AP® Computer Science A
2001 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.
(a) Write the free function `LessThan`, as started below. `LessThan` returns `true` if either

- `lowAge` of the first book is less than `lowAge` of the second book; or
- `lowAge` is the same for both books, and `highAge` of the first book is less than `highAge` of second book.

Otherwise, `LessThan` returns `false`.

For example:

<table>
<thead>
<tr>
<th>BookA</th>
<th>BookB</th>
<th>LessThan(BookA, BookB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowAge</td>
<td>highAge</td>
<td>lowAge</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>

Complete function `LessThan` below.

```cpp
bool LessThan(const Book & lhs, const Book & rhs)
// postcondition: returns true if lowAge of lhs < lowAge of rhs or
// if lowAge of lhs and rhs are equal
// and highAge of lhs < highAge of rhs;
// otherwise, returns false

return (lhs.lowAge < rhs.lowAge ||
        (lhs.lowAge == rhs.lowAge &&
         lhs.highAge < rhs.highAge));
```
Complete function InsertOne below.

```cpp
void BookList::InsertOne(const Book & bk)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan;
// bk is not already in this BookList
// postcondition: bk has been inserted into this BookList,
// maintaining its order by age range

if (myList.length() == myCount) // if there is no room left, double the amount of space
    myList.resize(myCount + 3);

int i = 0;
while (i < myCount && & LessThan(myList[i], bk))
    i++;

for (int k = myCount, k > i, k --)
    myList[k] = myList[k - 1];

myList[i] = bk;
myCount++;
```
Complete function `InsertMany` below.

```c++
void BookList::InsertMany(const avector<Book> & second)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan; second contains
// second.length() books in arbitrary order;
// none of the books in second are in this BookList
// postcondition: all the books from second have been inserted into
// this BookList, maintaining its order by age range
{
    for (int i = 0; i < second.length(); i++)
        InsertOne(second[i]);
```
(a) Write the free function `LessThan`, as started below. `LessThan` returns true if either

- `lowAge` of the first book is less than `lowAge` of the second book; or
- `lowAge` is the same for both books, and `highAge` of the first book is less than `highAge` of second book.

Otherwise, `LessThan` returns false.

For example:

<table>
<thead>
<tr>
<th>BookA</th>
<th>BookB</th>
<th>LessThan(BookA, BookB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Complete function `LessThan` below.

```cpp
bool LessThan(const Book & lhs, const Book & rhs) {
    // postcondition: returns true if lowAge of lhs < lowAge of rhs or
    // if lowAge of lhs and rhs are equal
    // and highAge of lhs < highAge of rhs;
    // otherwise, returns false

    return ((lhs.lowAge < rhs.lowAge) || (lhs.lowAge == rhs.lowAge) &&
            (lhs.highAge < rhs.highAge));
}
```
Complete function InsertOne below.

```cpp
void BookList::InsertOne(const Book & bk)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan;
// bk is not already in this BookList
// postcondition: bk has been inserted into this BookList,
// maintaining its order by age range

bool flag = true;
int i = 0;
int j;

while (flag == true)
{
    if (LessThan(myList[i], bk) == 1)
    {
        i++;
    }
    else
    {
        flag = false;  // i is the index where the book will be inserted.
        myList.resize(myCount + 1);  // resize the list
        myCount += 1;  // resize myCount
        for (j = myCount; j > i; j--)
        {
            myList[j] = myList[j-1];  // shift the list over
        }
        myList[i] = bk;  // add bk
    }
}
```
Complete function InsertMany below.

```cpp
void BookList::InsertMany(const apvector<Book> & second)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan; second contains
// second.length() books in arbitrary order;
// none of the books in second are in this BookList
// postcondition: all the books from second have been inserted into
// this BookList, maintaining its order by age range

    int i;

    for(i=0; i<second.length; i++)
    {
        InsertOne (second[i]);
    }
```
(a) Write the free function `LessThan`, as started below. `LessThan` returns `true` if either

- `lowAge` of the first book is less than `lowAge` of the second book; or
- `lowAge` is the same for both books, and `highAge` of the first book is less than `highAge` of second book.

Otherwise, `LessThan` returns `false`.

For example:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BookA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lowAge</td>
<td>highAge</td>
<td>lowAge</td>
<td>highAge</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
<td>14</td>
<td>true</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>true</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>10</td>
<td>15</td>
<td>true</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>8</td>
<td>15</td>
<td>false</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
<td>11</td>
<td>false</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>9</td>
<td>12</td>
<td>false</td>
</tr>
</tbody>
</table>

Complete function `LessThan` below.

```c++
bool LessThan(const Book & lhs, const Book & rhs)
// postcondition: returns true if lowAge of lhs < lowAge of rhs or
//                 if lowAge of lhs and rhs are equal
//                 and highAge of lhs < highAge of rhs;
//                 otherwise, returns false
{
    if (Book::lowAge(lhs, rhs))
        return true;
    if (Book::lowAge(lhs, rhs))
        if (Book::highAge(lhs, rhs))
            return true;
        return false;
}
```
Complete function InsertOne below.

```cpp
void BookList::InsertOne(const Book & bk)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan;
// bk is not already in this BookList
// postcondition: bk has been inserted into this BookList,
// maintaining its order by age range
```

```cpp
int x;

myList.Resize(myList.Length() + 1)
for (int i = 0; i < myList.Length() - 1; i++)
  if (LessThan(i, i + 1))
    myList[i] = bk;
  Break;
for (x = 0; x < myList.Length(); x++)
  myList[x] = myList[x + 1];
for (x = 0; x > 0; x++)
  myList[x] = myList[x - 1];
```
Complete function InsertMany below.

```cpp
void BookList::InsertMany(const apvector<Book> & second)
// precondition: this BookList is in sorted order by age range
// as defined by LessThan; second contains
// second.length() books in arbitrary order;
// none of the books in second are in this BookList
// postcondition: all the books from second have been inserted into
// this BookList, maintaining its order by age range

for (int i = 0; i < second.length(); i++)
    InsertOne(second[i]);
```