North South University

Department of Electrical and Computer Engineering

CSE 225L.13 (Data Structures and Algorithms Lab)

Lab 15: Sorted Lists (Linked List Based)

Instructor: Syed Shahir Ahmed Rakin, Arfana Rahman

Objective:

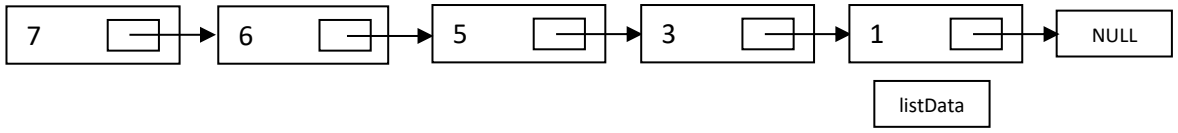
- Learn how the Sorted Lists work when made with linked lists.

Sorted Lists:

A sorted list is an abstract data structure where the values are given in a sorted manner; here, we are using an array to make a sorted list. The elements are inserted in this order:

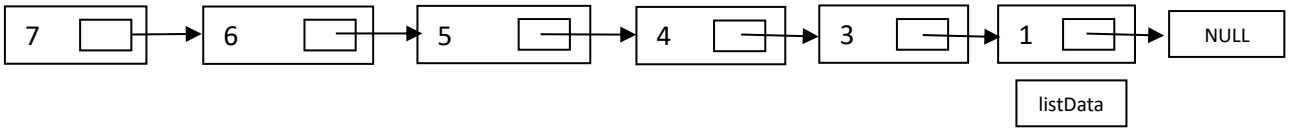
5	3	7	6	1

Then, the elements will be readjusted based on the size of the elements of the array, and the new list will be as follows:



Here, you can see that a linked list represents the sorted list; these items, upon insertion, get sorted based on the size of the elements provided. The best thing is that the memory is the limit, not the limit of 5 back in the days of arrays.

Now, when you insert an element such as 4, the sorted linked list would be represented as follows:



Prototype of Sorted Lists:

The header and source file of the Sorted Linked Lists are given as follows.

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>::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>
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>
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;
}
```

Tasks:

Generate the ***Driver file (main.cpp)*** and check your program with the following outputs

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Write a class ***timeStamp*** that represents a time of the day. It must have variables to store the number of seconds, minutes, and hours passed. It also must have a function to print all the values. You will also need to overload a few operators.		
• Create a list of objects of class ***timeStamp.***		
• Insert five-time values in the format ssmmhh (seconds, minutes, hours)	15 34 23 13 13 02 43 45 12 25 36 17 52 02 20	
• Delete the timestamp 25 36 17		
• Print the list		13:13:02 15:34:23 43:45:12 52:02:20