GCSE Computer Science Knowledge Organiser
SLR 2.5 Programming languages and IDEs:
Characteristics and purpose of different levels of programming language

Key Terminology	BCS Definition
High-level language	"Designed to allow the expression of a computer program in a way that reflects the problem being solved rather than the details of how the solution is produced. One-to-many."
Low-level language	"Close to machine code and closely related to the design of the machine. One-to-one."

High- and low-level languages

Machine code

- Binary representation of instructions in a format that the CPU can decode and execute.
- Includes an operation code (opcode) instruction and address or data to use (operand).

Low-level languages

- Written in assembly language.
- Converted into machine code using a translator called an assembler.
- Used for embedded systems and device drivers where it is necessary to instruct hardware directly.
- One instruction translates into one machine code instruction.
- Code only works with one specific type of processor.
- Programmers work with memory directly.
- Code is harder to write and understand.
- Memory-efficient.
- Code is quick to execute.

High-level languages

- Written in languages like Python, C++, Java and Visual Basic.
- Converted into machine code using a translator called a compiler or interpreter.
- Makes writing programs easier by utilising commands that are like English.
- One source code instruction translates to many machine code instructions.
- Code will run on different types of processors.
- Programmers can use many different data structures.
- Code is quicker and easier to understand and write.
- Less memory-efficient.
- Code can be slower to execute if it is not optimised.

```
100
101
102
103
104
105
106
107 00000030 B9FFFFFFFF
108
109
110 00000035 41
111 00000036 803C0800
112
113
114 0000003A 75F9
115
116
117
118
119
120
121
122 0000003C C3
```

```
;-------------------------------------------------------------
; zstr_count:
; Counts a zero-terminated ASCII string to determine its size
; in:   eax = start address of the zero terminated string
; out:  ecx = count = the length of the string

zstr_count:                 ; Entry point
    mov  ecx, -1            ; Init the loop counter, pre-decrement
                            ;  to compensate for the increment
.loop:
    inc  ecx                ; Add 1 to the loop counter
    cmp  byte [eax + ecx], 0  ; Compare the value at the string's
                            ;  [starting memory address Plus the
                            ;  loop offset], to zero
    jne  .loop              ; If the memory value is not zero,
                            ;  then jump to the label called '.loop',
                            ;  otherwise continue to the next line
.done:
                            ; We don't do a final increment,
                            ;  because even though the count is base 1,
                            ;  we do not include the zero terminator in the
                            ;  string's length
    ret                     ; Return to the calling program
```

Low-level: Assembly

Low-level languages like assembly allowed programmers to express programs using simple commands, which could then be translated into machine code.

These languages were written for a specific processor and closely mapped to machine architecture.

Programs written in assembly language are incredibly efficient.

However, assembly language requires a great deal of intellectual effort to use, as it is difficult to write and understand.

High-level: Source code

Two factors led to an explosion in the use of high-level languages:

- Increased processor speed
- Increased memory capacity

Assembly languages are now reserved for specialist situations like:

- Embedded systems
- Device drivers

GCSE Computer Science Knowledge Organiser
SLR 2.5 Programming languages and IDEs:
The purpose of translators

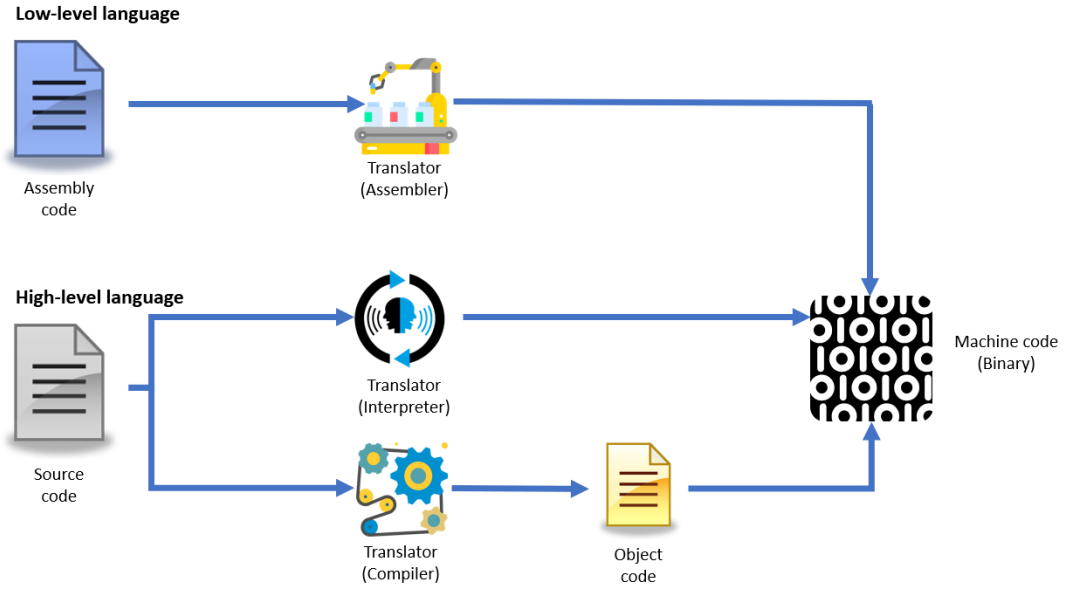
Key Terminology	BCS Definition
Translator	"Takes a program written in one programming language and converts it to another."
Compiler	"Translates high-level language source code into a computer's machine code."
Interpreter	"Translates and executes a program one statement at a time."

The purpose of translators

Code written in both low-level **assembly code** and high-level **source code** is converted into binary **machine code** ahead of execution – this is the purpose of a **translator**.

There are two types of **translators** used to convert **source code** into **machine code** – interpreter and compiler.

Assembly code is always translated into **machine code** using another type of **translator** called an **assembler**.



Compilers and interpreters

Compiler: Translates source code from high-level languages into object code and then into machine code ready to be processed by the CPU. The whole program is translated into machine code before it is run.

Advantages:

- No need for translation software at runtime.
- Faster to execute.
- Code is usually optimised.
- Original source code can be kept secret.

Disadvantages:

- Source code is easier to write in a high-level language, but the program will not run with syntax errors, which can make writing the code more difficult.
- Code needs to be recompiled following any changes.
- Designed for a specific type of processor.

Interpreter: Translates source code from high-level languages into machine code ready to be processed by the CPU. The program is translated line by line as the program is running.

Advantages:

- Easy to write source code – the program will always run and only stop when it finds a syntax error.
- Code does not need to be recompiled following changes, making it is easier to try out commands.
- A very easy way for beginner programmers to learn how to write code.

Disadvantages:

- Translation software is required at runtime.
- Slower to execute.
- Code is not optimised.
- Source code must be available.

GCSE Computer Science Knowledge Organiser
SLR 2.5 Programming languages and IDEs:
Integrated Development Environments (IDEs)

Key Terminology	BCS Definition
IDE	Integrated Develop Environment: "A software application that provides comprehensive facilities for software development. Normally consists of a source code editor, build automation tools and a debugger."
IDE: Error diagnostics	"IDE tools that provide detailed feedback on errors in code."
IDE: Run-time environment	"A configuration of hardware and software. Includes the CPU type, operating system and any runtime engines or system software required by a particular category of application."

IDE's provide the following functions:

- Debugging tools for finding logic errors:
  - Breakpoints to stop a program during execution when an error is found.
  - Stepping through lines of code one at a time to check which lines are executing.
  - Tracing through a program to output the values of variables.
- Help with preventing and identifying syntax errors:
  - Illustrating keyword syntax and auto-completing command entry.
  - Error highlighting.
  - The compiler produces an output of the error message to help identify it.
- Run-time environment:
  - Output window.
  - Simulating the different devices that the program can run on.
- Usability functions:
  - Navigation, showing/hiding sections of code.
  - Formatting source code.
  - Find and replace.
  - Commenting or indenting regions of code.

Common Features of an IDE:

- Easily comment out sections of code.
- Automatically indent regions of code.
- Provide a run-time environment and output window.
- Provide an editor
- Line numbers
- Keyword colour highlighting
- Auto-complete

Debugging features:

- Automatic highlighting of syntax errors.
- Breakpoints
- Variable tracing
- Step though line by line and…
- …watch variables change