Notes on constant referencing (`const &`)

Constant referencing, i.e. `const &,` means that a memory address is passed.

When we use `const &` for a parameter, we indicate that we want to use a reference parameter **but** that the parameter may not be changed inside the body of the function.

Remember that a memory address is a pointer. For a reference parameter, the memory address of the parameter (a pointer) is passed when the function is executed. This allows the function to change the value of the parameter in the main function or calling function. When we add `const` to that, we still pass the memory address, but we indicate that the value of the parameter may not be changed. This cause the `const &` parameter to be treated like a value parameter.

We use a constant reference (`const &`) to save memory space and time, because the compiler does not have to copy the entire parameter to new memory space for the function, as it would do for a value parameter.

See for example the difference between the following functions:

```
void aFunction(anObject a)
{
   //do something
}

void anotherFunction(& anObject b)
{
   //do something
}

void yetAnotherFunction(const & anObject c)
{
   //do something
}
```

In function `aFunction` we pass object **`a`** as a **value** parameter. This means that the compiler will copy the object `a` to the stack frame in CPU memory where the function is executed, using a certain amount of CPU time and memory space. The function may change the parameter, but this will be local to the function, so that no changes to the actual parameter will be in effect after the function has executed.

In function `anotherFunction` we pass object **`b`** as a **reference** parameter. This means that the compiler will pass the memory address of the object `b` (a pointer) to the stack frame in CPU memory where the function is executed. The only additional time or memory space used, is for passing the memory address or pointer and the time necessary to do that. This also allows function `anotherFunction` to change the value of the parameter and return that value to the calling function. Any changes made to the parameter, will therefore be reflected in the value of the actual parameter after the function has executed.

In function `aFunction` we pass object **`c`** as a **constant reference** parameter. This means that the compiler will pass the memory address of the object `c` (a pointer) to the stack frame in CPU memory where the function is executed. The only additional time or memory space used, is for passing the memory address or pointer and the time necessary to do that. The `const` modifier will prevent any changes to the parameter, so that no changes to the actual parameter will be in effect after the function has executed. We have saved memory space and time, and protected the actual parameter against any changes.