Programming and Data Analysis for Scientists

C++ Workshop 2

Introduction to the basic syntax of C++

Prof Stephen Clark

SCIF20002

Introduction to the basics of C++

The purpose of this workshop is to introduce the key syntax of the C++ programming language. The *learning objectives* are:

- Learn variable types, conditional logic and looping syntax in C++
- Basic input and output to the screen
- Write a simple programs, compile them and run them
- Solve some mathematical problems using C++

Introduction to the basics of C++

All programming languages have basic primitives required for any code:

- Variables
- Arithmetic expressions and maths functions
- Basic text input and output
- Logical branching
- Looping and iterations

To start C++ we need to learn its version of each of these primitives. This workshop’s **online lecture notes** give detailed descriptions. Here I will give a brief overview of each before you solve some problems …

Variables in C++

C++ is a **typed** language – this means you have to declare what a variable is before you use it and cannot change its type at runtime.

- `int` integers (at least 16 bits, but can be 32 bits or 64 bits)
- `char` character (8 bits)
- `float` single precision floating point (32 bits)
- `double` double precision floating point (64 bits)
- `boolean` true or false (often 8 bits because CPU's address memory in bytes)

Variables are declared like:

```
int i, j, k;
float speed, distance;
```

```
double pi = 3.14159;
```

Arithmetic expressions

All the usual arithmetic expressions work on variables:

```
x = 1;      // sets x to 1
x = 3*y;    // sets x to 3 times the value of y
x = y+1;    // sets x to 1 more than y
x = x+1;    // increases x by 1
x = pi/2;   // sets x to pi (defined previously) divided by 2
x = y-1;    // sets x to 1 less than y
```

Needs x to be a double to work as expected!

C++ also has some neat shortcuts for other common operations:

```
x++;        // also increases x by 1
x += 5;     // increases x by 5
x *= 3;     // triples the value of x
x /= y;     // set x to be x divided by y
y = x++;    // set y to x and then increase x by 1
y = ++x;    // increases x by 1 then set to y
x = 5%3;    // sets x to the remainder of 5 divided by 3
x--;        // decrease x by 1
x %= 2;     // set x to be remainder of x divided by 2
```

Mathematical functions

C++ is **lightweight** so basic maths functions are not included in the core language. By including the library `<cmath>` most common functions are available:

```cpp
#include <iostream>
#include <cmath>

int main()
{
    double x = 0.91211;
    double y, z;

    y = exp(-acos(x));
    z = cbrt(fabs(atan(x)));
    std::cout << "Answer = " << z <<  std::endl;

    return EXIT_SUCCESS;
}
```

Include maths library

Use exponential and inverse-cosine functions

Use cube root, absolute value and inverse-tangent

Basic text input and output

We display text to the shell by “<<“ inserting it into the `std::cout` object:

```
int M = 13;
float G = 5.91;
char X = 'P';

std::cout << "M = " << M << " and G = " << G << " with X = " << X;
```

Can insert multiple variables of different types for display, along with strings

Outputs: `M = 13 and G = 5.91 with X = P`

There are lots of formatting options.

We get input from the keyboard by “>>“ extracting it into the `std::cin` object:

```
#include <iostream>

int main()
{
  int i; // Define a variable to store input
  std::cout << "Please enter an integer value: ";
  std::cin >> i; // Extract a value from the keyboard
  std::cout << "The value you entered is " << i;
  std::cout << " and its double is " << i*2 << ".\n";
  return EXIT_SUCCESS;
}
```

Logical branching

The main branching primitive in any language is an `if-then` statement:

```
if ( logical expression X )
{
  **block of code** – executed only if X is TRUE
}
else if ( logical expression Y )
{
  **block of code** – executed only if X is FALSE and Y is TRUE
}
else
{
  **block of code** – executed only if X and Y are FALSE
}
```

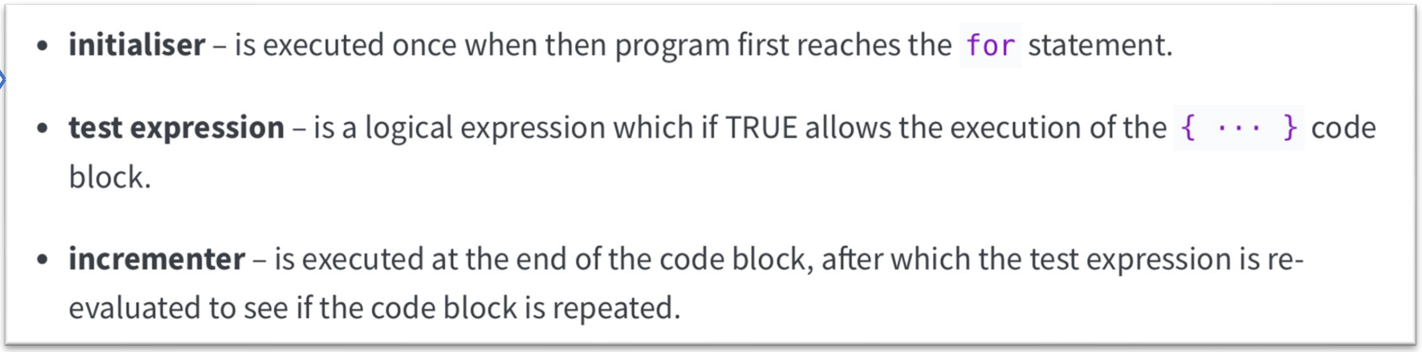
example logical expressions:

```
x=5;              // Set some values for x and y
y=0;
if (x==y)         // x is not equal to y => FALSE
if (y=x)          // sets y to x (non-zero) => TRUE
if (!x)           // ! is the NOT operator => FALSE
if (x||y)         // x OR y => TRUE
if (x&&y)         // x AND y => FALSE
if (x&&(y=x))     // x AND x => TRUE
if (x=>5 && y==x) // => FALSE
if (x!=3 && y<x)  // => TRUE
```

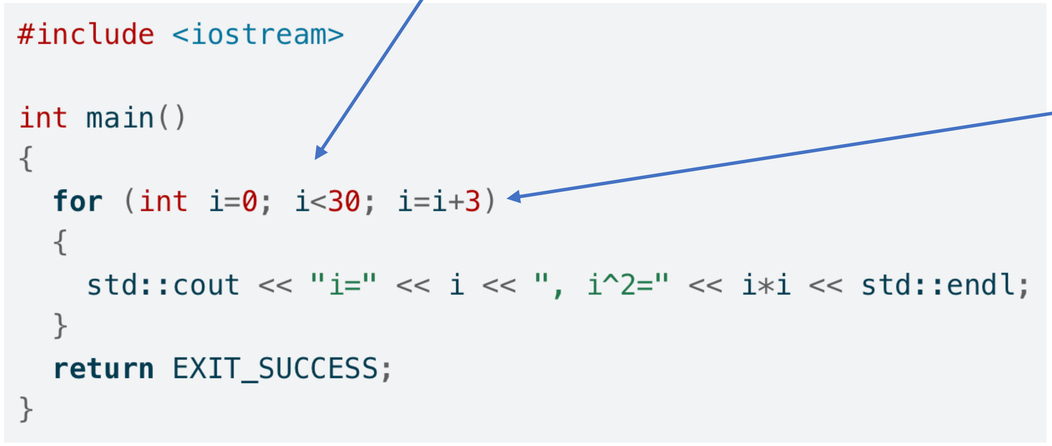
Looping and iterations

A common looping primitive is the `for-loop`. In C++ it has the form:

```
for ( initialiser ; test expression ; incrementer )

{

  **block of code**

}
```



- **initialiser** – is executed once when then program first reaches the `for` statement.
- **test expression** – is a logical expression which if TRUE allows the execution of the `{ ··· }` code block.
- **incrementer** – is executed at the end of the code block, after which the test expression is re-evaluated to see if the code block is repeated.



```
#include <iostream>

int main()
{
  for (int i=0; i<30; i=i+3)
  {
    std::cout << "i=" << i << ", i^2=" << i*i << std::endl;
  }
  return EXIT_SUCCESS;
}
```

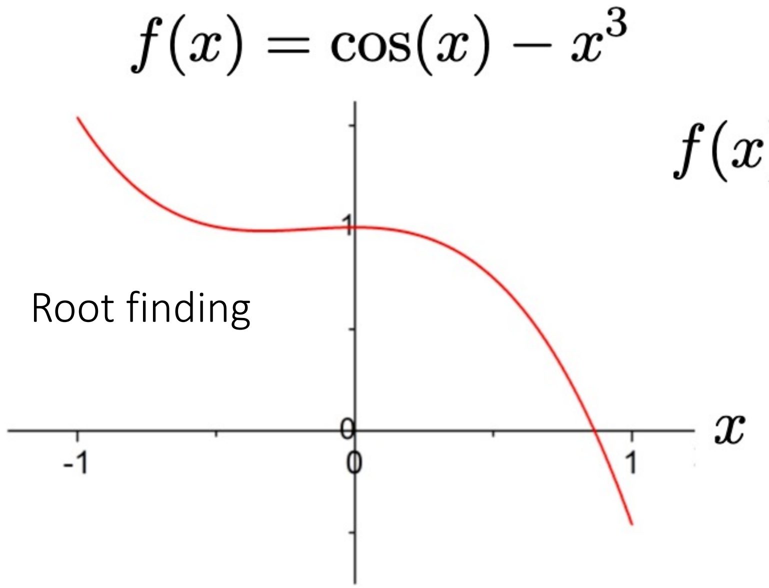
Note that the initialiser can declare the iterating variable

Also `do` and `while` loops are available.

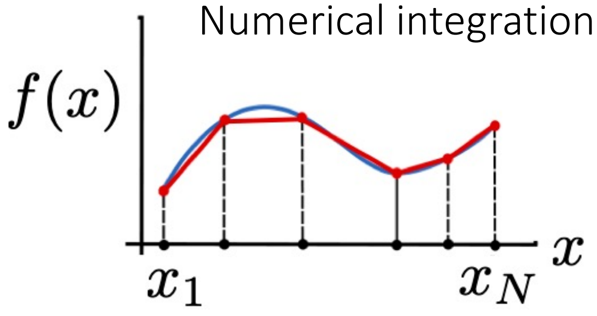
Workshop exercises

Let’s do some C++ coding …

Using the basic syntax covered here the workshop exercises ask you to write code to solve some well-known mathematical problems.

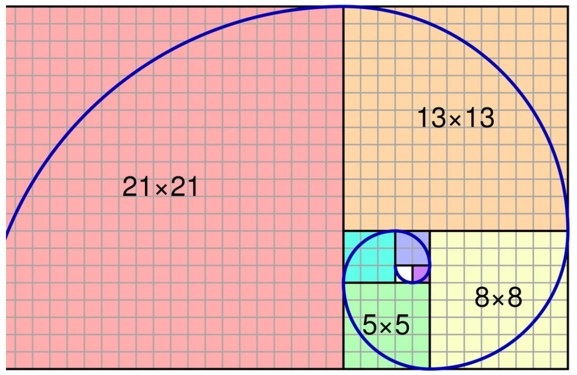


Root finding



Numerical integration

Fibonacci sequence



Some philosophy …

When you write a program, think of it primarily as a work of literature. You're trying to write something that human beings are going to read. Don't think of it primarily as something a computer is going to follow. The more effective you are at making your program readable, the more effective it's going to be: You'll understand it today, you'll understand it next week, and your successors who are going to maintain and modify it will understand it.

Donald Knuth