

OCR Computer Science A Level

1.2.2 Applications Generation Intermediate Notes



Specification:

Specification:

1.2.2 a)

- **Nature of applications**

1.2.2 b)

- **Utilities**

1.2.2 c)

- **Open source vs closed source**

1.2.2 d)

- **Translators**
 - Interpreters
 - Compilers
 - Assemblers

1.2.2 e)

- **Stages of compilation**
 - Lexical analysis
 - Syntax analysis
 - Code generation
 - Optimisation

1.2.2 f)

- **Linkers, loaders and use of libraries**



Nature of applications

Software can either be categorised as applications software or systems software.

Applications software

Designed to be **used by the end-user to perform one specific task**. Application software requires systems software in order to run.

Examples: *desktop publishing, word processing, spreadsheets, web browsers.*

Systems software

Low-level software responsible for managing the computer resources and maintaining consistently high performance.

Examples: *library programs, utility programs, operating system, device drivers.*

Utilities

Utilities ensure the **consistent, high performance** of the operating system. Each utility program has a **specific function** linked to the **maintenance of the operating system**.

Examples include:

- Compression
 - Used to decompress and compress files.
 - Examples: *decompressing files downloaded online, compressing large files for transmission across the internet, compressing scanned files*
- Disk defragmentation
 - As the hard disk becomes full, read/write times slow down. This is because files can no longer be stored contiguously. The disk defragmenter utility **rearranges the contents of the hard drive** so they can be accessed faster.
- Antivirus
 - Detects potential threats** to the computer, alerts the user and removes these threats.
- Automatic updating
 - Automatically installs any updates to the operating system. Ensures there are no security issues so the **system is less vulnerable** to malware and hacking threats.
- Backup
 - Routinely creates copies of files** specified by the user. In the event of a power failure, malicious attack or other accident, **files can be recovered**.

Synoptic Link

Compression techniques are explained in detail in 1.3.



Open source vs closed source

Source code is code written by a programmer and is **object code** before it has been **compiled**.

	Open source	Closed Source
Definition	Can be used by anyone without a license and is distributed with the source code .	Requires the user to hold an appropriate license to use it. Users cannot access the source code as the company owns the copyright license .
Advantages	Can be modified and improved by anyone	Thorough, regular and well-tested updates
	Technical support from online community	Company owning software provides expert support and user manuals.
	Can be modified and sold on	High levels of security as developed professionally.
Disadvantages	Support available online may be insufficient or incorrect. No user manuals.	License restricts how many people can use the software at once
	Lower security as may not be developed in a controlled environment	Users cannot modify and improve software themselves

Translators

A translator is a program that **converts high-level source code into low-level object code**, which is then ready to be executed by a computer. There are three types of translator:

Compiler

Translate high-level code into machine code **all at once**.

The **initial compilation process is longer** than using an interpreter or an assembler.

Compiled code is **specific to a particular processor type and operating system** but can be run **without a translator** present.

Synoptic Link

Compilers are also used to generate **intermediate code** which you will have come across in 1.2.1.



Interpreter

Translate and execute code line-by-line. They stop and produce an error if a line contains an error.

Initially appear faster than compilers, but are slower than running compiled code as code must be interpreted each time it is executed. Code also requires the correct interpreter in order to run on different devices.

Interpreters are useful for testing code, as time is not wasted compiling code with errors. Code is also platform-independent, making interpreted code more portable.

Assembler

Assembly Code

A low-level language and is the 'next level up' from machine code. Assembly code is platform specific.

Assemblers translate assembly code into machine code. Each line of assembly code is equivalent to almost one line of machine code.

Synoptic Link

LMC is an example of intermediate code, discussed further in 1.2.4.

Stages of compilation

High level code goes through four stages before it is fully compiled:

Lexical Analysis

Whitespace and comments are removed from the code. Keywords and names of variables and constants are replaced with tokens. Information about tokens is stored in a symbol table.

Syntax Analysis

Tokens are analysed against the rules of the programming language. Tokens that break the rules are flagged up as syntax errors.

Examples of syntax errors: *undeclared variable type, incomplete set of brackets.*

An abstract syntax tree is produced, which is a representation of the source code in the form of a tree.

Semantic analysis is carried out where logic mistakes within the program are detected.

Examples of semantic errors: *multiple declaration, undeclared identifiers*

Code Generation

The abstract syntax tree is used to produce machine code.



Optimisation

Searches through the code for areas it could be made more efficient to **reduce execution time** but is a **very time-consuming** of compilation.

Redundant parts of code are removed. Repeated sections of code are replaced with more efficient code. Excessive optimisation may alter the way in which the program behaves.

Linkers, Loaders and Use of Libraries

Linkers

A piece of software responsible for **linking external modules and libraries included within the code.**

Static Linker

Modules and libraries are **added directly** into the main file. This **increases the size of the file.**

Dynamic Linker

Addresses of modules and libraries are included in the file. **File size does not change** and **external module/library updates automatically feed through to the main file.**

Loaders

Programs provided by the operating system. When a file with dynamically linked addresses is executed, the **loader retrieves the library or module from the specified memory location.**

Use of Libraries

Pre-compiled programs which can be incorporated within other programs. They are **ready-to-use and error free**, so **save time** developing and testing modules. Libraries **can be reused** across multiple programs and **save programmers from having to 'reinvent the wheel'** and instead make use of others' expertise.

