CSE225L – Data Structures and Algorithms Lab
Lab 05
Sorted List (array based)

In today's lab we will design and implement the List ADT where the items in the list are sorted.

Header file:

```
#ifndef SORTEDLIST_H_INCLUDED
#define SORTEDLIST_H_INCLUDED

const int MAX_ITEMS = 7;

template <class ItemType> class SortedList
{
public :
    SortedList();
    void MakeEmpty();
    bool IsFull();
    int LengthIs();
    bool InsertItem(ItemType);
    bool DeleteItem(ItemType);
    bool RetrieveItem(ItemType&);
    void ResetList();
    bool GetNextItem(ItemType&);
private:
    int length;
    ItemType info[MAX_ITEMS];
    int currentPos;
};
#include "SortedList.tpp"
#endif // SORTEDLIST_H_INCLUDED
```

Implementation file:

```
#include "SortedList.h"
#include <iostream>

using namespace std;
template <class ItemType>
SortedList<ItemType>::SortedList()
{
    length = 0;
    currentPos = - 1;
}

template <class ItemType> void SortedList<ItemType>::MakeEmpty()
{
    length = 0;
}

template <class ItemType>
bool SortedList<ItemType>::IsFull()
{
    return (length == MAX_ITEMS);
}

template <class ItemType> int SortedList<ItemType>::LengthIs()
{
    return length;
}

template <class ItemType> void SortedList<ItemType>::ResetList()
{
    currentPos = - 1;
}
```

```
template <class ItemType>
bool SortedList<ItemType>::GetNextItem(ItemType& item)
{
    if(currentPos<length-1)
    {
        currentPos++;
        item = info [currentPos];
        return true;
    }
    return false;
}

template <class ItemType>
bool SortedList<ItemType>::InsertItem(ItemType item)
{
    if(IsFull())
        return false;

    int location = 0;

    while (location < length)
    {
        if(item >= info[location])
        {
            location++;
        }
        else if(item < info[location])
            break;
    }

    for (int index = length; index > location; index--)
        info[index] = info[index - 1];

    info[location] = item;
    length++;

    return true;
}

template <class ItemType>
bool SortedList<ItemType>::DeleteItem(ItemType item)
{
    int location = 0;
    bool deleted = false;

    while (location < length){

        if(item == info[location]){
            deleted = true;
            break;
        }
        location++;
    }

    if(!deleted)
        return deleted;

    for (int index = location + 1; index < length; index++)
        info[index - 1] = info[index];

    length--;
    return deleted;
}
```

```
template <class ItemType>
bool SortedList<ItemType>::RetrieveItem(ItemType& item)
{
    int midPoint, first = 0, last = length - 1;
    bool found = false;

    while ((first <= last) && !found)
    {
        midPoint = (first + last) / 2;
        if(item < info[midPoint])
        {
            last = midPoint - 1;
        }
        else if(item > info[midPoint])
        {
            first = midPoint + 1;
        }
        else
        {
            found = true;
            item = info[midPoint];
        }
    }
    return found;
}
```

Task:

Generate the **driver file (Main.cpp)** where you perform the following tasks. Note that you cannot make any change to the header file or the source file.

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a list of integers		
• Print length of the list		0
• Insert five items	5 7 4 2 1	
• Print the list		1 2 4 5 7
• Retrieve 6 and print whether found		Item is not found
• Retrieve 5 and print whether found		Item is found
• Print if the list is full or not		List is full
• Delete 1		
• Print the list		2 4 5 7
• Print if the list is full or not		List is not full
• Delete 2, 4, 5, 7		
• Print the list		Should print nothing
• Keep inserting 5 until the list is full (you cannot use IsFull() method)	6	
• Print the list		6 6 6 6 6