Translation units & the Preprocessor

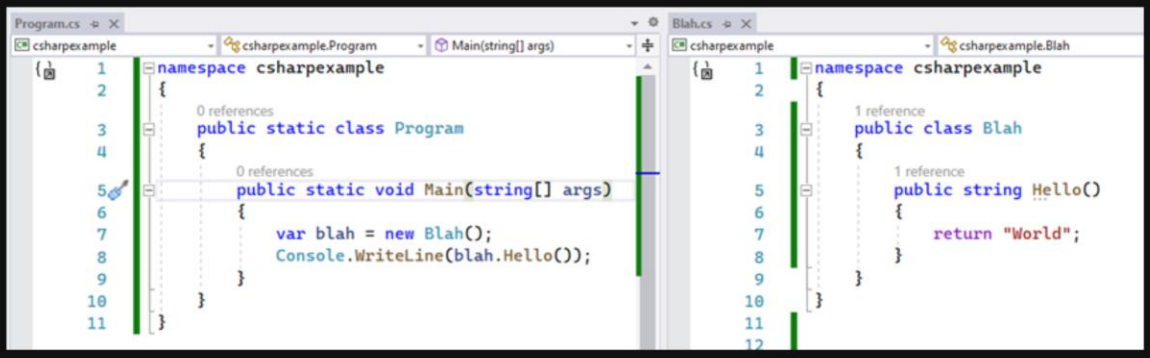
REDPOINT

Welcome!

- This is the fundamental "how does C++ compilation work" session
- C++ compiles very differently to most other modern programming languages
- If you don't understand how things are compiled, you're going to get confused very quickly
- We're going to look at how C++ compiles, how it compares with C# and everything that goes on under the hood
- **Aim:** You understand the path your code takes from editor to the compiled binary

REDPOINT 

How does C# compile?

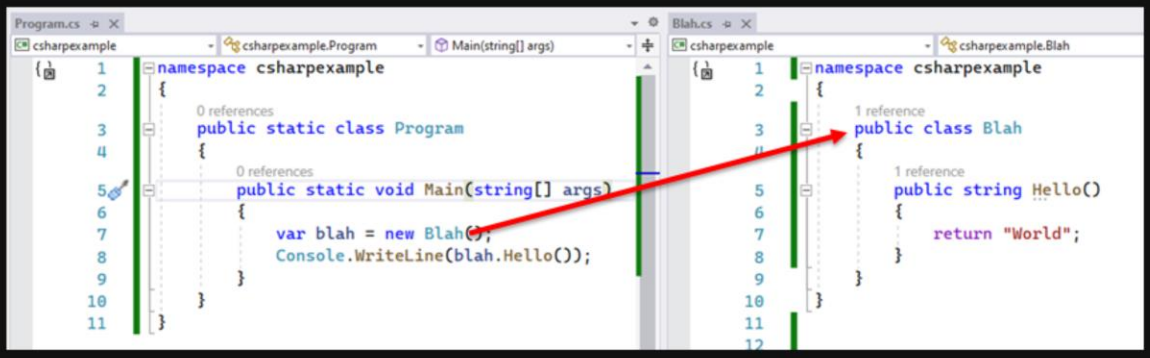


This works because the C# compiler reads all files at once:

```
csc.exe /out:obj\Debug\net6.0\csharpexample.dll Blah.cs Program.cs
```

REDPOINT

How does C# compile?



Because it reads and parses all files at once, it can know what "Blah" refers to in Program.cs!

REDPOINT

C++ does not compile this way

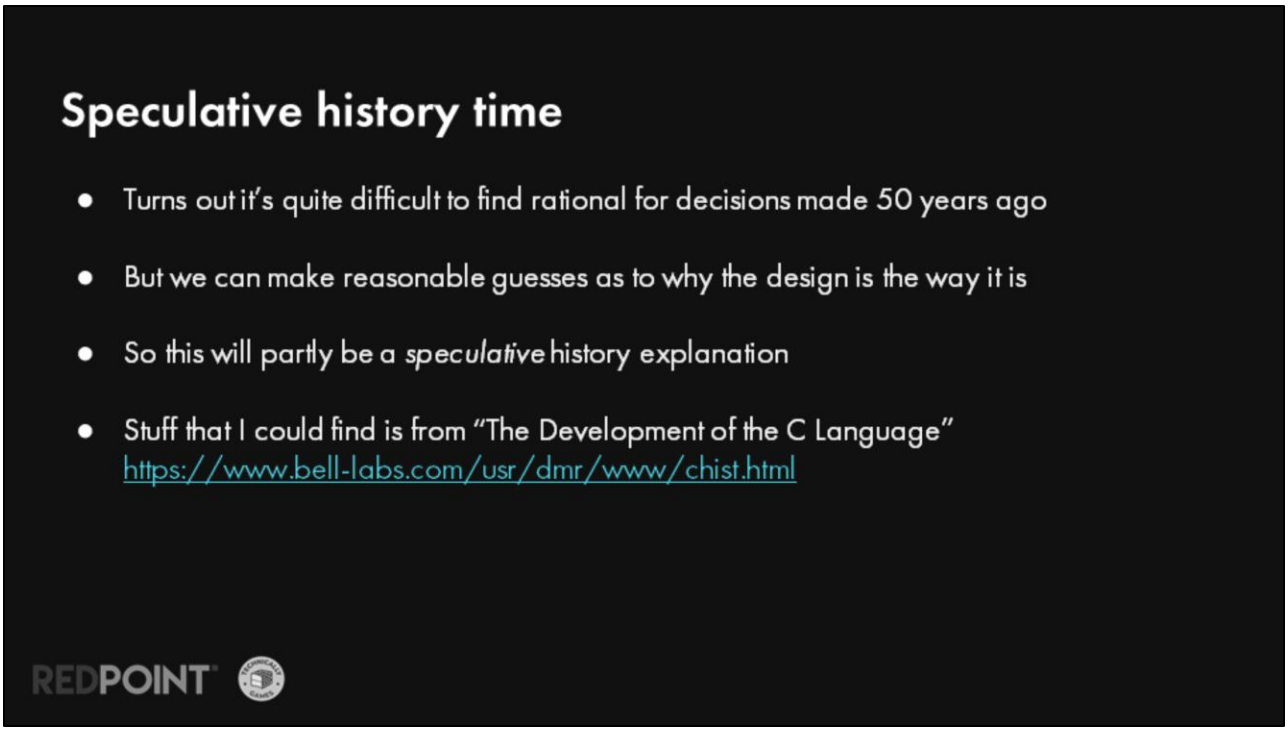
REDPOINT

C++ compiles one file at a time

Each file is a “translation unit”

REDPOINT


digital
pdp11/70
1972



- The Development of the C Language was written in 1993
- Even they don't know why they did some of the things they did

Constraints of PDP-7/PDP-11

- C originally written on PDP-7, later ported to PDP-11
- PDP-7 had **8KB memory**. The PDP-11 had **24KB memory**. Total.
- Extremely memory constrained for compilers to operate in.

What does a compiler need to do?

- Read files
- Lex (tokenise)
- Parse
- Abstract syntax tree
- Type resolution
- Emit machine code

- If you had all the source code in one file, you would run into a problem.

- While the resulting executable would work within memory limitations, there was not enough memory to compile the whole program at once.

Split up the work

Compiler
Compile individual .c files to object files

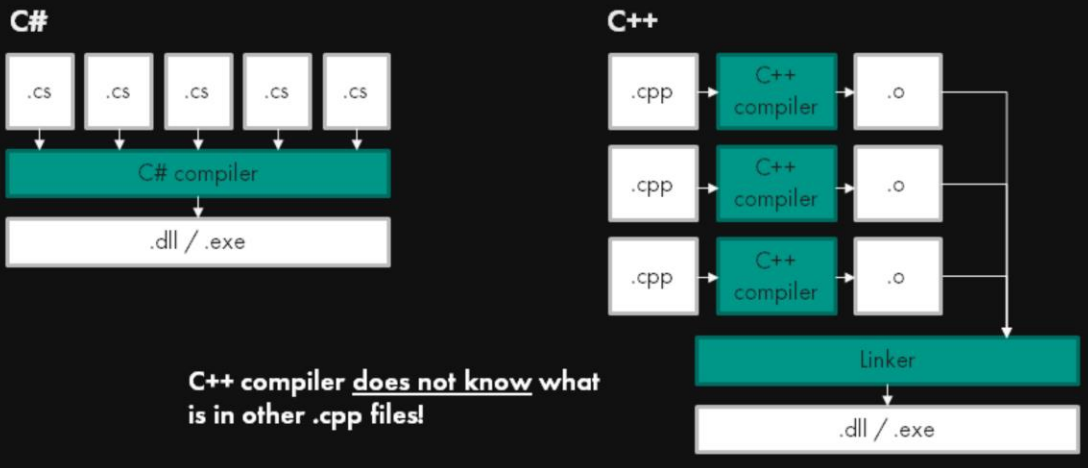
- Read one file
- Lex (tokenise)
- Parse
- Abstract syntax tree
- Type resolution
- Emit machine code as objects

Only need enough memory for compiler to compile each .c file in sequence...

Linker
Combine object files to get final program

- Read machine code in files
- Stitch machine code together
- Replace placeholder function pointers with actual function locations
- Emit resulting executable

Comparison again...



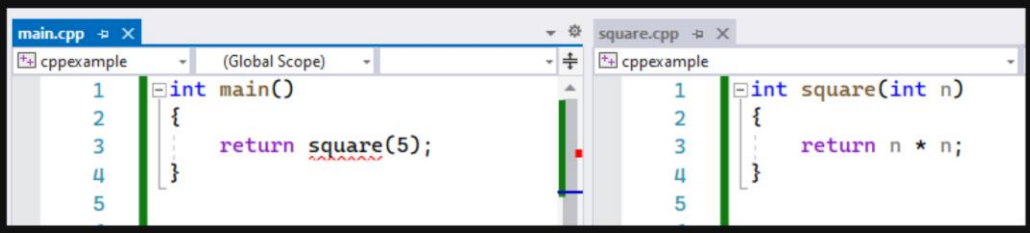
C++ compiler does not know what is in other .cpp files!

REDPOINT

How do we reference across files?

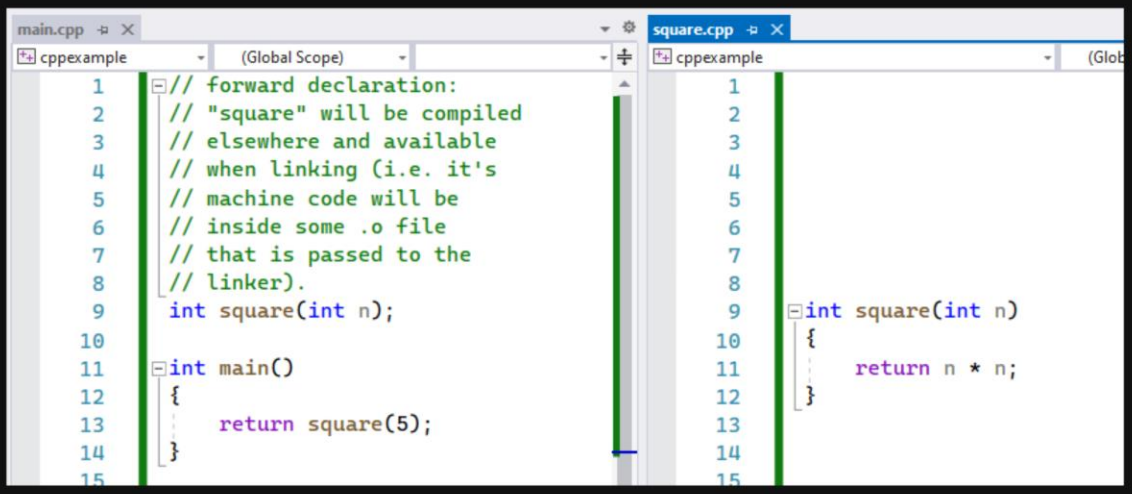
Let's say we want to call square() from main().

Given these files will be compiled separately, how can we tell the compiler what "square" looks like in main.cpp?

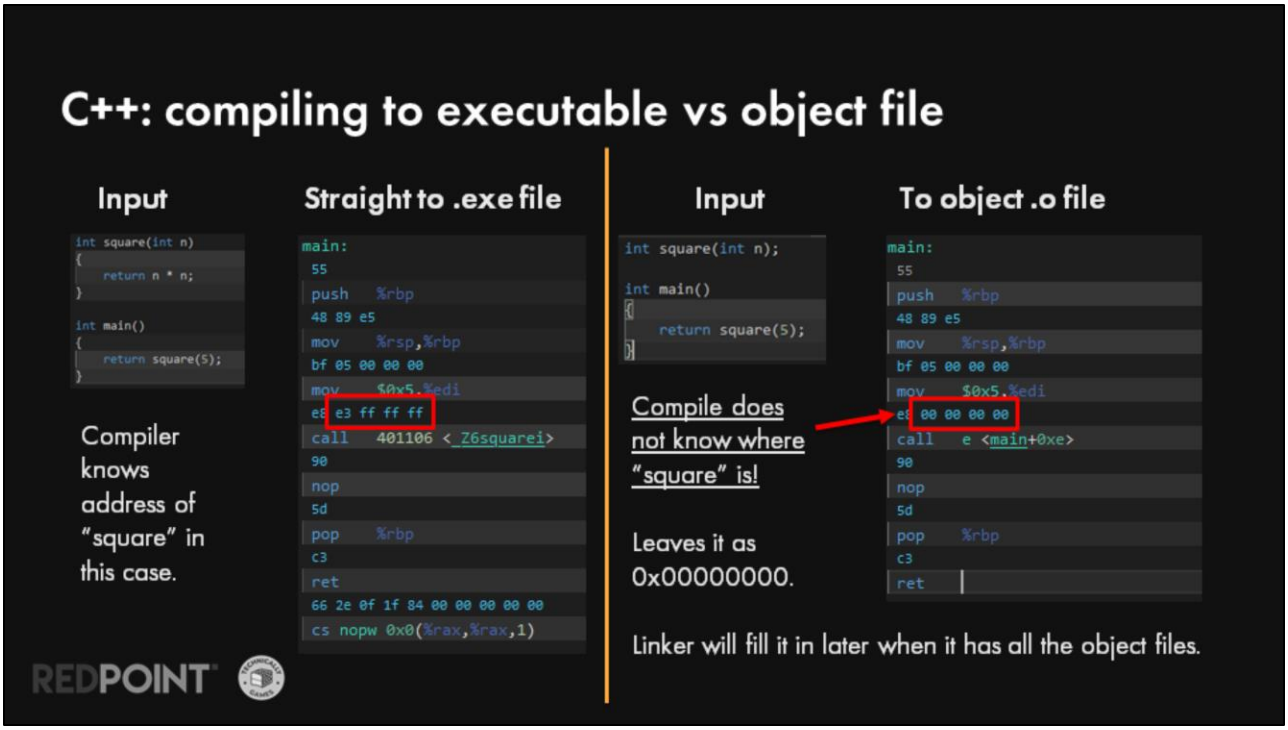


REDPOINT

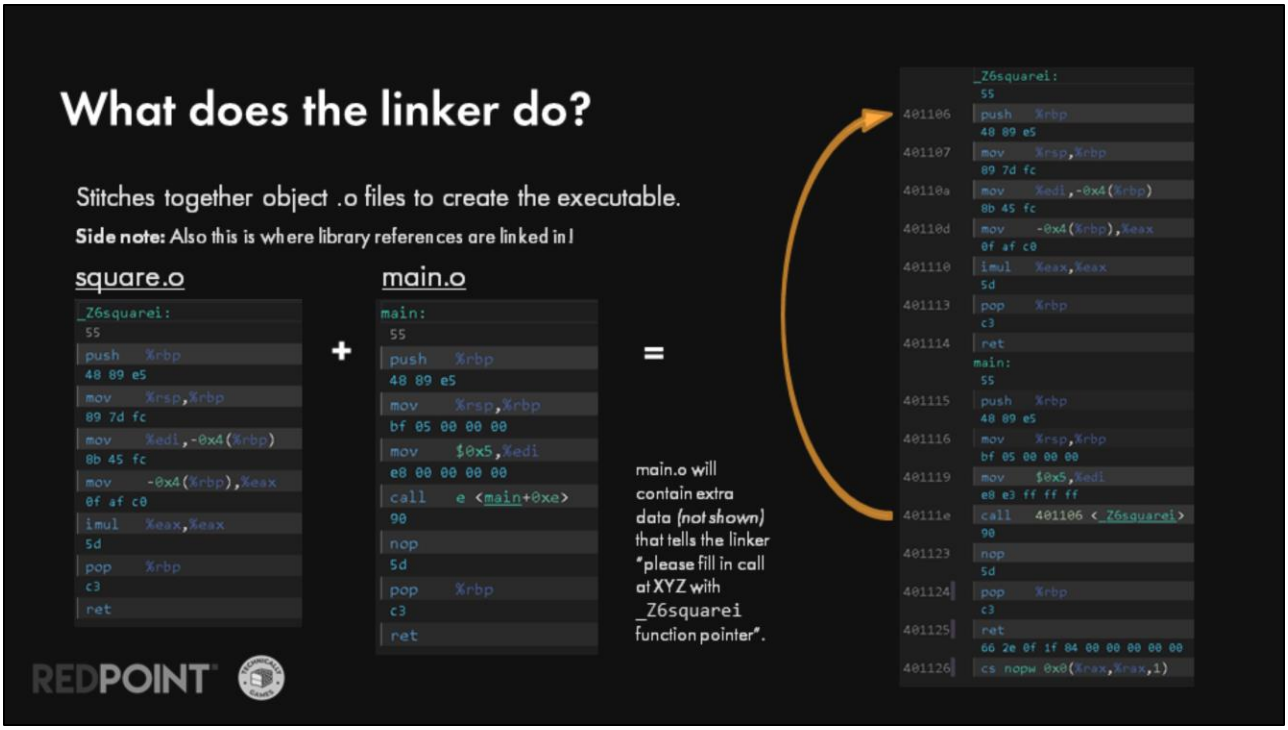
Forward declarations!



REDPOINT



- Left hand side: compiler knows the address of square so it can just put it straight in there
- Right hand side: compiler does not know where square is! It leaves it blank for the linker to fill in later.

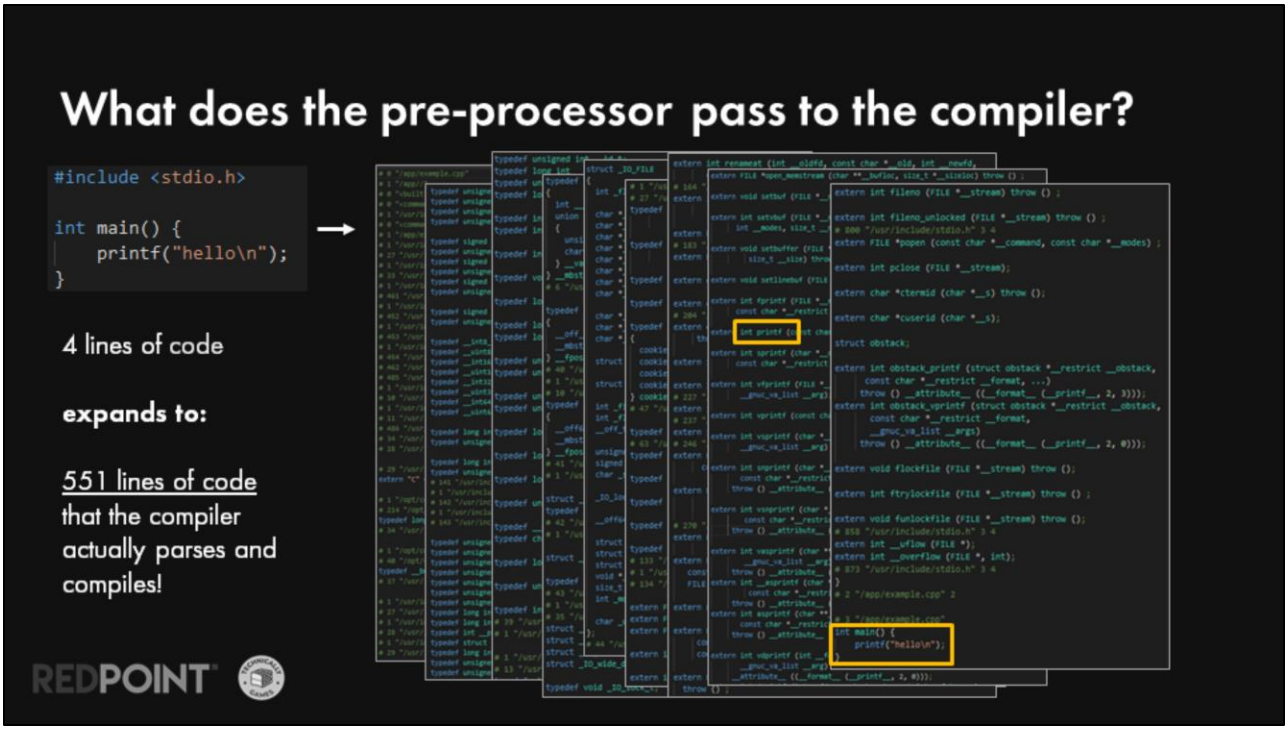


- Note how the linker replaced the call location with the actual location of the square() function.

- This is why if you forward declare something but then *don't actually implement it* you get linker errors.
- As far as the C++ compiler is concerned, the function you declared will be available but then the linker can't find it.
- The linker only has machine code and the raw symbol names though: **that's why linker errors can be super cryptic!**

What to do when lots of .cpp files need to forward declare the same function?

- Could forward declare in every file, but tedious.
- Pre-processor created so you can #include common code into multiple translation units
- This is a LITERAL TEXT INCLUDE – nothing fancy. You can even run the pre-processor on non-C++ code because it doesn't care about C/C++ code at all.
- So what does the compiler actually see when you compile some normal piece of code?



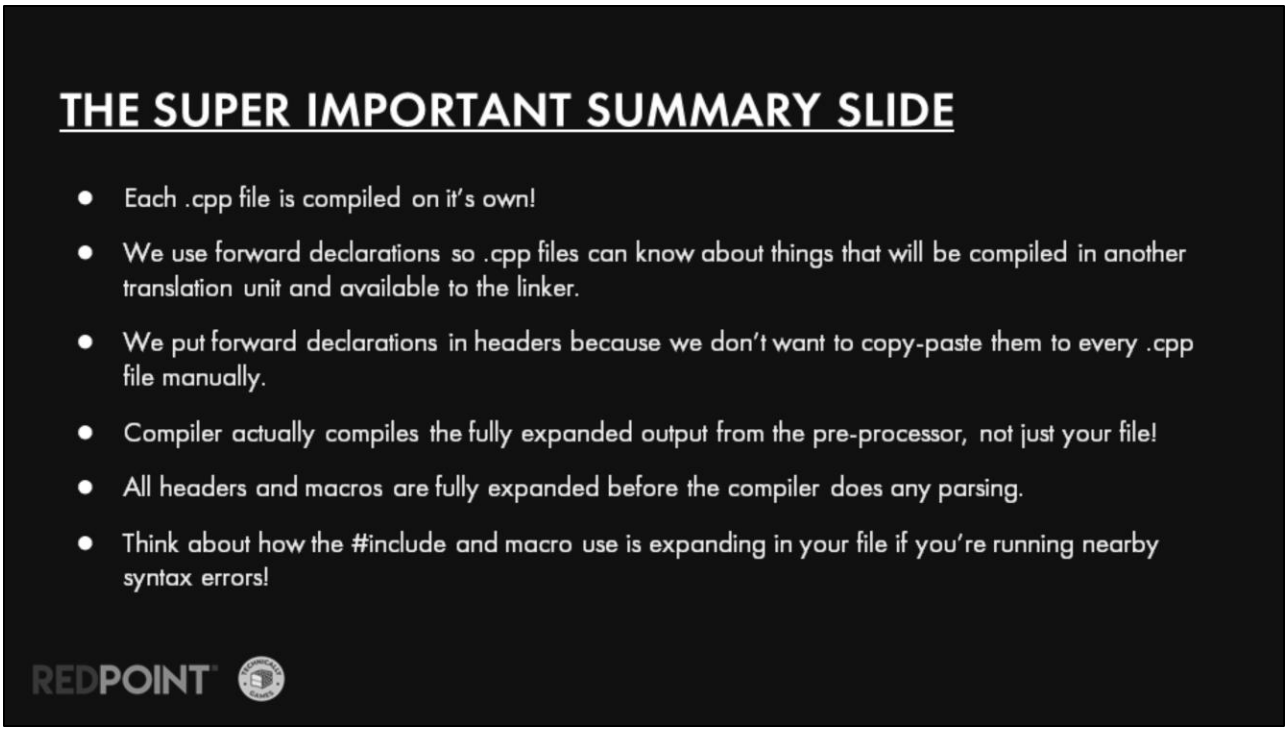
- Ran out of space on this slide for all the emitted code…!
- This is for a single, very core include of “stdio.h”.
- Imagine how many lines of code the Unreal Engine headers expand out to.
- This is why syntax errors in headers can cause problems elsewhere!
    - If you’re missing a brace or semicolon in the header, when it all gets expanded into a single file that can cause syntax problems in the files further down.

Side note: Seems like a lot of redundant effort...

- Imagine lots of files that are all including the same headers.
- Each compiler invocation has to parse all of that header code every time.
- Slow and redundant!
- Introducing PCHs: Pre-compiled headers.
- Special compiler mode to compile a set of headers down into a special binary format. Probably a serialized AST (not machine code!)
- Each compiler then loads PCH instead of re-parsing all common headers – speeds up compilation.

Other things the preprocessor does

- Pre-processor also lets you define macros with #define FUNC(...) ...
- Again this is just text processing, so you can emit anything with the pre-processor including things that aren't syntactically valid.
- Macros are very very important to the UCLASS() / UObject system which we will talk about later.
- But for now important take-away points are...



- This is the super important summary slide.
- If you pay attention to only one thing in this session, make sure it is this slide!

- Sure seems like C++ is hacks upon hacks upon hacks….. That's because it is :P