CSE225L – Data Structures and Algorithms Lab
Lab 10
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_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>
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(ItemType
item)
{
    NodeType* location;
    location = new NodeType;
    location->info = item;
    location->next = listData;
    listData = location;
    length++;
}
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>
UnsortedType<ItemType>::~UnsortedType()
{
    MakeEmpty();
}
```

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

Now generate the **Driver file (main.cpp)** where you perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a list		
• Insert four items and print the list	5, 7, 6, 9	5 7 6 9
• Print the length of the list		4
• Insert one item and print the list	1	5 7 6 9 1
• Retrieve 4 and print whether found or not		Item is not found
• Retrieve 5 and print whether found or not		Item is found
• Retrieve 9 and print whether found or not		Item is found
• Retrieve 10 and print whether found or not		Item is not found
• Print if the list is full or not		List is not full
• Delete 5 and then print if the list is full or not		List is not full
• Delete 1 and print the list		7 6 9
• Delete 6 and print the list		7 9