Effective C++

Note: these examples go from quite simple to some mildly tricky features. If your C++ is rusty, try the basics first, otherwise skip to “Logging …” and on to “Effective STL” and onwards.

Recap of C++

- 01 A simple example using everything you should know

- This program recaps the content of the primer, to make sure you’re ready to go
- We’re going to code a simple “guess the number game”, which will use stream IO, loops and conditionals

Info:

If you’ve never used stream *input* in C++ there’s two things to know - first you can read from the terminal into a given variable type using `` `std::cin >> my_var;` `` and the value the user enters will be interpreted to the type of the given my_var. If the value is incompatible (e.g. you want an int and the user presses ‘a’) the stream will be put into an error state. C++ requires that a stream in this state, interpreted as a boolean, will be false, otherwise it will be true. So we can use it as the condition in an if or in a while loop, and the body will run only when valid input is given. You can also test the stream directly with functions such as .fail()

Try:

Write a program which asks the user to guess a number, and tells them whether their guess is too big or too small. You can use any range you like, but 0-10 is a good example.

You can exit the program if the user does not enter a number at all, or you can give them another chance.

You can hard-code the number-to-be-guessed, or look up how to pick a random number, or do something like check the current time and turn this into a number somehow

- 02 The classic interview problem - FizzBuzz

- This is a classic "can you program in language x" question, because it uses loops, conditions and simple problem solving. For this reason, it's a really GOOD familiarisation exercise.
- The problem is as follows: Your code will output the numbers from 1 to 100 in order, BUT if the number divides by 3, it will INSTEAD print the word "Fizz", and if the number divides by 5, it will INSTEAD print the word "Buzz". If the number divides by 3 AND 5, it will print just the word "Fizzbuzz".

Info:

There are many possible ways to solve this problem, although most people immediately go to one of two basic approaches. For one of these, you might find the Modulo division operator, '%', helpful as this means "give me only the remainder from this division", so for example `i%2` will be 0 for even and 1 for odd numbers.

Try:

Write a simple program to meet the brief above. If you have any trouble with the algorithm, please do ask!

If you like, try and find some different or more esoteric approaches. Try finding a different way to think about the problem, or to characterise what the output should be. (My best C/C++ attempt is 3 lines of actual code with no conditionals…, but it is also terrible C++)

Logging - A simple idea with myriad implementations

- 03 A simple "logging" function

- Lots of libraries exist to do "proper" error logging. We're ignoring those for now. Consider the much simpler task of adding "severity" to errors.
- Don't worry about the fancy aspects, such as colour, font, size etc. Let's just propose 3 levels of severity - "Info", "Warning", "Error"

Try:

Write a function which takes a string and sends it to std::out. This should also take a single integer parameter, of 0, 1 or 2 for the severity level, and pre-pend (display before) the message above (with punctuation etc as you like).

For example log(“Hello world”, 2) -> “Error: Hello world”

This is hopefully trivial, but it’s a good first step and we’ll expand on it below

Getting the Hang of Reference and Auto

- 04 Reference as a return type

- Sometimes rather than returning a value, which will be copied* into the “capturing” variable (the one the result gets assigned to), we want to return something else
- For example, if we have a string, we may wish to access a character in it directly, and read/write it. Similarly for an array, a vector etc. A function to access an element that way must return us a “reference” not a copy, otherwise we couldn’t change the value in place
- Or suppose we have good reasons to want to create something inside a function and return it, we want it to be on the heap (i.e. created with new), and we want to avoid pointers

Info:

*Actually, this is not true. One thing a compiler is allowed to do is to “elide” or skip copies in certain circumstances, called “copy elision”. For a return from a function, the compiler can decide to skip the copy by filling in the result directly. Normally this is completely invisible, but it is one of the few things where that “As If” rule we’ve mentioned can be violated - even if the copying has observable side effects, it can be skipped! Look up Return Value Optimisation (RVO) or Copy Elision for more.

Try:

We’re going to write a very silly function just to see how we would do this! Write a function, get_element, which takes an array of type int and an integer parameter i and returns the i-th value of the array by reference. Consider what you will do if i is outside the array bounds. For this, you can use a global const int for the length of array, and then use `int arr[len];` to create one.

What happens if you capture the return from your function into a plain int? What about an integer reference?

Can you understand the following line of code?`get_element(arr, 1) *= -1;` Why can the function go on the LHS of the expression here?

- 05 Wrinkles in reference variables

- Sometimes, often for clarity, we want to refer to an existing variable using a different name - we can use a reference to do this
- For example, we might want to select something using a complicated rule, and then change it in place (c.f. returning a reference before)
- There is a tiny bit of trickery in this - a reference must be initialised on the line where it is created, and an if/else block will open a new scope

Try:

Suppose we have 3 integer values, x, y, and z. Part of our program has to select between x and z according to whether x+y < z or not. We decide to use a reference to do this.

Why will the following code block not solve this problem in general? Hint - what is the scope of my_ref?

```
if(x+ y < z){
     int & my_ref = x;
}else{
     int & my_ref = z;
}
```

There are two ways we can fix this. We just saw one - we could write a function that takes x, y, and z (x and z at least by reference) and returns a reference to the correct one. Write this. Does it do what you expect?

The other way uses a special operator called the ternary, or inline-if, which can choose between two options and "return" the result. Rather than if-else, the syntax is "`condition ? true-result : false result`" This lets us do this:

```
int & val = (x + y < z ? x : z);
```

Can you see how this works? Is this readable code to you? Sometimes the ternary is useful, but it has to be used with caution!

- 06 Auto as a variable type

- Recall that auto as a variable type can be used when a variable is initialised on the same line that it is created, AND the initialiser unambiguously dictates the type

- Do keep in mind that sometimes unexpected things can happen, especially with code that is being actively developed, so don't use auto "just because" it's newer. Sometimes being explicit is best.
- Also remember that you'll need to compile to at least C++11 or newer - some compilers require this explicitly, others are more forgiving

Try:

Take some of the code you've now written and see where you could use auto instead of explicit types. Watch out for references! That's not technically part of the type! How can and should you tackle that?

Recall that auto pretty much just "stands in" for a basic type, like int, double etc. Experiment with adding extra qualifiers to the type, such as const-ness.

Are there places where auto could confuse you? Is this a sign of bad code?

- 07 Extra: Auto as a function return type

- If a function is sufficiently simple, the compiler can infer what type is returned from the function body. It has to be the exact type (no conversions allowed)
- This is very powerful if you "wrap" one function with another
- Go back to that logging function above. Suppose instead of writing to std-out we call a function, call it "write" which does this for us.
- This would let us e.g. change how this part works without changing other places in the code - to change from stdout to stderr for example, or to file based io.

Info:

Here's what the log function could look like:

```
??? write(std::string message){std::cout<<message;}
```

But what can go in place of the ??? For that function body there's nothing much to return, but we might write it to return a bool or int status code etc

Try:

Modify your logging function to use the definition of 'write' above, with void return type. You'll want to insert the extra text into the "mess" string and then pass the whole thing to "write". Your function should return whatever the write method does and you should use auto return type. You'll need to compile to C++14 or newer for this!

Now modify the write function to return an error code. You can return a dummy value for now. Do you have to modify your wrapper function at all?

Beware: in this example we are using auto to "patch over" the fact that our interface is not well specified and might be changed, which is not a great idea in general. But we're also using it to avoid repeating redundant information (return types of write and log), which is a good thing. This example is just a demo, and it's the second point you should take away, not the first!

Effective STL

- 08 A simple problem with map

- The map data structure is used to associate a value with a (unique) key, while being fast to insert-into and look-up from. They can be iterated, but the order shouldn't be considered meaningful, so this is usually done just to go through every item
- NEVER iterate a map to find an item by key! You can just look it up! If you need to find items by a more complex property (e.g. partial key matching, find by value etc), you should probably use something like std::find or std::find_if

Try:

Create a map from string to string and put some data into it. For example, a program config might contain things like "location" and "language".

What happens if you use the [key] syntax to look up an item that is not in the map? How might you get around this?

Try iterating over the map, using whatever approach you wish, and printing each key-value pair. Remember that map items are given back to you as pairs, so item.first and item.second will give us the key and the value respectively.

Try creating another map, this time from string to int, with some data in. How would you increment one of the mapped values?

Tricky: how might you find all the items whose value contains a specific letter? There is a function to do the finding, std::find or std::find_if, but we're going to need a special “comparator” to decide if the value matches that criteria. We can declare a “lambda” function to do this, like: `auto compare = [](auto item){…}` where the body should contain code to return true or false. The find_if function will call this function with each map entry in turn, so inside the function, item.first and item.second will give us the key and the value respectively for a single map entry. Can you finish the code to do this search? Again you’ll need C++14 for this syntax.

- 09 A Caesar cipher

  - The traditional Caesar cipher, or monoalphabetic substitution cipher encodes text by replacing every letter with the letter from N digits later in the alphabet, looping around at the end
  - I.e. if we use an N of 2 then A -> C, B -> D, C -> E etc
  - We could obviously use either a vector of letters, or we could use a map. What are the pros and cons of these options? Try both and see.
  - Hint - if you are reading in the message to cipher, you will need to use std::getline to read something with spaces. std::cin >> str will stop at the first space. std::getline(std::cin, str); will read until Enter/Return is pressed.

Info:

For the interested: monoalphabetic ciphers are very weak to frequency analysis - counting occurrences of each letter - which most languages have very characteristic patterns for

Try:

Write a program which takes a message (hardcoded, command line or interactive input is fine), ciphers it and shows the result. You can assume all upper-case, and simply preserve any punctuation.

Try to use a solution which exploits the containers and algorithms in the STL, rather than brute force. We suggest trying a vector and a map separately, and considering the pros and cons.

To save you some typing, here’s the list of comma-separated letters which you can use to initialise a vector or plain array {“A”, "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", “M”, "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", “Z”}

Hint - for a string, you can use either [] or the at() method to get one character at a time, or a for(auto :) style loop. You will get a character though, not just a one-element string, so you might need to make a string from it like this: std::string{character}

Hint - you can create a shifted vector of letters in several ways OR you can use just the one vector and look up each letter by index and then look at index+shift (taking care to wrap around at the end).

Hint - to initialise a map for the cipher you probably want to create a vector or array first and then programmatically create the map almost the same way you dealt with messages

Fun with Templates

- 10 Multiple argument types, one function body

  - In the logging function above, we only took a string for the message. If we want to take several types we'd have to write one function for each, even though the body would be exactly the same for them all

  - This is obviously silly - we'd have a lot of duplication for no good reason. Templating is exactly what we want to fix this

Try:

Expand the logging function we wrote above to take any sort of argument using a template. If you keep using the "write" intermediary we had above, you will have to template this too, otherwise revert back to the original function that writes to stdout directly

What will happen if somebody called your function with a type that doesn't support stream io, printf, or whatever method you are using to show to the screen?

- 11 A simple template specialisation

  - Sometimes there are certain argument types where we want to have a different function body. We can do this by providing a "specialisation" of the template

  - For example, suppose writing was a costly operation and we would prefer not to invoke it twice without need: we might decide to implement the "string" argument version of our logging intermediary by pre-pending our extra string to the argument and writing the whole thing once; but we might not want to "stringify" other types of argument explicitly, so those would then be treated differently

Try:

Take your function from the previous example and add a string type specialisation. Remember to make sure it is a template specialisation, and not a function overload (ask if this is unclear)

You'll need to make sure you call the function with an actual std::string. HINT: "Hello world" the literal is NOT.

Can you think of other circumstances where one or a few argument types might be handled specially, but the majority are the same?

Do you see the difference between a templated function, a specialisation, and a function overload?

Simple Classes and OO

- 12 A cipher class

- In the Caesar cipher program, we need some data to en/de-cipher the message and this was probably either some global state, or it would be done afresh for each message
- A better, more flexible program could use a class for the cipher object, with en/de-cipher methods, and containing the alphabet data vector, and/or cipher map as member state

Try:

Write a class that:

- has methods to take a message string and encipher/decipher it
- can be configured with any value of shift from 1 to 26, and **does not** need configuring each time a message is enciphered or deciphered
- optimises the time taken to encipher/decipher (i.e. stores the shifted data or map internally)

You can use the vector or map based cipher algorithm, or any other approach you wish that meets the second and third bullets. Note that the vector version is quite

symmetric in encipher/decipher, but we either need an encipher and decipher map, or one direction will be slower and more tricky.
Also write a quick "driver program" which takes a message from a user (std::cin with a string for example), enciphers it, deciphers it, and prints the message, the cipher text and checks that the round-trip has given the same message back. Include a sensible error report if somehow encipher-decipher has not worked.

- 13 A simple stateful class (with templates)

    - The logging example we've been using is a good example of where we might want to "wrap" some function in a class to add behaviour
    - We can use class members for things like the text strings we are adding, and implement functions to do the work
    - Templated class member functions will save us a lot of typing, just as we saw in example 10.

    Try:

    Write a class, called e.g. Logger which has a log member function just like we wrote above. However, now the class should internally track the number of times it has been called with severity of 1 or of 2. Use the templated version of log for flexibility

    Add a member function to display the call stats info. You can also add an internal name identifier and print this along with the stats

    For the counters, you can initialise these in their definitions (because 0 is always the right initial value). For the name, you'll need a constructor with parameter

    Write a small main program which creates two loggers, logs some stuff and displays the stats. Verify that the two instance do not affect each other. In other words, data is now bound to the instance and is NOT global, which is a good thing.