Unit 3: Program construction

Terminology	Definition
Compilers	A program that converts an entire high-level language code program into machine code that can be executed by a computer.
Interpreter	A program that converts a high-level code program line by line into machine code.
Assembler	An assembler is a program that converts assembly language into machine code.

The function of translation programs

A translator or programming language processor is a term for a program that changes (translates) a program written in one programming language into an equivalent program written in a different programming language.

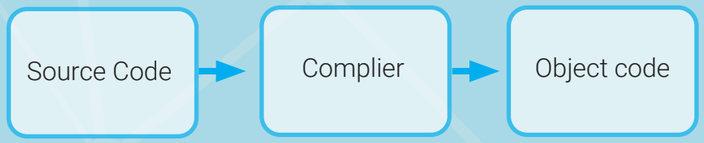
For example, a program written using the PASCAL programming language may be translated into a program written in one of the C programming languages, using a translator.

There are three specialised types of translator that convert high level languages into machine code that can be executed by a specific operating system.

Compilers

A compiler is a translator that converts high level language programs into machine code. It converts the whole program at one time, generating machine code from the source code. The translated code is stored. A compiler is a very complex piece of software.

Once a program has been compiled the machine code can be run time and time again without the need to translate the code each time.



There are four main stages of the compilation process that take place.

Lexical analysis

- Redundant parts of the source code, such as comments and unneeded spaces, are removed
- Keywords, constants and identifiers are replaced by 'tokens'. The code is said to have been tokenised. Keywords would include command words such as 'FOR', 'WHILE', 'SUBROUTINE'.
- A symbol table is created which holds the addresses of variables, labels and subroutines.
- Characters such as '+' and '/' are known as terminal characters.

Syntax analysis

- The structure of the source program is to see if it conforms with the 'grammar' rules of the source language.
- Tokens are checked to see if they match the spelling and grammar expected, using standard language definitions.
- Each token is parsed to determine if it uses the correct syntax for the programming language.
- If syntax errors are found, error messages are produced.

Semantic analysis

- Variables are checked to ensure that they have been properly declared and used.
- Variables are checked to ensure they are of the correct data type, e.g. real values are not being assigned to integers.
- Operations are checked to ensure that they are legal for the type of variable being used, e.g. you would not try to store the result of a division operation as an integer.

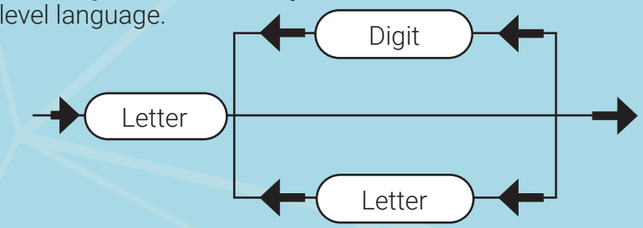
Code generation

- Machine code is generated.
- Code optimisation may be employed to make it more efficient/ faster/less resource intense.

Syntax Analysis

The syntax of any high-level language can be expressed as a set of rules. A syntax diagram can be used to express the syntax of a language.

This diagram shows the syntax for an identifier in a certain high-level language.



Using the above diagram, it can be seen that 'B', 'HAM' and 'CAT331' would all be valid identifiers. Whereas '1F', '34DD' would not be valid as they do not start with a letter.

It is during the syntax stage of the analysis that the dictionary is generated. The dictionary is a list kept by the compiler of the variables, their data types such as integer or real, and their location in memory.

Variable name	Variable type	Memory location
Radius	Interger	AB06
Π	Real	ABFF
Area	Real	DF01

Code Generation

Code generation is the phase of compilation where machine code is generated that is specific to the target machine (the computer on which the code is to be run).

Each line of the high-level language will produce multiple lines of machine code. For example:

The statement 'Let A = B + C' appears to be very simple as a high-level statement. 'Let' would have been removed during lexical analysis.

This simple statement, however, would involve the following processes, which, being executed as machine code:

- load the accumulator with B
- add C to the accumulator
- store the contents of the accumulator in the memory address of A.

Obviously, this is a very simple instruction and the more complex the high-level instruction, the more machine code actions would be required to carry out the instruction.

Code Optimisation

Just as when a person uses an online translation facility to translate from one spoken language to another, the translation is often not of the highest standard. This can be true of the code generated in the code generation stage of the compilation process.

It is often possible to improve the efficiency of the generated code through a process known as code optimisation.

Interpreters

An interpreter translates one line of a high-level language program and executes it without creating stored machine code instructions.

If, for example, a set of lines are contained in a FOR. .NEXT loop, then each iteration will require another translation of the same lines.

All stages of translation will have to be carried many times, meaning that interpreters are very slow compared to compilers.

Assembler

An assembly language is a type of low-level language which uses mnemonics to represent the machine code instruction set of a particular computer.

Mnemonics (symbols) are used to enable programmers to program at low level without having to use binary or hex codes. To code in binary or even hex is a very tedious process. Mnemonics make the coding easier to read and write. For example, LDA 33 would indicate the binary value of 33_{10} to the accumulator in the CPU.

Each instruction in an assembly language program translates to one machine code instruction.

Assemblers are far easier to write than compilers as the tasks carried out are far simpler. Less syntax checking takes place and usually instruction sets are smaller.