

technoteach

technocamps



Llywodraeth Cymru
Welsh Government

Cyngor Cyllido Addysg
Uwch Cymru
Higher Education Funding
Council for Wales

hefcw



Prifysgol
Abertawe
Swansea
University



Cardiff
Metropolitan
University

Prifysgol
Metropolitan
Caerdydd



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David



PRIFYSGOL
ABERYSTWYTH
UNIVERSITY

PRIFYSGOL
Glyndŵr
Wrexham

Wrexham
glyndŵr
UNIVERSITY

institute of
CODING
in wales
technocamps

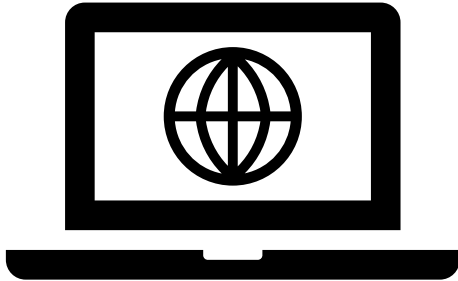
Program Construction





What is a
computer
program?

Programs

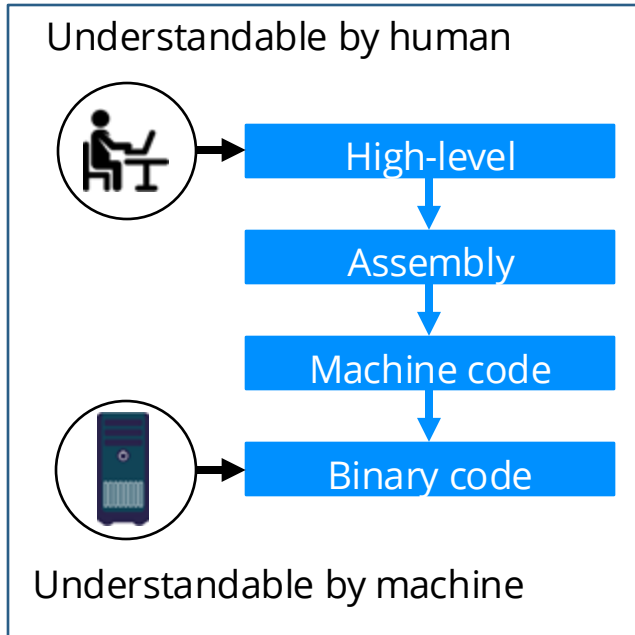


A computer program can be defined as a set of instructions that enables a computer to perform a specific task.



High & Low Level Languages

High & Low Level



Programs can be written in high-level or low-level languages, according to the requirements of the user.

High level languages

High-level languages enable a programmer to write programs for a computer without knowledge of the hardware and instruction sets of that computer

Portable programs can be used in different systems

For example: Java, C++ and Python

Same programming concept can be applied to different high-level languages

A typical statement in high-level language

```
Difference = number1 - number2;
```

ADVANTAGES

→ Easy to understand

→ Written in short time

→ Errors can be debugged at development stage

→ Programs can be maintained while in use

Low-level languages

Low-level languages are related to the hardware architecture and its instruction set

There are two types of low-level languages:

- Machine code, binary instructions that are understandable by the computer
- Assembly language that needs to be converted to machine code.

Assembly Language

Assembly language is used by programmers to make use of special hardware

Instructions used are dependent upon the type of machine

The code does not take up much space of primary memory and performs its task quickly

Assembly Language

Code to add two numbers:

Instruction	Operand	Function
MOV	AX, 22h	Move the hex number 22 to the register AX
ADD	AX, 15h	Add the number in register AX with hex number 15
HLT		Stop

In instruction, ADD AX, [15]. ADD is an example of instruction.
The numbers in registers AX and 15 are examples of operand.

What are Registers?

Registers are high-speed data storage areas in the computer

They are found in the CPU which means they can be accessed very quickly

But they are usually for storing very small amounts of data for performing simple arithmetic

Machine Code

Machine code is written in hexadecimal or binary form.

It is hard to understand and complicated to manage the storage and manipulation of data.

Machine Code

Code to add two numbers:

Opcode	Operand	Function
A1	22	Move the hex number 22 to the register AX
05	15	Add the number in register AX with hex number 15
F4		Stop

How easy is this to understand? Even assembly is more approachable!

Opcode tells the CPU which operation should be performed

Translators

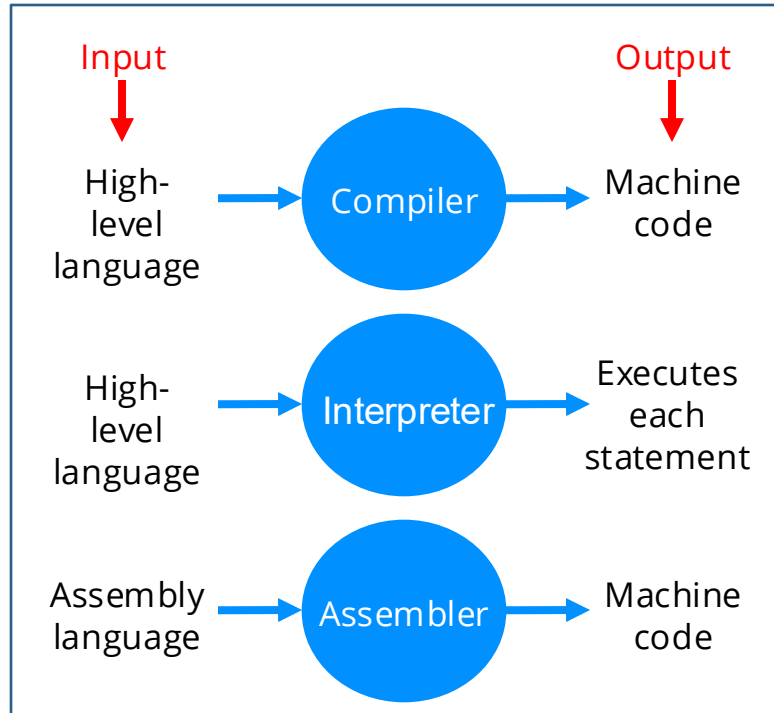


Translators

A utility program that translates programs written by programmers into binary form

The binary is understandable by the computer!

Literally **translates** human readable code into binary for computers

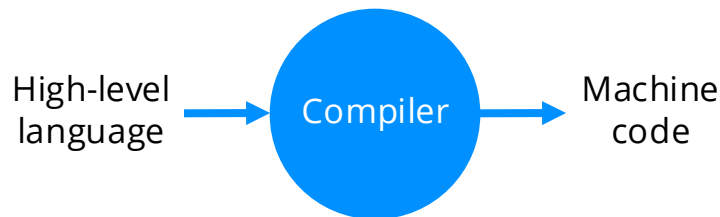


Compilers

A compiler translates a program written in high-level language into machine code that can be directly used by a computer to perform required tasks

Once compiled, the same code can be used again many times without recompilation.

A compiler optimises the code and errors are picked up only after the complete compilation of program

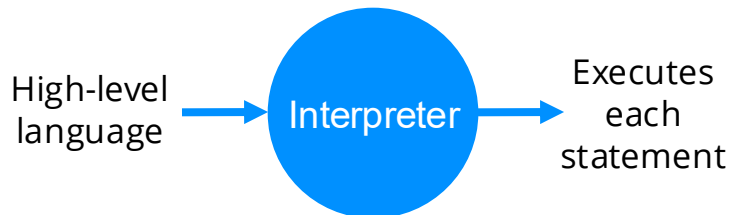


Interpreter

An interpreter is a computer program that reads a statement from a program written in high-level language, performs the action specified by the statement and then proceeds to the next statement and so on

In case an error is found, interpreter prompts the user to correct the error

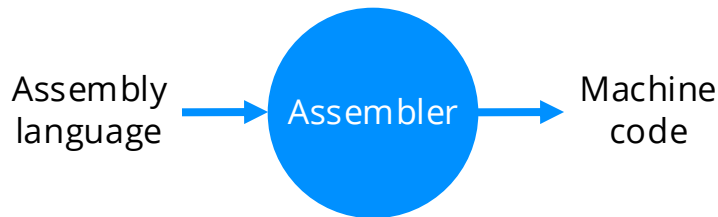
Code does not get optimised



Interpreter

An assembler is a computer program that translates a program written in an assembly language into machine code so that it can be directly used by the computer to perform the required task

Once assembled, the same code can be used again multiple times without re-assembly



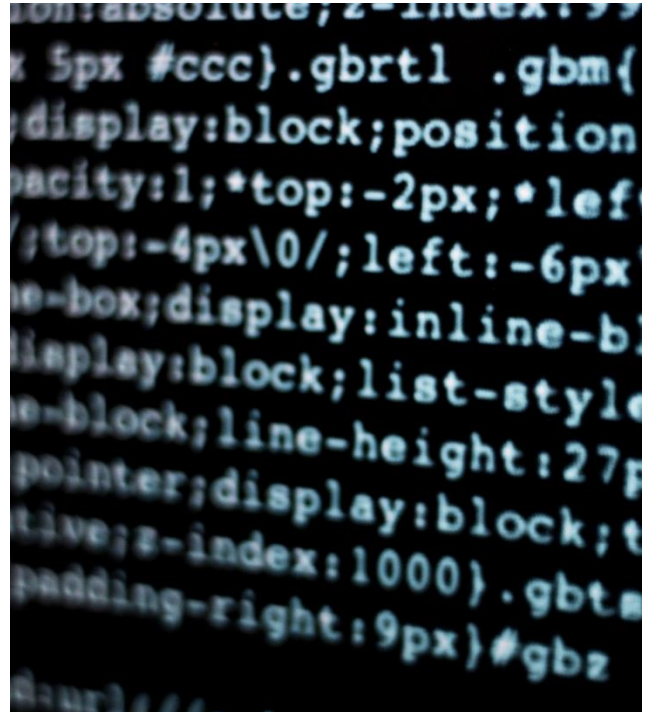
	Compiler	Interpreter	Assembler
What it does	Translates a high-level language into machine code.	Executes one statement of high-level language at a time and then proceeds to the next.	Translates a low-level language into machine code.
How it works	One statement in high-level language can be translated into several machine code instructions.	One statement in high-level language may require several machine code instructions to be executed.	One statement in low-level language is translated into one machine code instruction.

	Compiler	Interpreter	Assembler
Output	<p>An executable file of machine code is produced.</p> <p>Compiled programs work independently without re-compilation.</p>	<p>No executable file of machine code is produced.</p> <p>Interpreted programs can only be used with the interpreter.</p>	<p>The executable file of machine code is produced.</p> <p>Assembled programs work independently without re-assembly.</p>

Stages of Compilation

Compilers walk through 5 principle stages when compiling code:

1. Lexical analysis
2. Syntax analysis
3. Semantic analysis
4. Code generation
5. Code optimisation



Lexical Analysis

Lexical analysis is the process in which the source program's stream of characters is read from left to right and organised into tokens

Tokens are sequences of characters that represent a single unit of meaning

During lexical analysis, other tasks like removing comments, ignoring white space between tokens, and handling features like macros may also be performed for convenience

A symbol table is generated during this stage, which stores the addresses of variables, labels, and subroutines

Syntax Analysis



Syntax involves parsing which is analysing the sequence of tokens to identify the program's syntactic structure

The tokens are checked against the expected spelling and grammar based on the language's standard definitions

If any syntax errors are detected, error messages are generated

Semantic Analysis

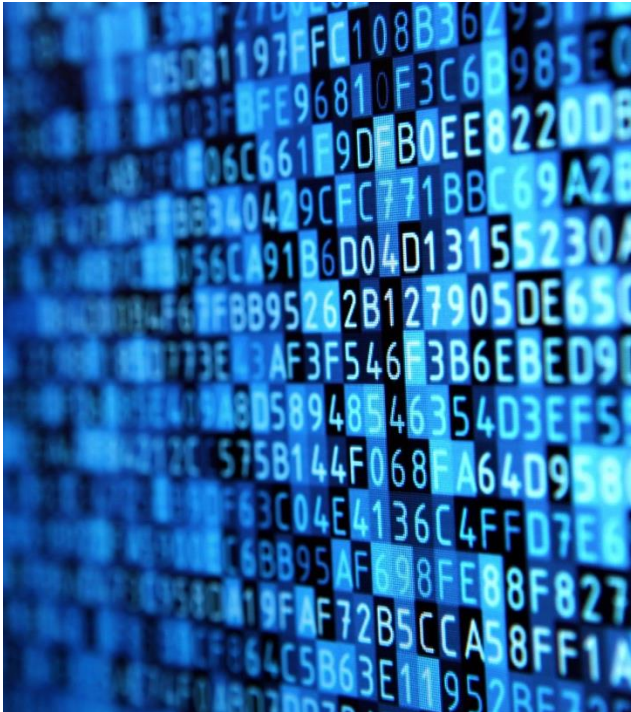
This phase conducts semantic checks, such as verifying data types and ensuring definite assignment confirming that all local variables are initialised before use

It ensures that variables are properly declared and used, and verifies that they are of the correct data type

Checks that operations are valid for the type of variable involved



Code Generation & Optimisation



Code Generation

- Machine code is generated

Code Optimisation

- Code optimisation may be employed to make it more efficient/faster/less resource intense



Programming Errors

Programming Errors

Syntax errors

- Programming language has a set of rules. If these rules are not followed, the program cannot be translated. This leads to syntax errors
- When a program is compiled, if syntax errors are found, the user is notified about the line numbers of the error
- When a program is interpreted, if a syntax error is found, the interpreter stops and user is notified about the line number of the error. After correction, the interpreter resumes


Programming Errors

< >
main.py
+ ↕

```

1 def myFunction():
2     list = [ ]
3     for x in range(1,6):
4         list.append(x)
5     return list
6 a = myFunction( )
7 print a
8
9
10
11
12

```

Powered by

trinket

File
"/tmp/sessions/c0e69338ddd59301/main.py",
line 7
print a
^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(a)?

Syntax Error

Programming Errors

Logic errors

- When a program runs, if it produces incorrect or unexpected results, it is said to contain logic errors
- To identify the correct position of the error, test data is used
- Tracing is done at each and every statement and compared with the expected results


Programming Errors

< >
main.py
+ ↕

```

1 def myFunction(x,y):
2     z=x/y
3     return z
4 a = myFunction(10,0)
5 print (a)
6
7
8
9
10
11
12

```

Powered by  trinket

Traceback (most recent call last):

File

"/tmp/sessions/5d28ab460881cb13/main.py",

line 4, in <module>

a = myFunction(10,0)

File

"/tmp/sessions/5d28ab460881cb13/main.py",

line 2, in myFunction

z=x/y

ZeroDivisionError: division by zero

Logic Error

Programming Errors

Runtime Error

- An error that only occurs when the program is running and is difficult to foresee before a program is compiled and run
- Example: Dividing a value by 0

Linking Error

- An error that occurs when a programmer calls a function within a program and the correct library has not been linked to that program
- Example: When the square root function is used and the library that calculates the square root has not been linked to the program

Programming Errors

Rounding Error

- Rounding is when a number is approximated to nearest whole number/tenth/hundredth
- Example: 34.5 rounded to nearest whole number is 35, an error of +0.5

Truncating Error

- Truncating is when a number is approximated to a whole number/tenth/hundredth
- Example: 34.9 truncated to whole number is 34, an error of -0.9

More on
Assembly
Language
Next Week!

