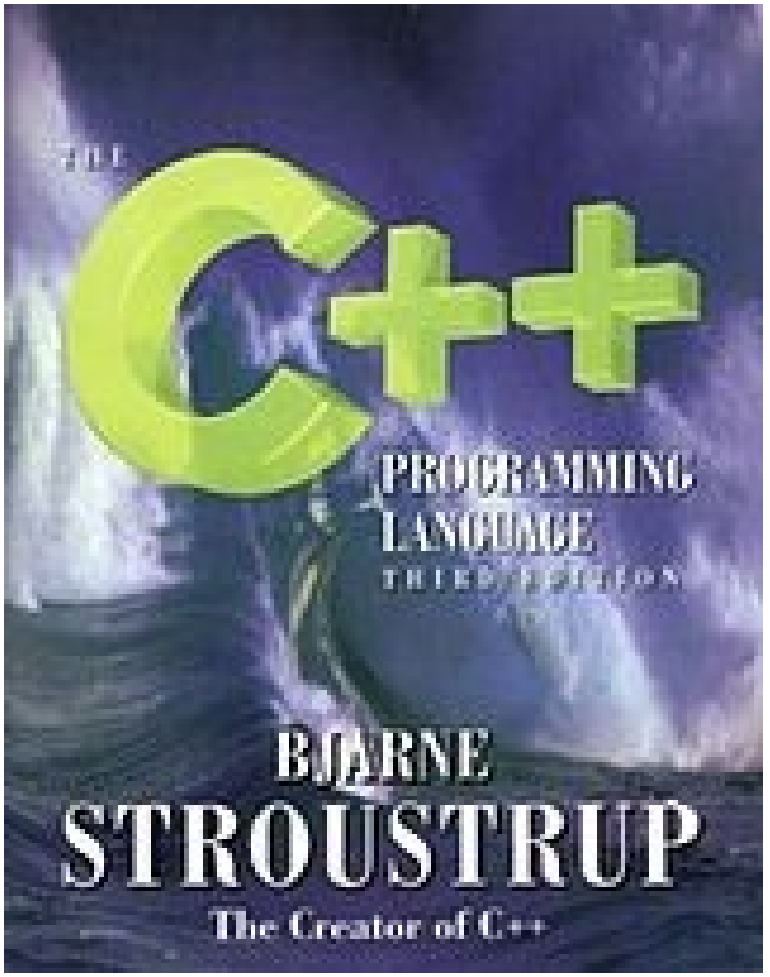
C++ Syntax and Compiler Usage

Mark Slater

UNIVERSITY OF BIRMINGHAM

Overview

1. Code Creation and Compilation
2. Types, Objects, Values and Variables
3. Operators
4. The Compiler and Pre-Processor



What is C++?

- C++ is a general purpose object oriented programming language widely used in the software industry and beyond. It was developed by Bjarne Stroustrup in 1979 as an enhancement to the C language ('C with classes')
- Though comprised of fairly basic syntax and conventions, it is incredibly powerful, especially with the addition of the 'standard libraries'. Anything you can think of to do on a computer can be (but not necessarily should be!) done in C++
- C++ has been adopted as the standard for most coding tasks in modern Particle Physics and so it's well worth getting to know!

1. Code Creation and Compilation

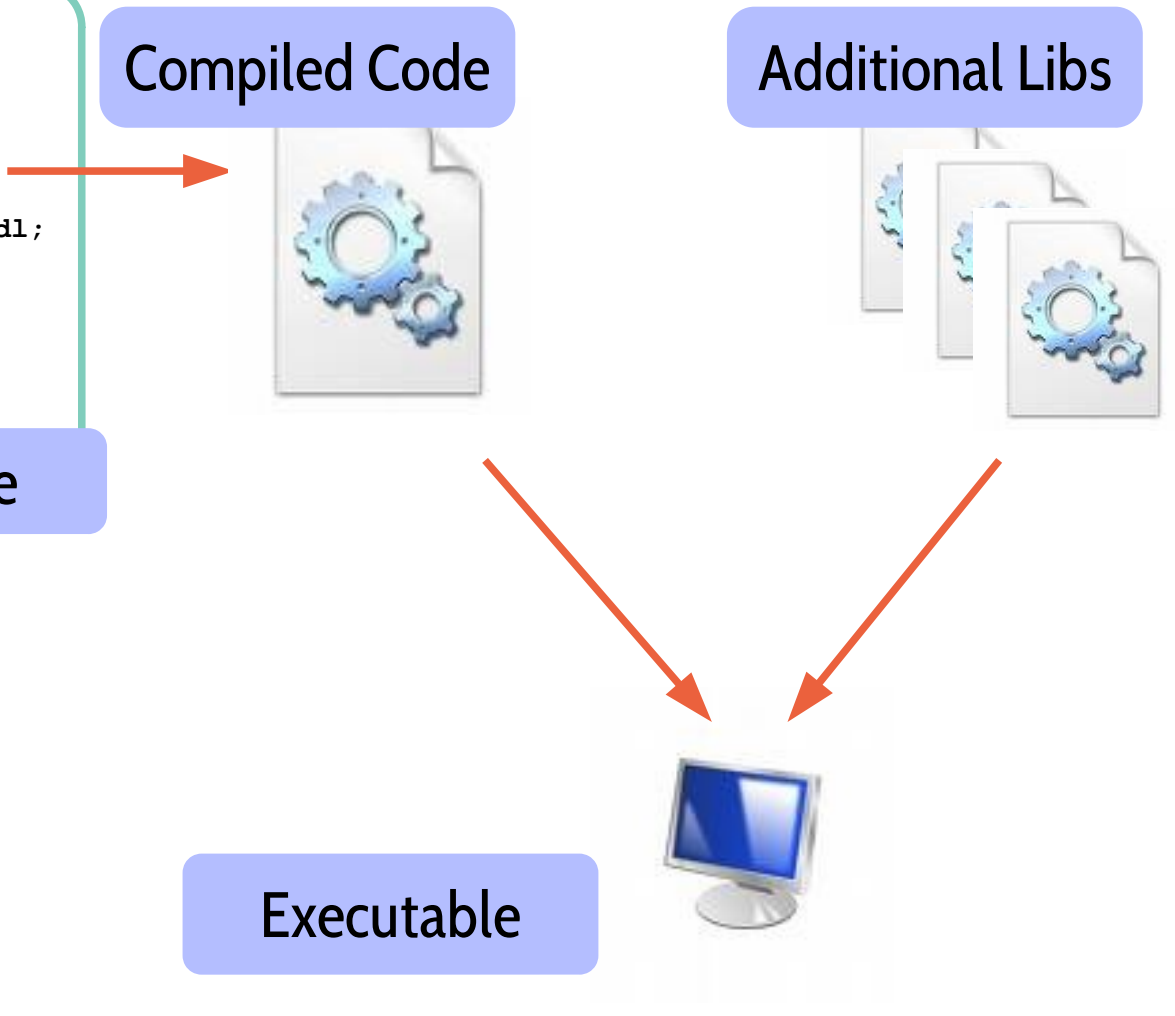
Writing and Compiling Code

As already stated, C++ code can be written using any text editor, but to create the actual programs requires a compiler that creates the machine-readable code

```
#include <iostream>

int main()
{
    // Read and print three
    // floating point numbers
    std::cout << "Give 3 nums" << std::endl;
    float a{0}, b{0}, c{0};
    std::cin >> a >> b >> c;
    std::cout << "You gave... ";
    std::cout << a << ", " << b << ", "
          << c << std::endl;
}
```

Raw Code



Exercise 3.1 - The Hello World Program

To demonstrate this process, you will create the ubiquitous 'Hello World' program:

1. Open your cloned repo using Visual Studio Code + Docker
2. Create a new file: File → New File:
3. Type in the below code and then save it as 'mpags-cipher.cpp'
4. Run the g++ compiler in the 'Terminal' as shown in the other box
5. Execute the program!

```
#include <iostream>
int main()
{
    // This is a comment
    /* This is a
     Multiline comment */

    std::cout << "Hello World!\n";

}
```

```
> g++ -std=c++11 -o mpags-cipher \
      mpags-cipher.cpp

> ./mpags-cipher

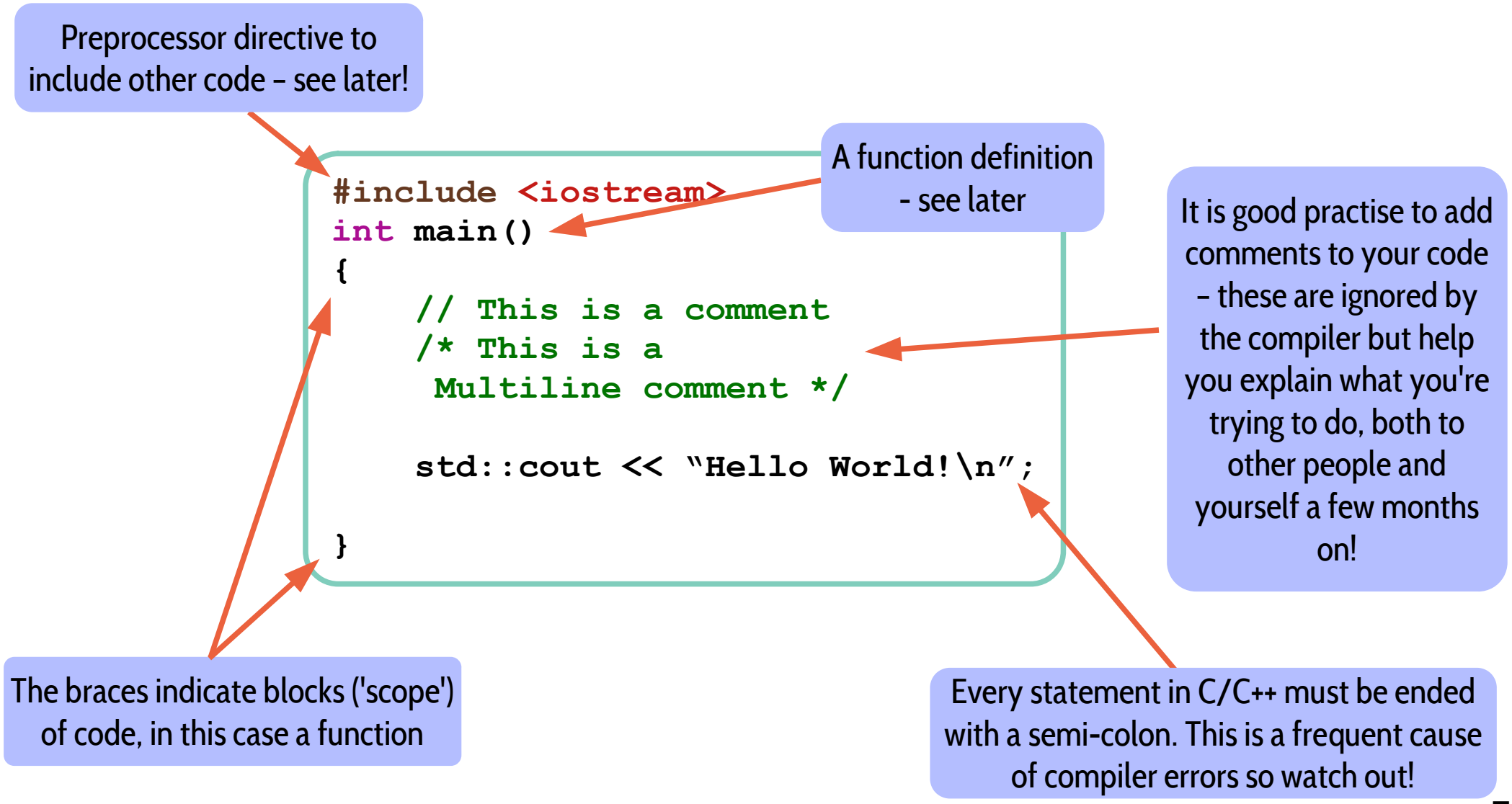
Hello World!

>
```

Note the trailing slash is just to indicate the line continues – you don't need to type it!

Basic Syntax of a C/C++ Program

Before we start looking in more detail at C++ coding, we will just cover the basic syntax of program you've just written




```
#include <iostream>
int main()
{
    // This is a comment
    /* This is a
     Multiline comment */

    std::cout << "Hello World!\n";

}
```

2. Types, Objects, Values and Variables

What are Types, Objects, Values and Variables?

C++ (and many other languages) have a defined way of holding and organising data within a computer's memory

The appropriate terms are:

- Types – How to interpret data in a memory location ('object') and what operations can be performed by it
- Object – Defined area of memory that holds the data ('values') associated with a type
- Value – Actual data/bits in memory interpreted by the 'type'
- Variable – A flag or name of an area of memory ('object')

These are a bit abstract at the moment, but we'll show examples in a few slides time!

What are Types, Objects, Values and Variables?

A way to understand this is to think of the Computer's memory like a room with things in it like tables and chairs:



What are Types, Objects, Values and Variables?

A way to understand this is to think of the Computer's memory like a room with things in it like tables and chairs:



The Chair and Table are ***OBJECTS*** in the room (Memory) that are different ***TYPES***

What are Types, Objects, Values and Variables?

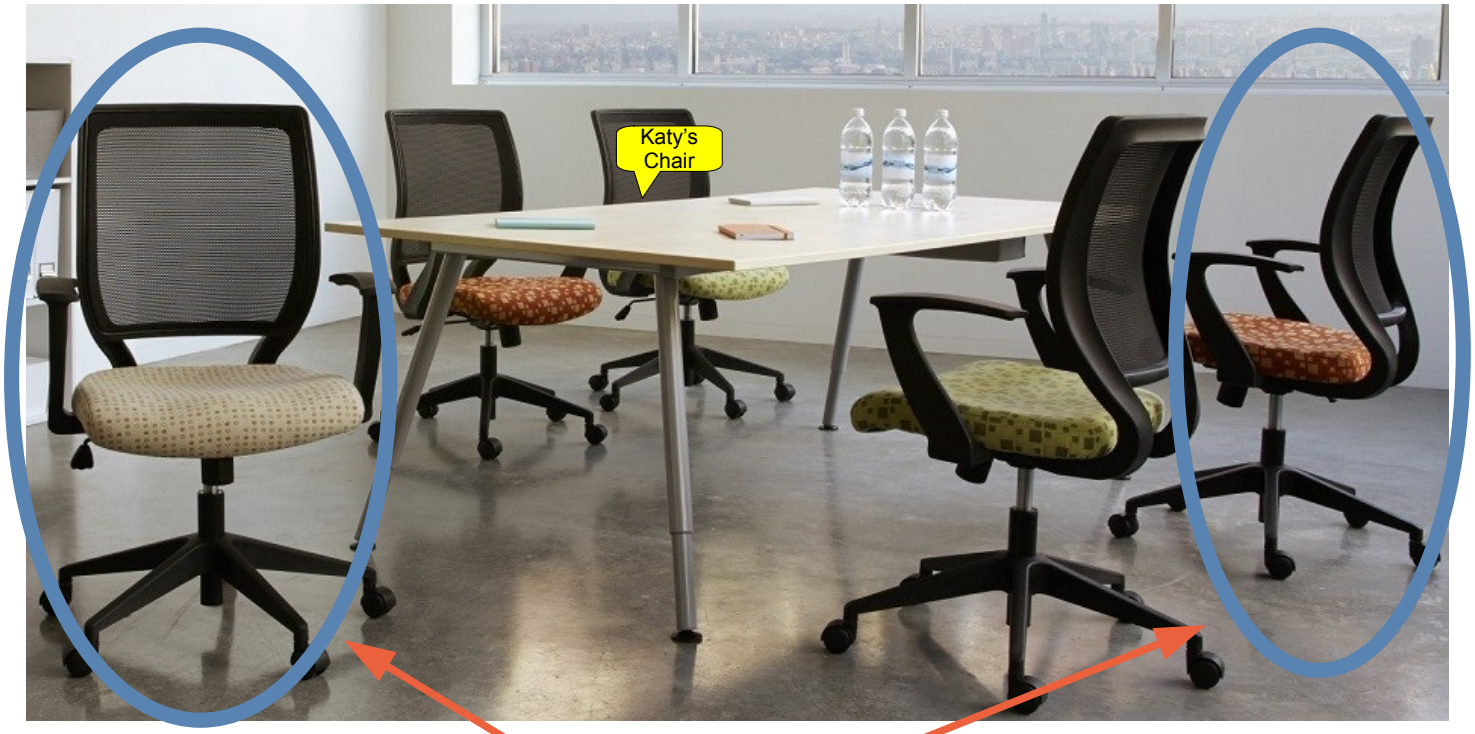
A way to understand this is to think of the Computer's memory like a room with things in it like tables and chairs



These two chairs are ***OBJECTS*** of the same ***TYPE*** and therefore have the same 'properties'

What are Types, Objects, Values and Variables?

A way to understand this is to think of the Computer's memory like a room with things in it like tables and chairs



What are Types, Objects, Values and Variables?

A way to understand this is to think of the Computer's memory like a room with things in it like tables and chairs



This chair (***OBJECT***) has a label (***VARIABLE)*** name on it so it can be identified easily

Basic Types and Initialising Variables

We will start by introducing the basic types that are available in C++ and showing what happens when these are created and destroyed

The built-in basic types are:

- A boolean (true/false) – 'bool'
- Integer number – 'int'
- Floating point number – 'float'
- Double precision number – 'double'
- Single Character/0-255 number – 'char'

Note that numerical variables can also be 'signed' (default) or 'unsigned'

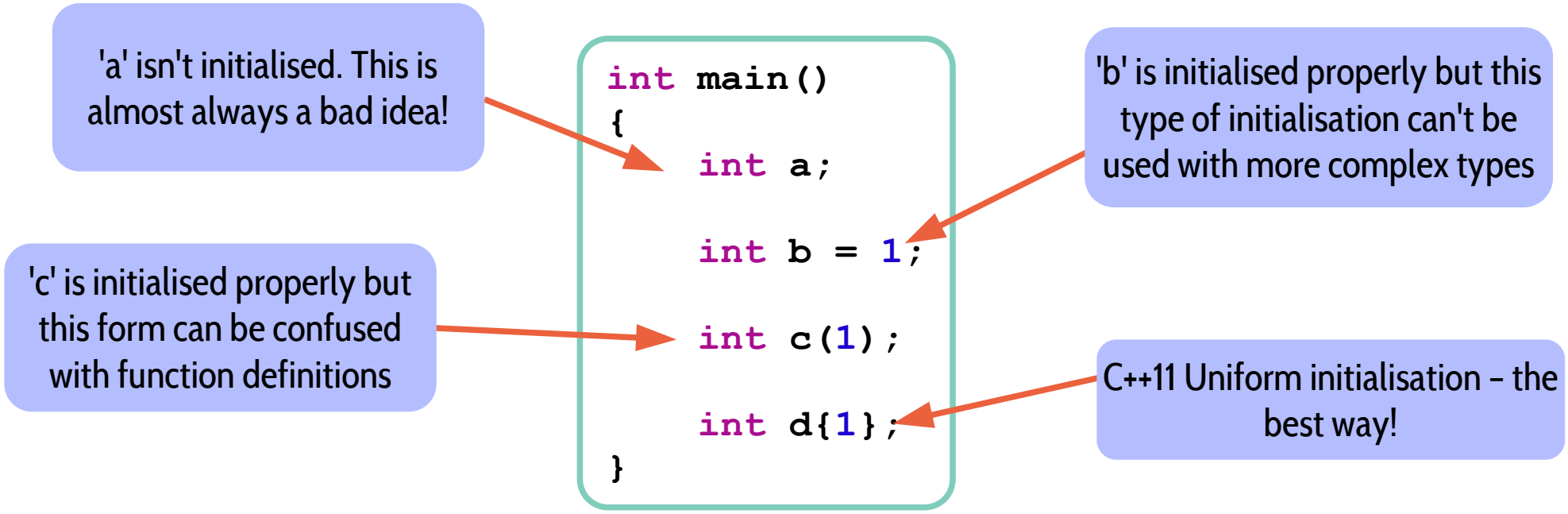
To declare a variable (a named object of this type), use the following syntax:

<object_type> <variable_name> {<initialisation_parameters>}

This will also create an object of the requested type (i.e. assign the appropriate memory) and initialise it with the given parameters

Aside: Other Forms of Initialisation

- Initialisation using the braces ('{}') is termed 'Uniform Initialisation'
- There are a number of different ways of initialising variables that you'll come across but this is the recommended method
- Other examples are shown below:



Variables in Action (1)

```
#include <iostream>

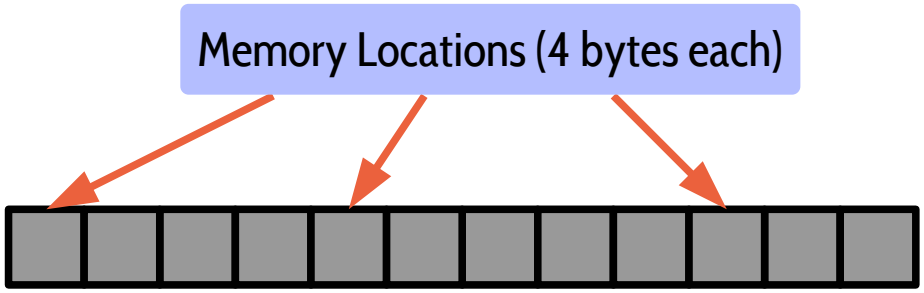
int main()
{
    int a;            // BAD!!
    double b = 1.2; // Not Great!
    double c{3.4};

    a = 43;
    b = 2.2;

    c = a * b;

    std::cout << c << std::endl;

    return 0;
}
```



To show you how variables work, we'll now go over a basic program that initialises some variables, does a calculation and outputs the result

It may seem a little basic at present, but it will help you to understand how the computer interprets variables, objects, etc.

Variables in Action (2)

```
#include <iostream>

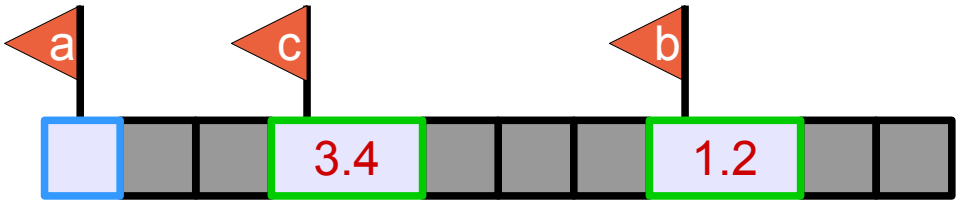
int main()
{
    int a;            // BAD!!
    double b = 1.2; // Not Great!
    double c{3.4};

    a = 43;
    b = 2.2;

    c = a * b;

    std::cout << c << std::endl;

    return 0;
}
```



The 3 variables are declared:

- The markers are the variables/names of each object
- The coloured outline represents the type – doubles take more space!
- The grey boxes themselves are the objects

Note that for 'a', all this does is create the object, NOT the actual value - the initial value is junk!

If b was an int, then this initialisation would only give a warning of the narrowing rather than an error

Variables in Action (3)

```
#include <iostream>

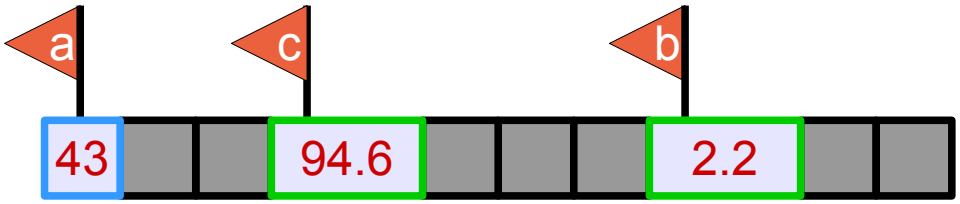
int main()
{
    int a;            // BAD!!
    double b = 1.2; // Not Great!
    double c{3.4};

    a = 43;
    b = 2.2;

    c = a * b;

    std::cout << c << std::endl;

    return 0;
}
```



We now assign values to the 3 variables

This gives the 'a' object a defined value and overwrites the others

Note that c is assigned correctly with double precision as the compiler will always use the highest accuracy type

The value of 'c' is then printed to the screen using the Standard Library 'std::cout' object as before

Variables in Action (4)

```
#include <iostream>

int main()
{
    int a;            // BAD!!
    double b = 1.2; // Not Great!
    double c{3.4};

    a = 43;
    b = 2.2;

    c = a * b;

    std::cout << c << std::endl;

    return 0;
}
```



At the end of the program/function the following happens:

- The return value is set to zero
- The objects associated with the variables are deleted
- The memory isn't reset so the values are still present

Exercise 3.2 - Getting Experience with Variables (1)

- Now edit the code in between the braces in your previous 'Hello World' program to do the following:
  - Create, modify and output an integer variable
  - Create a ‘double' variable and output this
  - Initialise another integer from this double – note the output from the compiler
- Note that to output things to the console, use the following:

```
std::cout << my_var << std::endl;
```

Using Strings

There is a basic type for a single character but not for a 'string' or 'array of characters'

This is because a string can be of variable length and requires much more complicated manipulation than the other types

Use the std::string type when dealing with strings. Though this looks a lot more complicated, you can generally treat it as a basic type

Note that you will need to #include the 'string' header!

Look at cppreference.com to find the headers to use for other types and functions

'std::' is required as the string type is part of the 'std' namespace ('group')

```
#include <iostream>
#include <string>

int main()
{
    std::string msg {"Hello"};

    std::cout << msg << "\n";

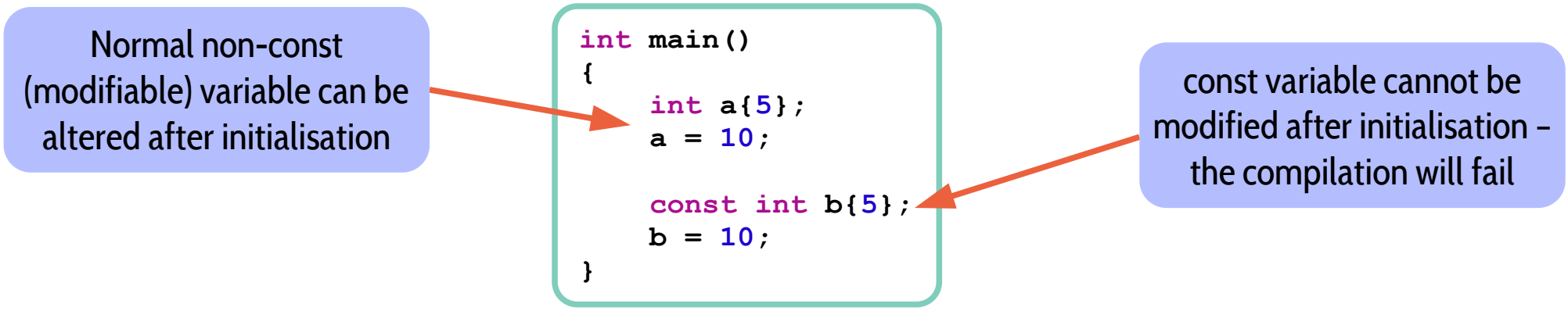
    return 0;
}
```

Initialise the string variable

Print the string as before

Constness

- Variables are used to store any information you wish to keep available to your program, however you may not want to modify some (most!) of these
- When declaring variables, it is good practise to tell the compiler if they should be kept constant or if modification is allowed
- This is done by using the '`const`' keyword before the variable type



A good rule of thumb is to declare everything const unless you are certain you have to modify it!

`constexpr` Variables

- There are a 'special class' of constant variable that can be useful in certain circumstances – one that can be determined at compile-time rather than runtime
- These are variables that are initialised from a 'constant expression', e.g. a hard-coded number, string or other compile-time determined variables
- This can save some time on execution as these variables are replaced when building the program rather than the look-up occurring at run time
- In addition they can be used in compile-time determined functions as well (see later!)
- Constant expression variables use the '`constexpr`' keyword before the variable type:

This will give an error as the value cannot be determined at compile time – 'c' is not a constexpr

```
int main()
{
    constexpr int a{5};
    constexpr double b{6.7 * 2.4 * a};

    int c{5};
    constexpr int d{c};
}
```

These are both valid constexpr variables and will be calculated when building the program

If you know a variable is initialised from other constant expressions, it's good practise to set it as 'constexpr'

Exercise 3.3 - Getting Experience with Variables (2)

- Again, edit the code in your mpags-cipher.cpp to do the following
  - Create a const 'double' variable and output this
  - Create another integer variable then modify it
  - Make the variable 'const' and attempt to compile
  - Create, initialise and output a string variable

3. Operators

Operators (1)

- In addition to variable declaration, we'll now introduce operators
- These are symbols that perform a specific operation on one or more objects. A subset of these are the arithmetic operations you're familiar with:
  - Multiplication: a * b
  - Addition: a + b
  - Subtraction: a - b
  - Division: a / b
- For example, 'a + b' is the addition operator being applied to the objects 'a' and 'b'
- Also note that operators in C++ are nothing 'special' – they are essentially shorthand for calling other bits of code

Operators (2)

As well as these arithmetic operators, there are several more language specific ones:

- Assignment: a = b
- Dec/Increment: a--, a++
- Bitwise shift/stream: a << b, a >> b
- Modulus: %
- Array: []

What each operator does is entirely a property of the type(s) they are operating on, e.g. The '<<' operator when used on an int will bit shift it but when used on 'cout' will output the object to the screen

The syntax for this depends on the operator, but a few examples are:

<object1> <operator> <object2> (e.g. *, +, -)

<object1><operator> (e.g. [], ++)

Note that operators also have precedence, associativity (left <-> right) and arity (# of operands). For a full description, see:

https://en.cppreference.com/w/cpp/language/operator_precedence

Single Character Access

As another example of simple operator use, we will look at the 'string' type

As mentioned, this type can be considered an array or list of characters (chars) and so to access a single character from that array, you use the '[]' operator

```
#include <iostream>
#include <string>

int main()
{
    std::string msg{"Hello World"};

    // output 'Hello World'
    std::cout << msg << std::endl;

    // output 'o'
    std::cout << msg[4] << std::endl;

}
```

The array operator references the zero-indexed list of letters in string and returns the appropriate element

Exercise 3.4 - Getting Experience with Variables (3)

- Once again, alter the code you've been writing so far to do the following:
  - Create double and integer variables
  - Output the product of these
  - See what happens when you divide a double and an int and then two ints
  - Create another string variable
  - Create a single char variable (type 'char'), assign it the value of one of the letters in your string and then output this variable.

4. The Compiler and Pre-Processor

Compiler Flags

- As with most programs, g++/clang/intel can take a number of different command line arguments and flags that can alter the behaviour of the program
- Some of the most useful ones are:
  - -std – This defines what C++ 'standard' to use. We will be using -std=c++17
  - -o – The filename to output to
  - -I – Also look in the given directory for include files
  - -L – Also look in the given directory for library files
  - -l – Link to this library (libMyLibrary.so → -lMyLibrary)
  - -c – This will run the compiler but NOT the linker. Used for compiling individual source files before creating an overall program
  - -save-temps – Save the intermediate files of compilation
  - -v – Print commands used at each stage of compilation
  - --version – Print the version of GCC being used
- You can get a full list of the flags using 'man g++'

Compiler Errors and Warnings

- You have already encountered a few errors and warning during compilation
- These can be the rather cryptic way of the compiler saying there's something wrong
- You should always aim for your code never to produce any warnings or errors and there are some flags to the compiler to help with this:
  - -Wall – Enables some common warnings (NB: Not all!)
  - -Wextra – Even more warnings
  - -Werror – Turn all warnings into errors
  - -Wfatal-errors – Stop compilation on the first error encountered
  - -pedantic(-errors) – Issue warnings(errors) from strict ISO compliance
  - -Wshadow – Enable warnings for 'shadowed' variables
- For this course, we will use all these flags and so our cmd line will be:

g++ -Wall -Wextra -Werror -Wfatal-errors -pedantic -Wshadow …

Preprocessor

- Before actual compilation, the compiler goes through a 'Preprocessor' step that can be very useful for changing what code is actually compiled
- The most widely used is the #include directive which you have already seen. This effectively inserts the contents of the given file at that part of the code
- There are several others which can be useful for situations like cross platform development:
  - #define – Definitions can be used to replace strings in the code or used in preprocessor conditionals
  - #ifdef/#else/#endif – Basic conditionals that will include or exclude certain code depending on some conditions

Generally these are used for advanced use cases though we will be using the conditionals later in the course