OCR Computer Science AS Level

1.2.2 Applications Generation

Advanced Notes

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

Nature of applications

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

Applications software

Applications software is 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

Systems software is low-level software that is responsible for running the computer system smoothly, interacting with hardware and generally providing a platform for applications software to run. The user does not directly interact with systems software but it ensures high performance for the user.
Examples: *library programs, utility programs, operating system, device drivers.*

Utilities

Utilities are a key piece of system software integral to ensuring 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

  Operating systems provide utilities that enable files to compressed and decompressed. This is used when compressing large files to be transmitted across the Internet and is commonly used to compress scanned files.

Synoptic Link

Compression techniques are explained in detail in 1.3.

- Disk defragmentation

  As the hard disk becomes full, read/write times slow down. This is because files become fragmented as they are stored in different parts of memory. The disk defragmenter utility rearranges the contents of the hard drive so they can be accessed faster, thus improving performance.

- Antivirus

  Antivirus is responsible for detecting potential threats to the computer, alerting the user and removing these threats.

- Automatic updating

  This utility ensures the operating system is kept up to date, with any updates being automatically installed when the computer is restarted. Updates tackle bugs or security flaws so this ensures the system is less vulnerable to malware and hacking threats.

- Backup

  The backup utility automatically creates routine copies of specific files selected by the user. How often files are backed up is also specified by the user. This means that in the event of a power failure, malicious attack or other accident, files can be recovered.

Open source vs closed source

Source code is written by a programmer and refers to object code before it has been compiled. When software is described to be 'open source' or 'closed source', this refers to whether or not the source code is accessible to the public.

	Open source	**Closed Source**
Definition	Open source code can be used by anyone without a license and is distributed with the source code.	Closed 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

Whether a user chooses to use open source or closed source software ultimately depends on the suitability of the software to the task they will be using it for. The user must also consider:

- Costs - *implementation, maintenance, training of staff, license*
- Functionality - *features available, ease of use*

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 that convert different types of code and work in different ways.

High-level code

Code that is written and understood by the programmer but not the computer. Low-level code can be executed by a computer but cannot be directly understood.

Note

Machine code is the same as object code.

Synoptic Link

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

Compiler

Compilers translate high-level code into machine code all at once, after carrying out a number of checks and reporting back any errors. This initial compilation process is longer than using an interpreter or an assembler. If changes need to be made, the whole program must be recompiled.
Once code has been compiled to produce machine code, it can only be executed on certain devices - compiled code is specific to a particular processor type and operating system. Code can be run without a translator being present.

Interpreter

Interpreters translate and execute code line-by-line. They stop and produce an error if a line contains an error. They may initially appear faster than compilers as code is instantly executed, but are slower than running compiled code as code must be translated each time it is executed with an interpreter.
This feature makes interpreters useful for testing sections of code and pinpointing errors, as time is not wasted compiling the entire program before it has been fully debugged.
Interpreted code requires an interpreter in order to run on different devices. However, code can be executed on a range of platforms as long as the right interpreter is available, thus making interpreted code more portable.

Assembler

Assembly Code

Assembly code is considered to be a low-level language as it is the 'next level up' from machine code. Assembly code is platform specific, as the instructions used are dependent on the instruction set of the processor.

Synoptic Link

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

Assemblers translate assembly code into machine code. Each line of assembly code is equivalent to almost one line of machine code so code is translated on almost a one-to-one basis.