1. Without using the STL implement a class to maintain a **stack** of doubles (`StackOfDoubles`)

a. The class should use the list of doubles you implemented last week, `ListOfDoubles` internally. (i.e. you don't have to code a stack from scratch).

It will have the usual stack methods:

- `push()` //pushes a double onto the stack
- `pop()` // removes the top item from the stack without returning it
- `top()` //returns the top item from the stack without removing it

b. The stack should also facilitate an overloaded insertion operator to insert the contents of the stack (tab separated) into an output stream. The signature of this overloaded operator is:

```
ostream & operator<<(ostream& str, const StackOfDoubles &stackobj)
```

c. Write a program which fully tests your class.

// StackOfDoubles.h

```
#pragma once
#include "DoubleListNode.h"
#include <string>
#include <iostream>
#include <Windows.h> // uses for Color
using namespace std;

class StackOfDoubles // LinkedList
{
	friend class DoubleListNode;

public:
	friend ostream& operator<< (ostream& str, const StackOfDoubles &stackObj);
	friend void setColor(int colorValue);

	StackOfDoubles();
	~StackOfDoubles();

	void push(double data);
	void pop();
	double getTop();

private:
	DoubleListNode *top;
};
typedef StackOfDoubles *List;
```

// StackOfDoubles.cpp

```
#include "StackOfDoubles.h"

StackOfDoubles::StackOfDoubles()
	: top(NULL)
{
	cout << "Stack Example: LIFO (Last In First Out)" << endl;
}

StackOfDoubles::~StackOfDoubles()
{
	ListNodePtr tempPtr;
	while (top)
	{
		tempPtr = top;
		top = top->next;
		delete tempPtr;
	}
}

// perform insert at the front/top of the stack
void StackOfDoubles::push(double data)
{
	DoubleListNode* newNode = new DoubleListNode(data);
	if (newNode)
	{
		newNode->next = top;
		top = newNode;
	}
	else
	{
		cout << "Push Could Not Be Done!" << endl;
	}
}

void StackOfDoubles::pop()
{
	double data = 0;
	if (!top) // empty list
	{
		cout << "Pop cannot be done with an empty stack!\n" << endl;
	}
	else // because insert at the front/start => most recent will be at front/start
	{
		ListNodePtr tempPtr = top;
		top = top->next;
		data = tempPtr->data;
		cout << "\nPoping the most recent! Data = " << data << endl;
		delete tempPtr;
	}
}
```

```
double StackOfDoubles::getTop()
{
	double data = 0;
	if (!top)
	{
		data = 0;
		cout << "It is an empty stack!\n" << endl;
	}
	else
	{
		data = top->data;
		cout << "Top data is: " << data << endl;
	}
	return data;
}

ostream& operator<< (ostream& str, const StackOfDoubles &stackObj)
{
	ListNodePtr tempPtr = stackObj.top;
	if (!tempPtr)
	{
		str << "The stack is empty, nothing to be displayed!" << endl;
	}
	else
	{
		setColor(11);
		str << "\nStack:" << "\n"
			<< "======" << endl;
		setColor(7);
		while (tempPtr != NULL)
		{
			setColor(10);
			str << "\t" << "| " << tempPtr->getData() << " |" << "\n"
				<< "\t" << "|___|" << endl;
			tempPtr = tempPtr->getNext();
		}
		setColor(7);
	}
	return str;
}

/*
 * usage: used/modified
 * availability: http://www.cplusplus.com/forum/beginner/77879/
 * on 03/11/2016, at 13:30
 */
void setColor(int value)
{
	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), value);
}
```

// DoubleListNode.h

```
#pragma once
#include "StackOfDoubles.h"
#include <string>

class DoubleListNode // ListNode
{
	friend class StackOfDoubles;

public:
	DoubleListNode();
	DoubleListNode(double data);
	double getData();
	DoubleListNode* getNext();

private:
	double data;
	DoubleListNode *next;
};
typedef DoubleListNode *ListNodePtr;
```

// DoubleListNode.cpp

```
#include "DoubleListNode.h"

DoubleListNode::DoubleListNode()
	: data(0), next(NULL)
{}

DoubleListNode::DoubleListNode(double data)
	: data(data), next(NULL)
{}

double DoubleListNode::getData()
{
	return data;
}

DoubleListNode* DoubleListNode::getNext()
{
	return next;
}
```

```
// Main.cpp

#include "StackOfDoubles.h"
#include <ostream>


int main()
{
	StackOfDoubles stack;

	operator<< (cout, stack);
	stack.pop();

	stack.push(1);
	stack.push(2);
	stack.push(3);
	operator<< (cout, stack);

	stack.pop();
	operator<< (cout, stack);

	stack.~StackOfDoubles();
	system("pause");
	return 0;
}
```

2. Using the STL implement a **queue** of doubles by

– Composition, using an STL list `(list<double>)` to hold the data

http://www.cplusplus.com/reference/queue/queue/

```
/* usage: used/modifed
 * availability: http://www.cplusplus.com/reference/queue/queue/queue/
 * on 21/11/2016, at 20:30
 */
#include <iostream>
#include <queue>
#include <list>
using namespace std;

int main()
{
	queue<double, list<double>> myQueue;

	myQueue.push(1);
	myQueue.push(2);
	myQueue.push(3);
	myQueue.push(4);
	myQueue.push(5);

	myQueue.pop();

	cout << "Front of queue is: " << myQueue.front() << endl;
	cout << "Back  of queue is: " << myQueue.back() << endl;
```

```
	cout << "\nSize  of queue is: " << myQueue.size() << endl;

	cout << "\nThe full queue is: ";
	while (!myQueue.empty())
	{
		cout << myQueue.front() << "  ";
		myQueue.pop(); // queue will be empty after loop finished
	}
	cout << endl;

	system("pause");
	return 0;
}
```

Now if we wanted to use an STL vector `(vector<double>)` to hold the data, we cannot as it does not have a `pop_front()` operation; such an operation would be slow, as it would have to move all the remaining elements. However if we want to make a priority queue we can use a vector. Therefore now.

3. Using the STL implement a **priority queue** of doubles by

– Composition, using an STL vector `(vector<double>)` to hold the data.

http://www.cplusplus.com/reference/queue/priority_queue/

http://www.cplusplus.com/reference/queue/priority_queue/priority_queue/

```
/* usage: used/modifed
 * availability: http://www.cplusplus.com/reference/queue/priority_queue/priority_queue/
 * on 21/11/2016, at 21:45
 */

#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

int main()
{
	priority_queue<int, vector<double>> pQueue;
	pQueue.push(1);
	pQueue.push(2);
	pQueue.push(3);

	cout << "Front: " << pQueue.top() << endl;

	pQueue.pop();
	cout << "Front: " << pQueue.top() << endl;

	system("pause");
	return 0;
}
```