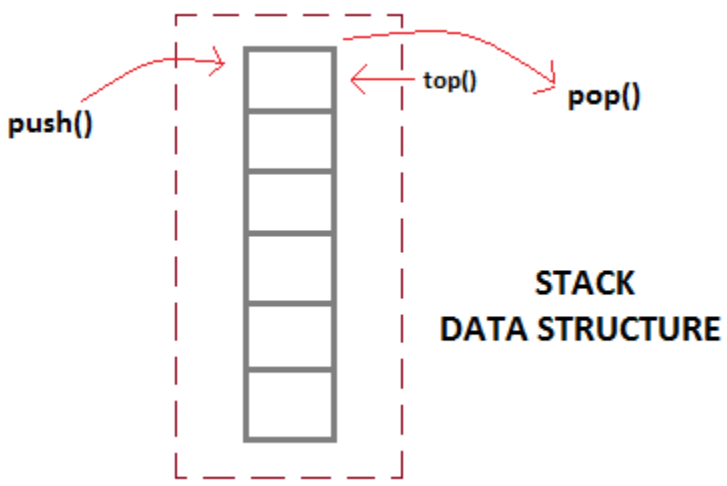
Stack using Linked List

Stack as we know is a **Last In First Out(LIFO)** data structure. It has the following operations :

- **push:** push an element into the stack
- **pop:** remove the last element added
- **top:** returns the element at top of stack



Implementation of Stack using Linked List

Stacks can be easily implemented using a linked list. Stack is a data structure to which a data can be added using the `push()` method and data can be removed from it using the `pop()` method. With Linked list, the **push** operation can be replaced by the `addAtFront()` method of linked list and **pop** operation can be replaced by a function which deletes the front node of the linked list.

In this way our Linked list will virtually become a Stack with `push()` and `pop()` methods.

First we create a class **node**. This is our Linked list node class which will have **data** in it and a **node pointer** to store the address of the next node element.

```
class node

{

     int data;

     node *next;

};
```

Copy

Then we define our stack class,

```
class Stack

{

     node *front;  // points to the head of list

     public:

     Stack()
```

```
	{
		front = NULL;
	}
	// push method to add data element
	void push(int);
	// pop method to remove data element
	void pop();
	// top method to return top data element
	int top();
};
```

Copy

Inserting Data in Stack (Linked List)

In order to insert an element into the stack, we will create a node and place it in front of the list.

```
void Stack :: push(int d)
{
	// creating a new node
	node *temp;
	temp = new node();
	// setting data to it
	temp->data = d;

	// add the node in front of list
	if(front == NULL)
	{
		temp->next = NULL;
	}
	else
	{
		temp->next = front;
```

```
	}

	front = temp;

}
```

Copy

Now whenever we will call the `push()` function a new node will get added to our list in the front, which is exactly how a stack behaves.

Removing Element from Stack (Linked List)

In order to do this, we will simply delete the first node, and make the second node, the head of the list.

```
void Stack :: pop()

{

	// if empty

	if(front == NULL)

		cout << "UNDERFLOW\n";


	// delete the first element

	else

	{

		node *temp = front;

		front = front->next;

		delete(temp);

	}

}
```

Copy

Return Top of Stack (Linked List)

In this, we simply return the data stored in the head of the list.

```
int Stack :: top()

{

	return front->data;
```

```
}
```

Copy

Conclusion

When we say "implementing Stack using Linked List", we mean how we can make a Linked List behave like a Stack, after all they are all logical entities. So for any data structure to act as a Stack, it should have `push()` method to add data on top and `pop()` method to remove data from top. Which is exactly what we did and hence accomplished to make a Linked List behave as a Stack.