First Programs in C++

Note: model solutions are given for these, but keep in mind there is often more than one way to do a thing, so these are really only a guide

Getting Started

- 01 Before we code

- When developing programs we want to compile them as often as we can. Tackling errors one at a time avoids having a million issues to try and resolve, many of which might depend on the others
- Turning on compiler warnings is also important as this will help us spot issues before they become bugs

Info:

These instructions assume you are on a system with a C++ compiler. Any linux or Mac system probably does. On windows we can use the WSL or try VSCode. Alternatively, for simple programs like this, use your favourite search engine to look for an online compiler

Try:

Get hold of the "HelloWorld.cpp" program from the examples pack in the Exercises directory. Compile and run this.

Compiling is probably going to look like `g++ HelloWorld.cpp -o HelloWorld` to create a program called HelloWorld

You can then run this using `./HelloWorld` and see what it says

Only once you can do this, go to the next steps

- 02 Errors

- The compiler will try to tell us what is wrong when we write bad code. These messages can be very helpful, but sometimes it can feel like the compiler is trying to make your life harder. We promise it isn't!

- Learning to read an error message is an art! Since the compiler can't read your mind, you have to try and put yourself in its place instead

Try:

The program "err.cpp" is an example of a compiler error. Here's a few results when I compile this:

Clang (OSX default compiler) and current Intel

```
<source>:4:3: error: use of undeclared identifier 'std'
    4 |   std::vector<int> a;
      |   ^
<source>:4:18: error: expected '(' for function-style cast or
type construction
    4 |   std::vector<int> a;
      |               ~~~^
<source>:4:20: error: use of undeclared identifier 'a'
    4 |   std::vector<int> a;
      |                    ^
3 errors generated.
Compiler returned: 1
```

Gcc

```
<source>: In function 'int main()':
<source>:4:8: error: 'vector' is not a member of 'std'
    4 |   std::vector<int> a;
      |        ^~~~~~
<source>:1:1: note: 'std::vector' is defined in header
'<vector>'; this is probably fixable by adding '#include
<vector>'
  +++ |+#include <vector>
    1 |
<source>:4:15: error: expected primary-expression before 'int'
    4 |   std::vector<int> a;
      |               ^~~
Compiler returned: 1
```

Intel's icc (which is now out of date)

```
icc: remark #10441: The Intel(R) C++ Compiler Classic (ICC) is
deprecated and will be removed from product release in the second
half of 2023….(message shortened here)
<source>(4): error: namespace "std" has no member "vector"
    std::vector<int> a;
         ^
```

```
<source>(4): error: type name is not allowed
    std::vector<int> a;
                ^

<source>(4): error: identifier "a" is undefined
    std::vector<int> a;
                     ^

compilation aborted for <source> (code 2)
Compiler returned: 2
```

Even this very simple bit of code with a very simple error produces 2 or 3 issues and the is the *first and most important lesson when compiling*!

Start at the first error.

In all of these we're getting some weird errors cascading from the first one and there's no point trying to understand them until we fix that. All that junk about an undefined identifier (variable) and Clang's complaint about a "cast" are because it doesn't know what a std::vector is.

Gcc is trying to be our friend here, and the note tells us exactly what is wrong - vector is not a thing until we include the header file for it.

Fix this, compile the code, and verify that it now works OK. Congratulations - you have fixed your first compile error. Was that too easy? Don't worry! There will be plenty more!

- 03 Warnings

- Turning on compiler warnings is also important as this will help us spot issues before they become bugs
- Lots of the time, if we make a typo it's either going to produce invalid code, for instance a variable name that does not exist, or something "unexpected" that looks like an odd thing to do
- The compiler can try and help you by flagging a range of things that people usually don't do deliberately

Info:

Because warnings are only “unexpected” and not “invalid” they wont stop code compiling, and can usually be turned on or off individually (by type, not line of code). This is to prevent spam when a programmer really does want to do a thing. HOWEVER, especially at first, we suggest fixing all warnings, because they often do indicate poor code

Try:

There are 3 examples of compiler warnings in the pack, warn1, warn2 and warn3. Each does a different “unusual” thing. Depending on your compiler, you may get warnings automatically, or you may have to turn them on with flags. For gcc and Clang, use `-Wall -Wextra` Otherwise consult docs for your compiler

warn3.cpp is an example of a very subtle issue that we wont get a warning about unless we enable it specially. In this case we are comparing a float number (i.e. a decimal) to another floating point number. This is unlikely to mean what we intend, and if you run the code on all the systems I checked you indeed get a surprise (ask why if you are interested). Try the flag `-Wfloat-equal` to warn you of code like this.

(Aside - if you are wondering, comparing two floats of the same precision (same number of bits used to store them) will probably work as we expect, and what we actually did here was compare two different types of float - one can hold more decimal places than the other, and so they are not equal. However in practice there are many ways we might not get exactly the result we expect in all decimal places so we should usually “soften” our comparison and say “as long as they are this close to each other, they are equal”.)

First Programs

- 01 Guided first program
  - Try this if your programming skills are rusty and you want some guided steps to get started
  - We’re going to write something to calculate and print the Fibonacci sequence.
  - Fibonacci is a “recurrence” relation - describing a sequence where the current term depends on the previous ones. Here s(n) depends on s(n-1) and s(n-2). We

start with s(1) = 1 and s(2) = 1 and s(n) = s(n-2) + s(n-1) This gives the “classic” Fibonacci of 1, 1, 2, 3, 5, 8, 13 etc

Guided Steps:

Once each steps compiles and runs, go to the next.

Start with an empty text file and fill in the skeleton of a main program.

Create a max_val for the loop. Initialise the latter to 20.

Write a loop running from 3 to max_val and print the value of the loop variable

Create two temporaries for the “previous” pair of values. Call them fib_first and fib_second. They should also be integers. Start them with values of 1. Also create ‘fib_current’.

Output these first two values.

Inside your loop, calculate fib_current = fib_first + fib_second. Now swap fib_first = fib_second and fib_second = fib_current. Output fib_current.

Now you have a simple program!

Add one last step - after you initialise max_val, check that it is valid. It needs to be positive, and you might want to limit the maximum size to e.g. 46. Fibonacci grows FAST - this is the last term that fits in a standard integer on most systems.

If you want to know why things go funny after that, please do ask!

- 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!

- 03 A program using a vector

- Vectors can grow as data is added
- This is really convenient when we want to take an input for the size of something

Info:

Because vectors are always "contiguous" (one thing after another) in the computers memory, sometimes to grow bigger they have to move to a new block of memory entirely, copying everything in them. If you are worried that push_back might cause this to happen every time you add to a vector, you don't need to be - the implementation of vector nearly always increase the capacity (number of elements it can hold without needing to move) by a factor of 2 every time.

Try:

Rewrite, or write, the Fibonacci program to put the values into a vector instead. Instead of printing them from the loop, print them all afterwards using the "container loop" style

- 04 Writing your own function

- Before we try and write actually useful functions, lets just practice the syntax
- Once again, skip this part if you already get it

Info:

If passing something which could be large to a function, we might want to use "pass-by-reference" to save needless copying. However,

Try:

Start by creating a function to take two integers, average them, and return the result. (There is a built-in way to do this, ignore that for now!)

Now do the same, but for a vector of integers. Since this is an input, we can pass it by value, by reference, or by a "constant reference" - a reference for the purposes of not being a copy, but one which cannot modify the original value. This is a good trick to learn early, because passing big vectors around by value might cost you a lot.

Going Further

- Strings

- We didn't discuss strings very much, although we did use one
- Strictly we should include the header <string> but this often has already been included by something else we use so we probably wont get any errors if we forget
- Working with strings properly is a bit messy so for this workshop we're just going to look at one way of building a string from some stuff

Try:

Write a simple program to create two strings. Add them together: what do you get?

There is a function, std::to_string() which turns integers etc into strings. Try this out, and use it to build the following message from variables std::string name = "Bob"; and int age = 21;

"Hello Bob. You are 21!"

- Simple function overloading

- One powerful feature of C++ is the ability to have two functions with the same name, but different parameters

- This is called “overloading”. The parameters must be different “enough” that the compiler knows which one we mean

Try:

Write a simple, very silly function, which takes a string and prints it to std::out

Now write a second overload for this which does the same for an int. Make sure the name is the same

What happens if you try and write two functions with the same name which both take ints?

- Structs

- We introduced structs to show how we could hold multiple pieces of data
- Structs are a “data type” so we can have variables of that type, pass them to functions, return them from functions and every thing else

Try:

Create a struct, person, which contains a name field and an age field.

Create a vector of these structures, called staff. Fill in some values

Create a simple function, taking a vector of “person”s and printing their information.

Now try creating a function which:

Takes a (possibly empty) vector of people, and a new person

Checks if any person with that name is in the vector

If not, adds them

If so, prints an error

Returns the vector, with the person added if we did that step

Try calling this function with a new person, and one who is already in the list and check it works