Lab 4: Template Stack

stack class

Implement template stack class that is capable of storing integer and string values. Write both header and implementation in one file (stack.h).

Private data fields:

- **data**: An array with maximum size of 20. (Declare a constant in stack.h called MAX_SIZE and set it to 20.)
- **size**: stores the current number of elements in the stack.

Public interface:

- **stack()**: constructs an empty stack.
- **push(T val)**: inserts a new element (val) of type T (T could be integer or string) into the data. If the data array is full, this function should throw an overflow_error exception with error message "Called push on full stack.".
- **pop()**: removes the last element from data. If the data array is empty, this function should throw an out*of*range exception with error message "Called pop on empty stack.".
- **pop_two()**: removes the last two elements from data. If the data array is empty or is of size 1, this function should throw an out_of_range exception. If empty then the error message should be "Called pop_two on empty stack." If the size is 1 then the error message should be "Called pop_two on a stack of size 1.".
- **top()**: returns the top element of stack (last inserted element). If stack is empty, this function should throw an underflow_error exception with error message "Called top on empty stack.".
- **empty()**: returns true if the stack is empty otherwise it returns false.

main.cpp test harness

Use this main.cpp file for testing your stack.

```
#include <iostream>
#include <string>
#include "stack.h"
#include <stdexcept>

using namespace std;


int main()
{
   cout << "Enter a test number(1-6): ";
    int test;
    cin >> test;
    cout << endl;
   //tests constructor, push, pop, top and empty
   if (test == 1) {
       try{
          cout << "\nstack1 constructor called";
          stack<int> stack1;
          if(stack1.empty()){
              cout<<"\nstack1 is empty.";
          }
          else{
              cout<<"\nstack1 is not empty";
          }
          cout << "\npush 10";
          stack1.push( 10 );
          cout << "\npush 20";
          stack1.push( 20 );
          cout << "\npush 30";
          stack1.push( 30 );
          cout << "\nstack1 top: ";
          cout<<stack1.top();
          cout << "\npop";
          stack1.pop();
```

```
            cout << "\nstack1 top: ";
            cout<<stack1.top();
            cout << "\npop";
            stack1.pop();
            cout << "\nstack1 top: ";
            cout<<stack1.top();
            cout << "\npop";
            stack1.pop();
            if(stack1.empty()){
                cout<<"\nstack1 is empty.";
            }
            else{
                cout<<"\nstack1 is not empty";
            }
            cout << endl;
        }
        catch(underflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }
    //tests top on empty stack
    if (test == 2) {
        try{
            cout << "\nstack2 constructor called";
            stack<int> stack2;
            cout << "\nstack2 top: ";
            cout<<stack2.top();
            cout << endl;
        }
        catch(underflow_error & e){
```

```
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }
    //tests pop from an empty stack
    if (test == 3) {
        try{
           cout << "\nstack3 constructor called";
           stack<int> stack3;
           cout<<"\npop from empty stack\n";
           stack3.pop();
           cout << endl;
        }
        catch(underflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }
    //tests push to a full stack
    if (test == 4) {
        try{
           cout << "\nstack4 constructor called";
           stack<int> stack4;
           cout << "\npush 20 elements";
           for(int i = 1; i <=20; ++i){
               stack4.push(i);
```

```
            }
            cout<<"\nstack4 top: ";
            cout<<stack4.top();
            cout<<"\npush 21\n";
            stack4.push(21);
            cout << endl;
        }
        catch(underflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }
    //tests stack of strings
    if (test == 5) {
        try{
            cout << "\nstack5 constructor called";
            stack<string> stack5;
            cout << "\npush A";
            stack5.push("A");
            cout << "\npush B";
            stack5.push("B");
            cout << "\npush C";
            stack5.push("C");
            cout << "\nstack5 top: ";
            cout<<stack5.top();
            cout << "\npop";
            stack5.pop();
            cout << "\nstack5 top: ";
            cout<<stack5.top();
            cout << "\npop";
            stack5.pop();
```

```
            cout << "\nstack5 top: ";
            cout<<stack5.top();
            cout << "\npop";
            stack5.pop();
            if(stack5.empty()){
                cout<<"\nstack5 is empty.";
            }
            else{
                cout<<"\nstack5 is not empty";
            }
            cout << "\nstack5 top: \n";
            stack5.top();
            cout << endl;
        }
        catch(underflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }

    //tests stack of strings
    if (test == 6) {
        try{
            cout << "\nstack6 constructor called";
            stack<string> stack6;
            cout << "\npush A";
            stack6.push("A");
            cout << "\npush B";
            stack6.push("B");
            cout << "\npush C";
            stack6.push("C");
```

```
            cout << "\nstack5 top: ";
            cout<<stack6.top();
            cout << "\npopping two items";
            stack6.pop_two();
            cout << "\nstack5 top: ";
            cout<<stack6.top();
            cout << "\npopping 2 items\n";
            stack6.pop_two();
            cout << "\nstack5 top: ";
            cout<<stack6.top();
            cout << "\npopping two items\n";
            stack6.pop_two();
            if(stack6.empty()){
                cout<<"\nstack6 is empty.";
            }
            else{
                cout<<"\nstack6 is not empty";
            }
            cout << "\nstack6 top: \n";
            stack6.top();
            cout << endl;
        }
        catch(underflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(overflow_error & e){
            cout<<e.what()<<endl;
        }
        catch(out_of_range & e){
            cout<<e.what()<<endl;
        }
    }
     return 0;
}
```