OCR Computer Science A Level

1.2.2 Applications Generation

Concise 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

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

Applications software

- Used by the end-user
- Perform one specific task

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

Systems software

- Manages computer resources
- Ensures consistently high performance.

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

Utilities

- Maintain a high-performing operating system.
- Each utility has a specific function

Examples: *compression, disk defragmentation, automatic updating, automatic backup*

Open source vs closed source

- Source code: code written by a programmer

	Open source	Closed Source
Definition	• Can be used without a license • Distributed with the source code.	• User must hold correct license • Users cannot access source code • Company owns the copyright license.
Advantages	Improved by community effort.	Regular, well-tested updates.
	Technical support from online community.	Company provides expert support and user manuals.
	Can be modified and sold on for profit.	High levels of security as developed professionally.
Disadvantages	Inadequate support available. No user manuals.	License has restrictions about use.
	Lower security.	Users cannot modify and improve code.

Translators

- A program that converts source code into object code. There are three types:

Compiler

- Translate high-level code into machine code all at once.
- Initial compilation process is longer than using other translators.
- Compiled code is platform-specific
- Compiled code can be run without a translator present.

Interpreter

- Translate and execute code line-by-line.
- Produce an error if a line contains an error.
- Slower than running compiled code.
- Correct interpreter required to run on different platforms.
- Code is platform-independent.
- Useful for testing code.

Assembler

- Assembly code is a low-level language that is platform specific.
- Assemblers translate assembly code into machine code.
- Each line of assembly code is equivalent to almost one line of machine code.

Stages of compilation

Lexical Analysis

- Whitespace and comments are removed.
- Keywords and identifiers are replaced with tokens.
- Information about tokens is stored in a symbol table.

Syntax Analysis

- Tokens analysed against rules of the programming language.
- Syntax errors are flagged up.
- Abstract syntax tree is produced.

Code Generation

- Abstract syntax tree used to produce machine code.

Optimisation

- Aims reduce execution time
- Is a very time-consuming stage.
- Redundant parts of code are removed.

Linkers, Loaders and Use of Libraries

Linkers

- Software that links external modules and libraries included within the code.

Static Linker

- Module/library code is copied directly into the file.
- Increases the size of the file.

Dynamic Linker

- Addresses of modules/ libraries are added to the file.
- External module/library updates automatically feed through to the main file.

Loaders

- Programs provided by the operating system.
- Fetches the library/ module from the given memory location.

Use of Libraries

- Pre-compiled programs which can be incorporated within other programs.
- Error-free
- Save time and effort of developing and testing modules.
- Can be reused across multiple programs
- Save programmers from 'reinventing the wheel'