

AQA Computer Science A-Level
4.6.3 Types of program translator
Intermediate Notes



Specification:

4.6.2.1 Classification of programming languages:

Understand the role of each of the following:

- Assembler
- Compiler
- Interpreter

Explain the differences between compilation and interpretation.

Describe situations in which each would be appropriate.

Explain why an intermediate language such as bytecode is produced as the final output by some compilers and how it is subsequently used.

Understand the difference between source code and object (executable) code.



Translators

In order for a program to be executed by a computer's processor, it must be in machine code. This means that programs written in **assembly language** or **high-level languages** need to be translated into machine code before they can be executed by a computer.

Synoptic Link

Programming languages can be **low-level** or **high-level**.

Different types of programming language are covered in **classification of programming languages** under **fundamentals of computer systems**.

Types of program translator

There are **three types** of program translator: assemblers, compilers and interpreters.

Assemblers

An assembler translates **assembly language** into **machine code**. Assemblers are **platform specific**, meaning that a **different assembler** must exist for each different type of processor.

Compilers

A compiler can be used to translate programs written in **high-level languages** like C# and Python into **machine code**. Compilers take a high-level program as their source code, **check it for any errors** and then translate **the entire program** at once. If the source code contains an error, it will not be translated. Because compilers produce machine code, they are said to be **platform specific**.

Interpreters

An interpreter translates **high-level languages** into **machine code line-by-line**. Rather than checking for errors **before** translation begins (like a compiler), interpreters check for errors **as they go**. This means that a program with errors in can be **partially translated** by an interpreter until the error is reached.

Comparison of compilers and interpreters

Compiler	Interpreter
Checks source code for errors line-by-line before translating	Translation begins immediately
Entire source code translated at once	Each line is checked for errors and then translated sequentially



Compilers with intermediate languages

Some compilers don't produce machine code straight away but instead translate source code into an **intermediate language** which allows for **platform independence**.

A compiler that uses an intermediate language will translate high-level code into an intermediate language such as bytecode and then use a **virtual machine** to execute the bytecode on different processors. Each different processor instruction set **will have its own** virtual machine.

Using an intermediate language allows the interpreter to translate the source code **just once**, while still being able to execute the translated code with a **variety of different processors**.

Source code and object code

Source code is the name given to the **input** to a translator. For an assembler, this is assembly language code and for compilers and interpreters, this will be code written in a high-level language. A translator's **output** is called object code.

