

- C++ **is** case sensitive, so capitalisation makes a difference
- Files should end in `.cxx` for C++, and `.c` for C. Here we'll be using C++ so use `.cxx`.

Programs

All C++ codes must have one `main` function where execution will begin:

```
int main() {
    return 0;
}
```

The `return 0;` at the end indicates no error occurred. To compile the program:

```
$ g++ mycode.cxx -o mycode
```

will produce an executable called `mycode`. To run the compiled program:

```
$ ./mycode
```

Comments

To start a single-line comment, use `//`. For multiple lines you can use `/* ... */`

```
/*
    You should explain what
    each part of your code does
*/
int main() {
    // Explain the main steps
    return 0;
}
```

Variables

Before using a variable, you need to *declare* it by giving it a type e.g.

```
double r;
```

creates a variable `r` of type `double`. This type represents a number with decimal places as in IDL. There's also the `int` type for integers, and `float` for lower precision real numbers. To use complex numbers you need to include the complex library:

```
#include <complex>
typedef std::complex<double> cmplx;
```

```
int main() {
    cmplx a(1, 4.2); // Define a = 1 + 4.2i
    return 0;
}
```

You can also create your own types by using **structures** or **classes** which allow you to group data together.

Printing and input

To print outputs you need the `iostream` library, which allows you to send data to `cout` representing the terminal:

```
#include <iostream>
using namespace std;

int main() {
    double r = 10;
    cout << "Hello World!" << endl;
    cout << "Result: " << r << endl;
    return 0;
}
```

You can use `cin >> ...` to get input from the user. To read and write files, include the `fstream` library.

Arrays

`vector<double>` is a vector (1D array) of double types:

```
#include <vector>
#include <iostream>
using namespace std;

int main() {
    vector<double> a;
    a.resize(10);
    // 'a' now has 10 values (elements),
    // numbered 0 ... 9
    a[9] = 3.14;
    cout << a[9] << endl;
}
```

Functions

Functions take zero or more inputs and return a result

```
return_type name( inputs ) {
    <commands>
}
```

where `return_type` can be any type, or `void` if it doesn't return anything. Usually when an input is altered in a function the new value is not sent back. To change this, use `&` in the function definition:

```
// Note '&' on a, not on b
void change(double &a, double b) {
    a = 10;
    b = 20;
}

int main() {
    double x = 0, y = 0;
    change(x, y);
    // x is now 10, y still 0
}
```

Expressions

C++ has the usual `+`, `-`, `*`, `/` operators, but no “to the power of”. For this and many other mathematical functions include the `cmath` library.

```
double x = 3;
double y = pow(x, 3); // y = x ^ 5.2
```

Conditionals

The `if` syntax is

```
if(<condition>) {
    <commands> // Run if <condition> is true
}else {
    <commands> // Run if <condition> is false
}
```

The `else` clause is optional. See also `switch` statements. Conditions include

Condition	Meaning
<code>a == b</code>	'a' equal to 'b'?
<code>a != b</code>	'a' not equal 'b'?
<code>a < b</code>	'a' less than 'b'?
<code>a > b</code>	'a' greater than 'b'?

Loops

`for` loops are usually used to iterate a variable (here `i`) between a minimum and maximum value e.g.

```
for( int i=0; i<10; i++) {
    <commands>
}
```

will repeat `<commands>` with `i=1, 2, ..., 9`. There is also the `do...while` loop:

```
do {
    <commands>
}while(<condition>)
```

which repeats `<commands>` while the condition is true.