**CSE225L – Data Structures and Algorithms Lab**
**Lab 11**
**Sorted List (linked list based)**

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

**sortedtype.h**

```
#ifndef SORTEDTYPE_H_INCLUDED
#define SORTEDTYPE_H_INCLUDED

template <class ItemType>
class SortedType
{
    struct NodeType
    {
        ItemType info;
        NodeType* next;
    };
    public:
        SortedType();
        ~SortedType();
        bool IsFull();
        int LengthIs();
        void MakeEmpty();
        void RetrieveItem(ItemType&,
bool&);
        void InsertItem(ItemType);
        void DeleteItem(ItemType);
        void ResetList();
        void GetNextItem(ItemType&);
    private:
        NodeType* listData;
        int length;
        NodeType* currentPos;
};

#endif // SORTEDTYPE_H_INCLUDED
```

**sortedtype.cpp**

```
#include "sortedtype.h"
#include <iostream>
using namespace std;

template <class ItemType>
SortedType<ItemType>::SortedType()
{
    length = 0;
    listData = NULL;
    currentPos = NULL;
}
template <class ItemType>
int SortedType<ItemType>::LengthIs()
{
    return length;
}
template<class ItemType>
bool SortedType<ItemType>::IsFull()
{
    NodeType* location;
    try
    {
        location = new NodeType;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType item)
{
    NodeType* newNode;
    NodeType* predLoc;
    NodeType* location;
    bool moreToSearch;

    location = listData;
    predLoc = NULL;
    moreToSearch = (location != NULL);
    while (moreToSearch)
    {
        if (location->info < item)
        {
            predLoc = location;
            location = location->next;
            moreToSearch = (location != NULL);
        }
        else moreToSearch = false;
    }
    newNode = new NodeType;
    newNode->info = item;

    if (predLoc == NULL)
    {
        newNode->next = listData;
        listData = newNode;
    }
    else
    {
        newNode->next = location;
        predLoc->next = newNode;
    }
    length++;
}
template <class ItemType>
void SortedType<ItemType>::DeleteItem(ItemType item)
{
    NodeType* location = listData;
    NodeType* tempLocation;
    if (item == listData->info)
    {
        tempLocation = location;
        listData = listData->next;
    }
    else
    {
        while (!(item==(location->next)->info))
            location = location->next;
        tempLocation = location->next;
        location->next = (location->next)->next;
    }
    delete tempLocation;
    length--;
}
```

```
template <class ItemType>
void
SortedType<ItemType>::RetrieveItem(ItemType
& item, bool& found)
{
    NodeType* location = listData;
    bool moreToSearch = (location != NULL);
    found = false;
    while (moreToSearch && !found)
    {
        if (item == location->info)
            found = true;
        else if (item > location->info)
        {
            location = location->next;
            moreToSearch = (location !=
NULL);
        }
        else
            moreToSearch = false;
    }
}
template <class ItemType>
void SortedType<ItemType>::MakeEmpty()
{
    NodeType* tempPtr;
    while (listData != NULL)
    {
        tempPtr = listData;
        listData = listData->next;
        delete tempPtr;
    }
    length = 0;
}
```

```
template <class ItemType>
SortedType<ItemType>::~SortedType()
{
    MakeEmpty();
}
template <class ItemType>
void SortedType<ItemType>::ResetList()
{
  currentPos = NULL;
}

template <class ItemType>
void
SortedType<ItemType>::GetNextItem(ItemType
& item)
{
    if (currentPos == NULL)
        currentPos = listData;
    else
        currentPos = currentPos->next;
    item = currentPos->info;
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

| **Operation to Be Tested and Description of Action** | **Input Values** | **Expected Output** |
|---|---|---|
| • Create a list | | |
| • Print Length | | 0 |
| • Insert five items and print | 5 7 4 2 1 | 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 not full |
| • Delete 1 and print | | 2 4 5 7 |
| • Print if the list is full or not | | List is not full |
| • Print Length | | 4 |