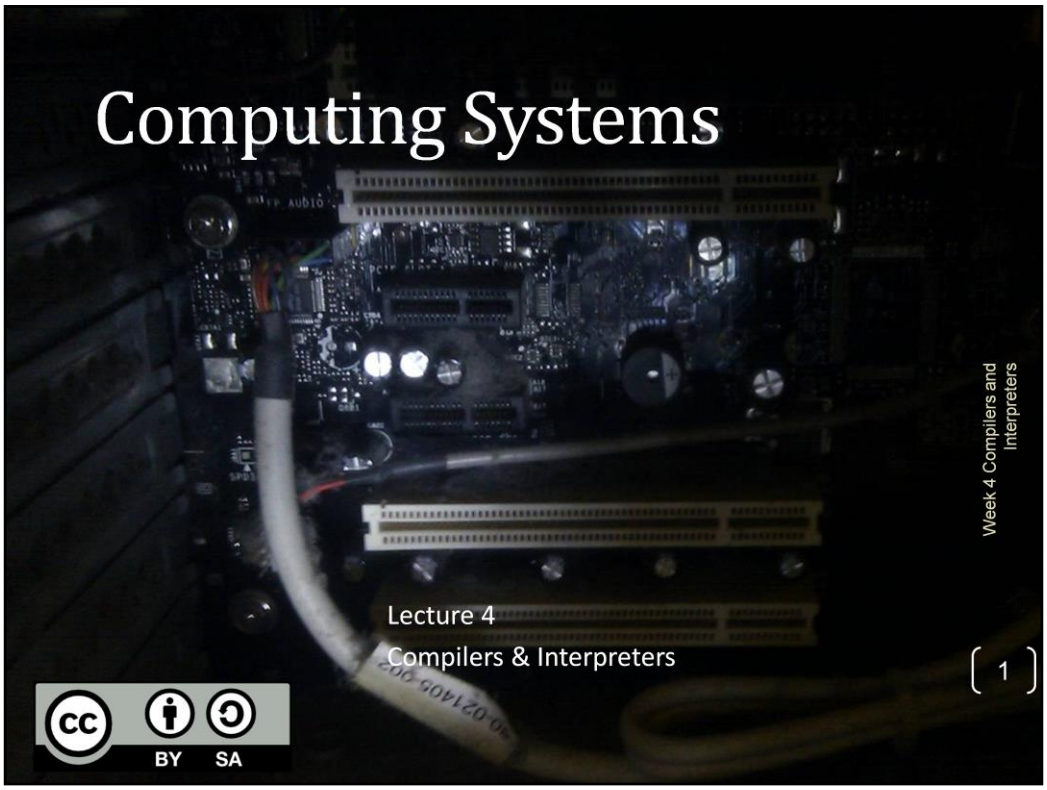
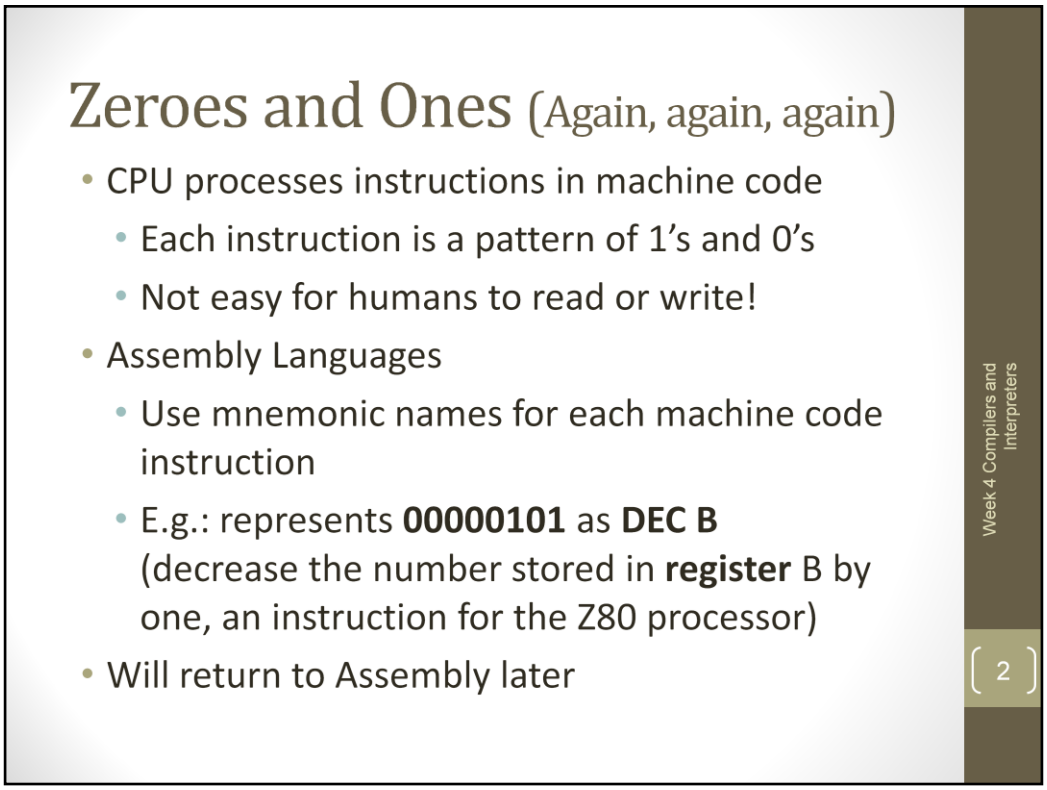

Computing Systems
Week 4 Compilers and Interpreters
Lecture 4
Compilers & Interpreters
1
BY SA

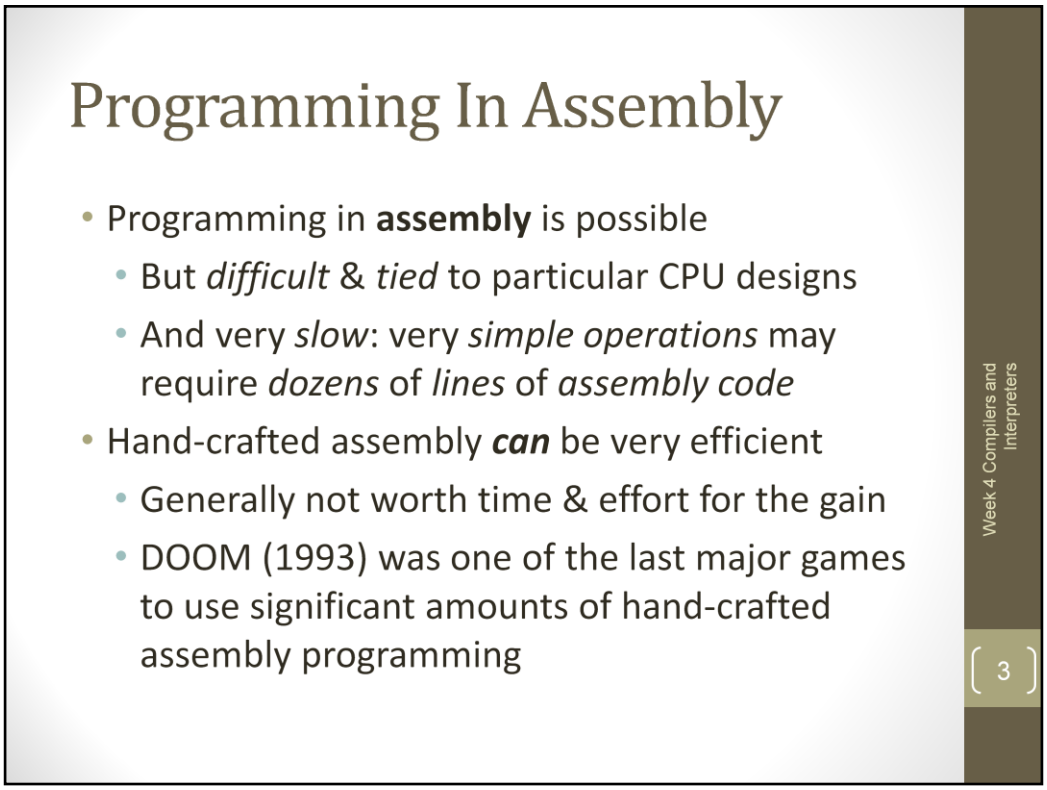


The CPU processes instructions in Machine Code.

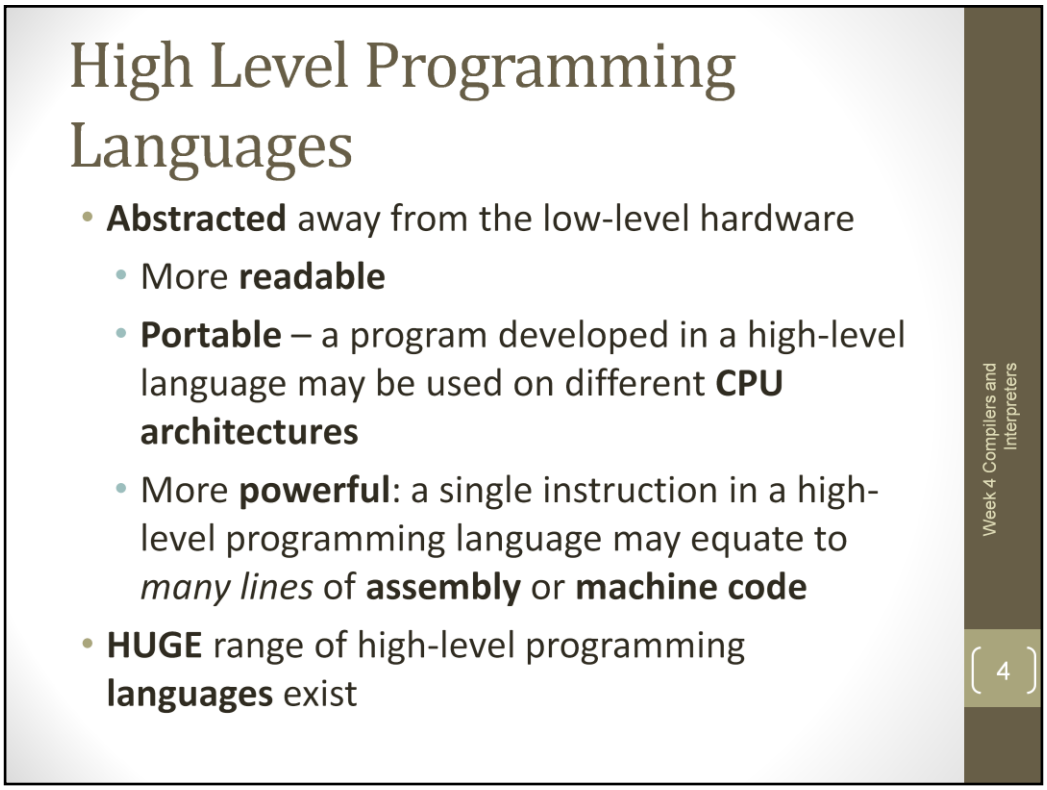
The transistors, the key components which make up the CPU, only understand ON / OFF.

All of the instructions the CPU processes are patterns of 0 / 1s, which are not easy for humans to read or write. Earliest programs were written in machine code.

People write using mnemonics which are easier than binary. The assembler can turn turn mnemonics into binary.

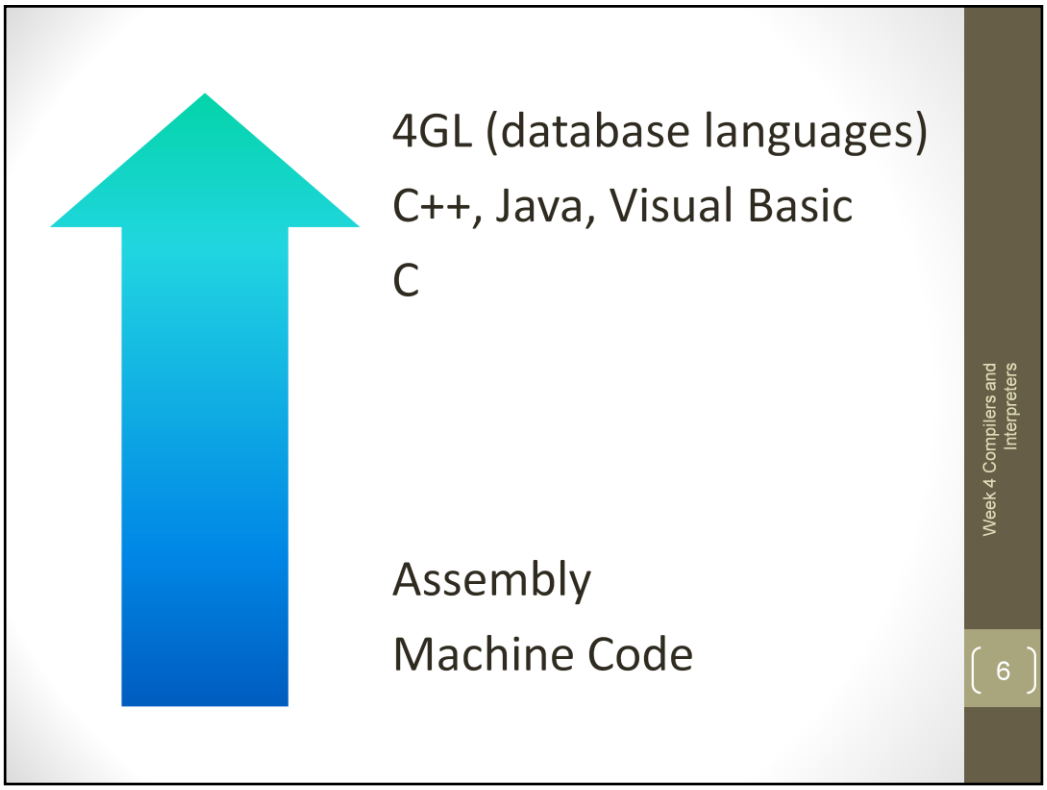


Quake also used assembly to program games.



Can write the program once then compile to run on different computers with different CPUs in them.

What Programming languages have you used?

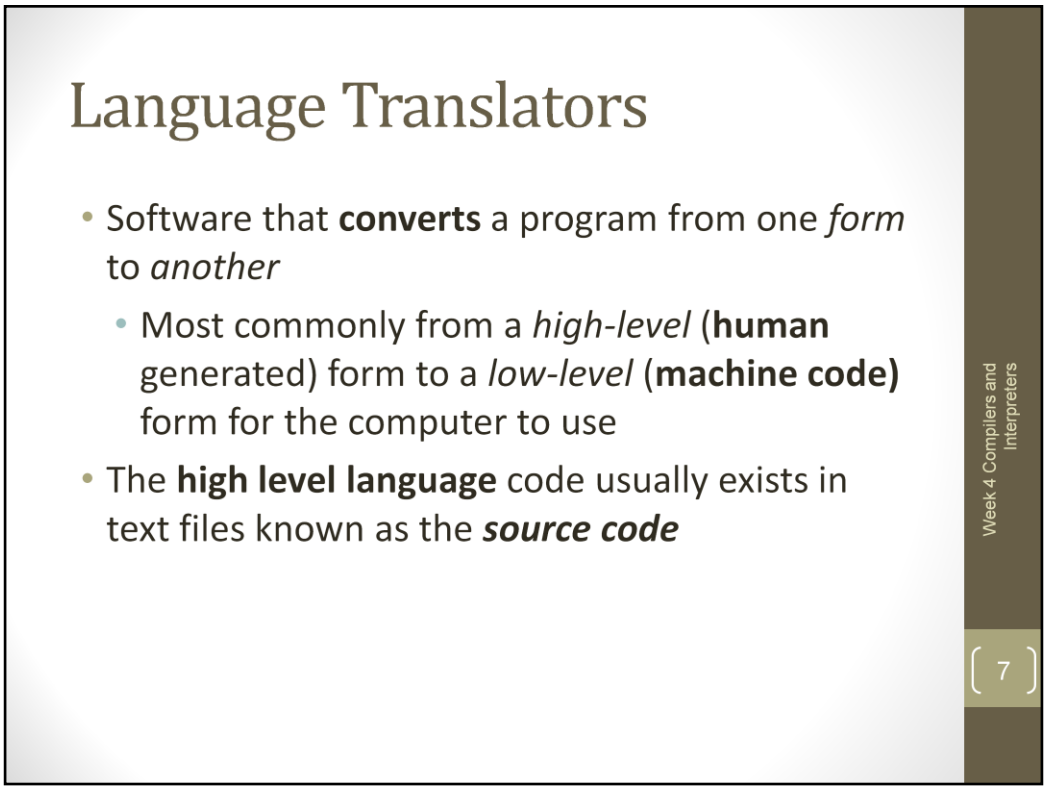


High level and low level programming.

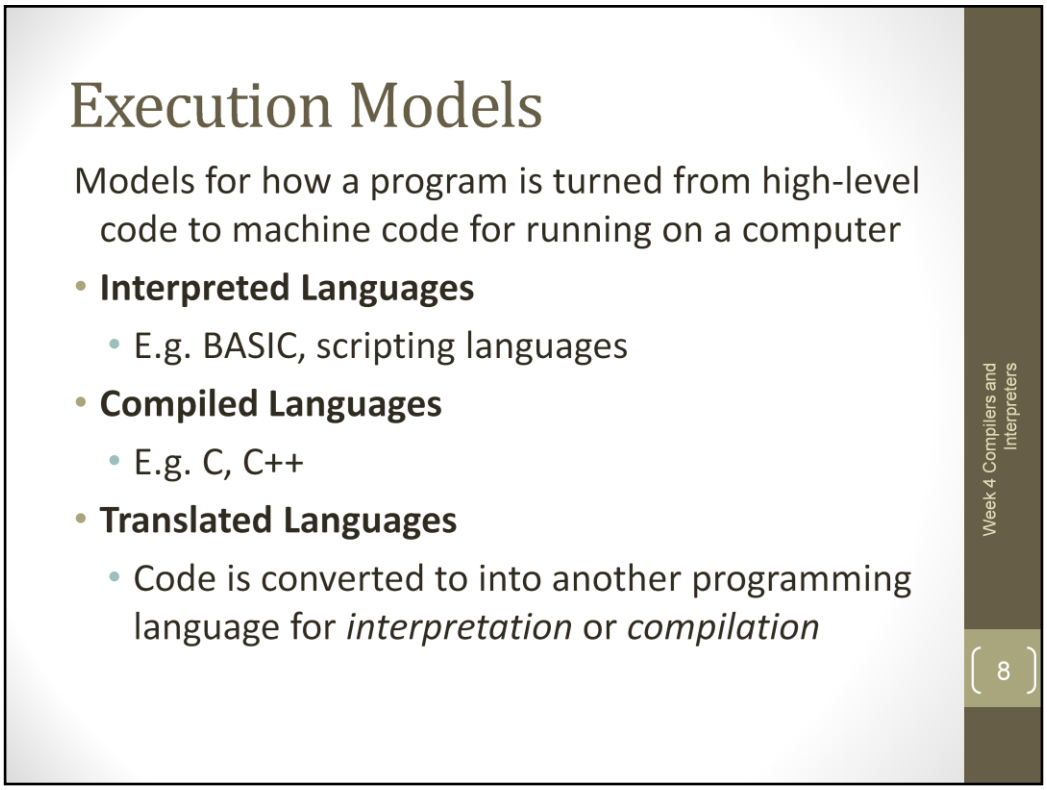
Assembly and Machine code are low level languages are closer to the processor.

C++ etc are high level language.

C is considered a lower level language as it allows a lot of direct access to the underlying hardware, and has some lower level language features.



Translator converts source code into machine code.



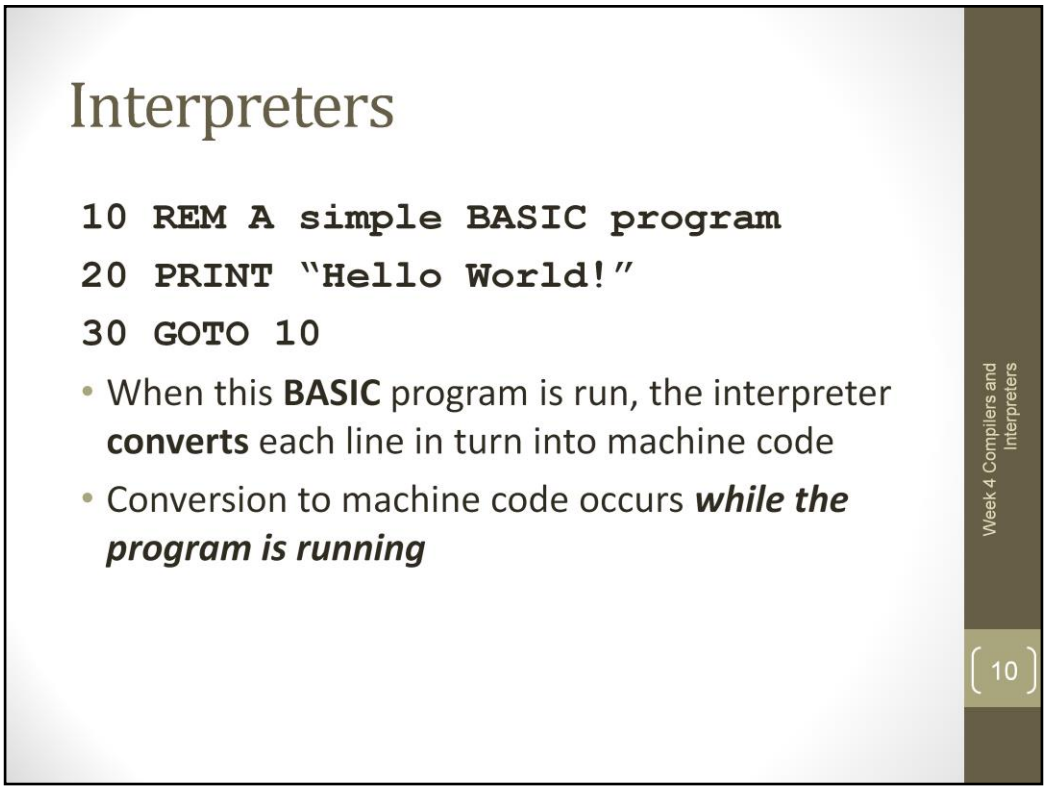
Translator can take a source code and convert into C for example from another programming language..

Languages that are usually interpreted:

BASIC (most versions), **Javascript**, **Lua**, **PhP**, **Python**, **Perl**, **Game Maker**, **Ruby**, ...

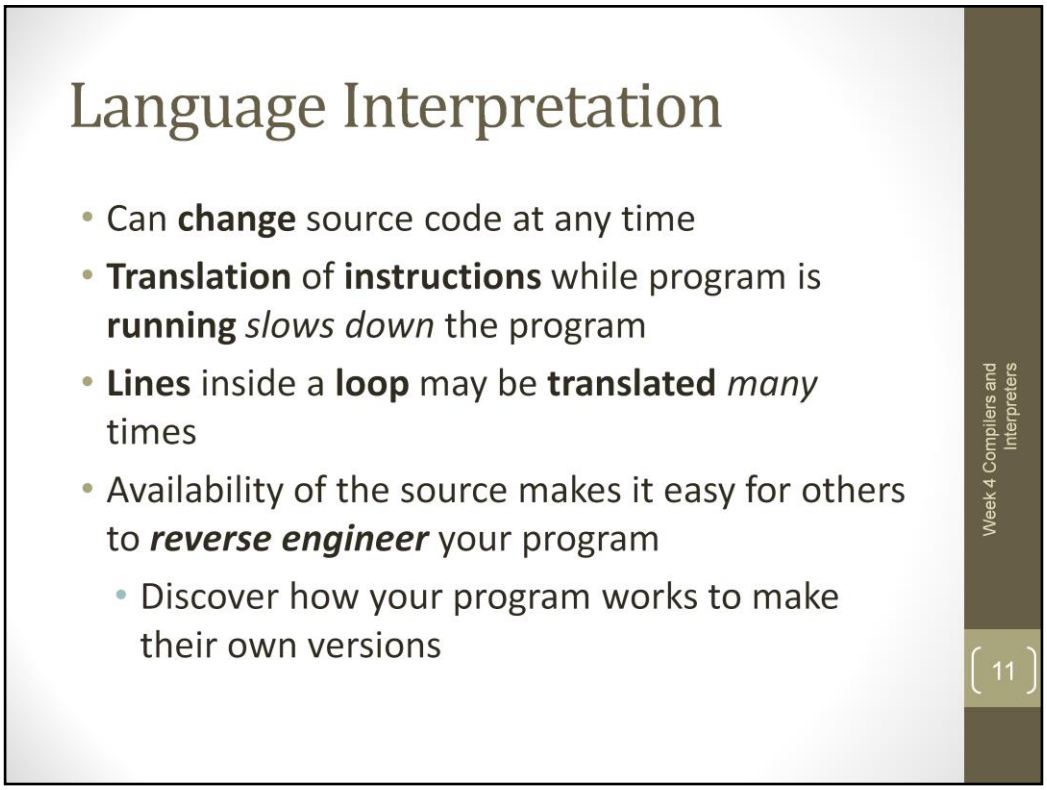
Languages that are usually compiled:

Algol, **C**, C++, **Objective-C**, **Pascal**, **Ada**, C#, **COBOL**, **Delphi**, **Eiffel**, **Common Lisp**, **Forth**, **Java**, **Scheme**, **Smalltalk**, **ML**, **OCaml**, **Visual Basic**, ...



Each line is converted from source code to machine code each time the program is run.

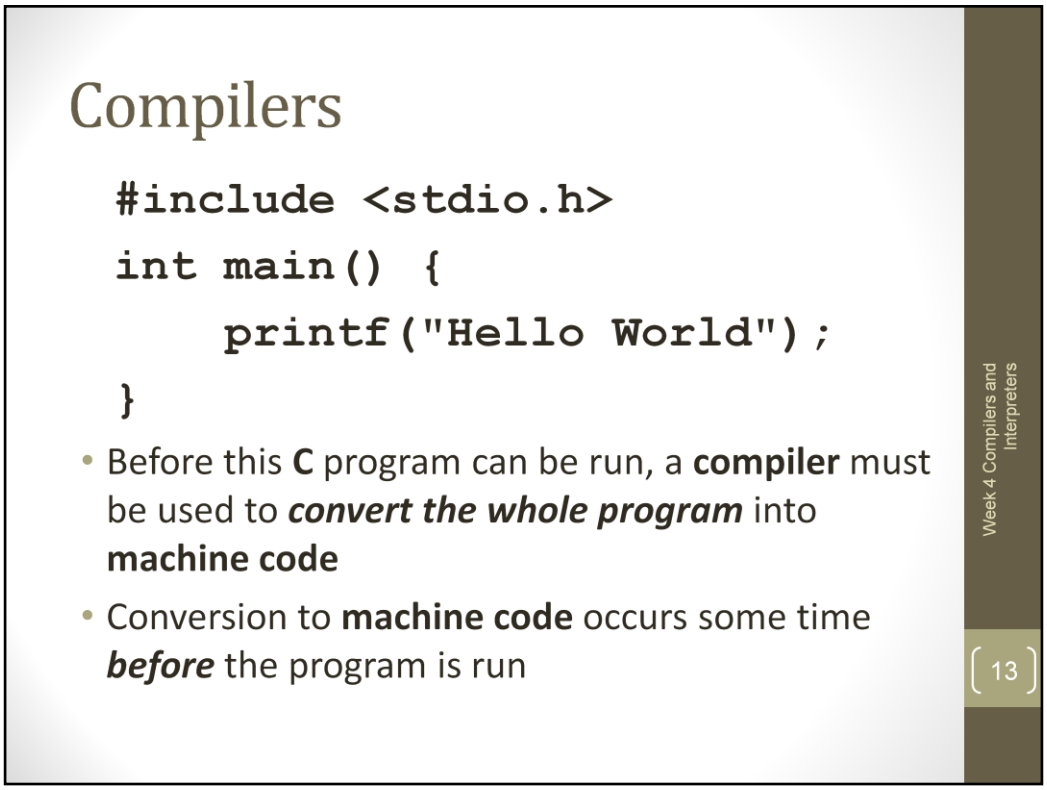
Advantages are: the code is often more flexible and can be changed at any time. But it is slower as some line e.g. the ones inside a loop are interpreted many times.



Flexible, easier to develop and test.

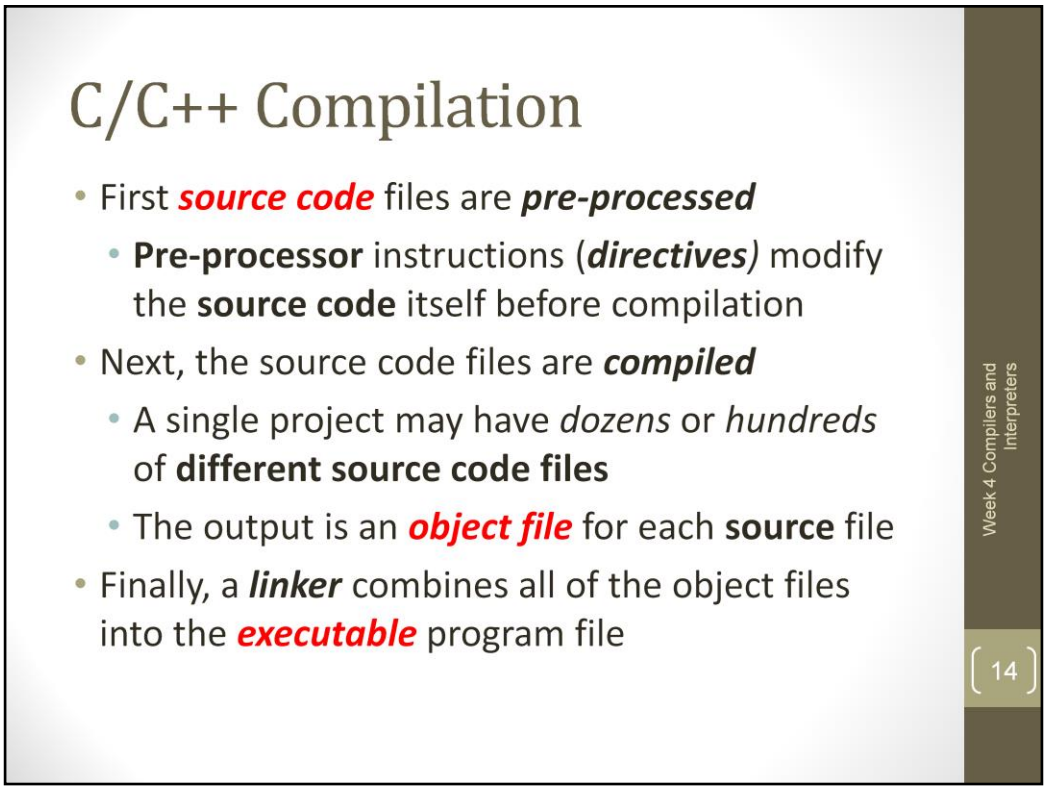
Quick quiz

- Join the ‘Socrative’ app ‘Room 642124’ and try the quick quiz.



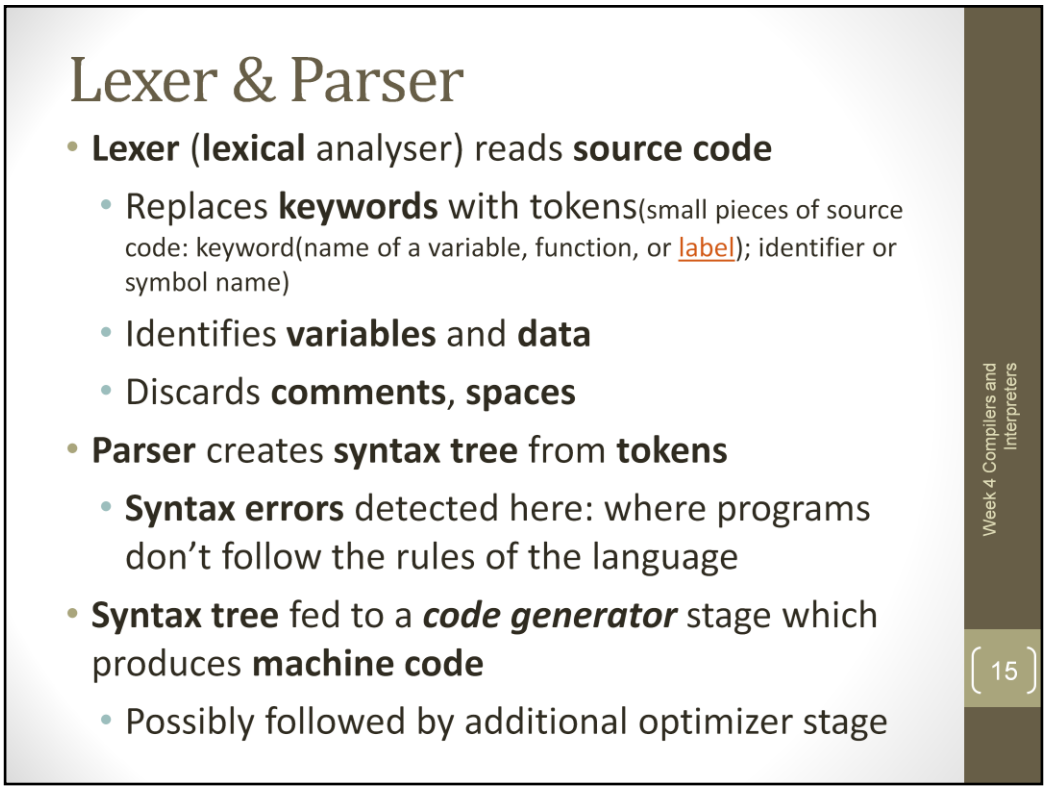
`#include <stdio.h>`

The pre-processor operates on this statement. It tells it that it needs to take the contents of the file stdio.h and adds it into the source code before compiling.



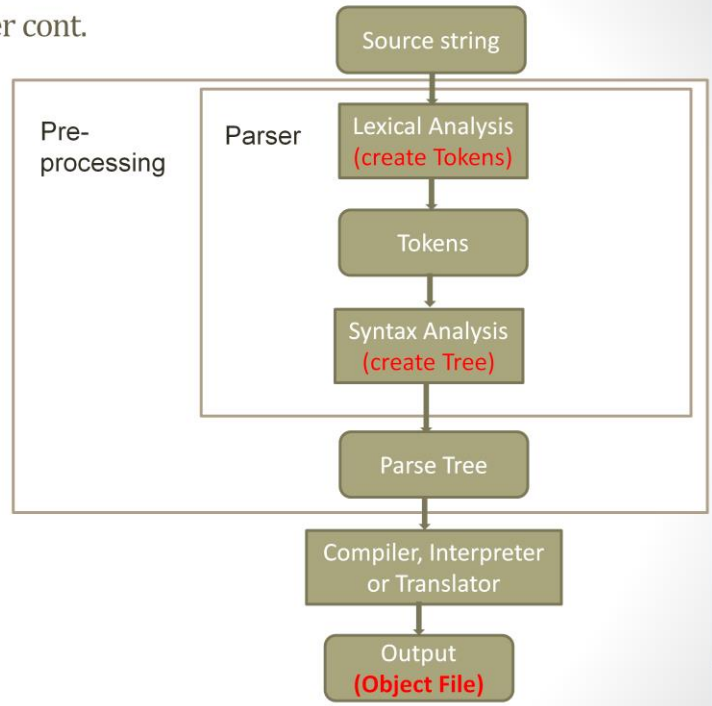
Each source code file that is compiled (may have dozens or hundreds) creates an object file (in machine code).

Compilation has two phases.

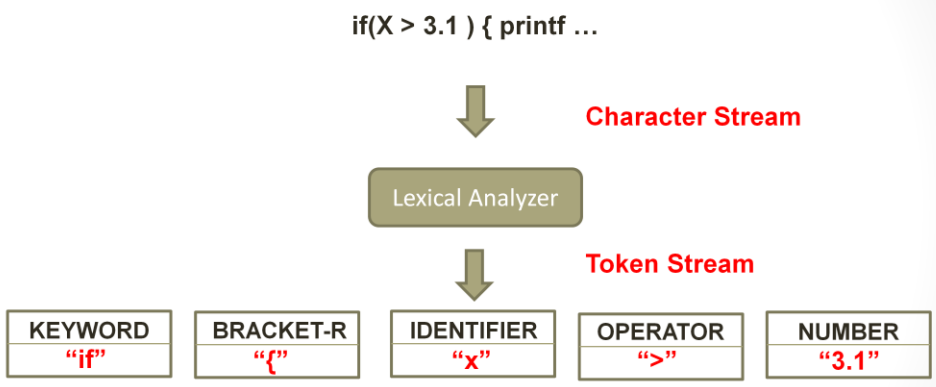


Lexer inputs into the Parser then Parser feeds into a Code Generator.

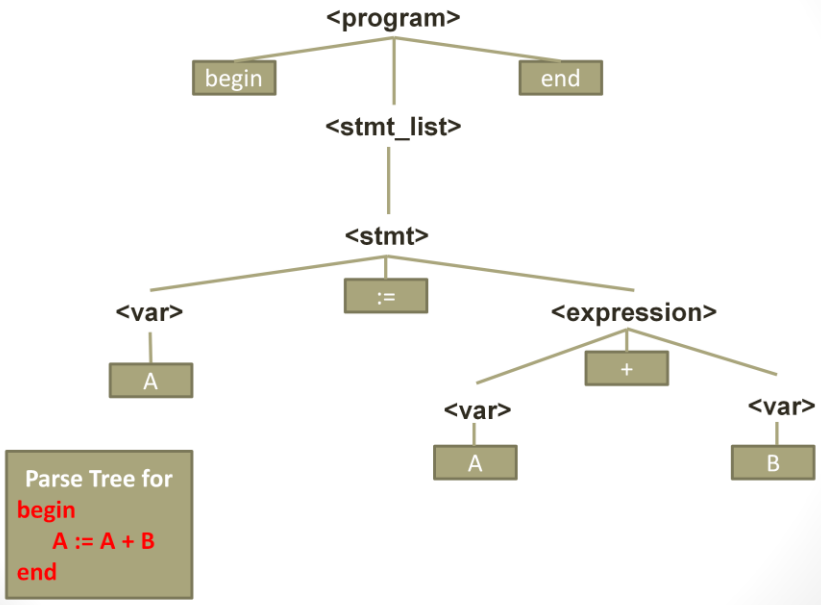
Lexer & Parser cont.



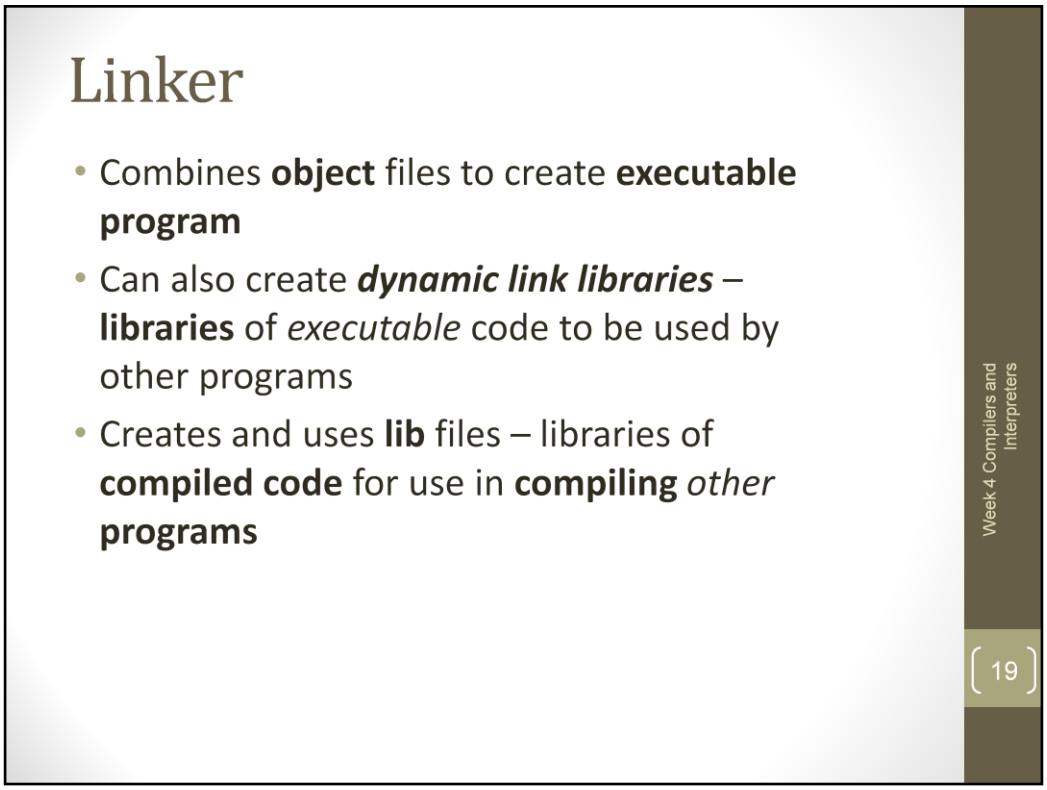
Lexical Analysis Tokens example



Parse Tree example



Parse Tree for
begin
A := A + B
end



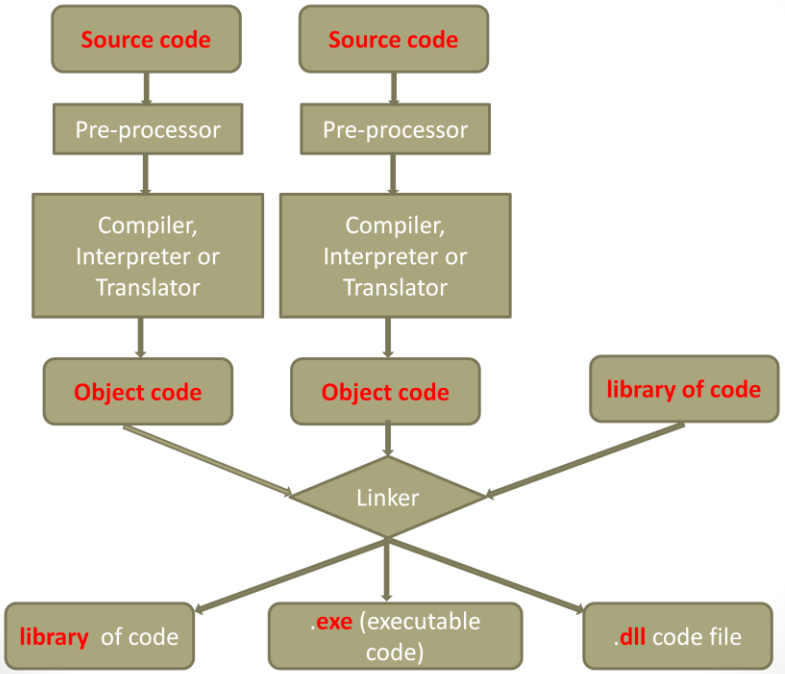
When we compile a file we end up with an object file.

The linker combines the Library (Lib, pre-existing object files) files with the Object files.

Produces program output – exe an executable file(can run itself), an object file(lib) stored in the library. lib files are compiled code that can be used in the linking stage of compiling a new project.
It is mainly used for creating code libraries for third party use - and means that the code library itself does not need to be recompiled every time it is used, it is added at the linking stage instead.

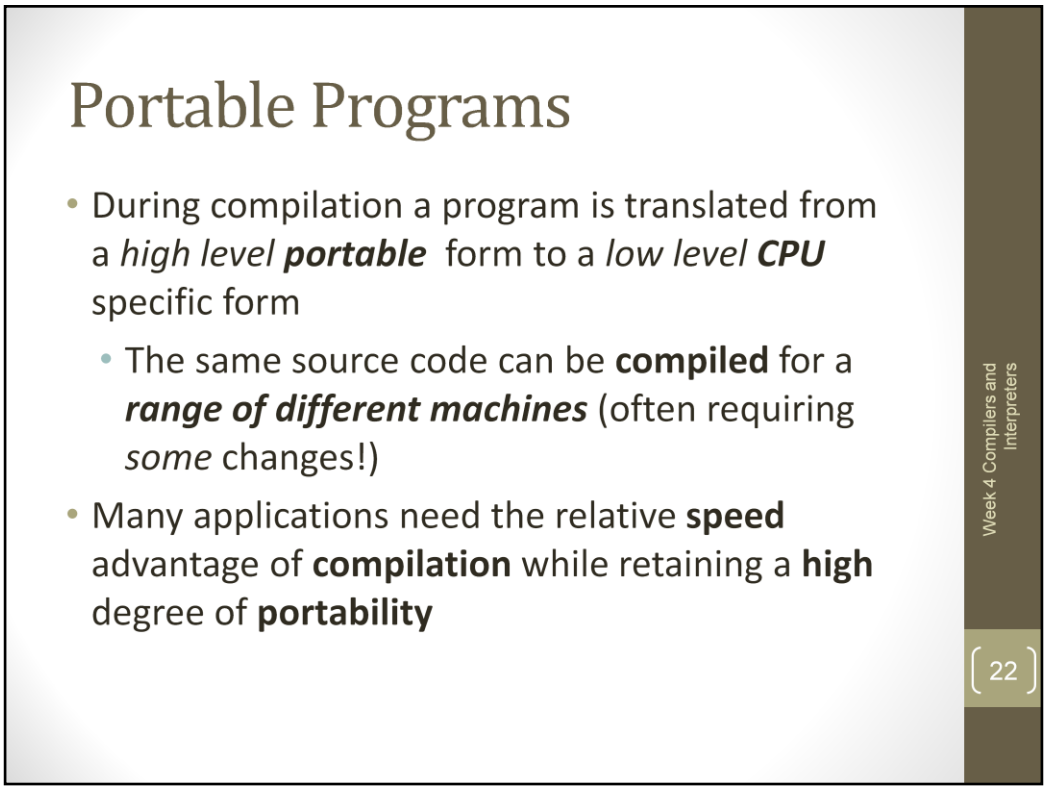
dll files, in contrast, are used at run-time, and allow third party libraries to be used in a different way - instead of being 'statically linked' into the project (and being included in the executable file), the library code is kept seperate. This can be particularly effective if the library is very commonly used - instead of having the code added to every project that uses it, the DLL only needs to be installed once. Additionally, some open source projects allow commercial use only where added as dynamic link libraries.

Linker cont.

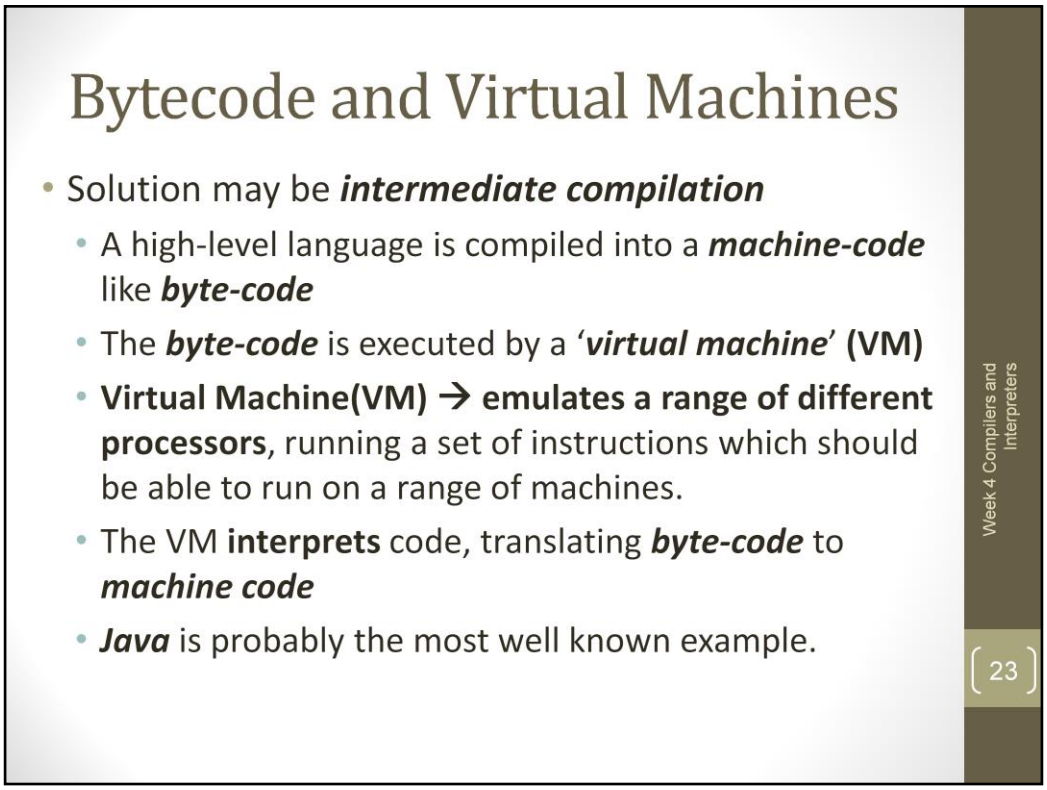

Source code
Source code
Pre-processor
Pre-processor
Compiler, Interpreter or Translator
Compiler, Interpreter or Translator
Object code
Object code
library of code
Linker
library of code
.exe (executable code)
.dll code file

Comparison of compiled and interpreted code

- Using a ***compiler*** separates *translation* and *execution* of a program.
  - In contrast of an *interpreted* program the *source code* is **translated** only **once**.
  - The *object code* is *machine-dependent* meaning that the *compiled* program can **only** be **executed** on a **machine** for which it has been **compiled**
- An ***interpreted*** program is **not** *machine-dependent* because the *machine-dependent* part is in the interpreter itself.
  - A pure interpreter **reads** the source text of a program, **analyzes** it, and **executes** it as it goes. This is usually **very slow**--the interpreter spends a lot of time analyzing strings of characters to figure out what they mean.
  - A pure interpreter must **recognize** and **analyze** each **expression** in the **source** text **each time** it is **encountered**, so that it knows what to do next.

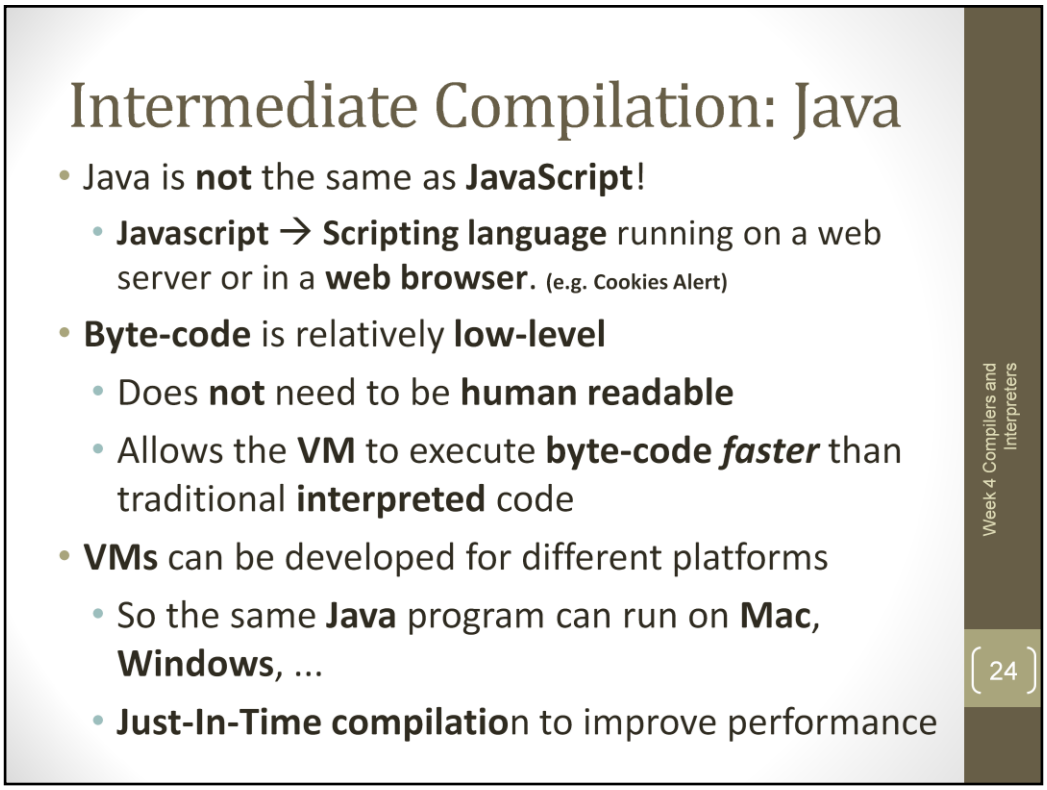


Pre-processor directives can be used to make changes to code e.g. **#include**. e.g. depending on compiling for a Mac or Windows machine, different Libraries might be included. i.e. we do not need to recompile for a different machine.



Byte code is executed on a **'virtual machine' (VM) → emulates a range of different processors**, running a set of instructions which should be able to run on a range of machines. Giving better portability.

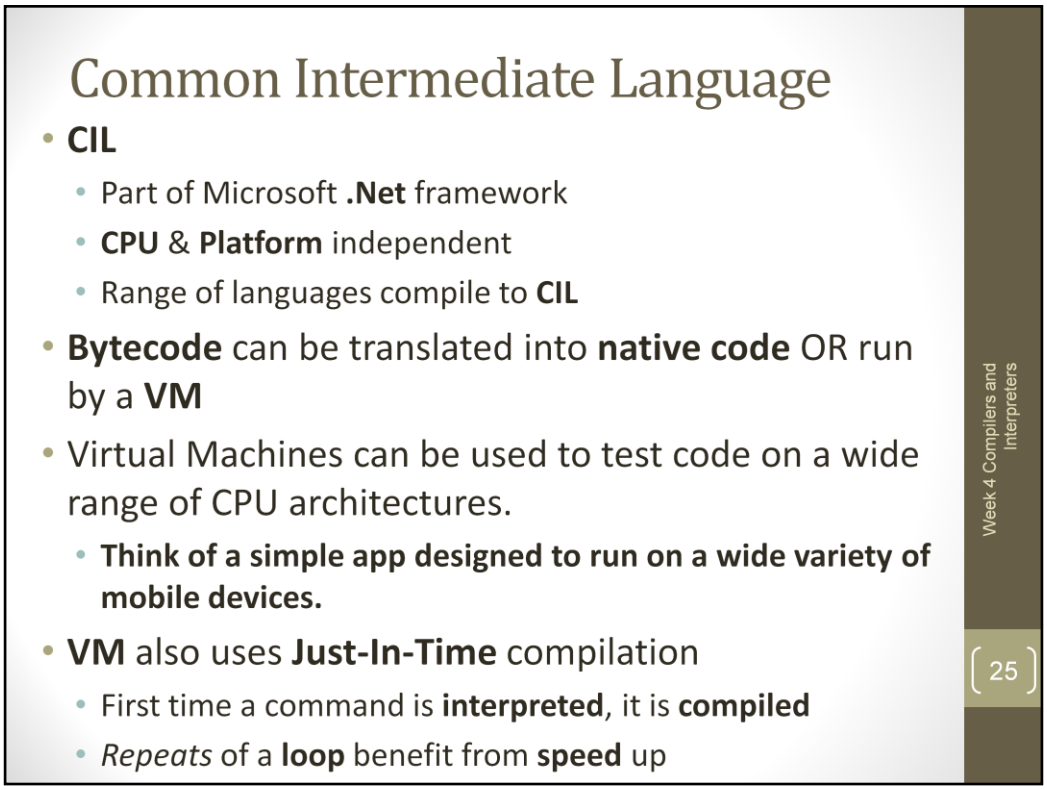
Java is a portable programming language, therefore Java can run on any computer as long as a virtual machine is available. Therefore it is more efficient.



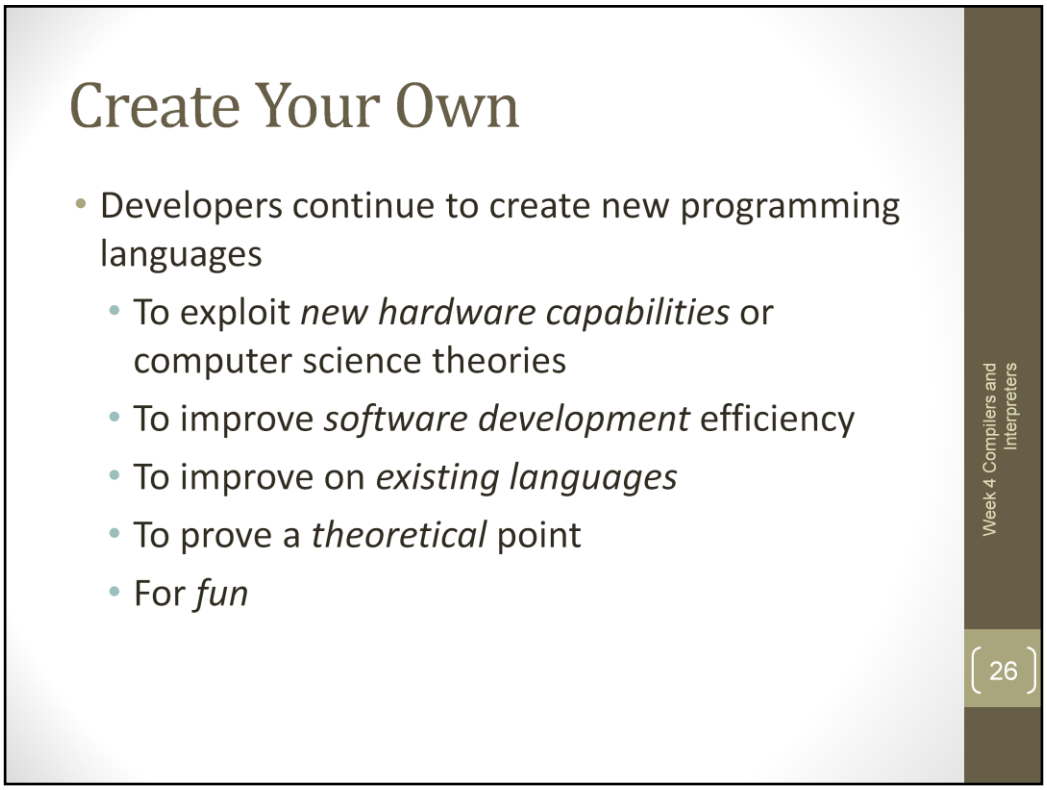
Java → programming language which gets compiled to byte – code which runs on a virtual machine, byte code is relatively low level and does not need to be human readable.

Javascript → Scripting language running on a web server or in a web browser.

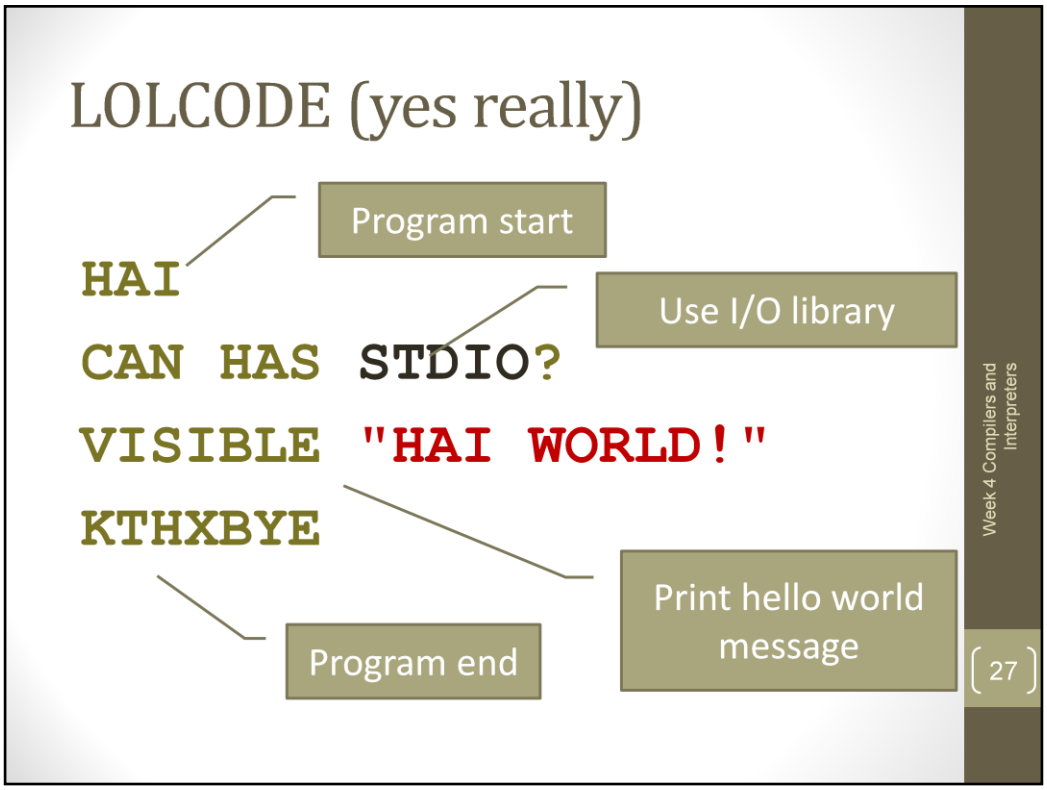
Just-In-Time→ allows processor to convert a command to machine code but it remembers the compiled version, therefore in a loop it only needs compiled once.



CIL → e.g. C-sharp and XNA, can use XNA to develop a game which can run on Windows, running on an Intel CPU, then run the same code to run the same game on an x-box, which uses a different processor architecture and has a different set of low level command which it understands.

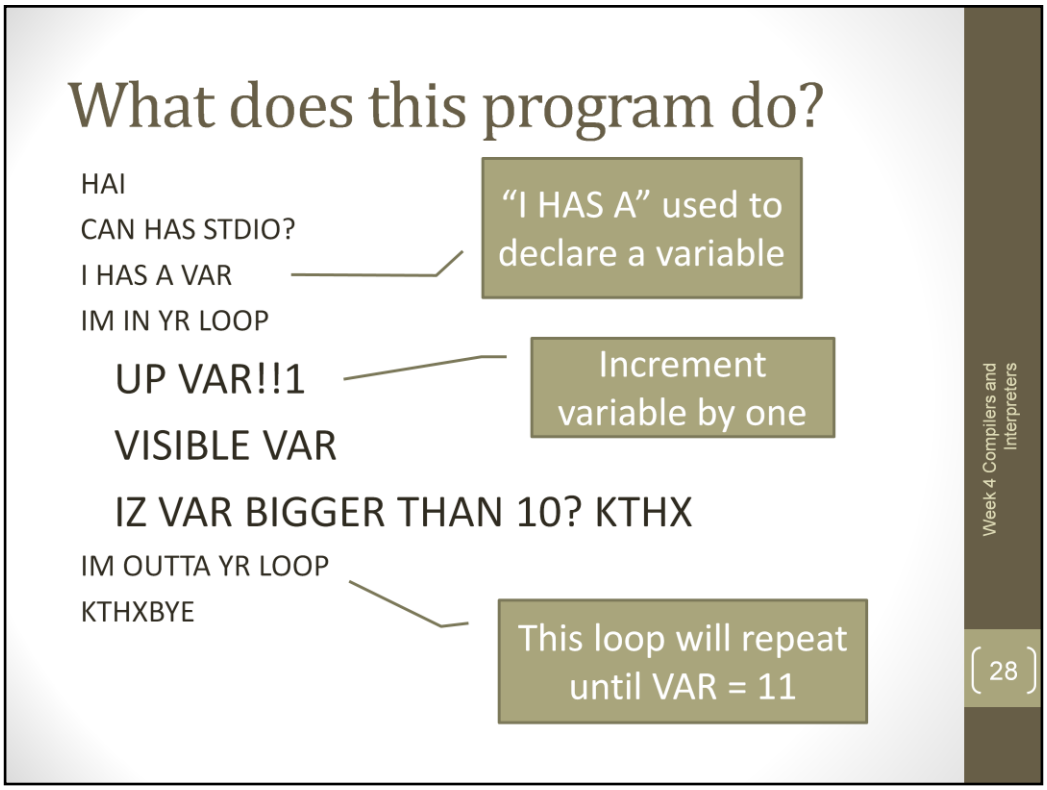


A virtual machine lets you use more portable code. But you could develop your own.



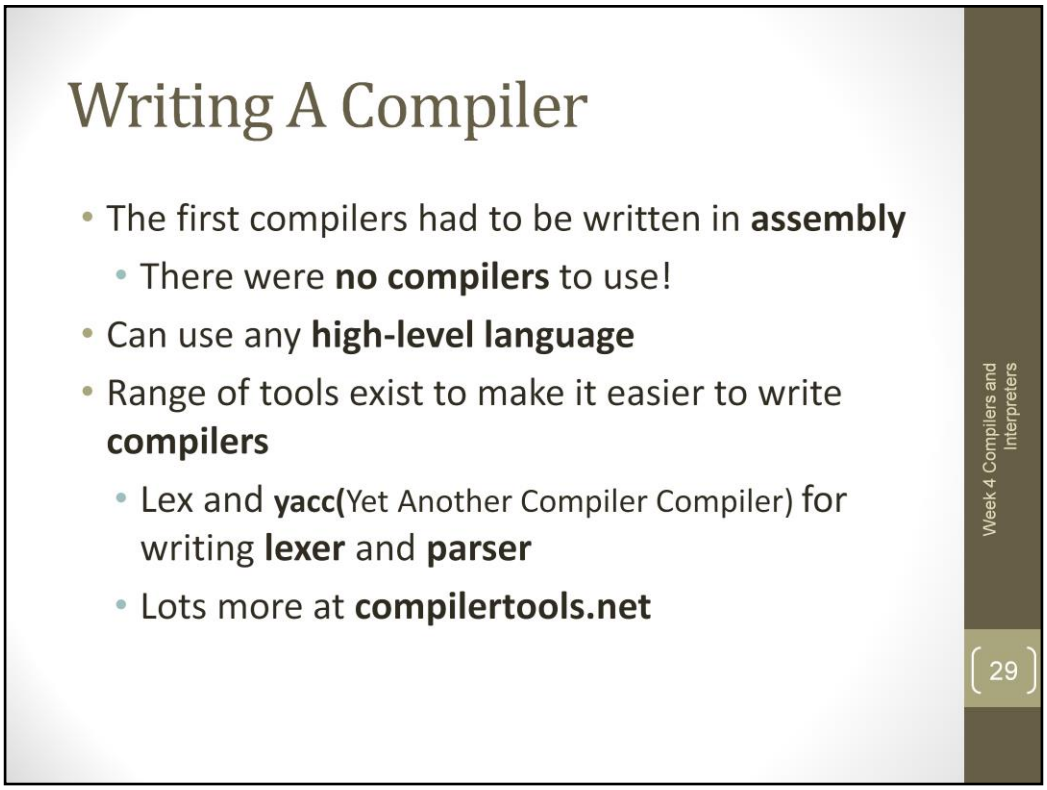
HAI → program start

KTHXBYE → program end



Loop to print numbers 1 to 10 – though code inspection makes it look like it will print numbers 1-11

Worth checking out...



Lex → Lexer

yacc → Parser, Yet Another Compiler Compiler

Quick quiz

- Join the ‘Socrative’ app ‘Room 642124’ and try the quick quiz.

Programming Paradigms

- Over the past half century a number of different **programming paradigms** have emerged
- Increasing **complexity** and **size** of software systems required changes to the manner of *software construction*
- A **programming paradigm** is a fundamental style of computer programming, a way of building the structure and elements of computer programs.

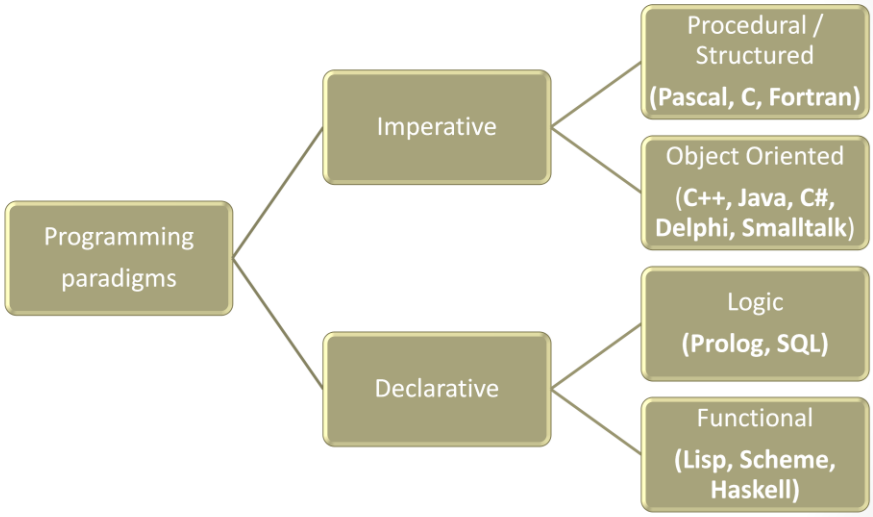
Programming Paradigms cont.

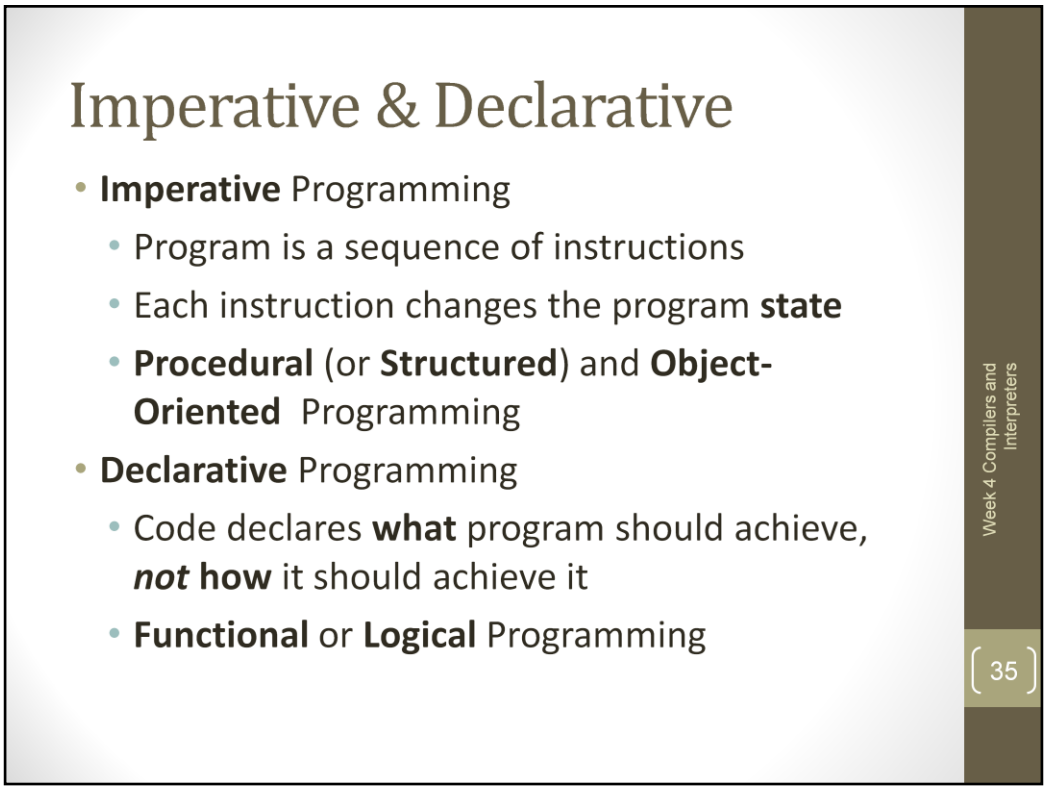
- Capabilities and styles of various programming languages are defined by their supported programming paradigms; some programming languages are designed to follow only *one* paradigm, while others support mul*ti*ple *paradigms*.

Major programming paradigms

- Major programming paradigms:
  - Imperative
  - Declarative
  - Functional
  - Object-oriented
  - Logical
  - Structured / Procedural

Main programming paradigms cont.





Imperative

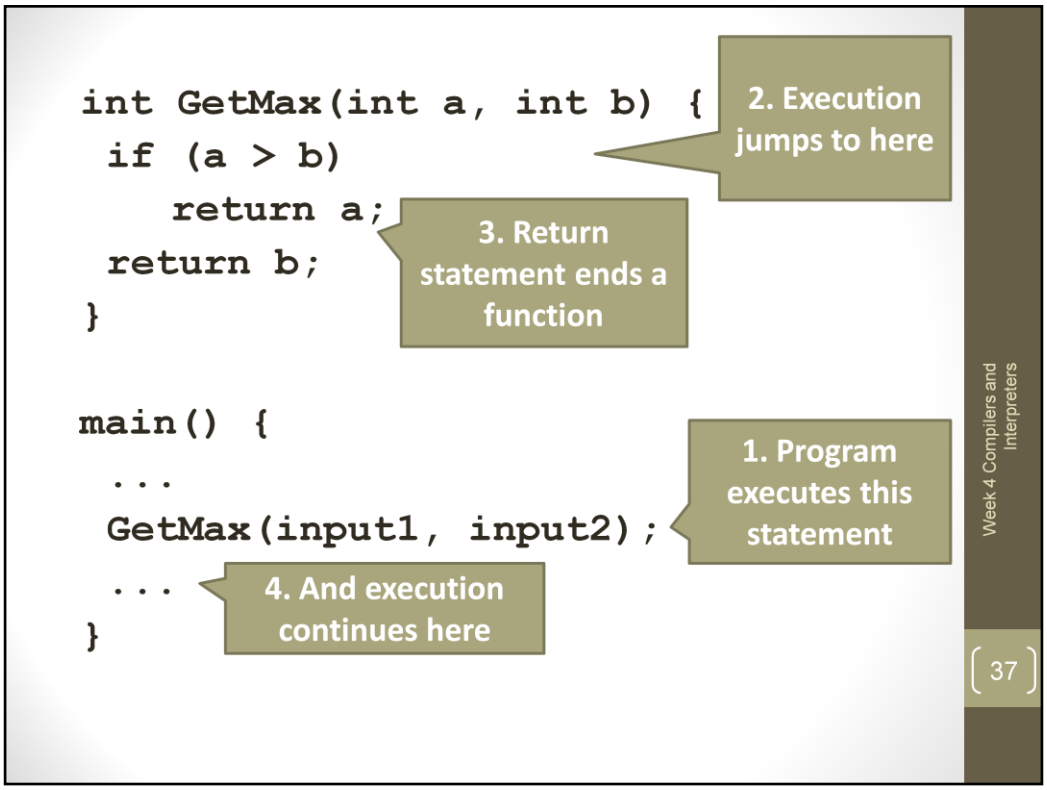
Procedural → C, Pascal

Object-Oriented →C++, C sharp, Java

Declarative →Functional, Logical

Structured Programming

- Examples: **C, Pascal**
- **Program** can contain *blocks* of code known as **functions** or **procedures**
- Statements may *call* other **functions** or **procedures**
  - Execution jumps to called procedure & returns to just after calling statement when procedure is completed



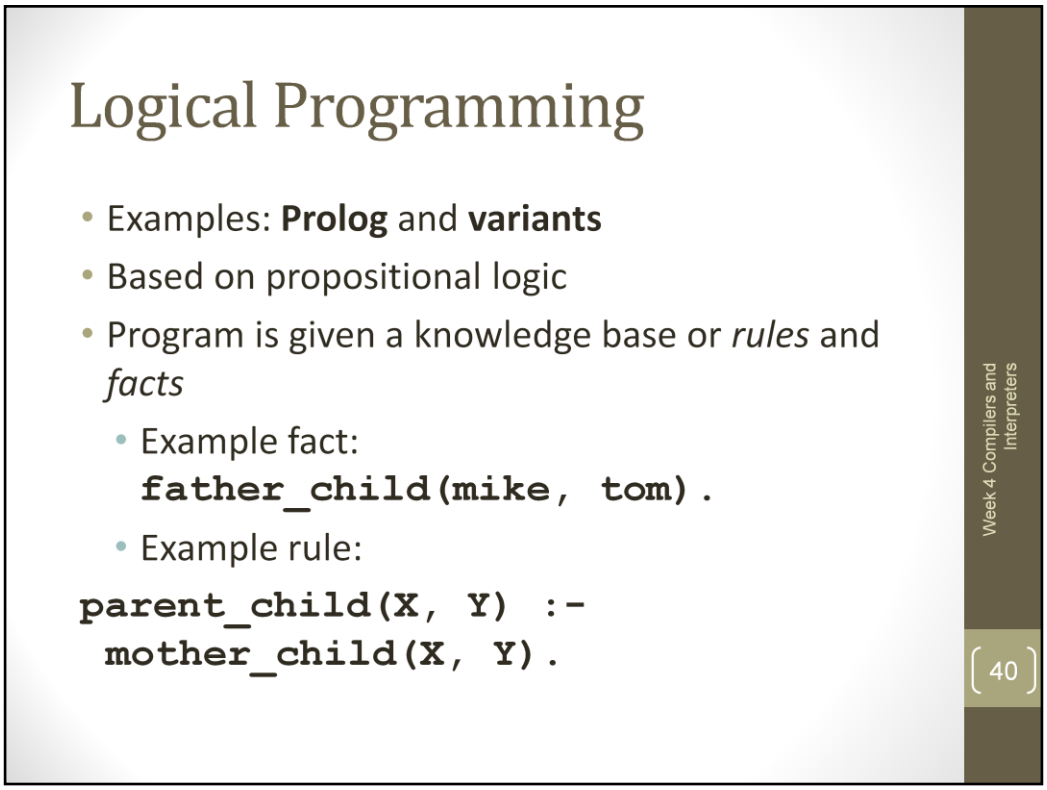
Start at Main. **GetMax** → is a calling statement

Object Oriented Programming

- Dominant programming paradigm?
- Examples: **C++, Java, C#, Delphi, Smalltalk**
- Groups data with related functions into *classes* and *objects*
  - Classes ***encapsulate*** design: to use a class, there is no need to understand how it is implemented
- **Objects** can communicate with other **objects** (*message passing*) & it is possible to derive new classes from existing ones (*inheritance*)

Functional Programming

- Examples: **Lisp, Scheme, Haskell**
- Emphasises functions where **imperative programs** emphasise program **state**
  - In a functional programming language, a function's result depends only on the **input parameters**, **not** on the **program state**
- May be **less efficient** in **memory** & **CPU** use
- Relatively **limited** use outside academia



Mike is the Father of Tom

If x is the Mother and y is child, then x is the parent and y is the child.

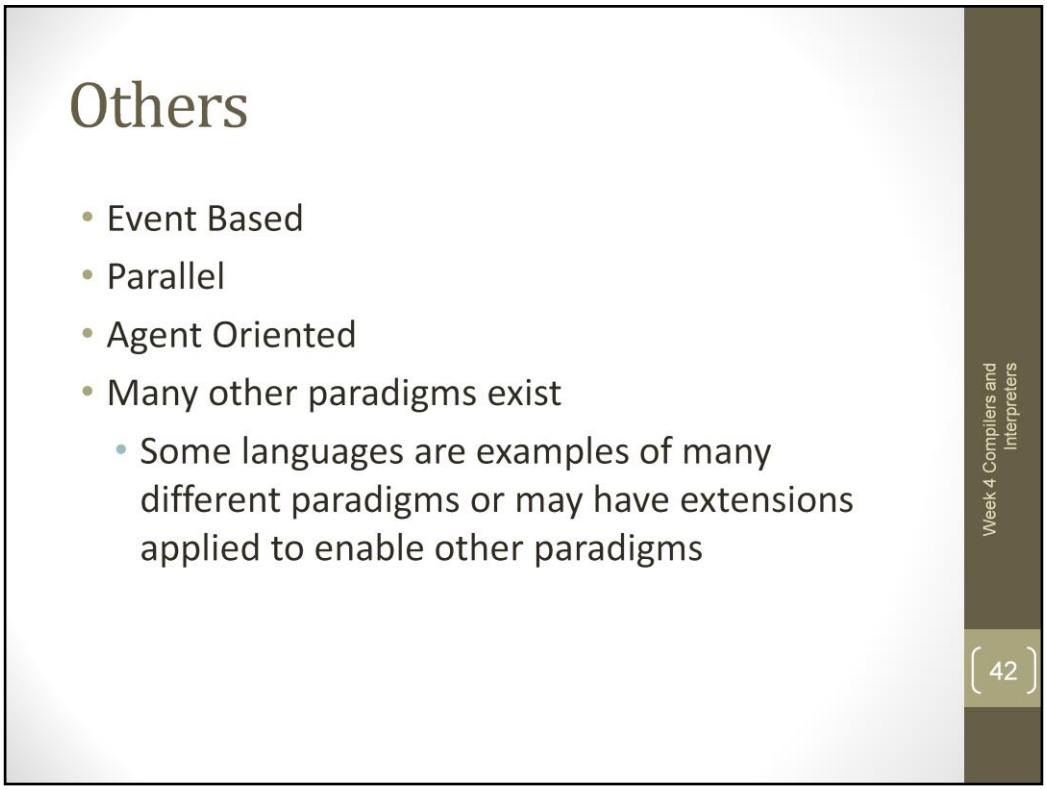
Prolog Example

```
mother_child(trude, sally).
father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X),
 parent_child(Z, Y).
parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

?- sibling(sally, erica).
Yes
```


Two siblings with same parents therefore sally and erica are siblings. Program is a collection of rules or facts.



Event Based → C, C++

Parallel → multiple things can happen at the same time.

Quick quiz

- Join the ‘Socrative’ app ‘Room 642124’ and try the quick quiz.

Further Reading

- Wikipedia
  - Compiler, High level language, List of programming languages, LOLCODE, ...
- Compilertools.net
- Programming Languages and Compilers
  - http://www-inst.eecs.berkeley.edu/~cs164/sp11/
  - A complete course on the construction of programming languages – quite advanced

Next Week

- ***Required*** *Reading* – HCW:
  - RAM p56-59
  - Data Storage, p148-155, p158-161, 167-171, 177-179, 185-193
- Additional Reading
  - HCW – Remaining sections on Data Storage
  - PCH – Chapter 12, Computer Memory

Credits:

Title Image, CC-BY-SA Daniel Livingstone

LOLCODE examples from: http://en.wikipedia.org/wiki/LOLCODE CC-BY-SA

Prolog example from http://en.wikipedia.org/wiki/Prolog CC-BY-SA