CSE225L – Data Structures and Algorithms Lab
Lab 06
Unsorted List (Linked-list based)

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

unsortedtype.h

```
#ifndef UNSORTEDTYPE_H
#define UNSORTEDTYPE_H

template <class T>
class UnsortedType
{
private:
    struct Node
    {
        T data;
        Node* next;
    };
    Node* head;
    Node* pointTo;
    int size;
public:
    UnsortedType();
    ~UnsortedType();
    int Length();
    void Insert(T value);
    void Search(T value, bool &found);
    void Delete(T value);
    void MakeEmpty();
    void GetNext(T &value);
    void Reset();
};
#endif // UNSORTEDTYPE_H
```

unsortedtype.cpp

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

template <class T>
UnsortedType<T>::UnsortedType()
{
    head = NULL;
    pointTo = NULL;
    size = 0;
}

template <class T>
int UnsortedType<T>::Length()
{
    return size;
}
```

```cpp
template <class T>
void UnsortedType<T>::Insert(T value)
{
    Node* temp = new Node;
    temp->data = value;
    temp->next = head;
    head = temp;
    size++;
}

template <class T>
void UnsortedType<T>::Search(T value, bool &found)
{
    found = false;

    Node* i = head;

    while(i != NULL)
    {
        if (value == i->data)
        {
            found  = true;
            break;
        }
        else
        {
            i = i->next;
        }
    }
}

template <class T>
void UnsortedType<T>::Delete(T value)
{
    Node* i = head;
    Node* prev = NULL;
    bool found = false;

    while(i != NULL)
    {
        if (value == i->data)
        {
            found = true;
            break;
        }
        else
        {
            prev = i;
            i = i->next;
        }
    }

    if (found)
    {
        if (prev == NULL) // first node / no previous nodes
            head = i->next;
        else
            prev->next = i->next;
        delete i;
        size--;
    }
}
```

```
template <class T>
void UnsortedType<T>::MakeEmpty()
{
    Node* i = head;
    Node* nextNode;

    while (i != NULL)
    {
        nextNode = i->next; // Store the next node
        delete i;           // Delete the current node
        i = nextNode;       // Move to the next node
    }

    head = NULL;
    size = 0;
}

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

template <class T>
void UnsortedType<T>::GetNext(T &value)
{
    if (pointTo == NULL)
    {
        pointTo = head;
        value = pointTo->data;
    }
    else
    {
        value = pointTo->data;
    }
    pointTo = pointTo->next;
}

template <class T>
void UnsortedType<T>::Reset()
{
    pointTo = NULL;
}
```

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		
Insert four items	5 7 6 9	
Print the list		9 6 7 5
Print the length of the list		4
Insert one item	1	
Print the list		1 9 6 7 5
Search 4 and print whether found or not		Item is not found
Search 5 and print whether found or not		Item is found
Search 9 and print whether found or not		Item is found
Search 10 and print whether found or not		Item is not found
Delete 5		
Print the list		1 9 6 7
Delete 1		
Print the list		9 6 7
Delete 6		
Print the list		9 7

Operation to Be Tested and Description of Action	**Input Values**
You have two lists of numbers. Your job is to merge them into one list that is sorted in ascending order. You need to do this in **O(N)** time, and you must ensure that there are no duplicate numbers.	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