Templates C++

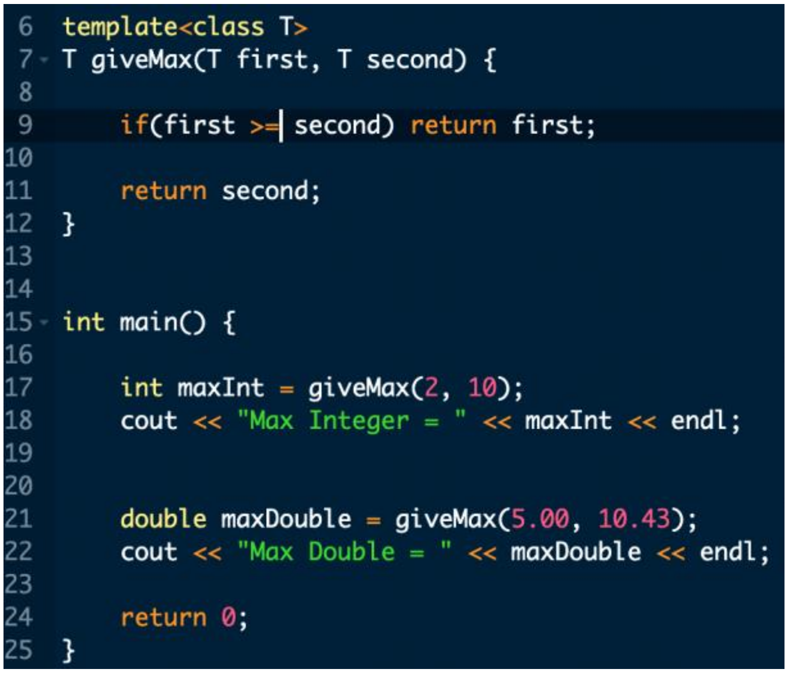
Thursday, October 10, 2019 7:14 PM

1. There are 2 kinds of templates:

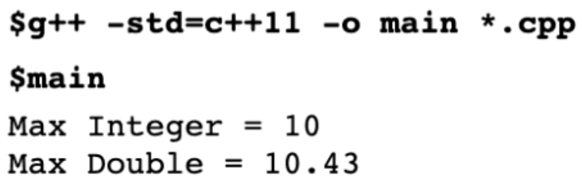
a.

<table>
<tr>
<td>Function Templates</td>
<td>

1. Allows us to create a function template whose functionality can be adapted to more than one type without repeating the code for each type

2. 

```cpp
template<class T>
T giveMax(T first, T second) {

    if(first >= second) return first;

    return second;
}


int main() {

    int maxInt = giveMax(2, 10);
    cout << "Max Integer = " << maxInt << endl;


    double maxDouble = giveMax(5.00, 10.43);
    cout << "Max Double = " << maxDouble << endl;

    return 0;
}
```



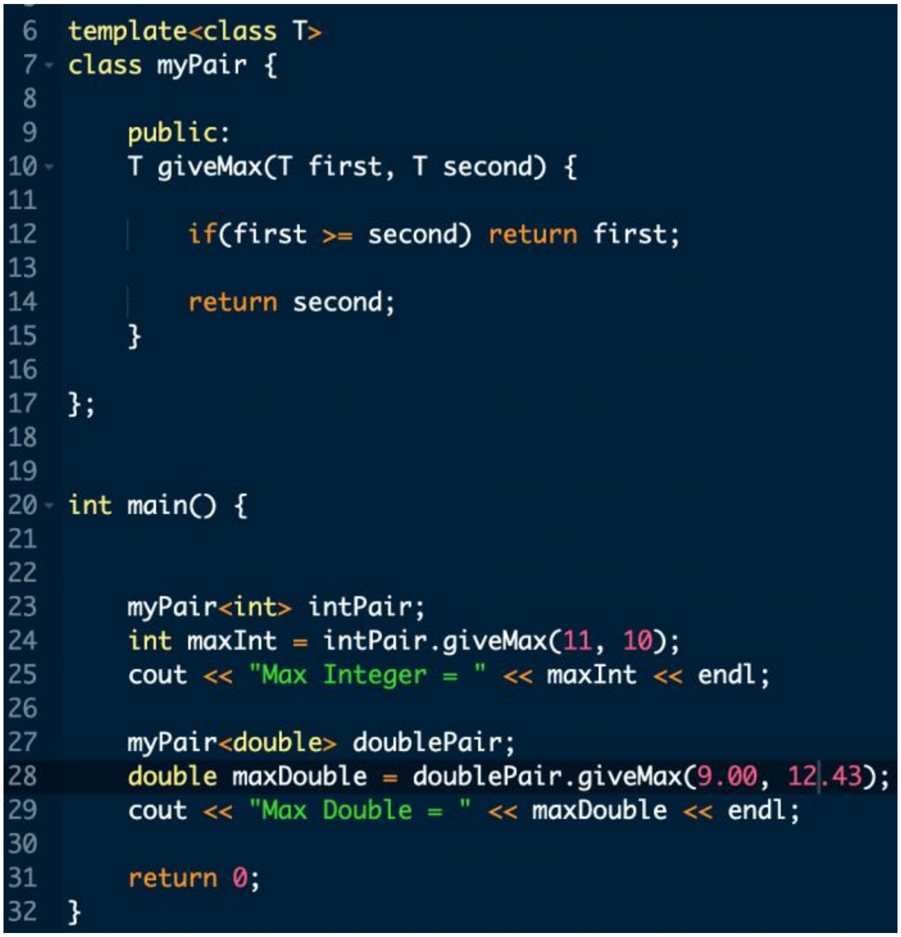
```
$g++ -std=c++11 -o main *.cpp
$main
Max Integer = 10
Max Double = 10.43
```

</td>
</tr>
<tr>
<td>Class Templates</td>
<td>

1. We may even write class templates, where a class can have members that use template parameters as types

</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>

2. 

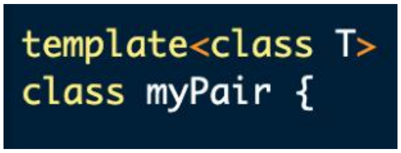
```
template<class T>
class myPair {

    public:
    T giveMax(T first, T second) {

        if(first >= second) return first;

        return second;
    }

};


int main() {


    myPair<int> intPair;
    int maxInt = intPair.giveMax(11, 10);
    cout << "Max Integer = " << maxInt << endl;

    myPair<double> doublePair;
    double maxDouble = doublePair.giveMax(9.00, 12.43);
    cout << "Max Double = " << maxDouble << endl;

    return 0;
}
```

```
$g++ -std=c++11 -o main *.cpp
$main
Max Integer = 11
Max Double = 12.43
```

</td>
</tr>
</table>

2. There are mainly 2 things to worry about in template writing:

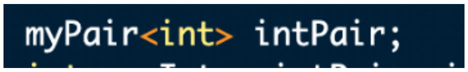
<table>
<tr>
<td>Declaration</td>
<td>



```
template<class T>
class myPair {
```

</td>
</tr>
<tr>
<td>Usage</td>
<td>

1. Every time we use the class, we have to specify the template parameters separately

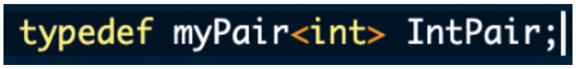
   1. 

```
myPair<int> intPair;
```

</td>
</tr>
</table>

a.

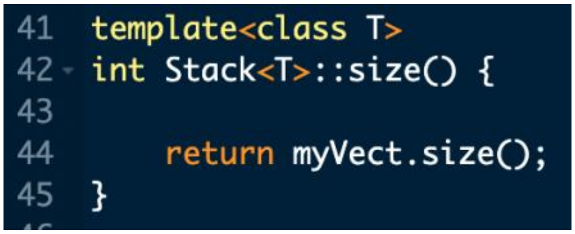
2. Or, we can use something like typedef to avoid repetitive usages:



```
typedef myPair<int> IntPair;
```

3. Scope and writing a function outside of its class:

4.

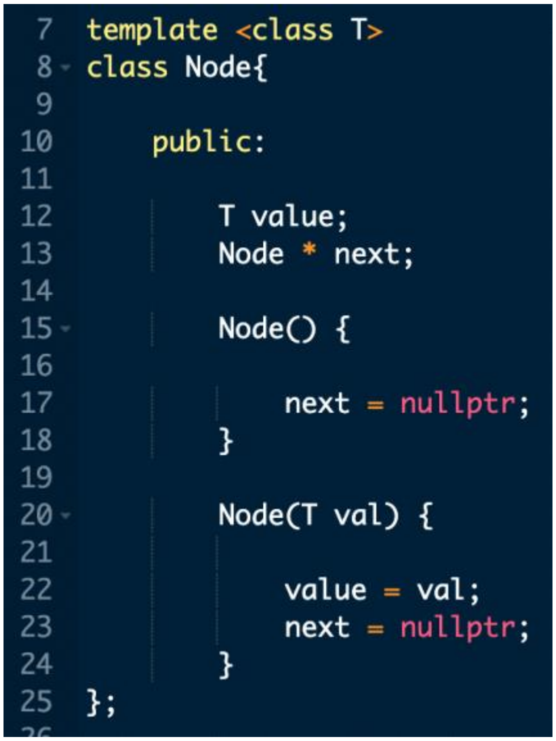


```
template<class T>
int Stack<T>::size() {

    return myVect.size();
}
```

3. Example Linked List code using Templates:

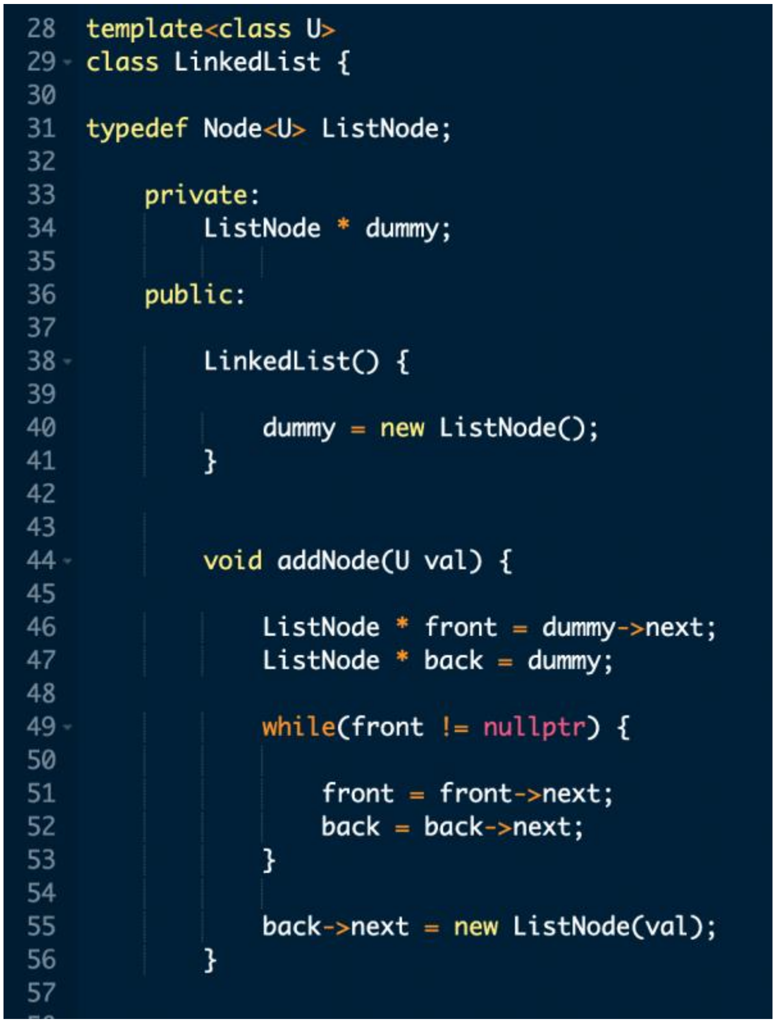
a. Linked List Node using templates:

b.

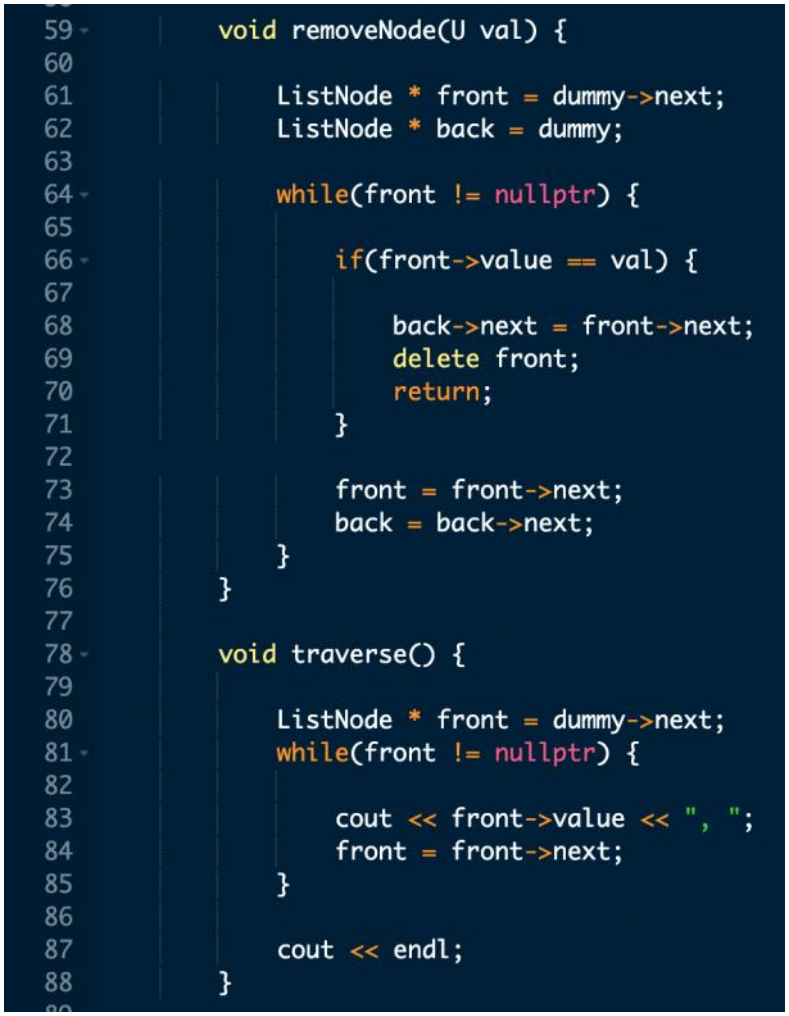


```
template <class T>
class Node{

    public:

        T value;
        Node * next;

        Node() {

            next = nullptr;
        }

        Node(T val) {

            value = val;
            next = nullptr;
        }
};
```

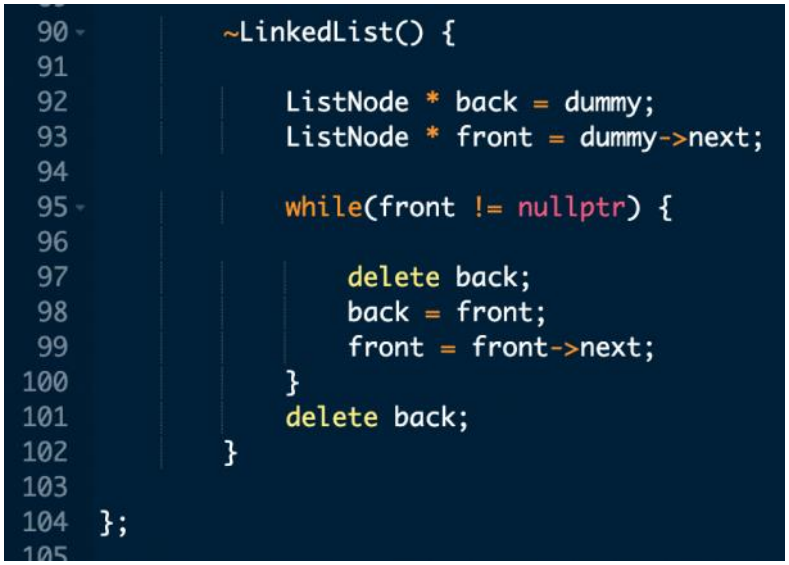
Using the above node as a template for further use in the Linked List



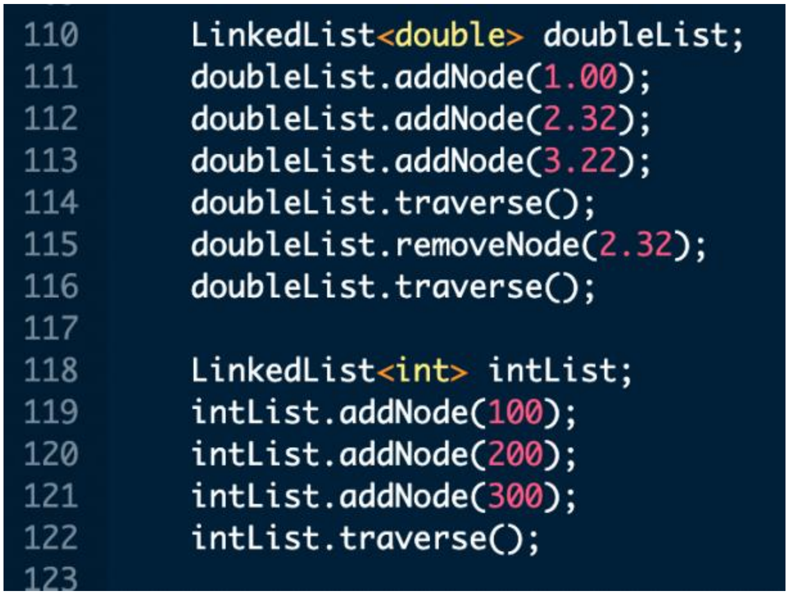
```cpp
template<class U>
class LinkedList {

typedef Node<U> ListNode;

    private:
        ListNode * dummy;

    public:

        LinkedList() {

            dummy = new ListNode();
        }


        void addNode(U val) {

            ListNode * front = dummy->next;
            ListNode * back = dummy;

            while(front != nullptr) {

                front = front->next;
                back = back->next;
            }

            back->next = new ListNode(val);
        }

```



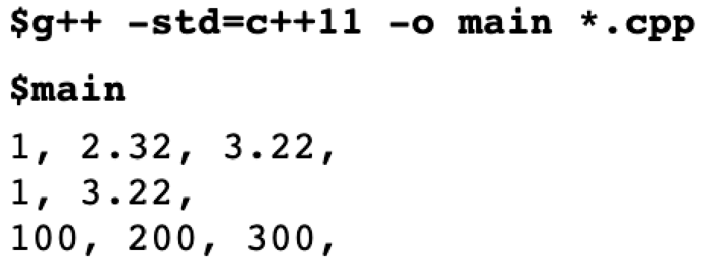
```cpp
        void removeNode(U val) {

            ListNode * front = dummy->next;
            ListNode * back = dummy;

            while(front != nullptr) {

                if(front->value == val) {

                    back->next = front->next;
                    delete front;
                    return;
                }

                front = front->next;
                back = back->next;
            }
        }

        void traverse() {

            ListNode * front = dummy->next;
            while(front != nullptr) {

                cout << front->value << ", ";
                front = front->next;
            }

            cout << endl;
        }
```



```cpp
        ~LinkedList() {

            ListNode * back = dummy;
            ListNode * front = dummy->next;

            while(front != nullptr) {

                delete back;
                back = front;
                front = front->next;
            }
            delete back;
        }

};
```

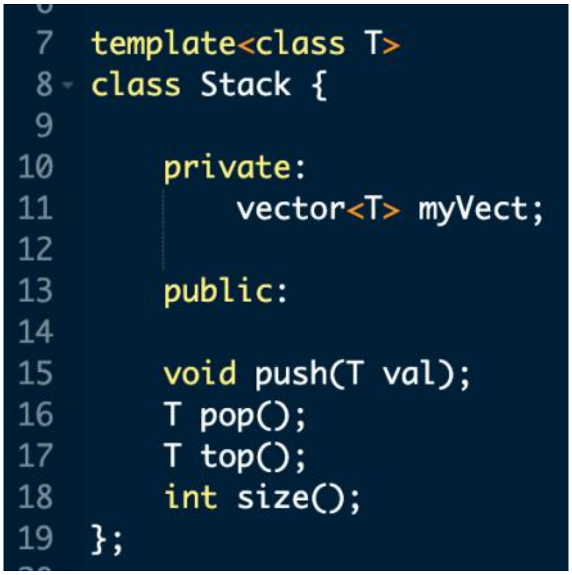


```
    LinkedList<double> doubleList;
    doubleList.addNode(1.00);
    doubleList.addNode(2.32);
    doubleList.addNode(3.22);
    doubleList.traverse();
    doubleList.removeNode(2.32);
    doubleList.traverse();

    LinkedList<int> intList;
    intList.addNode(100);
    intList.addNode(200);
    intList.addNode(300);
    intList.traverse();
```

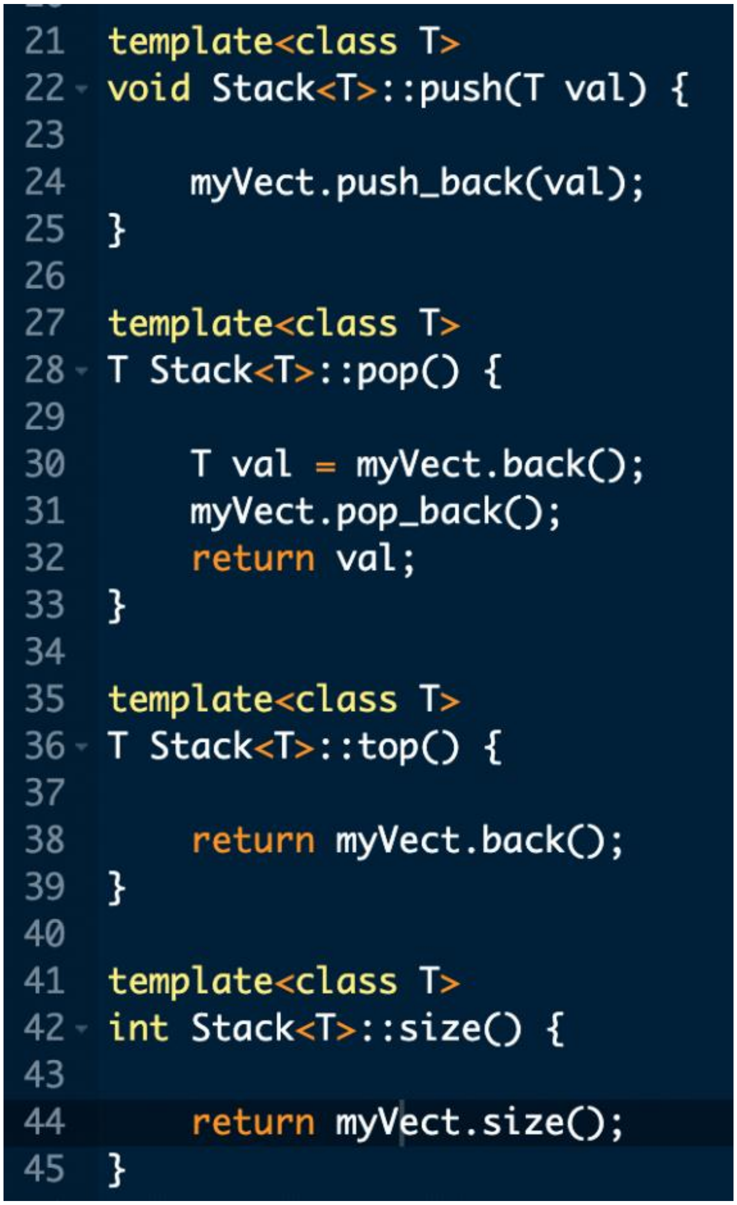


```
$g++ -std=c++11 -o main *.cpp
$main
1, 2.32, 3.22,
1, 3.22,
100, 200, 300,
```

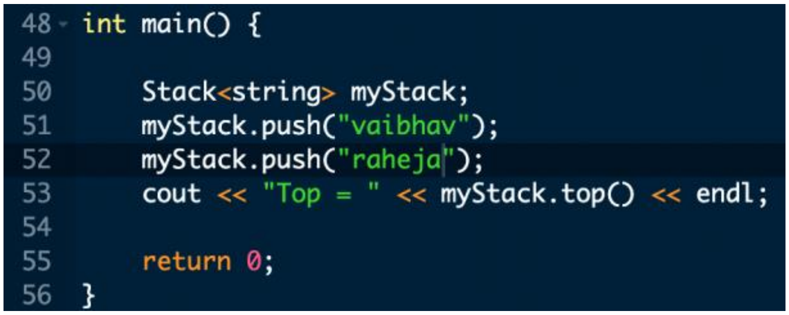
4. **General Template Stack using Vectors:**



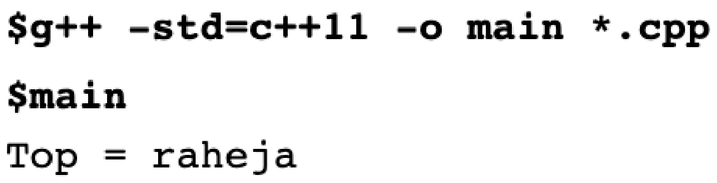
```
template<class T>
class Stack {

    private:
        vector<T> myVect;

    public:

    void push(T val);
    T pop();
    T top();
    int size();
};
```



```cpp
template<class T>
void Stack<T>::push(T val) {

    myVect.push_back(val);
}

template<class T>
T Stack<T>::pop() {

    T val = myVect.back();
    myVect.pop_back();
    return val;
}

template<class T>
T Stack<T>::top() {

    return myVect.back();
}

template<class T>
int Stack<T>::size() {

    return myVect.size();
}
```



```cpp
int main() {

    Stack<string> myStack;
    myStack.push("vaibhav");
    myStack.push("raheja");
    cout << "Top = " << myStack.top() << endl;

    return 0;
}
```



```
$g++ -std=c++11 -o main *.cpp
$main
Top = raheja
```