>GPA</th>
<th>Result of the call IsSenior(student)</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>45</td>
<td>171</td>
<td>3.8</td>
<td>false (not enough credit hours)</td>
</tr>
<tr>
<td>Norton</td>
<td>128</td>
<td>448</td>
<td>3.5</td>
<td>true</td>
</tr>
<tr>
<td>Solo</td>
<td>125</td>
<td>350</td>
<td>2.8</td>
<td>true</td>
</tr>
<tr>
<td>Kramden</td>
<td>150</td>
<td>150</td>
<td>1.0</td>
<td>false (GPA too low)</td>
</tr>
</tbody>
</table>

Complete function IsSenior below.

```cpp
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.

```cpp
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
    {
        string 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 >> roster[a].name;
            cin >> roster[a].creditHours;
            cin >> roster[a].gradePoints;
            if (roster[a].creditHours == 0)
                roster[a].GPA = 0;
            else
                roster[a].GPA = roster[a].gradePoints / roster[a].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:

<table>
<thead>
<tr>
<th>student</th>
<th>creditHours</th>
<th>gradePoints</th>
<th>GPA</th>
<th>Result of the call IsSenior(student)</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>45</td>
<td>171</td>
<td>3.8</td>
<td>false (not enough credit hours)</td>
</tr>
<tr>
<td>Norton</td>
<td>128</td>
<td>448</td>
<td>3.5</td>
<td>true</td>
</tr>
<tr>
<td>Solo</td>
<td>125</td>
<td>350</td>
<td>2.8</td>
<td>true</td>
</tr>
<tr>
<td>Kramden</td>
<td>150</td>
<td>150</td>
<td>1.0</td>
<td>false (GPA too low)</td>
</tr>
</tbody>
</table>

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 i = 0; i < roster.length(); i++)
if (StudentInfo.creditHours[i] ≥ 125 && StudentInfo.GPA[i] ≥ 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.

```c
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++;  
          }
      }  
  }
```

GO ON TO THE NEXT PAGE