North South University
CSE-225.1L (Spring-2018)
Lab-01 (Objects & Classes in C++)

Course Details:

- **Course:** CSE-225 Lab (Data Structures and Algorithms)
- **Section:** 01
- **Time-slot:** ST 08:00 AM : 09:30 AM
- **Instructor:**

  Mir Tahsin Imtiaz,
  email : tahsin.nsu30@gmail.com
  cell-phone no. : 01751212910
- **Facebook Group:**
  - **Name:** CSE225L Sec 1 SFM1 Spring 18
  - **Link:** https://www.facebook.com/groups/1636531979800983/

Pre-requisites:

- CSE-115
- CSE-215

Class and Course Policy:

- Each lab class will carry attendance mark.
- Starting from the third lab class and onwards, there will be **graded practice in each class**.
- **Make-up policy**:
  - **Make-up exam due to medical reason**: You must take permission from the corresponding theory course faculty by writing an application for sitting for the makeup lab exam along with a set of copy of your valid medical documents.
  - **Make-up exam due to emergency/ personal/ family reasons:** You must take permission from the corresponding theory course faculty by writing an application (explaining the situation) for sitting for the makeup lab exam.
  - No make-up for ‘lab practice’

- **Tentative Percentage Breakdown:**
  - Attendance: **10%**
  - Lab-evaluation: **20%**
  - Midterm: **30%**
  - Lab Final Exam/Project: **40%**

‘Academic Honesty’ policies:

- Honest academic behavior will be of utmost importance.
- Any form of **dishonest academic behaviour** (copying of source codes, cheating during exams/ lab-evaluations) **will be very harshly dealt**.
- In both the cases of lab practices and lab exams, **the person copying** and **the person letting copy his/ her code**, will be **awarded zero as their lab practice/ exam score** during that class/ exam. Suspiciously similar code structure/ variable names/ solving techniques will be considered ‘copy’ works.

How to write a class in C++:

In C++, the following is the general format for a class declaration and definition:

```
        class class-name{

                private data variables and functions

        access-modifiers:
                respective data and functions

        access-modifiers:
                respective data and functions

};
```

Here, access-modifiers can be: public/ private/ protected (just like in JAVA). **By default**, functions and data declared within a C++ class are private to that class.

Suppose, **in JAVA**, you have written the following class named **DynamicArray-**

```
        public class DynamicArray{

        private int[] data;

        public DynamicArray(int size)
        {
                data = new int[size];
        }

        public void insertItem(int index, int item)
        {
                data[index] = item;
        }

        public int getItem(int index)
        {
                return data[index];
        }

        }
```

Now, in the main method, you create an object of that above class like this:

```
        public static void main(String[] args)
        {

          //create a dynamic array object with
         //size = 10
          DynamicArray d = new DynamicArray(10);

        // calling the JAVA garbage collector to free the
        // allocated memories
        System.gc();

        }
```

Now, if you convert the above JAVA class into a C++ class, it'll consist of the following different parts:

- The first part is the 'header' file **(**with the file extension **.h)** which will contain only the declarations of all the class variables and class functions, no implementation here.

<u>dynamicarray.h</u>

```
#ifndef DYNAMICARRAY_H_INCLUDED
#define DYNAMICARRAY_H_INCLUDED

class DynamicArray{

private:
        int* data;

public:
        DynamicArray(int);
        ~DynamicArray();
        void insertItem(int, int);
        int getItem(int);
};

#endif
```

- The second part is the cpp file **(**with the file extension **.cpp)** which will contain only the definitions of all the class variables and class functions 'declared' in the previous class header file. You **MUST** have to **include** the header file inside this cpp file.

<u>dynamicarray.cpp</u>

```
#include "dynamicarray.h"

DynamicArray::DynamicArray(int size)
{
        data = new int[size];
}
void DynamicArray::insertItem(int index, int item)
{
        data[index] = item;
}

int DynamicArray::getItem(int index)
{
        return data[index];
}

DynamicArray::~DynamicArray()
{
        delete[] data;
}
```

Now, in the main c++ file (also sometimes called the **driver file**) named **main.cpp**, you create and manipulate a DynamicArray class object as described below:

<u>main.cpp</u>

```
#include “dynamicarray.cpp”
#include <iostream>
using namespace std;

int main()
{
        // Prompting the user to enter the size of the array
        cout<<“Enter the size of the array: ”<<endl;
        int size;

        // Taking the input from the user and assigning that value to the int variable named size
        cin>>size;


        // Creating the DynamicArray class object with the specified size
        DynamicArray d(size);


        // Taking 10 inputs from the user and saving them inside the DynamicArray object created
        // above

        int temp;

        for(int i=0;i<size;i++)
        {
                cout<< “Enter value to be inserted at index = ”<<i<<endl;
                cin>>temp;
                d.insertItem(i, temp);
        }

        // Printing all the integer values saved in the DynamicArray class object


        cout<< “The values stored are: ”;

        int temp2;

        for(int i=0;i<size;i++)
        {
                temp2 = d.getItem(i);
                cout<< “Index = ”<<i; cout<< “, Value = ”<<temp2<<endl;

        }

        return 0;
}
```

Home Assignment (Submit handwritten hardcopy on the next class):

Write down in point form all the steps required for creating and adding the **header** and **cpp** files to an already created CodeBlocks project as demonstrated during the Lab-01 class to avoid the ‘precompiled header’ dilemma.

Hint: Remember how the **dynamicarray.h** and **dynamicarray.cpp** files were manually created as text files, then extensions were changed to **.h** and **.cpp** extensions and then how they were added to the project.

North South University
CSE-225.1L (Spring-2018)
Lab-02 (Template Classes in C++)

What is 'Template Class' in C++:

"Template Class" is an important feature of C++ which enables the coder to write **generic** functions or classes. In a **generic function or class**, the type of data (i.e: int, float, double, etc.) upon which the function or class operates is specified as a parameter.

Why 'Template Class'?

By creating a templated class/ function, you can define the nature of your algorithm to be independent of any kind of data types.

Once you have written a templated code, your compiler will automatically generate the correct code for the type of data that is actually used when you execute the function.

Format for writing a 'Template Class' in C++

Remember the simple **DynamicArray** class we discussed in our **Lab-01** where we created a simple C++ class to create a dynamically allocated array for only holding integer type of values. If we convert that simple class into a templated class, then that class object will be able to hold any valid type of numeric values (int, float, double). Now, the format for writing a template function in C++ (in the source .cpp file) is as follows:

```
template <class ItemType>
return-type Class_Name<ItemType>::functionName(parameters)
{
        // your code goes here
}
```

Now, if we convert the header file of that DynamicArray class to a templated version, it will be like as given below:

dynamicarray.h

```
#ifndef DYNAMICARRAY_H_INCLUDED
#define DYNAMICARRAY_H_INCLUDED

template <class ItemType>
class DynamicArray{

private:
        ItemType* data;

public:
        DynamicArray(int);
        ~DynamicArray();
        void insertItem(int, ItemType);
        ItemType getItem(int);
};

#endif
```

If we convert the cpp file of that DynamicArray class to a templated version, it will be like as given below:

dynamicarray.cpp

```
#include "dynamicarray.h"
```

```
template <class ItemType>
DynamicArray<ItemType>::DynamicArray(int size)
{
        data = new ItemType[size];
}

template <class ItemType>
void DynamicArray<ItemType>::insertItem(int index, ItemType item)
{
        data[index] = item;
}

template <class ItemType>
ItemType DynamicArray<ItemType>::getItem(int index)
{
        return data[index];
}

template <class ItemType>
DynamicArray<ItemType>::~DynamicArray()
{
        delete[] data;
}
```

<u>Creating and using template class objects in the driver (main.cpp) file</u>:

<u>main.cpp</u>

```
#include “dynamicarray.cpp”
#include <iostream>
using namespace std;

int main()
{

        int defaultSize = 3;

// Creating and using a DynamicArray object
// dealing with integer type of data

DynamicArray<int> intArray(defaultSize);

for (int index=0,data=10;index<3; index++, data += 10)
{
        intArray.insertItem(index,data);
}


int temp;
cout<< “Integer Values: ”;
for(int index=0;index<3;index++)
{
        temp = intArray.getItem(index);
        cout<< temp<< “ ”;
}

cout<<endl;




// Creating and using a DynamicArray object
// dealing with char type of data

DynamicArray<char> charArray(defaultSize);

for(int index=0, value = ‘A’; index<3; index++, value++)
{
        charArray.insertItem(index,value);
}


char tempChar;
cout<< “Character type Values: ”;
for(int index=0; index<3; index++)
{
        tempChar = charArray.getItem(index);
        cout<< tempChar<<“ ”;
}

cout<<endl;


        return 0;
}
```

North South University
CSE-225.1L (Fall-2017)
Lab-03 (Lab-evaluation on C++ Template)

NAME:	
ID:	

Time: 20 Minutes
Marks: 10

Convert the following JAVA class into a C++ template class and perform the mentioned tasks:

MinMax.java

```
public class MinMax
{
        private int maxElement;
        private int minElement;

        public MinMax()
        {
                maxElement = -1;
                minElement = -1;
        }

public void initializeMinMax(int[] numbers,int size)
{
        maxElement = numbers[0];
        minElement = numbers[0];

        for(int i=1;i<size;i++)
        {
                if(numbers[i]<minElement)
                        minElement = numbers[i];

                if(numbers[i]>maxElement)
                        maxElement = numbers[i];
        }
}

public int getMax()
{
        return maxElement;
}

public int getMin()
{
        return minElement;
}

} // MinMax.java class ends here
```

Task

In the main.cpp file, declare an array for holding **5 double** type values and **assign the following values** to the array and using a MinMax class object, determine and print the minimum and maximum values in that array.

values to be stored in the array:
29.75, -23.01, -23.001, 29.757, -1.032

Expected output:

```
Maximum double Element is 29.757
Minimum double Element is -23.01
```

North South University
CSE-225L Fall-2017
Lab 04: Unsorted List (Array Based)

unsortedtype.h

```
#ifndef UNSORTEDTYPE_H_INCLUDED
#define UNSORTEDTYPE_H_INCLUDED


const int  MAX_ITEMS = 5;


template <class ItemType>
class UnsortedType
{

public:
        UnsortedType();
        void makeEmpty();
        bool isFull();
        int lengthIs();
        void insertItem(ItemType);
        void deleteItem(ItemType);
        void retrieveItem(ItemType&, bool&);
        void resetList();
        void getNextItem(ItemType&);

private:
        int length;
        ItemType data[MAX_ITEMS];
        int currentPosition;

};

#endif
```

unsortedtype.cpp

```
#include “unsortedtype.h”

template <class ItemType>
UnsortedType<ItemType>::UnsortedType()
{
        length = 0;
        currentPosition = -1;
}

template <class ItemType>
void UnsortedType<ItemType>::makeEmpty()
{
        length = 0;
}

template <class ItemType>
bool UnsortedType<ItemType>::isFull()
{
        return (length==MAX_ITEMS);
}


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

template <class ItemType>
void UnsortedType<ItemType>::insertItem(ItemType item)
{
        data[length] = item;
        length++;

}

template <class ItemType>
void UnsortedType<ItemType>::deleteItem(ItemType item)
{
        int location = 0;

        while(item != data[location])
        {
                location++;
        }

        data[location] = data[length-1];
        length--;

}

template <class ItemType>
void UnsortedType<ItemType>::retrieveItem(ItemType& item,bool& found)
{
        int location = 0;
        bool moreToSearch = (location<length);
        found = false;

        while( (moreToSearch) && (!found) )
        {
                if (item == data[location])
                {
                  found = true;
                  item = data[location];
                }

                else
                {
                  location++;
                  moreToSearch = (location<length);
                }
        }

}

template <class ItemType>
void UnsortedType<ItemType>::resetList()
{
        currentPosition = -1;

}

template <class ItemType>
```

```
void
UnsortedType<ItemType>::getNextItem(ItemType&
item)
{
        currentPosition++;
        item = data[currentPosition];
}
```

<u>Tasks to be performed</u>:

Now, generate the driver file main.cpp and in that file, perform the following tasks (you cannot change anything in the given source code):

Task Description	Input Values	Expected Output	Allotted Marks
Create a list for integers	-	-	1
Check if the list is empty or not	-	List Empty	1
Insert 4 items in the list	23, -57, 25, 78	-	1
Print all the items in the list using any loop statement	-	23, -57, 25, 78	1
Add another item to the list and print the whole list	96	23, -57, 25, 78, 96	1
Print the length of the list	-	List Length = 5	1
Retrieve 96 and print whether 96 is found or not	-	Item 96 is found	1
Retrieve -69 and print whether -69 is found or not	-	Item -69 not found	1
Delete 25 and print the whole list	-	23,-57,96,78	1
Empty the list and check whether the list is full or not	-	List is not full	1

CSE225L – Data Structures and Algorithms Lab
Lab 04
Unsorted List (array 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

const int MAX_ITEMS = 5;

template <class ItemType>
class UnsortedType
{
    public :
        UnsortedType();
        void MakeEmpty();
        bool IsFull();
        int LengthIs();
        void InsertItem(ItemType);
        void DeleteItem(ItemType);
        void RetrieveItem(ItemType&, bool&);
        void ResetList();
        void GetNextItem(ItemType&);
    private:
        int length;
        ItemType info[MAX_ITEMS];
        int currentPos;
};
#endif // UNSORTEDTYPE_H_INCLUDED
```

unsortedtype.cpp

```
#include "UnsortedType.h"

template <class ItemType>
UnsortedType<ItemType>::UnsortedType()
{
    length = 0;
    currentPos = -1;
}
template <class ItemType>
void UnsortedType<ItemType>::MakeEmpty()
{
       length = 0;
}
template <class ItemType>
bool UnsortedType<ItemType>::IsFull()
{
    return (length == MAX_ITEMS);
}
template <class ItemType>
int UnsortedType<ItemType>::LengthIs()
{
    return length;
}
template <class ItemType>
void UnsortedType<ItemType>::ResetList()
{
    currentPos = -1;
}
template <class ItemType>
void
UnsortedType<ItemType>::GetNextItem(ItemType&
item)
{
    currentPos++;
    item = info [currentPos] ;
}
```

```
template <class ItemType>
void
UnsortedType<ItemType>::RetrieveItem(ItemType&
item, bool &found)
{
    int location = 0;
    bool moreToSearch = (location < length);
    found = false;
    while (moreToSearch && !found)
    {
        if(item == info[location])
        {
            found = true;
            item = info[location];
        }
        else
        {
            location++;
            moreToSearch = (location < length);
        }
    }
}
template <class ItemType>
void UnsortedType<ItemType>::InsertItem(ItemType
item)
{
    info[length] = item;
    length++;
}
template <class ItemType>
void UnsortedType<ItemType>::DeleteItem(ItemType
item)
{
    int location = 0;
    while (item != info[location])
        location++;
    info[location] = info[length - 1];
    length--;
}
```

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 of size 5		
• Insert four items	5, 7, 6, 9	
• Print the list		5 7 6 9
• Print the length of the list		4
• Insert one item	1	
• Print the list		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 full
• Delete 5		
• Print if the list is full or not		List is not full
• Delete 1		
• Print the list		7 6 9
• Delete 6		
• Print the list		7 9

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

CSE225L – Data Structures and Algorithms Lab
Lab 05
Sorted List (array based)

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

sortedtype.h

```
#ifndef SORTEDTYPE_H_INCLUDED
#define SORTEDTYPE_H_INCLUDED

const int MAX_ITEMS = 5;
template <class ItemType>
class SortedType
{
    public :
        SortedType();
        void MakeEmpty();
        bool IsFull();
        int LengthIs();
        void InsertItem(ItemType);
        void DeleteItem(ItemType);
        void RetrieveItem(ItemType&,
bool&);
        void ResetList();
        void GetNextItem(ItemType&);
    private:
        int length;
        ItemType info[MAX_ITEMS];
        int currentPos;
};
#endif // SORTEDTYPE_H_INCLUDED
```

sortedtype.cpp

```
#include "sortedtype.h"
template <class ItemType>
SortedType<ItemType>::SortedType()
{
    length = 0;
    currentPos = - 1;
}
template <class ItemType>
void SortedType<ItemType>::MakeEmpty()
{
    length = 0;
}
template <class ItemType>
bool SortedType<ItemType>::IsFull()
{
    return (length == MAX_ITEMS);
}
template <class ItemType>
int SortedType<ItemType>::LengthIs()
{
    return length;
}
template <class ItemType>
void SortedType<ItemType>::ResetList()
{
    currentPos = - 1;
}
template <class ItemType>
void
SortedType<ItemType>::GetNextItem(ItemType&
item)
{
    currentPos++;
    item = info [currentPos];
}
```

```
template <class ItemType>
void SortedType<ItemType>::InsertItem(ItemType
item)
{
    int location = 0;
    bool moreToSearch = (location < length);

    while (moreToSearch)
    {
        if(item > info[location])
        {
            location++;
            moreToSearch = (location < length);
        }
        else if(item < info[location])
            moreToSearch = false;
    }
    for (int index = length; index > location;
index--)
        info[index] = info[index - 1];
    info[location] = item;
    length++;
}
template <class ItemType>
void SortedType<ItemType>::DeleteItem(ItemType
item)
{
    int location = 0;

    while (item != info[location])
        location++;
    for (int index = location + 1; index < length;
index++)
        info[index - 1] = info[index];
    length--;
}
template <class ItemType>
void SortedType<ItemType>::RetrieveItem(ItemType&
item, bool& found)
{
    int midPoint, first = 0, last = length - 1;
    bool moreToSearch = (first <= last);
    found = false;
    while (moreToSearch && !found)
    {
        midPoint = (first + last) / 2;
        if(item < info[midPoint])
        {
            last = midPoint - 1;
            moreToSearch = (first <= last);
        }
        else if(item > info[midPoint])
        {
            first = midPoint + 1;
            moreToSearch = (first <= last);
        }
        else
        {
            found = true;
            item = info[midPoint];
        }
    }
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a list of size 5		
• Print length of the list		0
• Insert five items	5 7 4 2 1	
• Print the list		1 2 4 5 7
• Retrieve 6 and print whether found		Item is not found
• Retrieve 5 and print whether found		Item is found
• Print if the list is full or not		List is full
• Delete 1		
• Print the list		2 4 5 7
• Print if the list is full or not		List is not full

CSE225L – Data Structures and Algorithms Lab
Lab 11
Sorted List (linked list based)

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

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>::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>
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>
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;
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a list		
• Print Length		0
• Insert five items and print	5 7 4 2 1	1 2 4 5 7
• Retrieve 6 and print whether found		Item is not found
• Retrieve 5 and print whether found		Item is found
• Print if the list is full or not		List is not full
• Delete 1 and print		2 4 5 7
• Print if the list is full or not		List is not full
• Print Length		4

CSE225L – Data Structures and Algorithms Lab
Lab 06
Stack (array based)

In today's lab we will design and implement the Stack ADT using array.

stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED

const int MAX_ITEMS = 5;

class FullStack
// Exception class thrown
// by Push when stack is full.
{};
class EmptyStack
// Exception class thrown
// by Pop and Top when stack is emtpy.
{};

template <class ItemType>
class StackType
{
    public:
        StackType();
        bool IsFull();
        bool IsEmpty();
        void Push(ItemType);
        void Pop();
        ItemType Top();
    private:
        int top;
        ItemType  items[MAX_ITEMS];
};

#endif // STACKTYPE_H_INCLUDED
```

stacktype.cpp

```
#include "StackType.h"
template <class ItemType>
StackType<ItemType>::StackType()
{
    top = -1;
}
template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (top == -1);
}
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    return (top ==  MAX_ITEMS-1);
}
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if( IsFull() ) throw FullStack();
    top++;
    items[top] = newItem;
}
template <class ItemType>
void StackType<ItemType>::Pop()
{
    if( IsEmpty() ) throw EmptyStack();
    top--;
}
template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty()) throw EmptyStack();
    return items[top];
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a stack of size 5		
• Check if the stack is empty		Stack is Empty
• Push four items	5 7 4 2	
• Check if the stack is empty		Stack is not Empty
• Check if the stack is full		Stack is not full
• Print the values in the stack		2 4 7 5
• Push another item	3	
• Print the values in the stack		2 4 7 5 3
• Check if the stack is full		Stack is full
• Pop two items		
• Print top item		7
• Write a function that returns the sum of all odd numbers in the stack. `int sumOdd(StackType s);` Example: If the stack contains 4, 3, 1, 2 and 5, then the function will return 9.		

CSE225L – Data Structures and Algorithms Lab
Lab 08
Stack (Linked List)

In today’s lab we will design and implement the Stack ADT using linked list.

stacktype.h

```
#ifndef STACKTYPE_H_INCLUDED
#define STACKTYPE_H_INCLUDED
class FullStack
{};
class EmptyStack
{};
template <class ItemType>
class StackType
{
    struct NodeType
    {
        ItemType info;
        NodeType* next;
    };
    public:
        StackType();
        ~StackType();
        void Push(ItemType);
        void Pop();
        ItemType Top();
        bool IsEmpty();
        bool IsFull();
    private:
        NodeType* topPtr;
};
#endif // STACKTYPE_H_INCLUDED
```

stacktype.cpp

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

template <class ItemType>
StackType<ItemType>::StackType()
{
    topPtr = NULL;
}

template <class ItemType>
bool StackType<ItemType>::IsEmpty()
{
    return (topPtr == NULL);
}

template <class ItemType>
ItemType StackType<ItemType>::Top()
{
    if (IsEmpty())
        throw EmptyStack();
    else
        return topPtr->info;
}
```

```
template <class ItemType>
bool StackType<ItemType>::IsFull()
{
    NodeType* location;
    try
    {
        location = new NodeType;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
void StackType<ItemType>::Push(ItemType newItem)
{
    if (IsFull())
        throw FullStack();
    else
    {
        NodeType* location;
        location = new NodeType;
        location->info = newItem;
        location->next = topPtr;
        topPtr = location;
    }
}
template <class ItemType>
void StackType<ItemType>::Pop()
{
    if (IsEmpty())
        throw EmptyStack();
    else
    {
        NodeType* tempPtr;
        tempPtr = topPtr;
        topPtr = topPtr->next;
        delete tempPtr;
    }
}
template <class ItemType>
StackType<ItemType>::~StackType()
{
    NodeType* tempPtr;
    while (topPtr != NULL)
    {
        tempPtr = topPtr;
        topPtr = topPtr->next;
        delete tempPtr;
    }
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
- Create a stack		
- Check if the stack is empty		Stack is Empty
- Push four items	5 7 4 2	
- Check if the stack is empty		Stack is not Empty
- Check if the stack is full		Stack is not full
- Print the values in the stack		2 4 7 5
- Push another item	3	
- Print the values in the stack		2 4 7 5 3
- Check if the stack is full		Stack is not full
- Pop two items		
- Print top item		7

- Add a function **`ReplaceItem`** to the `StackType` class which replaces all occurrences of **`oldItem`** with **`newItem`** in the Queue.

  `void ReplaceItem(int oldItem, int newItem);`

  <u>Sample Input &Output:</u>

  Stack items: 21 26 13 26 29 $\xrightarrow{\texttt{ReplaceItem(26, 9)}}$ Stack items: 21 9 13 9 29

CSE225L – Data Structures and Algorithms Lab
Lab 07
Queue (array based)

In today's lab we will design and implement the Queue ADT using array.

quetype.h

```
#ifndef QUETYPE_H_INCLUDED
#define QUETYPE_H_INCLUDED

class FullQueue
{};
class EmptyQueue
{};
template<class ItemType>
class QueType
{
    public:
        QueType();
        QueType(int max);
        ~QueType();
        void MakeEmpty();
        bool IsEmpty();
        bool IsFull();
        void Enqueue(ItemType);
        void Dequeue(ItemType&);
    private:
        int front;
        int rear;
        ItemType* items;
        int maxQue;
};

#endif // QUETYPE_H_INCLUDED
```

quetype.cpp

```
#include "quetype.h"

template<class ItemType>
QueType<ItemType>::QueType(int max)
{
    maxQue = max + 1;
    front = maxQue - 1;
    rear = maxQue - 1;
    items = new ItemType[maxQue];
}
template<class ItemType>
QueType<ItemType>::QueType()
{
    maxQue = 501;
    front = maxQue - 1;
    rear = maxQue - 1;
    items = new ItemType[maxQue];
}
template<class ItemType>
QueType<ItemType>::~QueType()
{
    delete [] items;
}
template<class ItemType>
void QueType<ItemType>::MakeEmpty()
{
    front = maxQue - 1;
    rear = maxQue - 1;
}
template<class ItemType>
bool QueType<ItemType>::IsEmpty()
{
    return (rear == front);
}
template<class ItemType>
bool QueType<ItemType>::IsFull()
{
    return ((rear+1)%maxQue == front);
}
template<class ItemType>
void QueType<ItemType>::Enqueue(ItemType newItem)
{
    if (IsFull())
        throw FullQueue();
    else
    {
        rear = (rear +1) % maxQue;
        items[rear] = newItem;
    }
}
template<class ItemType>
void QueType<ItemType>::Dequeue(ItemType& item)
{
    if (IsEmpty())
        throw EmptyQueue();
    else
    {
        front = (front + 1) % maxQue;
        item = items[front];
    }
}
```

Generate the **Driver file (main.cpp)** and perform the following tasks:

Operation to Be Tested and Description of Action	**Input Values**	**Expected Output**
• Create a queue of size 5		
• Print if the queue is empty or not		Queue is Empty
• Enqueue four items	5 7 4 2	
• Print if the queue is empty or not		Queue is not Empty
• Print if the queue is full or not		Queue is not full
• Enqueue another item	6	
• Print the values in the queue		5 7 4 2 6
• Print if the queue is full or not		Queue is Full
• Enqueue another item	8	Queue Overflow
• Dequeue two items		
• Print the values in the queue		4 2 6
• Dequeue three items		
• Print if the queue is empty or not		Queue is Empty
• Dequeue an item		Queue Underflow
• Add a function **`ReplaceItem`** to the QueType class which replaces all occurrences of **`oldItem`** with **`newItem`** in the Queue. **`void ReplaceItem(int oldItem, int newItem);`** **Sample Input &Output:** Queue Items: 21 26 13 26 29 $\xrightarrow{\texttt{ReplaceItem(26, 9)}}$ Queue Items: 21 9 13 9 29		

CSE225L – Data Structures and Algorithms Lab
Lab 09
Queue (Linked List)

In today's lab we will design and implement the Queue ADT using linked list.

quetype.h

```
#ifndef QUETYPE_H_INCLUDED
#define QUETYPE_H_INCLUDED
class FullQueue
{};
class EmptyQueue
{};
template <class ItemType>
class QueType
{
    struct NodeType
    {
        ItemType info;
        NodeType* next;
    };
    public:
        QueType();
        ~QueType();
        void MakeEmpty();
        void Enqueue(ItemType);
        void Dequeue(ItemType&);
        bool IsEmpty();
        bool IsFull();
    private:
        NodeType *front, *rear;
};

#endif // QUETYPE_H_INCLUDED
```

quetype.cpp

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

template <class ItemType>
QueType<ItemType>::QueType()
{
    front = NULL;
    rear = NULL;
}
template <class ItemType>
bool QueType<ItemType>::IsEmpty()
{
    return (front == NULL);
}
template<class ItemType>
bool QueType<ItemType>::IsFull()
{
    NodeType* location;
    try
    {
        location = new NodeType;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
void QueType<ItemType>::Enqueue(ItemType newItem)
{
    if (IsFull())
        throw FullQueue();
    else
    {
        NodeType* newNode;
        newNode = new NodeType;
        newNode->info = newItem;
        newNode->next = NULL;
        if (rear == NULL)
            front = newNode;
        else
            rear->next = newNode;
        rear = newNode;
    }
}
template <class ItemType>
void QueType<ItemType>::Dequeue(ItemType& item)
{
    if (IsEmpty())
        throw EmptyQueue();
    else
    {
        NodeType* tempPtr;
        tempPtr = front;
        item = front->info;
        front = front->next;
        if (front == NULL)
            rear = NULL;
        delete tempPtr;
    }
}
template <class ItemType>
void QueType<ItemType>::MakeEmpty()
{
    NodeType* tempPtr;
    while (front != NULL)
    {
        tempPtr = front;
        front = front->next;
        delete tempPtr;
    }
    rear = NULL;
}
template <class ItemType>
QueType<ItemType>::~QueType()
{
    MakeEmpty();
}
```

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**
• Print if the queue is empty or not		Queue is Empty
• Enqueue four items	5 7 4 2	
• Print if the queue is empty or not		Queue is not Empty
• Print if the queue is full or not		Queue is not full
• Enqueue another item	6	
• Print the values in the queue		5 7 4 2 6
• Print if the queue is full or not		Queue is not Full
• Enqueue another item	8	
• Dequeue two items		
• Dequeue		
• Print the values in the queue		2 6 8
• Dequeue three items		
• Print if the queue is empty or not		Queue is Empty
• Dequeue an item		Queue Underflow
• Add a function `Length` to the `QueType` class which returns the number of items in the Queue. `int Length();` **Sample Input &Output:** Queue Items: n o w y h `Length()` → Length is : 5		

CSE225L – Data Structures and Algorithms Lab
Lab 12
Recursion

1. Write a recursive function that returns the nth Fibonacci number from the Fibonacci series.

   ```
   int fib(int n);
   ```

2. Write a recursive function to find the factorial of a number.

   ```
   int factorial(int n);
   ```

3. Write a recursive function that returns the sum of the digits of an integer.

   ```
   int sumOfDigits(int x);
   ```

4. Write a recursive function that find the minimum element in an array of integers.

   ```
   int findMin(int a[], int size);
   ```

5. Write a recursive function that converts a decimal number to binary number.

   ```
   int DecToBin(int dec);
   ```

6. Write a recursive function that find the sum of the following series.

   $1 + 1/2 + 1/4 + 1/8 + ... + 1/2^n$

CSE225L – Data Structures and Algorithms Lab
Lab 15
Graph

In today’s lab we will design and implement the Graph ADT.

graphtype.h

```
#ifndef GRAPHTYPE_H_INCLUDED
#define GRAPHTYPE_H_INCLUDED
#include "stacktype.h"
#include "quetype.h"
template<class VertexType>
class GraphType
{
    public:
        GraphType();
        GraphType(int maxV);
        ~GraphType();
        void MakeEmpty();
        bool IsEmpty();
        bool IsFull();
        void AddVertex(VertexType);
        void AddEdge(VertexType,
VertexType, int);
        int WeightIs(VertexType,
VertexType);
        void GetToVertices(VertexType,
QueType<VertexType>&);
        void ClearMarks();
        void MarkVertex(VertexType);
        bool IsMarked(VertexType);
        void DepthFirstSearch(VertexType,
VertexType);
        void BreadthFirstSearch(VertexType,
VertexType);
    private:
        int numVertices;
        int maxVertices;
        VertexType* vertices;
        int **edges;
        bool* marks;
};
#endif // GRAPHTYPE_H_INCLUDED
```

heaptype.cpp

```
#include "graphtype.h"
#include "stacktype.cpp"
#include "quetype.cpp"
#include <iostream>
using namespace std;
const int NULL_EDGE = 0;

template<class VertexType>
GraphType<VertexType>::GraphType()
{
    numVertices = 0;
    maxVertices = 50;
    vertices = new VertexType[50];
    edges = new int*[50];
    for(int i=0;i<50;i++)
        edges[i] = new int [50];
    marks = new bool[50];
}
template<class VertexType>
GraphType<VertexType>::GraphType(int maxV)
{
    numVertices = 0;
    maxVertices = maxV;
    vertices = new VertexType[maxV];
    edges = new int*[maxV];
    for(int i=0;i<maxV;i++)
        edges[i] = new int [maxV];
    marks = new bool[maxV];
}
```

```
template<class VertexType>
GraphType<VertexType>::~GraphType()
{
    delete [] vertices;
    delete [] marks;
    for(int i=0;i<maxVertices;i++)
        delete [] edges[i];
    delete [] edges;
}
template<class VertexType>
void GraphType<VertexType>::MakeEmpty()
{
    numVertices = 0;
}
template<class VertexType>
bool GraphType<VertexType>::IsEmpty()
{
    return (numVertices == 0);
}
template<class VertexType>
bool GraphType<VertexType>::IsFull()
{
    return (numVertices == maxVertices);
}
template<class VertexType>
void GraphType<VertexType>::AddVertex(VertexType
vertex)
{
    vertices[numVertices] = vertex;
    for (int index=0; index<numVertices; index++)
    {
        edges[numVertices][index] = NULL_EDGE;
        edges[index][numVertices] = NULL_EDGE;
    }
    numVertices++;
}
template<class VertexType>
int IndexIs(VertexType* vertices, VertexType
vertex)
{
    int index = 0;
    while (!(vertex == vertices[index]))
        index++;
    return index;
}
template<class VertexType>
void GraphType<VertexType>::ClearMarks()
{
    for(int i=0; i<maxVertices; i++)
        marks[i] = false;
}
template<class VertexType>
void GraphType<VertexType>::MarkVertex(VertexType
vertex)
{
    int index = IndexIs(vertices, vertex);
    marks[index] = true;
}
template<class VertexType>
bool GraphType<VertexType>::IsMarked(VertexType
vertex)
{
    int index = IndexIs(vertices, vertex);
    return marks[index];
}
```

```
template<class VertexType>
void GraphType<VertexType>::AddEdge(VertexType fromVertex, VertexType toVertex, int weight)
{
    int row = IndexIs(vertices, fromVertex);
    int col= IndexIs(vertices, toVertex);
    edges[row][col] = weight;
}
template<class VertexType>
int GraphType<VertexType>::WeightIs(VertexType fromVertex, VertexType toVertex)
{
    int row = IndexIs(vertices, fromVertex);
    int col= IndexIs(vertices, toVertex);
    return edges[row][col];
}
template<class VertexType>
void GraphType<VertexType>::GetToVertices(VertexType vertex, QueType<VertexType>& adjVertices)
{
    int fromIndex, toIndex;
    fromIndex = IndexIs(vertices, vertex);
    for (toIndex = 0; toIndex < numVertices; toIndex++)
        if (edges[fromIndex][toIndex] != NULL_EDGE)
            adjVertices.Enqueue(vertices[toIndex]);
}
```

```
template<class VertexType>
void
GraphType<VertexType>::DepthFirstSearch(Vertex
Type startVertex, VertexType endVertex)
{
    StackType<VertexType> stack;
    QueType<VertexType> vertexQ;
    bool found = false;
    VertexType vertex, item;

    ClearMarks();
    stack.Push(startVertex);
    do
    {
        vertex = stack.Top();
        stack.Pop();
        if (vertex == endVertex)
        {
            cout << vertex << " ";
            found = true;
        }
        else
        {
            if (!IsMarked(vertex))
            {
                MarkVertex(vertex);
                cout << vertex << " ";
                GetToVertices(vertex,vertexQ);
                while (!vertexQ.IsEmpty())
                {
                    vertexQ.Dequeue(item);
                    if (!IsMarked(item))
                        stack.Push(item);
                }
            }
        }
    } while (!stack.IsEmpty() && !found);
    cout << endl;
    if (!found)
        cout << "Path not found." << endl;
}
```

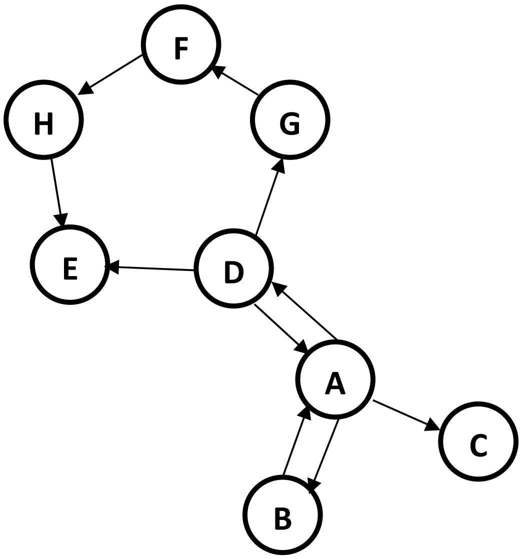
```
template<class VertexType>
void
GraphType<VertexType>::BreadthFirstSearch(Vertex
Type startVertex, VertexType endVertex)
{
    QueType<VertexType> queue;
    QueType<VertexType> vertexQ;

    bool found = false;
    VertexType vertex, item;

    ClearMarks();
    queue.Enqueue(startVertex);
    do
    {
        queue.Dequeue(vertex);
        if (vertex == endVertex)
        {
            cout  << vertex << " ";
            found = true;
        }
        else
        {
            if (!IsMarked(vertex))
            {
                MarkVertex(vertex);
                cout  << vertex << " ";
                GetToVertices(vertex, vertexQ);

                while (!vertexQ.IsEmpty())
                {
                    vertexQ.Dequeue(item);
                    if (!IsMarked(item))
                        queue.Enqueue(item);
                }
            }
        }
    } while (!queue.IsEmpty() && !found);
    cout << endl;
    if (!found)
        cout << "Path not found." << endl;
}
```

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**
• Generate the following graph. Assume that all edge costs are 1.  		
• Outdegree of a particular vertex in a graph is the number of edges going out from that vertex to other vertices. For instance the outdegree of vertex **B** in the above graph is 1. Add a member function `OutDegree` to the `GraphType` class which returns the outdegree of a given vertex. `int  OutDegree(VertexType v);`		
• Add a member function to the class which determines if there is an edge between two vertices. `bool FoundEdge(VertexType u, VertexType v);`		
• Print the outdegree of the vertex **D**.		3
• Print if there is an edge between vertices **A** and **D**.		There is an edge.
• Print if there is an edge between vertices **B** and **D**.		There is no edge.
• Use depth first search in order to find if there is a path from **B** to **E**.		B A D G F H E
• Use depth first search in order to find if there is a path from **E** to **B**.		E Path not found.
• Use breadth first search in order to find if there is a path from **B** to **E**.		B A C D E
• Use breadth first search in order to find if there is a path from **E** to **B**.		E Path not found.
• Modify the `BreadthFirstSearch` function so that it also prints the length of the shortest path between two vertices.		
• Determine the length of the shortest path from **B** to **E**.		3

CSE225L – Data Structures and Algorithms Lab
Lab 13
Binary Search Tree

In today's lab we will design and implement the Binary Search Tree ADT.

binarysearchtree.h

```
#ifndef BINARYSEARCHTREE_H_INCLUDED
#define BINARYSEARCHTREE_H_INCLUDED
#include "quetype.h"
template <class ItemType>
struct TreeNode
{
    ItemType info;
    TreeNode* left;
    TreeNode* right;
};
enum OrderType {PRE_ORDER, IN_ORDER,
POST_ORDER};
template <class ItemType>
class TreeType
{
    public:
        TreeType();
        ~TreeType();
        void MakeEmpty();
        bool IsEmpty();
        bool IsFull();
        int LengthIs();
        void RetrieveItem(ItemType& item,
bool& found);
        void InsertItem(ItemType item);
        void DeleteItem(ItemType item);
        void ResetTree(OrderType order);
        void GetNextItem(ItemType& item,
OrderType order, bool& finished);
        void Print();
    private:
        TreeNode<ItemType>* root;
        QueType<ItemType> preQue;
        QueType<ItemType> inQue;
        QueType<ItemType> postQue;
};
#endif // BINARYSEARCHTREE_H_INCLUDED
```

binarysearchtree.cpp

```
#include "binarysearchtree.h"
#include "quetype.cpp"
#include <iostream>
using namespace std;
template <class ItemType>
TreeType<ItemType>::TreeType()
{
    root = NULL;
}
template <class ItemType>
void Destroy(TreeNode<ItemType>*& tree)
{
    if (tree != NULL)
    {
        Destroy(tree->left);
        Destroy(tree->right);
        delete tree;
        tree = NULL;
    }
}
template <class ItemType>
TreeType<ItemType>::~TreeType()
{
    Destroy(root);
}
template <class ItemType>
void TreeType<ItemType>::MakeEmpty()
{
    Destroy(root);
}
template <class ItemType>
bool TreeType<ItemType>::IsEmpty()
{
    return root == NULL;
}
template <class ItemType>
bool TreeType<ItemType>::IsFull()
{
    TreeNode<ItemType>* location;
    try
    {
        location = new TreeNode<ItemType>;
        delete location;
        return false;
    }
    catch(bad_alloc& exception)
    {
        return true;
    }
}
template <class ItemType>
int CountNodes(TreeNode<ItemType>* tree)
{
    if (tree == NULL)
        return 0;
    else
        return CountNodes(tree->left) +
CountNodes(tree->right) + 1;
}
template <class ItemType>
int TreeType<ItemType>::LengthIs()
{
    return CountNodes(root);
}
template <class ItemType>
void Retrieve(TreeNode<ItemType>* tree, ItemType&
item, bool& found)
{
    if (tree == NULL)
        found = false;
    else if (item < tree->info)
        Retrieve(tree->left, item, found);
    else if (item > tree->info)
        Retrieve(tree->right, item, found);
    else
    {
        item = tree->info;
        found = true;
    }
}
template <class ItemType>
void TreeType<ItemType>::RetrieveItem(ItemType&
item, bool& found)
{
    Retrieve(root, item, found);
}
```

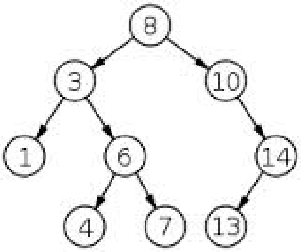
```
template <class ItemType>
void Insert(TreeNode<ItemType>*& tree,
ItemType item)
{
    if (tree == NULL)
    {
        tree = new TreeNode<ItemType>;
        tree->right = NULL;
        tree->left = NULL;
        tree->info = item;
    }
    else if (item < tree->info)
        Insert(tree->left, item);
    else
        Insert(tree->right, item);
}
template <class ItemType>
void TreeType<ItemType>::InsertItem(ItemType
item)
{
    Insert(root, item);
}
template <class ItemType>
void Delete(TreeNode<ItemType>*& tree,
ItemType item)
{
    if (item < tree->info)
        Delete(tree->left, item);
    else if (item > tree->info)
        Delete(tree->right, item);
    else
        DeleteNode(tree);
}
template <class ItemType>
void DeleteNode(TreeNode<ItemType>*& tree)
{
    ItemType data;
    TreeNode<ItemType>* tempPtr;

    tempPtr = tree;
    if (tree->left == NULL)
    {
        tree = tree->right;
        delete tempPtr;
    }
    else if (tree->right == NULL)
    {
        tree = tree->left;
        delete tempPtr;
    }
    else
    {
        GetPredecessor(tree->left, data);
        tree->info = data;
        Delete(tree->left, data);
    }
}
template <class ItemType>
void GetPredecessor(TreeNode<ItemType>*
tree, ItemType& data)
{
    while (tree->right != NULL)
        tree = tree->right;
    data = tree->info;
}
template <class ItemType>
void TreeType<ItemType>::DeleteItem(ItemType
item)
{
    Delete(root, item);
}
```

```
template <class ItemType>
void PreOrder(TreeNode<ItemType>* tree,
QueType<ItemType>& Que)
{
    if (tree != NULL)
    {
        Que.Enqueue(tree->info);
        PreOrder(tree->left, Que);
        PreOrder(tree->right, Que);
    }
}
template <class ItemType>
void InOrder(TreeNode<ItemType>* tree,
QueType<ItemType>& Que)
{
    if (tree != NULL)
    {
        InOrder(tree->left, Que);
        Que.Enqueue(tree->info);
        InOrder(tree->right, Que);
    }
}
template <class ItemType>
void PostOrder(TreeNode<ItemType>* tree,
QueType<ItemType>& Que)
{
    if (tree != NULL)
    {
        PostOrder(tree->left, Que);
        PostOrder(tree->right, Que);
        Que.Enqueue(tree->info);
    }
}
template <class ItemType>
void TreeType<ItemType>::ResetTree(OrderType
order)
{
    switch (order)
    {
        case PRE_ORDER:
            PreOrder(root, preQue);
            break;
        case IN_ORDER:
            InOrder(root, inQue);
            break;
        case POST_ORDER:
            PostOrder(root, postQue);
            break;
    }
}
template <class ItemType>
void TreeType<ItemType>::GetNextItem(ItemType&
item, OrderType order, bool& finished)
{
    finished = false;
    switch (order)
    {
        case PRE_ORDER:
            preQue.Dequeue(item);
            if(preQue.IsEmpty())
                finished = true;
            break;
        case IN_ORDER:
            inQue.Dequeue(item);
            if(inQue.IsEmpty())
                finished = true;
            break;
        case POST_ORDER:
            postQue.Dequeue(item);
            if(postQue.IsEmpty())
                finished = true;
            break;
    }
}
```

```
template <class ItemType>
void PrintTree(TreeNode<ItemType>* tree)
{
    if (tree != NULL)
    {
        PrintTree(tree->left);
        cout << tree->info << " ";
        PrintTree(tree->right);
    }
}
template <class ItemType>
void TreeType<ItemType>::Print()
{
    PrintTree(root);
}
```

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 tree object		
• Print if the tree is empty or not		Tree is empty
• Insert ten items	4 9 2 7 3 11 17 0 5 1	
• Print if the tree is empty or not		Tree is not empty
• Print the length of the tree		10
• Retrieve 9 and print whether found or not		Item is found
• Retrieve 13 and print whether found or not		Item is not found
• Print the elements in the tree (inorder)		0 1 2 3 4 5 7 9 11 17
• Print the elements in the tree (preorder)		4 2 0 1 3 9 7 5 11 17
• Print the elements in the tree (postorder)		1 0 3 2 5 7 17 11 9 4
• Make the tree empty		
• Build the following tree inserting the elements, one by one 		
• Add a member function to the TreeType class which returns the minimum element in the tree. `int findMin();`		1
• Add a function to the TreeType class which returns the number of leaves in the tree. `int numLeaves();`		4

CSE225L – Data Structures and Algorithms Lab
Lab 14
Priority Queue

In today's lab we will design and implement the Priority Queue ADT.

heaptype.h

```
#ifndef HEAPTYPE_H_INCLUDED
#define HEAPTYPE_H_INCLUDED
template<class ItemType>
struct HeapType
{
    void ReheapDown(int root, int bottom);
    void ReheapUp(int root, int bottom);
    ItemType* elements;
    int numElements;
};
#endif // HEAPTYPE_H_INCLUDED
```

heaptype.cpp

```
#include "heaptype.h"
template<class ItemType>
void Swap(ItemType& one, ItemType& two)
{
    ItemType temp;
    temp = one;
    one = two;
    two = temp;
}
template<class ItemType>
void HeapType<ItemType>::ReheapDown(int root, int bottom)
{
    int maxChild;
    int rightChild;
    int leftChild;

    leftChild = root*2+1;
    rightChild = root*2+2;
    if (leftChild <= bottom)
    {
        if (leftChild == bottom)
            maxChild = leftChild;
        else
        {
            if(elements[leftChild]<=elements[rightChild])
                maxChild = rightChild;
            else
                maxChild = leftChild;
        }
        if (elements[root] < elements[maxChild])
        {
            Swap(elements[root], elements[maxChild]);
            ReheapDown(maxChild, bottom);
        }
    }
}
template<class ItemType>
void HeapType<ItemType>::ReheapUp(int root, int bottom)
{
    int parent;
    if (bottom > root)
    {
        parent = (bottom-1) / 2;
        if (elements[parent] < elements[bottom])
        {
            Swap(elements[parent], elements[bottom]);
            ReheapUp(root, parent);
        }
    }
}
```

pqtype.h

```
#ifndef PQTYPE_H_INCLUDED
#define PQTYPE_H_INCLUDED
#include "heaptype.h"
#include "heaptype.cpp"
class FullPQ
{};
class EmptyPQ
{};
template<class ItemType>
class PQType
{
    public:
        PQType(int);
        ~PQType();
        void MakeEmpty();
        bool IsEmpty();
        bool IsFull();
        void Enqueue(ItemType);
        void Dequeue(ItemType&);
    private:
        int length;
        HeapType<ItemType> items;
        int maxItems;
};
#endif // PQTYPE_H_INCLUDED
```

pqtype.cpp

```
#include "pqtype.h"
template<class ItemType>
PQType<ItemType>::PQType(int max)
{
    maxItems = max;
    items.elements=new ItemType[max];
    length = 0;
}
template<class ItemType>
PQType<ItemType>::~PQType()
{
    delete [] items.elements;
}
template<class ItemType>
void PQType<ItemType>::MakeEmpty()
{
    length = 0;
}
template<class ItemType>
bool PQType<ItemType>::IsEmpty()
{
    return length == 0;
}
template<class ItemType>
bool PQType<ItemType>::IsFull()
{
    return length == maxItems;
}
```

```
template<class ItemType>
void PQType<ItemType>::Enqueue(ItemType newItem)
{
    if (length == maxItems)
        throw FullPQ();
    else
    {
        length++;
        items.elements[length-1] = newItem;
        items.ReheapUp(0, length-1);
    }
}
```

```
template<class ItemType>
void PQType<ItemType>::Dequeue(ItemType& item)
{
    if (length == 0)
        throw EmptyPQ();
    else
    {
        item = items.elements[0];
        items.elements[0] =
items.elements[length-1];
        length--;
        items.ReheapDown(0, length-1);
    }
}
```

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**
• Add a member function `PrintQueue` to the `PQType` class which prints the content of the heap		
• Create a `PQType` object		
• Print if the queue is empty or not		Queue is empty
• Insert ten items, in the order they appear	4 9 2 7 3 11 17 0 5 1	
• Print if the queue is empty or not		Queue is not empty
• Print the elements in the heap		17 7 11 5 3 2 9 0 4 1
• Dequeue one element and print the dequeued value		17
• Dequeue one element and print the dequeued value		11
• Print the elements in the heap		9 7 4 5 3 2 1 0
• Dequeue three more elements		
• Print the elements in the heap		4 3 2 0 1
• Modify the `ReheapUp` and the `ReheapDown` functions in such a way that the `PQType` class now works as a min-heap		
• Insert ten items, in the order they appear	4 9 2 7 3 11 17 0 5 1	
• Print the elements in the heap		0 1 4 3 2 11 17 9 5 7
• Dequeue one element and print the dequeued value		0
• Dequeue one element and print the dequeued value		1
• Print the elements in the heap		2 3 4 5 7 11 17 9
• Dequeue three more elements		
• Print the elements in the heap		5 7 11 9 17