Singly Linked List with Classes & Struct

Structs are used to create the Linked List Nodes

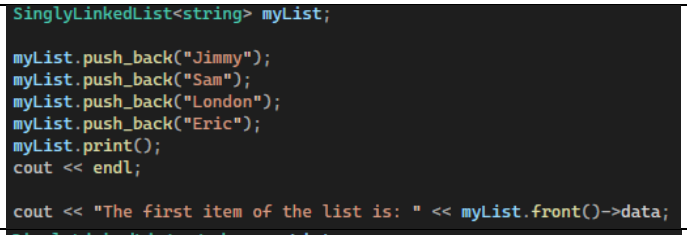
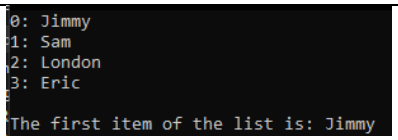
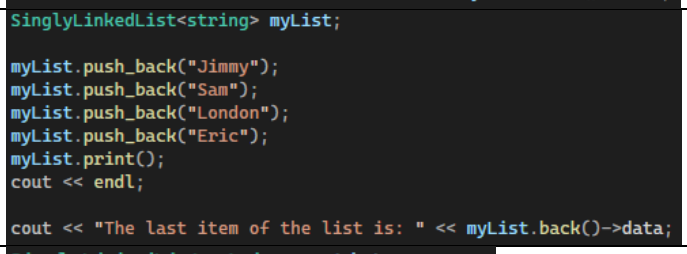
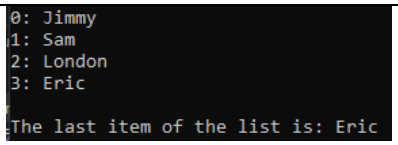
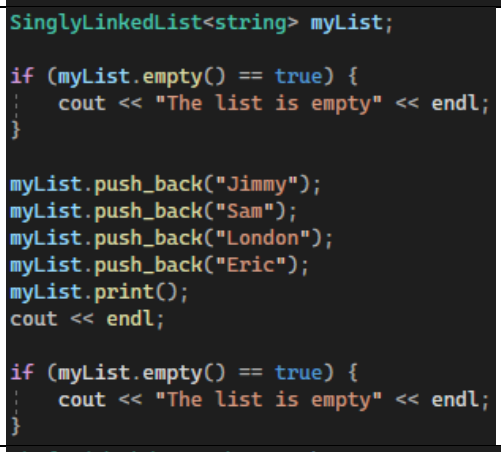
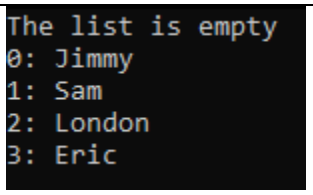
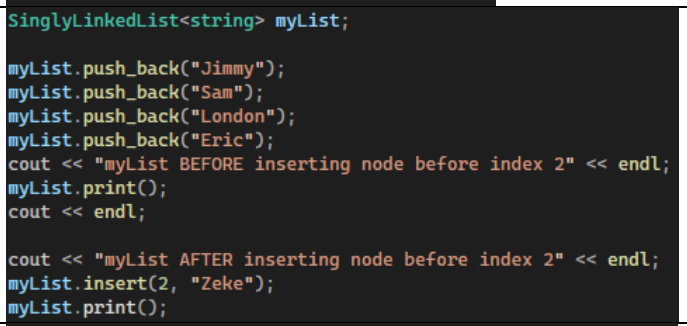
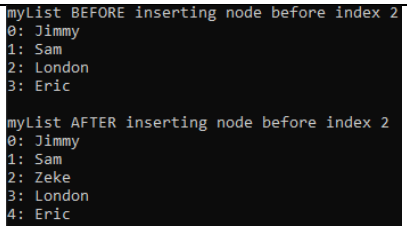
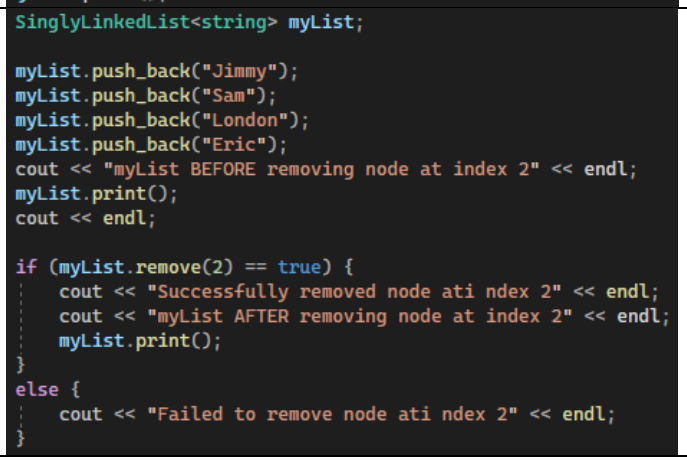
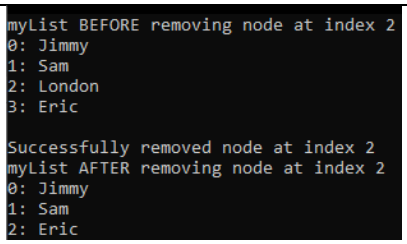
```
template <typename data_type>
struct Node {
    // Data Fields
    data_type data;
    Node* next; // The pointer to the next node.

    // Constructor
    // Creates a new Node that points to another node
    // Creates a Node containing user inputed data
    Node(const data_type& data_type, Node* next_ptr = NULL) : data(data_type), next(next_ptr) {}
};
```

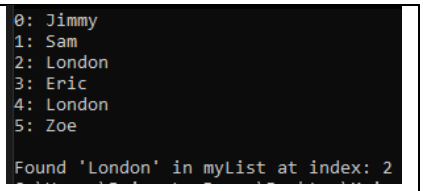
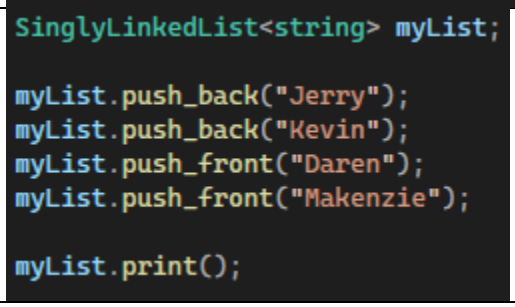
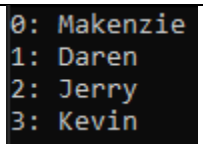
Properties & Methods	Code Snippet	Code Output

push_front(T data) -
- add a node to the front of the list
- create a new head node if list is empty
	`SinglyLinkedList<string> myList;`
	
	`myList.print();`
	`cout << endl;`
	
	`myList.push_front("Bianca");`
	`myList.print();`
	`cout << endl;`
	
	`myList.push_front("Cody");`
	`myList.print();`
		`The singly linked list is empty`
		
		`0: Bianca`
		
		`0: Cody`
		`1: Bianca`

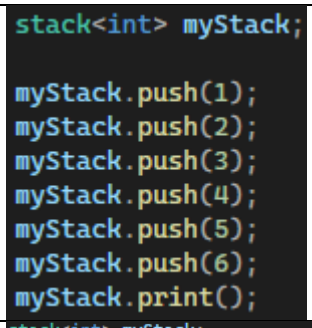
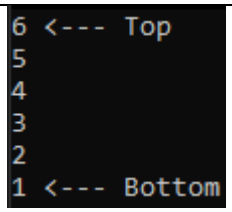
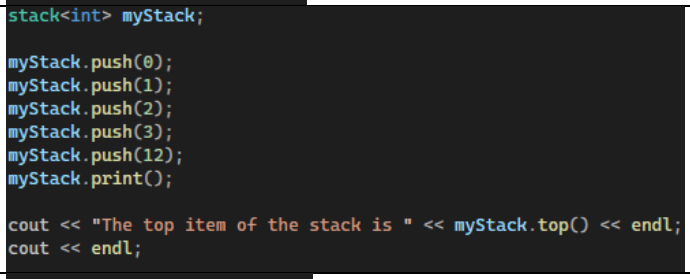
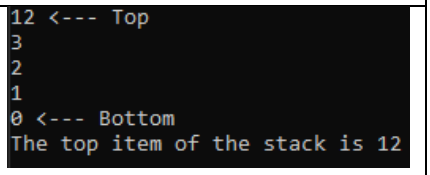
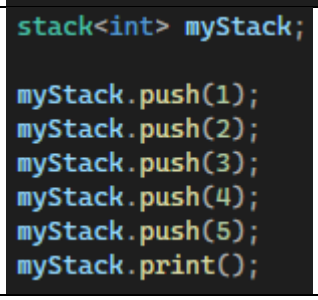
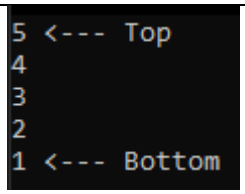
push_back(T data) -
- add a node to the back of the list
- create a new head node if list is empty
	`SinglyLinkedList<string> myList;`
	
	`myList.print();`
	`cout << endl;`
	
	`myList.push_back("Brian");`
	`myList.print();`
	`cout << endl;`
	
	`myList.push_back("Eric");`
	`myList.print();`
		`The singly linked list is empty`
		
		`0: Brian`
		
		`0: Brian`
		`1: Eric`

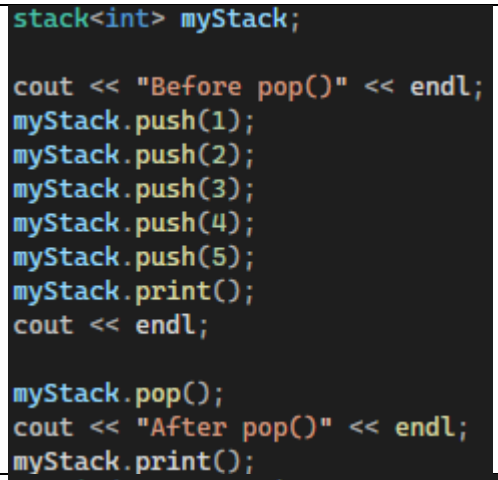
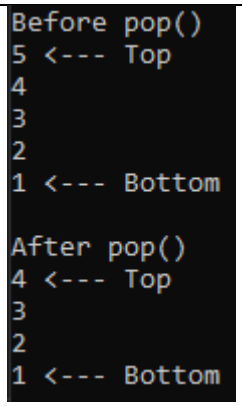
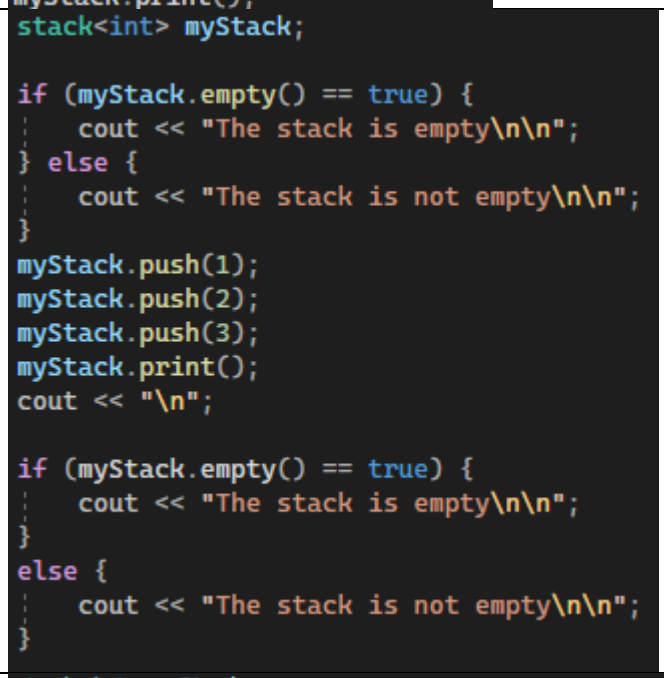
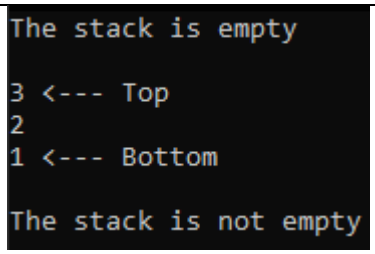
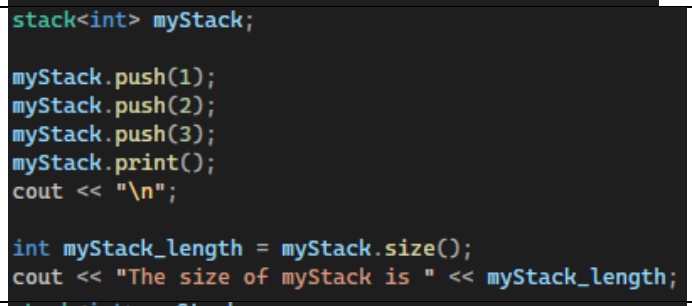
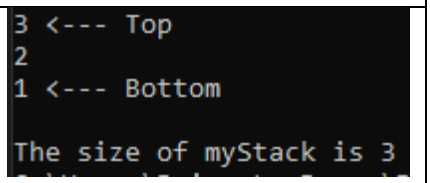
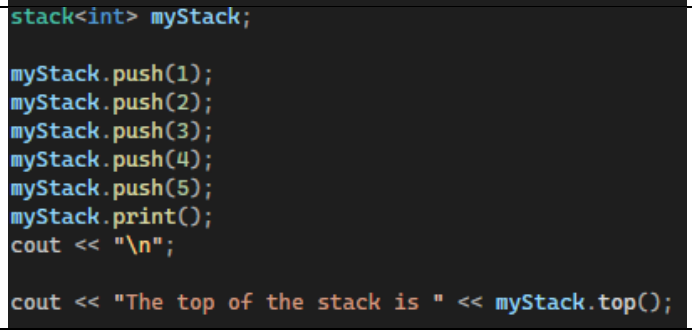
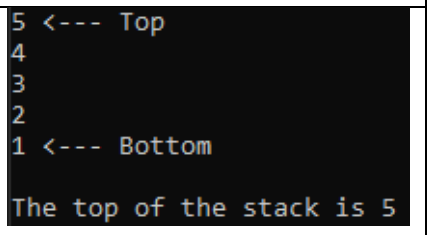
pop_front() -
- Remove the node that's in front of the list
	`SinglyLinkedList<string> myList;`
	
	`myList.push_back("Tommy");`
	`myList.push_back("Sam");`
	`myList.push_back("London");`
	`myList.push_back("Eric");`
	`myList.print();`
	`cout << endl;`
	
	`myList.pop_front();`
	`myList.print();`
	`cout << endl;`
	`SinglyLinkedList<string> myList;`
	
	`myList.push_back("Tommy");`
	`myList.print();`
	`cout << endl;`
	
	`myList.pop_front();`
	`myList.print();`
		`0: Tommy`
		`1: Sam`
		`2: London`
		`3: Eric`
		
		`0: Sam`
		`1: London`
		`2: Eric`
		
		`0: Tommy`
		
		`The singly linked list is empty`

front() - Returns the head node of the list		
back() - Returns the tail node of the list		
empty() - Returns true if the linked list has 0 nodes inside of it. Returns false if the list has at least 1 item.		
insert(T data) Inserts a new node before the index the user specifies		
remove(int index) Removes the node that's located at an index the user specifies		

find(T data) - Returns the position of the first occurrence of the item if it's found - If the item is not found, the size of the list will be returned		
print() - prints out all of items in the list in ascending order		

Stack Using Vectors

Properties & Methods	Code Snippet	Code Output
print() - prints all of the items in the stack from top to bottom		
top() - returns the node that is at the top of the stack		
push() - adds a new node at the top of the stack. Nodes are only added from bottom to top.		

pop() - Removes a node from the top of the stack. In a stack, nodes can only be removed from the top of the stack.		
empty() - Returns true if the stack has 0 items. Returns false, if the stack is not empty.		
size() - Returns an integer that represents the number of items that are in the stack.		
top() - Returns the node that is at the very top of the stack.		

average() - Returns the average value of the stack items.