Lab 6

Goals-
Build linked structures using **pointers**

You will create simple structures using dynamic memory. You will implement a Stack class and a Queue class. Each will have a pointer to the appropriate struct and the designated functions.

1. Stack-like behavior. A stack is a first in last out (FILO) structure. So the top of the stack will be the last item entered. You are not required to implement any other features of the formal Stack data structure. Be careful when you research this. You could make your program more difficult.

You will create a singly linked Stacknode. You will need to add an element and to take off an element. You should have functions:

```
void add(Type_of_Data Value)
Type_of_Data remove()
```

Type_of_Data is whatever data type you want to use; int, float, char, string, or even a class. You are only required to use one data type! So using int, add() will take an integer value, create a node, set your value to the data part of the node, and adjust pointers as necessary. The remove() function will return the data in the top node. It will then remove the node and adjust pointers.

This is all that is required.

2. Queue-like behavior. A queue is a first in first out (FIFO) structure. You will need a pointer to the front and to the back of the queue. You are not required to implement any other features of the formal Queue data structure. Be careful when you research this. You could make your program more difficult.

You will create a doubly linked Queuenode. You will need a front pointer and a back (or rear) pointer. You will need to add an element at the back node and to take off an element from the front node. You should have functions:

```
void add(Type_of_Data Value)
Type_of_Data remove()
```

Type_of_Data is whatever data type you want to use; int, float, char, string, or even a class. You are only required to use one data type! So using int, add() will take an integer value, create a node, set your value to the data part of the node, and adjust the back pointer to point to it. The remove() function will return the data in the front node. It will then remove the node and adjust the front pointer.

NOTE: De-allocate memory as appropriate.
HINT: You will use your code as part of assignment 4.

Modular Grading

We are using modular grading. Each lab will be divided into specific modules. Each module will be graded pass/fail. It either works properly or it does not. 10% of every lab or assignment grade is style/comments or other elements of self-documenting code and clarity. Remember the labs are worth 10 points total.

Programming style- 1 point

Create the nodes that use links properly- 1 point

Simple first in last out structure- 3 points
- Create it
- Correctly implement add()
- Correctly implement remove()

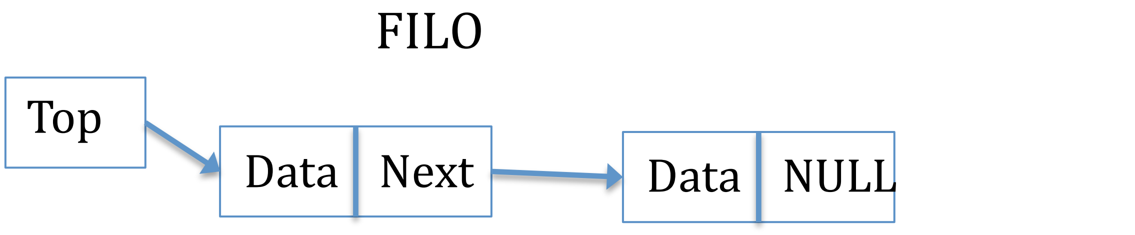
Simple first in first out structure - 3 points
- Create it
- Correctly implement add()
- Correctly implement remove()

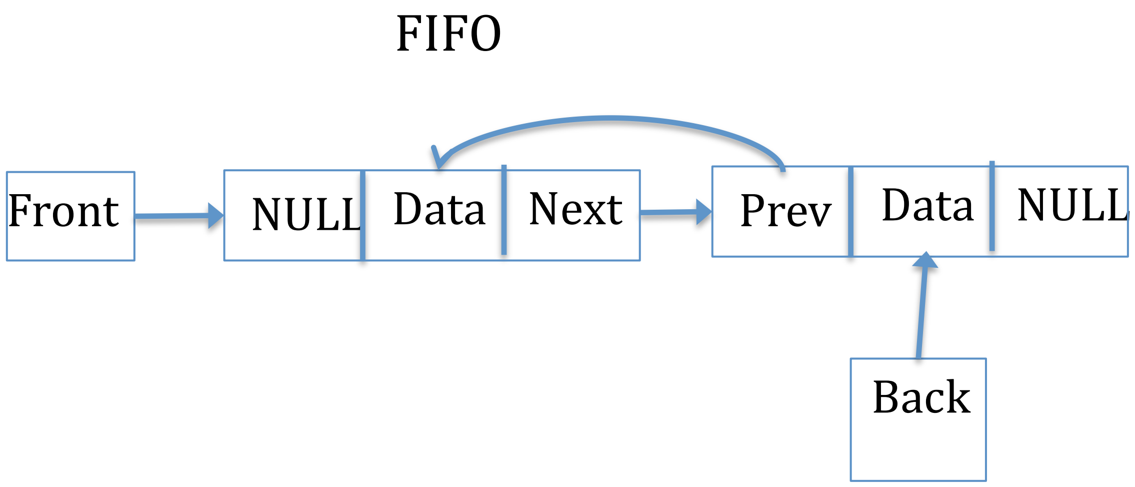
Proper use of dynamic memory- 1 point
--If there serious memory management issues deductions can be made from creating or implementing a structure.

Brief description of how you tested your structures- 1 point

Note: Remember that you cannot use or copy code from any source. Your work must be your own.

To help visualize the structures:





With a doubly linked list Front and Back can be either end.