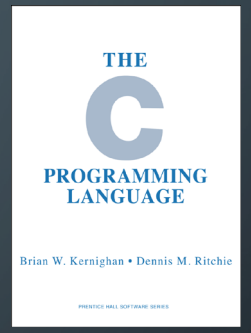
INTRODUCTION TO C++

PROGRAMMING WITH SCISSORS ☺

C++ ORIGIN

- Before OOP's
  - Assembly language CPU specific Z80,6502, 8080, 68000 – device dependant
  - LDA #52 ;Load the letter 'R' into Accumulator
- Started as plain C
  - Device independence, compiled to machine code by CPU specific compiler
  - Very closely coupled to assembly language




```
main()
{
        printf("Hello World\n"); //Call the print function to print Hello, followed by new line
}
```

C++ IS BUILT ON C

- C is requires programmer to be very specific with code
- Code soon became unwieldy as many real world objects are similar but not identical
- Would be helpful to layer code allowing high level objects to be described by a hierarchy of reusable low level objects
- C objects are called **struct** and are data only
- C++ object are called **class** they are data + code, they can also inherit

C++ IS

- C# is Microsoft's response to Oracle buying Sun Microsystems and hence owning Java
  - Java was a first attempt at a OS / system independent language
  - It is JIT (Just In Time) compiled to run safely within the Java VM (Virtual machine)
- C# was inspired by Java and C++, however it does not give the programmer direct access to memory, it is slower than C++ as it runs on a intermediate platform called .NET (like the Java VM)
- C++ is all of C plus all the OOP extensions of C++ (now up to V14)
- Access to C and memory pointers makes it fast (and dangerous)
- As C++ evolves, less and less needs to be done in pure C
- HOWEVER: the requirement to support C syntax, means some of the C++ additions have less intuitive syntax than C# (which was built ground up)

WHY C++ FOR GAMES?

- C++ allows direct access to memory, so its fast
- Most game engines are written in C++ so it does not need much data wrangling to be talk to the API (Application Program Interface)
- Its robust (i.e. has been around for a long time)
- Its 100% compatible with C (most if the internet is written in C)

NATIVE C++ VS. UNREAL C++

- Unreal C++ is a very customised version of standard C++
- C++ is governed by a standards committee who seek consensus above all https://isocpp.org/std/the-committee who move at a glacial pace
- Games companies move rapidly, so they have in some cases made their own versions of key language features as the needed them
- C++ historical big strength is also its weakness i.e. The programmer “owns” memory
  - Memory leaks, dangling pointers, failed allocations

C++ V11 ADDRESSED MOST OF THESE ISSUES

- Smart pointers: shared_ptr<> & unique_ptr<>
- Lambda expressions, to make inline callbacks
- auto (compiler guesses type)
- However these came to late for Epic so they made their own versions

WHAT'S A POINTER?

- Data is stored in memory
- The pointer is the actual memory address where the data "lives"
- This kind of addressing is FAST but dangerous, as code can accidentally (or deliberately) overwrite memory which it does not own

I HOPE YOU LIKE RED

- You will see a lot of red

```
int main()
{
    std::shared_ptr<std::string> tString = std::make_shared<std::string>("Hello");

    std::cout << *tString << std::endl;

}
```

- C++ is a very strict language, everything has to be perfect
- It's also very modular and you only include what you need to keep it lean

```
#include "pch.h"
#include <iostream>
#include <string>

int main()
{
    std::shared_ptr<std::string> tString = std::make_shared<std::string>("Hello");

    std::cout << *tString << std::endl;

}
```

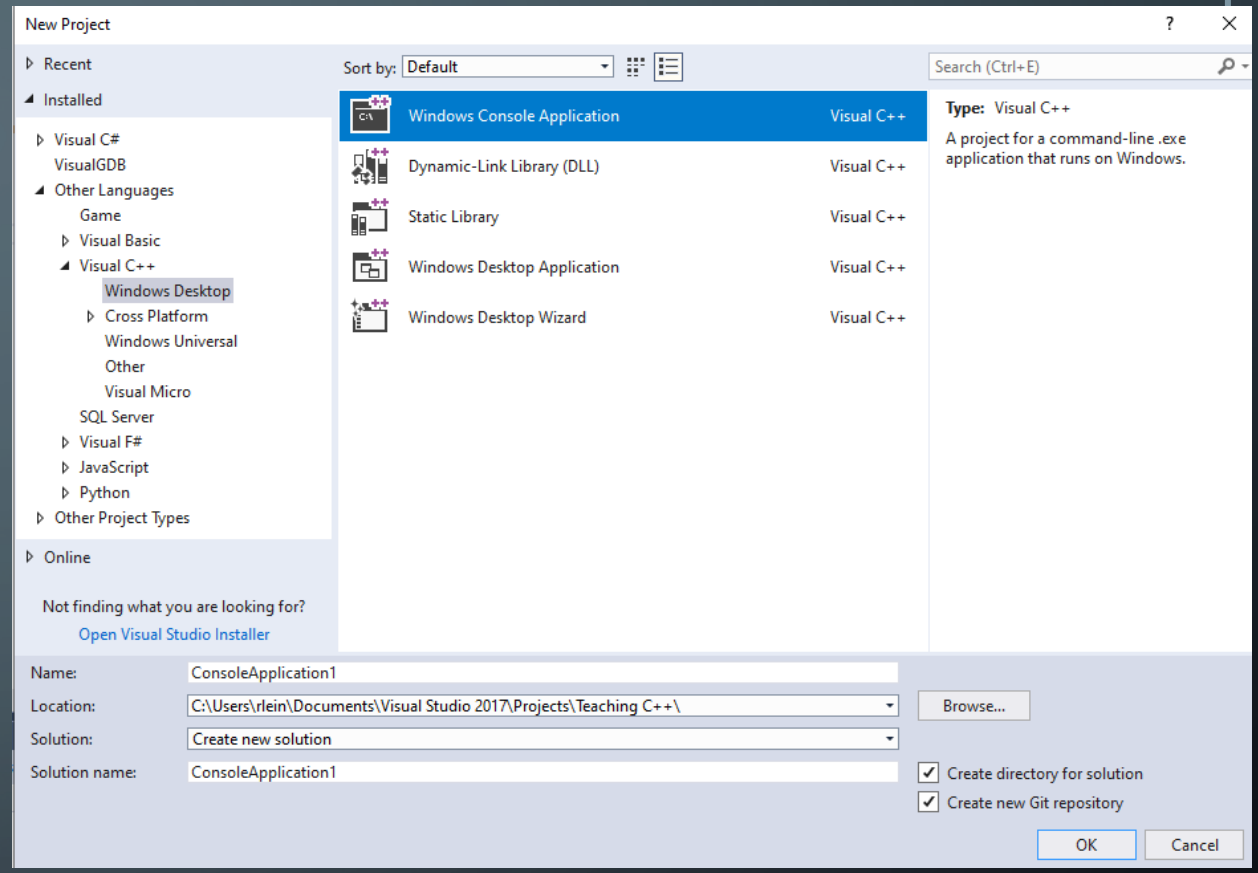
UNLIKE C# OR JAVA, IT USES #INCLUDE

- #include <> or #include “”
- These tell the pre-processor to literally include those text files
- Mostly they contain headers which allow the following code to understand the calling convention for classes, functions & types

HELLO WORLD

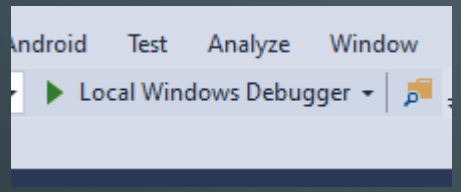
- Start Visual Studio
- Choose Visual C++
  - ➔Windows Desktop
  - ➔ Windows Console Application
- Select a folder to use

  NB: most system do not like network drives, so put it on a physical drive and at the end of the lecture copy it to your OneDrive



TYPE IN THIS CODE

- If you have no red then run it via the local Windows Debugger




```
int main()
{
    std::cout << "Hello World" << std::endl;
}
```

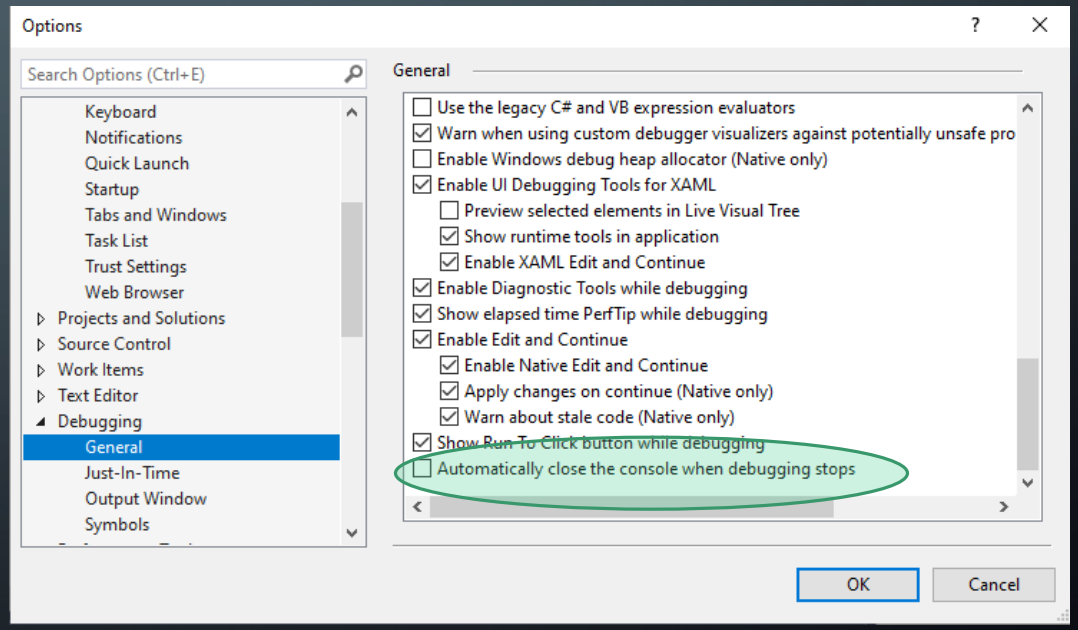
- You should get

```
Microsoft Visual Studio Debug Console
Hello World

C:\Users\rlein\Documents\Visual Studio 2017\Projects\Teaching C++\Intro to C++\Debug\Intro to C++.exe (process 10144) ex
ited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

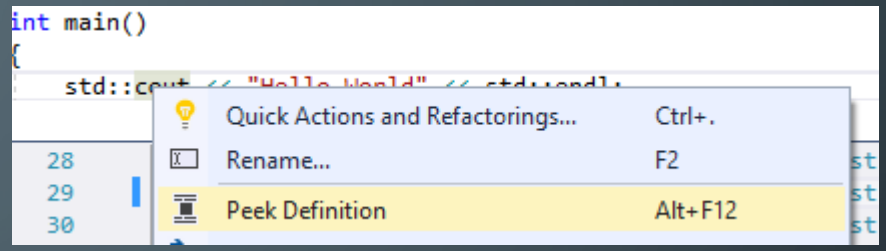
THIS IS CALLED CONSOLE PROGRAMMING

- Its text only
- If the window opens and then closes again too quite to see change this setting in Tools ➔ Options ➔ Debugging



PEEKING IS ENCOURAGED!

- Right click on a word and peek definition




```
#include "pch.h" //Precompiled header
#include <iostream> //where std::cout is defined

int main()
{
    std::cout << "Hello World" << std::endl;
```

iostream

```
28  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT istream cin, *_Ptr_cin;
29  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream cout, *_Ptr_cout;
30  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream cerr, *_Ptr_cerr;
31  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream clog, *_Ptr_clog;
32
33  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wistream wcin, *_Ptr_wcin
34  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wostream wcout, *_Ptr_wco
35  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wostream wcerr, *_Ptr_wce
36  __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wostream wclog, *_Ptr_wcl
37
38          // CLASS _Winit
39  class _CRTIMP2_PURE_IMPORT _Winit {
```

iostream
iostream - (16, 39)
iostream - (28, 56)

NAMESPACES

- Namespaces avoids clashes with names
- Any code, classes, etc in a namespace can only see entities in their own namespace
- std:: = standard namespace

  std::cout means cout is in the std name space
- using namespace std;

  allows those entities to be used without prefix

```
using namespace std;
int main()
{
    cout << "Hello World" << endl;
}
```

STANDARD TYPES

- int size depends on compiler (32/64bit)
- long long (64 bits, 8 bytes)
- long (32 bits, 4 bytes)
- short (16 bit,2 bytes)
- char (8 bits, 1 byte)
- All can be unsigned or signed (2's complement) (invert digits add 1), default is signed
- string Unicode text

https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2019

STATEMENTS & BLOCKS

- The end of a statement requires a ;
- A block is denoted by { //code in here }
  you do not need a ; at the end of a block
- You can chain statements with , (however this is rarely used & frowned upon)
- Control acts on statements or blocks

```
int main()
{
    for (int i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
    {
        cout << i << endl; //Print the number
    }
}
```

Same as

```
int main()
{
    for (int i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
        cout << i << endl; //Print the number
}
```

IF IN DOUBT USE A BLOCK

- Its very easy to make mistakes, e.g.
- What will this do?
- But why?
- The for binds to just the first line so it runs this 10 times and then moves to next one
- Braces remove the ambiguity

```
1
2
3
4
5
6
7
8
9
10
55
```

```
int main()
{
    int i;
    for (i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
        cout << i << endl; //Print the number
        cout << i * 5 << endl; //Print 5 times the same number
}
```

```
int main()
{
    int i;
    for (i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
    {
        cout << i << endl; //Print the number
        cout << i * 5 << endl; //Print 5 times the same number
    }
}
```

USING COUT & CIN

- cout is a “stream” and it can consume input via the << operator and this is printed to the console
- endl is just a shorthand to tell the console to go to the next line
- cin is the input stream, it can consume input and put it into variables
- What happens when input is not a number?

```
int main()
{
    int tTable; //Variable to store value
    cout << "Which times table?"; //Ask user for value
    cin >> tTable; //Get value from user
    for (int i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
    {
        cout << tTable << " x " << i*tTable << endl; //Print the number, some text then the result
    }
}
```

INPUT IS ALWAYS PRONE TO ERROR

- Most of a programmers time effort is making sure input is “safe”
  - This is called input validation, we will come back to this

NESTED LOOPS

- Challenge:
  - Modify your code to print all the times tables from 1-10
  - Hint: you will need a nested for loop

```
int main()
{
    for (int j = 1; j <= 10; j++) //Outer loop
    {
        for (int i = 1; i <= 10; i++) //Start at 1 and go to 10 inclusive in steps of 1
        {
            cout << j << " x " << i << "=" << i * j << endl; //Print the number, some text then the result
        }
    }
}
```

CONDITIONS

- if() {} //do this if true
- else {} //do that
- All conditions need to be boolean (true/false)
- if(1>0) //true
- if("richard" == "fred") //false

OTHER LOOP CONTROLS

- while()
  //will loop next statement / block while condition is true, but won't run at all if its false to start with
- do {} while(); //Will run at least once, and continue while condition is true
- for(set initial value; while(condition true); next value)
  // just shortcut for a while loop

BRACE WARS

- Braces and indenting, just for readability, compiler does not care
- However Unreal coding standards are

Brace on new line

```
if ("richard" == "fred") {
}

if ("richard" == "fred")
{
}
```

- Choose yours, just be consistent, there are options in VSS to help with automatically indenting etc.