Introduction to Programming

Computer System Components

(a) Hardware: Physical equipment used to process, store or transmit computer programs or data.

(b) Software: Computer programs, procedures, associated documentation and data pertaining to the operation of a computer system.

The function of a computer system is to take input, process it and produce output.

INPUT → PROCESS → OUTPUT→ Storage

A Computer Program

A computer program is a set of instructions that will enable a computer to carry the above function.

How Do Computers Follow Instructions?

All computers are designed to follow instructions. To understand how they do this, we need to know some background.

All computer operations must be given to the Central Processing Unit (**CPU**) before anything happens. CPUs can execute hundreds of thousands of these individual **instructions** every second. We describe CPUs according to how quickly they can execute instructions. Simply moving your mouse across a screen requires the CPU to handle around 10 000 separate instructions.

A CPU is a tiny collection of thousands of **transistor** switches and controls all the other parts of the computer. A transistor is a very special high-speed switch that has no moving parts. These switches can be used to store information and change it simply by turning combinations of switches on or off. For example, a CPU would recognise the word ANT as 01000001 01001110 01010100 where a “1” is a switch turned **on** and a “0” is a switch turned **off**.

A number system that only uses 1s and 0s is ideal for computers since any combination of switch settings can be written. This counting system is called **binary**. Every 1 or 0 is called a binary digit or **bit** for short. A **byte**, made of 8 bits, is a common amount needed to store a single character. A single instruction for a CPU is about 32 bits (4 bytes) long. CPUs must receive their instructions in binary code. They can only understand ons and offs. Each single step of a task, written in binary, is called a **machine code instruction**. Millions of instructions may be required to do one job. All these instructions are grouped together and called a **program**. Before a computer can execute or run a program, all its binary code must be loaded from a disk into the computer’s Random Access Memory (**RAM**). The CPU can then fetch each instruction from RAM one after the other.

Computer languages

Computers can only use binary numbers, but humans find binary difficult to write and understand. Can you read the word ANT in binary? However, all programs must be presented to the computer in binary form.

Fortunately humans have already written programs called **translators**. These programs enable humans to write programs in human type instructions and can convert them into machine code instructions.

Instructions written in human understandable form are called **high-level instructions** and the translator program will convert all these instructions into binary or machine code which are called **low level instructions**.

A translator programs are generally specific for a certain CPU.

The Early Development of Computer Languages

A **computer language**, also called **computer programming language**, is a strict set of instructions composed of a strict set of words, numbers and symbols that you must use in the computer that enables the user to operate it. This is called the SYNTAX of the programming language.

Many of the early computer programmers were extraordinary characters. Ada Augusta Byron, Countess of Lovelace, is thought to have been the world's first programmer. She was a brilliant mathematician who became fascinated with the analytic engine, which was invented by Charles Babbage in 1834.

Grace Hopper was one of the first programmers with a real machine on which to run programs. In 1951 she invented the compiler and was a driving force in the development of the language COBOL. She is probably most famous for discovering the first "bug"; a moth had been beaten to death by a mechanical component inside a computer she was using. Since then, the term "debugging" has been used to describe the process of locating and correcting program errors or bugs during testing.

Development Since the Early 1950s

There are lots of different computer languages, many specially designed for carrying out specific tasks. Languages that use words similar to our own language are called **high-level languages**. **Low-level languages** are closer to the computer's own **binary code**.

Low-level Language First and Second Generation Languages

There are two main types of low-level language;

1. **machine code** and
2. **assembly language**

Programming in either is usually known as **machine code programming**.

Each computer instruction made up of lots of one or more groups of eight signals called bytes.

1. First Generation: Machine code

First Generation computer languages were developed for the earliest computers. These used "Machine Language", and were written in binary notation that is, 0 and 1. The term machine language, or machine code, is sometimes used to mean the computer's own binary code, into which all programs are finally converted.

The first generation of programmers (1951-58) had to write programs, which looked like this:

```
00001000 00000000 11010110 10000000
00110000 00000000 00011011 01000000
01110000 00000000 00011010 10101100
```

Machine code takes a great deal of patience and experience to write. Imagine trying to find the error in the above code if you should have entered:

```
00001000 00000000 11010110 10000000
00110000 00000000 00111011 01000000
01110000 00000000 00011010 10101100
```

Advantages of Machine Code:	**Disadvantages:**
• It runs very Fast! • Produces very efficient code • It takes up little disk or memory space.	• Tedious and slow to write • Requires an high level of expertise • Requires a high degree of knowledge about the computer's architecture ie had to understand precisely how the computer worked. • Highly machine dependent

2. Second Generation: Assembly language

Programming became a little simpler with the second generation of computers (1959-64).

The Second Generation of languages was Assembly Language. Assemblers, interpreters and compilers were available to represent machine code in a form, which was easier for programmers to recognise and understand.

Assembly Language instructions are written as **mnemonics**, which are English like abbreviations, such as **LD** for load or **JMP** for jump.

Assembly language programs need a special program called an **assembler** to convert the language into the code used by the computer.

This may be built into the computer, or you may have to load it separately.

Example of Assembly code;

```
; Assembler program to cause speaker to beep (8088 code)
;
MOV   DL,    7      ; Load value of 7 into register DL
MOV   AH,    2      ; Load value of 2 into register AH
INT   21            ; Call DOS interrupt 21 - speaker beeps!
INT   20            ; return control to DOS command level.
```

Advantages of Assembler:	Disadvantages:
• Fast! • Compact and efficient code • Easier than Machine Code – "Machine Language without tears'"	• Difficult because need to know the architecture of the computer's CPU • Many instructions, even for simple jobs • Machine orientated, rather than task orientated

High Level Languages Third to Fifth Generation Languages

3. Third generation Languages – Procedural Languages

The third generation of computers (1965-70) was associated with the introduction of many "high-level" languages that used codes containing English words.

These Third Generation Languages include languages such as BASIC, FORTRAN, COBOL, C, Java etc. Programmers of these languages do not need to do as much coding as they did with the lower-level languages because third generation languages are associated with "translators". A translator automatically does many of the tedious tasks such as allocating storage locations for different pieces of data, or shifting a piece of data from one location inside the computer to another so that it can be added or multiplied.

Third-generation programming languages still require the programmer to supply very detailed instructions. Because every step in the program needs to be spelled out, they are also known as **procedural languages**. Procedural languages use logical sequences of statements, known as procedures, to solve problems.

A brief description of some of the more common programming languages follows;

- **BASIC** (Beginners' All-purpose Symbolic Instruction Code) is a simple general-purpose language that is easily learnt, hence the name.

  Example code;

```
10    REM      * A BASIC PROGRAM in Microsoft Basic which accepts Name and age *
20    REM      * and calculates age next birthday. *
30    REM
40    PRINT    "What is your name ";
50    INPUT    $NAME
60    PRINT
70    PRINT    "And what is your age ";
80    INPUT    AGE
90    LET      AGE = AGE + 1
100   PRINT
110   PRINT    "You will be ";AGE; “next birthday, ";$NAME
120   END
```

- **Pascal** was designed in the late 1960s - early 1970s, primarily as an educational language that would allow students to develop good programming techniques. It is generally taught in introductory programming courses (such as this one) and is used as a foundation for learning other, more practically oriented, languages. Pascal was named after the 17th century mathematician and scientist Blaise Pascal.

  Example code:

  The program above written in BASIC, would be written as follows in Pascal

```
Program Birthday;

      {Program in Pascal, which accepts a name and age, and displays age next birthday}

Uses WinCrt;   {Enables I/O in a windows environment}

Var
      Name : String[40];
      Age    : Integer;

Begin
      Write(‘Please enter your name : ’);
      Readln(Name);
      Writeln;
      Write('Now enter your age : ’);
      Readln(Age);
      Writeln;
      Age := Age + 1;
      Write('You will be ',Age,' next birthday, ',Name);
End.
```

Features of High Level languages:

- “English-like” code
- Procedure-orientated, rather than machine orientated
- Can be transferred to other machines (provided a compiler or interpreter is available)
- Code is bulky
- Code runs relatively slow compared to first and second generation languages.

4. Fourth Generation Languages : Non-Procedural Languages

Since 1971 the introduction of the fourth generation of computers has allowed programmers to use very high-level languages known as 4GLs (4th generation languages). Programmers using a 4GL do not need to indicate every step the computer should make when making a calculation. Instead, they need only to specify the desired end result.

Third-generation languages require every step in the program to be spelled out. **Fourth generation languages**, which need much fewer instructions, are known as **“non-procedural languages”**. SQL is an example of a query language that was developed in conjunction with a particular kind of database known as a relational database.

5. Fifth Generation Languages

Since World War II, computer scientists have tried to develop “intelligent” computers.

The capability of a computer system to learn from its experiences and mimic Human intelligence in areas of judgment and reason is known as artificial intelligence (AI).

Programmers needed to understand the different ways that human experts actually solve problems. They discovered that experts in the fields of medicine, science and mineral exploration often make judgments on rules-of thumb called **heuristics**. **Expert systems** were then developed using collections of rules-of thumb judgments. These required software, which was built up from a database of facts and an **inference engine**. An expert system asks the user a series of questions. The next question depends on the answers to the previous ones, and the answers are continually compared with the knowledge base. In the end the computer either draws a firm conclusion, or guesses at a likely solution. The program uses thousands of **IF ... THEN** rules and processes known as backward chaining and forward chaining. Expert systems are particularly useful when an expert is not available to assess a situation.

Vast resources are being directed into **AI** research. In addition to expert systems, such areas as speech recognition, vision systems, robotics, natural language systems and computer imaging are developing rapidly.

Interpreters and Compilers

High-level languages have to be **translated** into the computer’s own code by a special program which may be either an **interpreter** or a **compiler**. The difference between the two lies in the way they translate a program.

1. Interpreters

An interpreted language such as most BASIC is “interpreted”. When you run a program, an **interpreter** takes one instruction at a time from the **source code**, translates it into **object code** (machine code) and then carries it out (executes the instruction).

Source code - This is the language a program is written in.

Object code - This is the code that a computer translates a program into.

Advantages:	**Disadvantages:**
• is that it is machine independent ie it can be run on any type of computer provided the computer has the correct Interpreter program	• Firstly, that because the source code is interpreted a line at a time, the process is slow • Secondly, the code will only work running with an interpreter • Thirdly, the code must be interpreted EVERY time it is run. No final object code file is saved so it can be re run.

1. Compilers

A **compiler** translates the entire source code to object code first, and then executes the object code.

Advantages:	**Disadvantages:**
• Compiled programs run much faster than interpreted ones. • The object code is stored as an executable file, which can be re run without being re-compiled.	• The code is machine type specific and cannot be run on different machine types

Languages such as Pascal, C and FORTRAN are compiled.

Summary - Generations of Programming Languages

Generation	Development	Description
First	1951-58	Low-level machine languages, use 0s and 1s
Second	1959-64	Low-level assemblers, compilers, interpreters
Third	1965-70	High-level-procedural languages, English-like codes for example, BASIC, FORTRAN, COBOL, Pascal, PL/1
Fourth	1971	Very high level-4GLs, non-procedural languages, for example, SQL, PARADOX
Fifth	1981	Natural language, artificial intelligence, expert systems, for example, PROLOG, LISP

The ASCII code for each character

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
32	<space>	56	8	80	P	104	h
33	!	57	9	81	Q	105	I
34	“	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	‘	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	11 6	t
45	-	69	E	93	]	11 7	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	£	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	9	127	<delete>

Do Activity 1- Introduction to Programming Principles

<u>END</u>