North South University
CSE-225L(Data Structures & Algorithm)
Fall - 2018
Lab-04 (Sorted List – Array Based)

Class ‘ItemType’:

itemtype.h

```
#ifndef ITEMTYPE_H_INCLUDED
#define ITEMTYPE_H_INCLUDED
#include <iostream>
#include <string>
#include <stdio.h>
using namespace std;

const int MAX_ITEMS = 10;

enum RelationType {LESS,EQUAL,GREATER};

class ItemType
{
    public:
        ItemType();
        RelationType ComparedTo(ItemType);
        void Initialize(int,string);
        int getValue();
        string getName();
    private:
        int value;
        string name;
};
#endif
```

itemtype.cpp

```
#include "itemtype.h"

ItemType::ItemType()
{
    value = 0;
}

RelationType ItemType::ComparedTo(ItemType otherItem)
{
    if (value < otherItem.value)
        return LESS; // this item is smaller
    else if (value > otherItem.value)
        return GREATER; // this item is greater
    else
        return EQUAL;
}
```

```
void ItemType::Initialize(int v,string n)
{
    value = v;
    name = n;
}

int ItemType::getValue()
{
    return value;
}

string ItemType::getName()
{
    return name;
}
```

Class 'SortedType':

sortedtype.h

```
#ifndef SORTEDTYPE_H_INCLUDED
#define SORTEDTYPE_H_INCLUDED
#include "itemtype.h"

class SortedType
{
     public :
        SortedType();
        void InsertItem(ItemType);
        bool SearchItem(ItemType);
        void DeleteItem(ItemType);
        ItemType GetNextItem();
        int LengthIs();
        bool IsFull();
        bool IsEmpty();
        void ResetList();
        void MakeEmpty();

    private:
        int length;
        ItemType info[MAX_ITEMS];
        int currentPos;
};

#endif // SORTEDTYPE_H_INCLUDED
```

sortedtype.cpp

```
#include "sortedtype.h"


SortedType::SortedType()
{
  length = 0;
  currentPos = -1;
}


void SortedType::InsertItem(ItemType item)
{
    int location = 0;

    bool locationInRange = (location<length);
    bool positionFound = false;

    while((locationInRange) && (!positionFound))
    {
        switch(item.ComparedTo(info[location]))
        {
         case GREATER:
            location++;
            locationInRange = (location<length);
            break;

         case LESS:
            positionFound = true;
            break;
        }
    }

    for(int index=length;index>location;index--)
    {
        info[index] = info[index-1]; // shifting items to right
    }

    info[location] = item;
    length++;

}
```

```
bool SortedType::SearchItem(ItemType item)
{

    bool found = false;

    for(int index = 0;index<length;index++)
    {
        if(info[index].ComparedTo(item)==EQUAL)
        {
            found = true;
            break;
        }
    }

    return found;
}

void SortedType::DeleteItem(ItemType item)
{

    if(SearchItem(item)==true)
    {
        int location = 0;

        while (item.ComparedTo(info[location]) !=  EQUAL)
        {
            location++;
        }
        info[location] = info[length - 1];
        length--;
    }
    else
    {
        cout<<"Item not in the list"<<endl;
    }

}

ItemType SortedType::GetNextItem()
{
    currentPos++;
    return info[currentPos];
}

int SortedType::LengthIs()
{
    return length;
}
```

```
bool SortedType::IsFull()
{
    return (length == MAX_ITEMS);
}

bool SortedType::IsEmpty()
{
    return (length == 0);
}

void SortedType::ResetList()
{
    currentPos = -1;
}

void SortedType::MakeEmpty()
{
    length = 0;
}
```