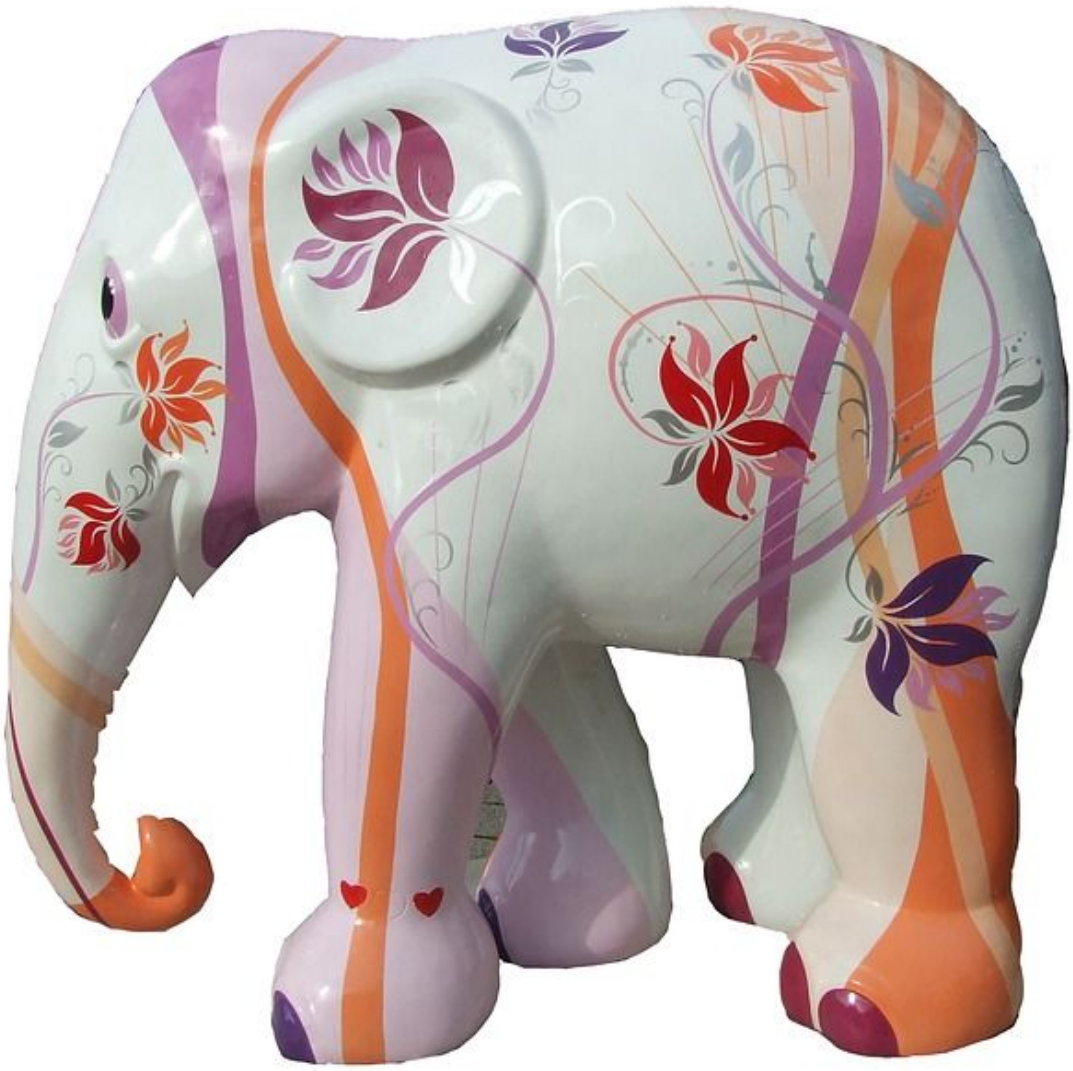


An Elephant Never Forgets...

Well, for starters, I have no idea if that's true.

I mean, if an elephant ever challenged me and said, "I never forget. So there", I'd be wise to say "Whatever you say, boss. Ain't no matter to me."

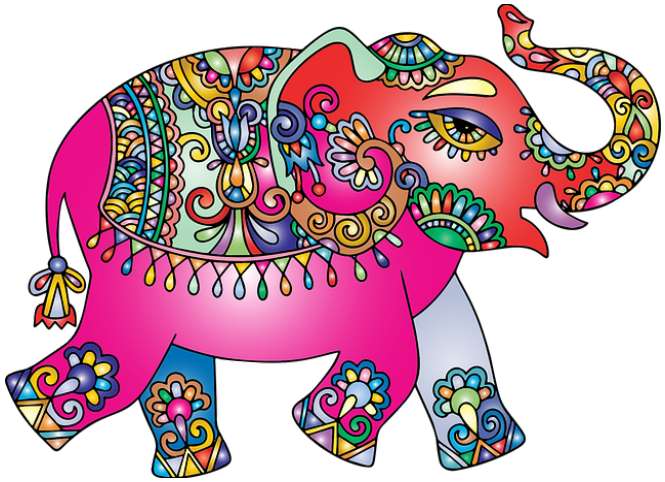
Anyway, that's the name of this quest (not its password).

In this relatively simple quest, you get to implement a stack abstract data type. In fact you get to implement two stacks for the price of one.

Some novel things you can discover in this quest:

1. Your entire implementation will be inline within the class definition. No need to create a separate .cpp file (obviously this is not a suitable strategy for everything you do, but you can get a feel for doing it this way so that in future you can make more informed decisions about how to segment your code.
2. You will actually implement both your required classes in the same header file. This header file will be called Stacks.h and should contain implementations of: `Stack_Int` and `Stack_String`.

Here's a cool thing about this quest. The code for your Stack_String class will be ALMOST identical to the code for your Stack_Int class. You'll find yourself copying and pasting huge chunks of your code. Or better yet, copying the whole thing and making minor edits here and there.



There's a reason I suggest doing this potentially redundant work. In the process of porting your Stack_Int into a Stack_String, you will find yourself thinking "Hmm... what a dumb mechanical thing to be doing. I wonder if there's a way to get the compiler to do it for me."

If so, that's awesome. As you continue your C++ romance into 2B, you'll find ways to get the compiler to do exactly that.

Here's another cool thing. You don't have to implement constructors or destructors for your classes. Instead, you will leverage the luxury of the compiler that gives you default ones for free. Your Stack_Int class should have a single private data member called _data which would be a

std::vector of ints. You will leverage standard library methods provided by the vector class to accomplish all of what a Stack does without compromising performance.

Stack Overview

A stack is what you may call a Last In First Out (LIFO) data structure. Typically a stack is implemented using a far more versatile data structure by stripping away(or hiding) some of the things you can do with the more versatile data structure.

For example, consider an array, which allows you to pretty much do whatever you want to any place in it - You can assign values to arbitrary locations in the array. But you can also take an array and say, "I'm going to wrap up this array within my own container and prevent others from accessing its entire power. Instead, I'm only going to allow users to access one end of the array."

You may be thinking, "Who in their right mind would do something like that?"

I mean, why would you take something as powerful as an array and dress it down so someone can only use a small subset of the features it provides? That's like, if I go on Amazon and see 10-packs and 2-packs of something I like for the same price, me saying "Wow! *That's really cool. Lemme buy five two-packs instead of a ten-pack.*"

Incredibly, as we have discovered time and time again, you gain an enormous amount of creative power by sacrificing a small bit of freedom. In effect, you're saying "I hereby sacrifice my ability to do random crazy things to my array. So give me a version of the array that doesn't let me do that, and in return absolves me of the headache of having to remember potential code-gotchas involving invocations of these (self) forbidden functions. You will be pleasantly surprised at how creative you can get once you stop worrying about a whole bunch of things that could break (but now won't because you've hidden them away from prying eyes). This, basically, is the core insight behind Object Oriented Programming.

So, we'll do exactly what far-sighted common sense tells us. We'll use a powerful array (vector) under the covers to implement everything that a stack interface provides to its users. But we'll hide the fact that it's an array from the user. Thus they can only ever manipulate it using our own public functions as an intermediary. Since we have absolute control over our own public methods, we get to decide what users can and can't do to our underlying data array.

The following functionality is what you will implement for your stack. Each of the following is a miniquest worth a possibly different number of rewards. Some of these are critical miniquests. That means you can't proceed further in this quest or to the next one unless you complete it. Some other miniquests will let you proceed, but you may not get the rewards for the failed miniquests.

Here is the specification for your Stack_int class:

```
class Stack_Int {
private:
    std::vector<int> _data;

public:
    // No explicit constructor or destructor. You must implement the remaining
    // methods below in-line (directly in this header file) within the class-def.
    // See the starter code for more detail.

    size_t size() const;
    bool is_empty() const;
    void push(int val);
    int top(bool& success) const;
    bool pop();
    bool pop(int& val);
    string to_string() const;
};
```

As you can see, it's quite simple. Below are your miniquests. There's at least one for each of the public methods you must implement.

Your first miniquest - Check if your stack is empty

Implement:

```
bool Stack_Int::is_empty() const;
```

It should return true if the stack (_data vector) is empty and false otherwise. Just return the value of the `vector::empty()` method. It's a one-liner. Hopefully the first of countless one-liners you will write in your programming life.

Your second miniquest - Check your stack's size

Implement:

```
size_t Stack_Int::size() const;
```

It should return the size of the _data vector. Another one-liner! Just return the value of the `vector::size()` method.

Your third miniquest - Push

Implement:

```
void push(int val);
```

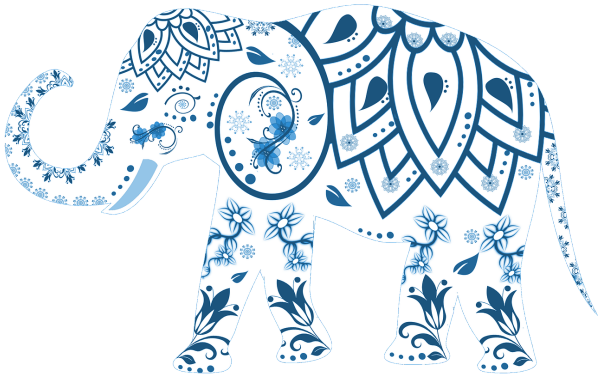
It should insert the given value into your stack. Choose which end of the stack you're going to treat as the top of your stack. This will become important as your stack grows in size (Why? Discuss it in the forums.)

Your fourth miniquest - Top

Implement:

```
int top(bool& success) const;
```

Note the unusual signature I've chosen for this method. What do you think is the reason for this? What problem am I trying to solve? What are the pros and cons of doing it this way as opposed to some other way?



When called on a `Stack_Int` object it must return the value currently occupying the top of the stack, without changing the stack data in any way (hence the const qualifier in the method signature).

If the stack is empty, you should return 0 and set the boolean parameter to `false`. Note that it's passed by reference. Otherwise, it should return the top element and set the parameter to `true`.

Your fifth miniquest - One kind of pop

Implement:

```
bool pop();
```

This method simply removes the top element off the data vector and return true.

You can do this using a one-line `std::vector` method. If the vector is empty, you would not do that and return false.

Your sixth miniquest - Another kind of pop

Implement:

```
bool pop(int& val);
```

This kind of `pop()` is virtually identical to your `top()` method, except that it has the side-effect of removing the element that is returned if the stack is not empty.

Your seventh miniquest - Stringify

Implement:

```
string to_string() const;
```

This method must return a string representation of the stack of integers in the following exact format. As usual, I've colored in the spaces. Each gray rectangle stands for exactly one space.

```
Stack (<N> elements):
<element 1>
<element 2>
<. . .>
Elements, if listed above, are in increasing order of age.
```

The parts in red above (also in angle brackets) must be replaced by you with the appropriate values.

There is no newline after the last line that ends with "`age.`"

You would print a maximum of 10 elements this way. If the stack has more than 10 elements, you must print the top (most recent) 10 and then print a single line of ellipses (`...`) in place of all other elements.

Your eighth and final miniquest - Stack o' Strings

You get to implement a whole other class: `Stack_String`.

Copy the code for your `Stack_Int` class and adapt it to work with strings instead. This is an exercise in porting code which requires a level of understanding about what the code does.

Your `Stack_String` should do everything your `Stack_Int` does... except, with strings.

Starter code

You will submit one file: `Stacks.h`. Here is `your` starter code

```
// Student ID: 12345678
// TODO - Replace the number above with your actual student ID
//
#ifndef Stacks_h
#define Stacks_h

#include <vector>
#include <sstream>

class Stack_Int {
private:
    std::vector<int> _data;

public:
    // No explicit constructor or destructor
    size_t size() const {
        // TODO - Your code here
    }

    bool is_empty() const {
        // TODO - Your code here
    }

    void push(int val) {
        // TODO - Your code here
    }

    int top(bool& success) const {
        // TODO - Your code here
    }

    bool pop() {
        // TODO - Your code here
    }

    bool pop(int& val) {
        // TODO - Your code here
    }

    std::string to_string() const {
        // TODO - Your code here
    }

    // Don't remove the following line
    friend class Tests;
};

class Stack_String {
      // TODO - Your code here. Ask in the forums if you're stuck.
};

#endif /* Stacks_h */
```

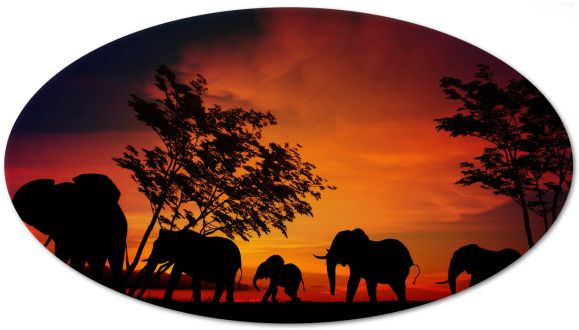
Testing your own code

You should test your functions using your own `main()` function in which you try and call your methods in many different ways and cross-check their return value against your expected results. But when you submit you must NOT submit your `main()` function. I will use my own and invoke your functions in many creative ways. Hopefully you've thought of all of them.

Submission

When you think you're happy with your code and it passes all your own tests, it is time to see if it will also pass mine.

1. Head over to https://quests.nonlinearmedia.org
1. Enter the secret password for this quest in the box.
2. Drag and drop your `Stacks.h` file into the button and press it. (Make sure you're not submitting a `main()` function)
3. Wait for me to complete my tests and report back (usually a minute or less).



Points and Extra Credit Opportunities

I monitor the discussion forums closely and award extra credit points for well-thought out and helpful discussions.

May the best coders win. That may just be all of you.

Happy Hacking,

&