```
// Header file ListP.h for the ADT list.
typedef  desired-type-of-list-item  ListItemType;
```

```
class List   {  // List accessed By-Position
public:
// constructors and destructor:
  List();                          // default
  List(const List& aList);         // copy
  operator=(const List& rhs);      // assignment op=
  ~List();                         // destructor

// list operations:
  bool isEmpty() const;
  int  getLength() const;

  bool insert(int index, ListItemType& NewItem);

  bool remove(int index);

  bool retrieve(int index, ListItemType& DataItem);




private:
   struct ListNode {
      ListItemType  item;
      ListNode     *next;
   };

   int       size;
   ListNode *head;
   ListNode *ptrTo(int index) const;
};
```

```
// Header file SortedListP.h for the ADT list.
typedef  desired-type-of-list-item  ListItemType;
```

```
class SortedList   {  // List sorted by items
public:
// constructors and destructor:
  SortedList();
  SortedList(const SortedList& aList);
  operator=(const SortedList& rhs);
  ~SortedList();

// list operations:
  bool isEmpty() const;
  int  getLength() const;

  bool insert(ListItemType& NewItem); // by value

  bool remove(int index);

  bool retrieve(int index, ListItemType& DataItem);

  // Find position of anItem in the list
  // Return -1 if item not in list
  int  find(ListItemType& anItem) const;


private:
   struct ListNode {
      ListItemType  item;
      ListNode     *next;
   };

   int       size;
   ListNode *head;
   ListNode *ptrTo(int index) const;
};
```

```
// Iterative SortedList Insert

bool SortedList::insert(ListItemType& newItem)
{

      ListNode *prev = NULL;
      ListNode *cur  = head;




      while((cur != NULL) &&
            (newItem > cur->item))
      {
            prev = cur;
            cur  = cur->next;
      }

      ListNode *newPtr = new ListNode;
      newPtr->item  =  newItem;

      newPtr->next  =  cur;

      if(prev == NULL)
            head = newPtr;
      else
            prev->next = newPtr;
}
```

```
// Recursive SortedList Insert

bool SortedList::insert(ListItemType& newItem)
{
      SLInsert( head, newItem );
}
```

```
// Private method
void SLInsert(ListNode*     &headPtr,
              ListItemType  newItem)
{
      if((headPtr != NULL) &&
         (newItem > headPtr->item))
      {
            SLInsert( headPtr->next, newItem );
      }
      else {
            ListNode *newPtr = new ListNode;
            newPtr->item  = newItem;

            newPtr->next  = headPtr;

            headPtr  = newPtr;
      }
}
```