

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 inputted 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) - <ul style="list-style-type: none">- add a node to the front of the list- create a new head node if list is empty	<pre>SinglyLinkedList<string> myList; myList.print(); cout << endl; myList.push_front("Bianca"); myList.print(); cout << endl; myList.push_front("Cody"); myList.print();</pre>	<pre>The singly linked list is empty 0: Bianca 0: Cody 1: Bianca</pre>
push_back(T data) - <ul style="list-style-type: none">- add a node to the back of the list- create a new head node if list is empty	<pre>SinglyLinkedList<string> myList; myList.print(); cout << endl; myList.push_back("Brian"); myList.print(); cout << endl; myList.push_back("Eric"); myList.print();</pre>	<pre>The singly linked list is empty 0: Brian 0: Brian 1: Eric</pre>
pop_front() - <ul style="list-style-type: none">- Remove the node that's in front of the list	<pre>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();</pre>	<pre>0: Tommy 1: Sam 2: London 3: Eric 0: Sam 1: London 2: Eric 0: Tommy The singly linked list is empty</pre>

front() - Returns the head node of the list	<pre>SinglyLinkedList<string> myList; myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); myList.print(); cout << endl; cout << "The first item of the list is: " << myList.front()->data;</pre>	<pre>0: Jimmy 1: Sam 2: London 3: Eric The first item of the list is: Jimmy</pre>
back() - Returns the tail node of the list	<pre>SinglyLinkedList<string> myList; myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); myList.print(); cout << endl; cout << "The last item of the list is: " << myList.back()->data;</pre>	<pre>0: Jimmy 1: Sam 2: London 3: Eric The last item of the list is: Eric</pre>
empty() - Returns true if the linked list has 0 nodes inside of it. Returns false if the list has at least 1 item.	<pre>SinglyLinkedList<string> myList; if (myList.empty() == true) { cout << "The list is empty" << endl; } myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); myList.print(); cout << endl; if (myList.empty() == true) { cout << "The list is empty" << endl; }</pre>	<pre>The list is empty 0: Jimmy 1: Sam 2: London 3: Eric</pre>
insert(T data) Inserts a new node before the index the user specifies	<pre>SinglyLinkedList<string> myList; myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); cout << "myList BEFORE inserting node before index 2" << endl; myList.print(); cout << endl; cout << "myList AFTER inserting node before index 2" << endl; myList.insert(2, "Zeke"); myList.print();</pre>	<pre>myList BEFORE inserting node before index 2 0: Jimmy 1: Sam 2: London 3: Eric myList AFTER inserting node before index 2 0: Jimmy 1: Sam 2: Zeke 3: London 4: Eric</pre>
remove(int index) Removes the node that's located at an index the user specifies	<pre>SinglyLinkedList<string> myList; myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); cout << "myList BEFORE removing node at index 2" << endl; myList.print(); cout << endl; if (myList.remove(2) == true) { cout << "Successfully removed node at index 2" << endl; cout << "myList AFTER removing node at index 2" << endl; myList.print(); } else { cout << "Failed to remove node at index 2" << endl; }</pre>	<pre>myList BEFORE removing node at index 2 0: Jimmy 1: Sam 2: London 3: Eric Successfully removed node at index 2 myList AFTER removing node at index 2 0: Jimmy 1: Sam 2: Eric</pre>

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	<pre>SinglyLinkedList<string> myList; myList.push_back("Jimmy"); myList.push_back("Sam"); myList.push_back("London"); myList.push_back("Eric"); myList.push_back("London"); myList.push_back("Zoe"); myList.print(); cout << endl; cout << "Found 'London' in myList at index: " << myList.find("London");</pre>	<pre>0: Jimmy 1: Sam 2: London 3: Eric 4: London 5: Zoe Found 'London' in myList at index: 2</pre>
print() - prints out all of items in the list in ascending order	<pre>SinglyLinkedList<string> myList; myList.push_back("Jerry"); myList.push_back("Kevin"); myList.push_front("Daren"); myList.push_front("Makenzie"); myList.print();</pre>	<pre>0: Makenzie 1: Daren 2: Jerry 3: Kevin</pre>

Stack Using Vectors

Properties & Methods	Code Snippet	Code Output
print() - prints all of the items in the stack from top to bottom	<pre>stack<int> myStack; myStack.push(1); myStack.push(2); myStack.push(3); myStack.push(4); myStack.push(5); myStack.push(6); myStack.print();</pre>	<pre>6 <--- Top 5 4 3 2 1 <--- Bottom</pre>
top() - returns the node that is at the top of the stack	<pre>stack<int> myStack; myStack.push(0); myStack.push(1); myStack.push(2); myStack.push(3); myStack.push(12); myStack.print(); cout << "The top item of the stack is " << myStack.top() << endl; cout << endl;</pre>	<pre>12 <--- Top 3 2 1 0 <--- Bottom The top item of the stack is 12</pre>
push() - adds a new node at the top of the stack. Nodes are only added from bottom to top.	<pre>stack<int> myStack; myStack.push(1); myStack.push(2); myStack.push(3); myStack.push(4); myStack.push(5); myStack.print();</pre>	<pre>5 <--- Top 4 3 2 1 <--- Bottom</pre>

<p>pop() - Removes a node from the top of the stack. In a stack, nodes can only be removed from the top of the stack.</p>	<pre>stack<int> myStack; cout << "Before pop()" << endl; myStack.push(1); myStack.push(2); myStack.push(3); myStack.push(4); myStack.push(5); myStack.print(); cout << endl; myStack.pop(); cout << "After pop()" << endl; myStack.print();</pre>	<pre>Before pop() 5 <--- Top 4 3 2 1 <--- Bottom After pop() 4 <--- Top 3 2 1 <--- Bottom</pre>
<p>empty() - Returns true if the stack has 0 items. Returns false, if the stack is not empty.</p>	<pre>stack<int> myStack; if (myStack.empty() == true) { cout << "The stack is empty\n\n"; } else { cout << "The stack is not empty\n\n"; } myStack.push(1); myStack.push(2); myStack.push(3); myStack.print(); cout << "\n"; if (myStack.empty() == true) { cout << "The stack is empty\n\n"; } else { cout << "The stack is not empty\n\n"; }</pre>	<pre>The stack is empty 3 <--- Top 2 1 <--- Bottom The stack is not empty</pre>
<p>size() - Returns an integer that represents the number of items that are in the stack.</p>	<pre>stack<int> myStack; myStack.push(1); myStack.push(2); myStack.push(3); myStack.print(); cout << "\n"; int myStack_length = myStack.size(); cout << "The size of myStack is " << myStack_length;</pre>	<pre>3 <--- Top 2 1 <--- Bottom The size of myStack is 3</pre>
<p>top() - Returns the node that is at the very top of the stack.</p>	<pre>stack<int> myStack; myStack.push(1); myStack.push(2); myStack.push(3); myStack.push(4); myStack.push(5); myStack.print(); cout << "\n"; cout << "The top of the stack is " << myStack.top();</pre>	<pre>5 <--- Top 4 3 2 1 <--- Bottom The top of the stack is 5</pre>

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

```
stack<int> myStack;  
  
myStack.push(1);  
myStack.push(2);  
myStack.push(3);  
myStack.push(4);  
myStack.push(5);  
myStack.print();  
cout << "\n";  
  
cout << "The average of the values" << endl;  
cout << "in the stack is " << myStack.average();
```

```
5 <--- Top  
4  
3  
2  
1 <--- Bottom  
  
The average of the values  
in the stack is 3
```