North South University

Department of Electrical and Computer Engineering

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

Lab 14: Unsorted Lists (Linked List Based)

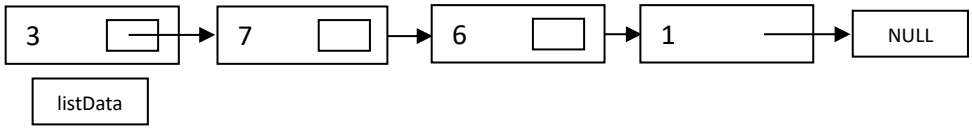
Instructor: Syed Shahir Ahmed Rakin, Arfana Rahman

Objective:

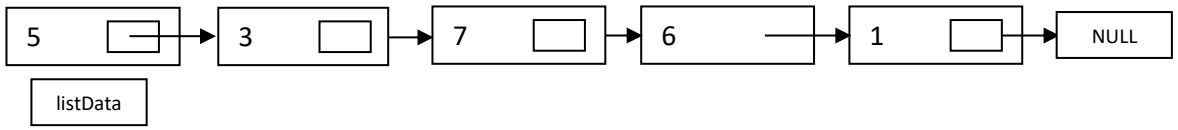
- Learn how the Unsorted Lists work when made with Linked Lists.

Unsorted Lists:

An unsorted list is an abstract data structure where the values are given in an unsorted manner, here, we are using a linked list to make an unsorted list. An example of an unsorted list is given as follows:



Now, let us insert '5' into the unsorted list, which will look like this.



Here, you can see that a linked list represents the unsorted list, and you can delete and retrieve any item as you see fit. The best thing is that the memory is the limit, not the limit of 5 back in the days of arrays.

Prototype of Unsorted Lists:

The header and source file of the Unsorted Linked List is given as follows.

unsortedtype.h

```
#ifndef UNSORTEDTYPE_H_INCLUDED
#define UNSORTEDTYPE_H_INCLUDED
template <class ItemType>
class UnsortedType
{
    struct NodeType
    {
        ItemType info; NodeType* next;
    };
public:
    UnsortedType(); ~UnsortedType();
    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 // UNSORTEDTYPE_H_INCLUDED
```

unsortedtype.cpp

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

template <class ItemType>
UnsortedType<ItemType>::UnsortedType()
{
    length = 0;
    listData = NULL;
    currentPos = NULL;
}

template <class ItemType>
UnsortedType<ItemType>::~UnsortedType()
{
    MakeEmpty();
}

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

template<class ItemType>
bool UnsortedType<ItemType>::IsFull()
{
    NodeType* location;
    try
    {
        location = new NodeType;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}

template <class ItemType>
void
UnsortedType<ItemType>::InsertItem(ItemT
ype item)
{
    NodeType* location;
    location = new NodeType;
    location->info = item;
    location->next = listData;
    listData = location;
    length++;
}

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

```
template <class ItemType>
void UnsortedType<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    UnsortedType<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
        {
            location = location->next;
            moreToSearch = (location != NULL);
        }
    }
}
template <class ItemType>
void UnsortedType<ItemType>::MakeEmpty()
{
    NodeType* tempPtr;
    while (listData != NULL)
    {
        tempPtr = listData;
        listData = listData->next;
        delete tempPtr;
    }
    length = 0;
}

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

Tasks:

Generate the **driver file (main.cpp)** where you perform the following tasks.

Operation to Be Tested and Description of Action
▪ You are given two sequences of integers arranged in ascending order. Your task is to combine the sequences into one ascending sequence. You can safely assume that no integer will be repeated. Input starts with a positive integer **m**, which specifies the number of elements in the first sequence. Next **m** values are the elements in the first sequence. The next positive integer **n** specifies the number of elements in the second sequence. Next, **n** values are the elements in the second sequence. The output is the combined sequence.

Input Values
10 1 5 6 10 14 20 25 31 38 40 12 2 4 7 9 16 19 23 24 32 35 36 42

Expected Output
1 2 4 5 6 7 9 10 14 16 19 20 23 24 25 31 32 35 36 38 40 42