AP® Computer Science A
2001 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.

(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:

BookA		BookB		LessThan(BookA, BookB)
lowAge	highAge	lowAge	highAge	
9	12	9	14	true
9	12	10	11	true
9	12	10	15	true
9	12	8	15	false
9	12	9	11	false
9	12	9	12	false

Complete function `LessThan` below.

```
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.

```
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*2);
  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.

```
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]);
}
```

(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:

BookA		BookB		LessThan(BookA, BookB)
lowAge	highAge	lowAge	highAge	
9	12	9	14	true
9	12	10	11	true
9	12	10	15	true
9	12	8	15	false
9	12	9	11	false
9	12	9	12	false

Complete function `LessThan` below.

```
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 ((BookA.lowAge < BookB.lowAge) || ((BookA.lowAge == BookB.lowAge) &&
                                              (BookA.highAge < BookB.highAge));
}
```

Complete function InsertOne below.

```
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 = 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.

```
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:

BookA		BookB		LessThan(BookA, BookB)
lowAge	highAge	lowAge	highAge	
9	12	9	14	true
9	12	10	11	true
9	12	10	15	true
9	12	8	15	false
9	12	9	11	false
9	12	9	12	false

Complete function `LessThan` below.

```
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.

```
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
```

```
{ 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=i; x<Mylist.length(); x++)
      myList[x] = myList[x+1];

  for (x=i; x>0; x++)
      myList[x] = myList[x-1];

}
```

Complete function InsertMany below.

```
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 < books.length(); i++)
        InsertOne(Book[i]);
}
```