CS319 – Scientific Computing

# Lab 3: Classes

The goal of the lab this week is to further develop the `stack` class that we introduced during last week's lectures, which you can download from `http://www.maths.nuigalway.ie/~niall/CS319/Week04/03StackConstructor.cpp`.

## 1 The `stack` Class

(i) Review the `stack` example from Week 4. Read it carefully and make sure you understand how it works.

Modify it so that it uses a *destructor* to de-assign memory for the `contents` array.

(ii) Add a member function (method) that checks if the stack is empty. Return type should be `bool`.

(iii) The maximum number of items we can add to the stack is controlled by `MAX_STACK`. Add a method that can be used to check if the stack is full. Again, the return type should be `bool`.

(iv) Change the `stack::pop()` code so that, if you try to pop an empty stack, it gives a warning message and returns `NULL`.
*Tip:* Rather than writing the warning to `cout`, write it to `cerr` – the standard error stream.

(v) Modify the `stack::push()` code in a similar way, so that you can't put more items onto a stack than there is space for. Here (for now) you don't need a return value.

## 2 The `string` class

Now we'll look at the "built-in" `string` class in C++. By including the `string` header file, the programmer can declare objects of the class `string`, for storing words etc.

There are numerous methods associated with `string`. For example, if *`Name`* is of type `string`, then `Name.size()` returns the number of bytes used to store it (which is essentially the same as the number of `char`s stored in it). The return type is `unsigned int`.

Because of *operator overloading*, we can do the following with `string`s

- use = to perform an assignment.
- use == and `!=` to compare two strings,
- add a single character to a string using +=
- access the $k$th character of, say, a string called *`Name`* with `Name[i]`

We'll study this in more detail in a few weeks time.

## 3 Exercises

Q1. Test your `stack` class, and use of strings, by writing a program that prompts the user to input a word and then checks if it is a palindrome.

Q2. Test your `stack` class by writing a programme that prompts the user to input an mathematical arithmetic expression, and checks if it is correctly parsed. For example, the following is correct:
$(1 + (3 - 5))/(123 + x)$, but $(1 + (3 - 5)/(123 + x)$ is not. At the very least, your code should check that all parenthesis are matched.

Q3. One can use a stack when converting a number form decimal to binary. This is described at
`http://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queues`;
see Section 1.2.1

Test your class stack by writing a programme that prompts the user for an (unsigned) integer, and then calls a function that implements the algorithm described in WikiBooks to compute a `string` that is the binary representation of the decimal integer.

## 4 FYI: Applications of stacks

Some classic applications of stacks include

- Solving puzzles, such as the Towers of Hanoi problem;
- Evaluating expressions in programming and mathematics, particularly ones involving lost of parentheses;
- Syntax checking. For example, checking through the source code of a C++ program to verify that all `(`, `{`, `[` and < are matched with the closing counter parts and in the right order.

## 5 Homework

Submit your solution to **Q3** through the labs section of Blackboard by 9am, Monday 13th February. Next week, you'll develop a more sophisticated application for your `stack` class, but first we'll need to study how to manipulate files in C++.