# CO2401 Advanced Programming with C++

## Lab Worksheet 5 – Stacks

## Introduction

This worksheet starts with the *Stack* class Chiara talked about in the lecture. You can find the code on the lecture slides.

* Again, you are using pointers and dynamic memory to create a *Stack* variable.
* Remember:
  + **An** **Object is an instance of a Class!**
  + **A pointer is only a variable, which’s value is a memory address.**

## Essential Exercises

**Exercise 1: starting with the Stack class**

Copy over the code for the Stack class from the lecture slides. Compile and run it.

One aspect of object-oriented code is that the user does not need to know the details of *how* the class has been implemented.

To start with just try pushing and popping data within main in order to see what happens.

Now examine the class member functions. Make sure you understand what they are doing. Try popping data when the stack is empty and pushing data when the stack is full.

**Exercise 2: use dynamic memory**

As we talked about pointers and dynamic array the last couple of weeks, I want you to change the whole Stack class so it is using pointers. That means mData is an array of pointers and mTop is a pointer, pointing to the top of the stack.

Additionally, the array should be dynamic, meaning that you’d determine the size of the array by whatever you set by using the constructor.

**Tip:** Remember pointer arithmetic (Week 3&4 lecture, slide 23)

**Exercise 3: write a GetLength function**

Write a new public member function that returns the length of the stack (as top is now the pointer to the top element of the stack, have a think about how to keep track of the size!).

Note that GetLength is a "getter" and, as such, you could refer back to the first week’s lecture slides.

**Exercise 4: displaying the stack in reverse order**

In the lecture slides I gave a member function that displays each data item on the stack in order from bottom to top.

Write a new member function that displays each data item on the stack from top down to bottom, i.e. in the reverse order to the DisplayStack function.

**Exercise 4: CountData**

Write a new member function called countData. The function prototype looks like this:

int CStack::CountData( string searchString );

Remember to place the function prototype inside the class definition and then write the definition of the member function separately.

The new member function counts the number of times the given string can be found on the stack. The function returns 0 if no instances of the given string can be found.

## Another Essential Exercise

If you haven’t done the pointers quiz on Blackboard yet, it would be a good time to do it now. This might sound like a suggestion, but trust me, it isn’t. DO THE QUIZ!

## Additional Essential Exercises

As reflection week is around the corner, I’ll give you some extra exercises, so you won’t forget where to put a semicolon…

I want you to use your own stack to implement either of these coding concepts:

* Infix to Prefix/Postfix
  + Here’s a link for with a converter and an explanation: <https://www.web4college.com/converters/infix-to-postfix-prefix.php>
* Depth-first search
  + Here’s a link for an explanation:  
    <https://www.codecademy.com/article/depth-first-search-conceptual>

These are just example links, and you’d probably do more research before you give it a go.

**Pro tip:** I’m not a fool, and I know there are probably thousands of implementations of both concepts which you can find online and just copy. You can do that; I really don’t care. However, at the end of the day you’re just kidding yourself as you’re missing out on good programming exercises. And trust me, you need it. This exercise is not about completing a task and ticking it off, it is about learning. You won’t learn programming by just attending the classes (or not even that) and trying to pass. You need to exercise is regularly. You can trust me with that too! Or just keep fooling yourself.

**Exercise 5: findData**

Write a new member function called findData. The function prototype of the function is:

int findData( string searchString );

Remember to place the function prototype inside the class definition and then write the definition of the member function separately.

The new member function finds the first occurrence of the given string. The function returns the index of the array at which the first occurrence of the string is found. If the string is not found on the stack then return -1 to indicate not found.

Note that the string class allows the double-equals as the test for equality, so implementing the find operation should not be too problematic.

Try to write the function as efficiently as possible - the return statement allows a dynamic exit from a loop. In this way you can use return to exit from the loop (and the function) as soon as the first instance of the string has been found.

**Exercise 6: display nth element**

Write a new member function that displays the nth element of the stack. In other words, the function is given a number and it then displays the data at that array index number. Obviously the user could specify a number that is outside the bounds of the stack so you need to ensure that the number is within the bounds of the stack. Write an "out of bounds" error message if the number is outside the bounds of stack.

The function prototype of the function is:

void displayN( int n );

Remember to place the function prototype inside the class definition and then write the definition of the member function separately.

**Exercise 7: find nth element**

This is a variant on the previous exercise. Write a new member function that finds and returns the nth element of the stack. In other words, the function is given a number and it then finds the data at that array index number. The found string is assigned to a reference parameter (why?). Obviously the user could specify a number that was outside the bounds of the stack so you need to ensure that the number is within the bounds of the stack. Return false if the number is outside the bounds of stack, otherwise return true.

The function prototype of the function is:

bool findN( int n, string &foundString );

Remember to place the function prototype inside the class definition and then write the definition of the member function separately.

**Exercise 8: all the same...**

Write a function called allSame. The function returns true if all the elements on the stack are identical to one another, otherwise it returns false.

## Advanced Exercises

**Exercise 9: constructors**

We mentioned in the lecture that the job currently carried out by the initialise function would be done using a constructor.

Delete the initialise function. Write a constructor for the Stack class that initialises top to 0.

**Exercise 10: a copy function**

It would be useful to have a function that copies the contents of one stack onto another stack.

In order to do this you need to pass an object as a parameter. A class object can be passed as a parameter in the same way as any other data type. The function prototype would look like this:

void Stack::copyStack( Stack\* newStack )

I am assuming that copyStack makes the new stack an exact copy of the old stack. In other words, inside the copy function you initialise the new stack (so that it has a top of 0) before the copy operation occurs.

You could write a variant on this in which the function concatenates the contents of one stack with that of a second stack.

**Exercise 11: starting thinking about copy constructors**

A class can have more than one constructor. Write a second constructor for the Stack class that takes an existing stack object as a parameter and initialises the new stack to the values held on the existing stack.