In this lab you will implement a stack data structure, described in class, using the object-oriented features of C++.
The class declarations are the following (stack.h in the lab files):

```cpp
class ListNode {
    private:
        int data;
        ListNode *next;

    public:
        ListNode(int data, ListNode *next);
        int getData();
        void setData(int data);
        ListNode* getNext();
        void setNext(ListNode *next);
};

class Stack {
    private:
        ListNode *head;

        // Private member functions
        void insertHead(int data);
        void deleteHead();

    public:
        // Constructors
        Stack();
        Stack(int a[], int length);
        ~Stack();

        // Member functions
        int pop();
        void push(int data);
        int peek();
        bool isEmpty();
        void printContents(ostream &os);

        // Operator overload
        void operator<<(int data);
        void operator>>(int &data);
};
```

A stack.cpp file is provided that includes the implementation of all the ListNode member functions, and of all the Stack private member functions.
To test your list implementation, a main.cpp is provided in the lab files. Running this program with a correct stack implementation should yield the following:

```
9 8 7 6 5 4 3 2 1
First element is 9
9 8 7 6 5 4 3 2 1
Popped element 9
8 7 6 5 4 3 2 1
Popped element 8
7 6 5 4 3 2 1
Popped element 7
6 5 4 3 2 1
Popped element 6
5 4 3 2 1
Popped element 5
4 3 2 1
Popped element 4
3 2 1
Popped element 3
2 1
Popped element 2
1
Popped element 1
Stack is empty!
5 4 3 2 1
8 7 6 5 4 3 2 1
Popped element 8
7 6 5 4 3 2 1
```

You should be able to do the following exercises (except the last one) simply by reusing code from the list implementation seen in class and the stack code provided in the midterm. Do not overthink this lab.

**Exercise 1** <<3 points>>

Implement the default constructor:

```
Stack();
```

**Exercise 2** <<5 points>>

Implement the following constructor:

```
Stack(int a[], int length);
```

This constructor takes the values in array a (of length length), and pops them into the stack (starting with the first element in the array, and ending with the last element)
Exercise 3 <<5 points>>
Implement the following standard stack operations (described in class):

```cpp
int pop();
void push(int data);
int peek();
```

These three functions can assume that the stack is not empty.

Exercise 4 <<2 points>>
Implement the `isEmpty` member function:

```cpp
bool isEmpty();
```

This function return true is the stack is empty, and false otherwise.

Exercise 5 <<2 points>>
Implement the `printContents` function:

```cpp
void printContents(ostream &os);
```

This function must print the contents of the stack starting at the top of the stack and ending at the bottom of the stack.

Notice how, unlike previous list traversal functions seen in class, this one expects a parameter of type `ostream`. This is done so the programmer will be able to specify what output stream to use (see the main.cpp file for an example).
Exercise 6 <<8 points>>

Overload the following operators:

```cpp
void operator<<(int data);
void operator>>(int &data);
```

The `<<` operator must be overloaded in such a way that the following operation results in having value 17 pushed into stack `st`:

```cpp
Stack st;
st << 17;
```

The `>>` operator must be overloaded in such a way that the following operation results in having a value popped from stack `st` and stored in variable `n`:

```cpp
Stack st;
// Initialize stack with values
int n;
st >> n;
```

Exercise 7 <<15 points>>

You are asked to do some modifications to your stack implementation. Copy all your files to a new directory called Exercise7 and make your modifications there.

➢ Rewrite the class declarations in such a way that `Stack` becomes a friend class of `ListNode`, so that the `Stack` member functions don't have to use get/set methods to access the `data` and `next` member variables of a `ListNode` object.

➢ Eliminate the get/set methods from the `ListNode` class, and modify your `Stack` implementation accordingly.

➢ Overload the `<<` operator in such a way that you can print out the contents of the stack to an output stream. For example:

```cpp
Stack s1;
s1.push(5);
s1.push(6);
s1.push(7);
cout << s1; // Should write “7 6 5”
```

A `main_friend.cpp` file is provided to test these modifications. The result of running this file with your stack implementation should yield the same output shown on page 3.