CHAPTER 1

INTRODUCTION TO STRUCTURED PROGRAMMING

Programming means to convert problem solutions into instructions for the computer. It also refers to the process of developing and implementing various sets of instructions to enable a computer to do a certain task.

Structured programming (sometimes known as *modular programming*) is an approach to writing programs that are easier to test, debug, modify and maintain by enforcing a modular approach which breaks a large complex problem into sub-problems.

A programming language is a vocabulary and set of grammatical rules designed for instructing a computer to perform specific tasks.

HISTORY OF PROGRAMMING LANGUAGES

First-Generation Programming Languages – Machine Language

A first-generation of programming languages includes **machine-level** programming languages. These languages were introduced in the 1940s and had the following characteristics:

- Instructions were entered directly in binary format (1s and 0s) and therefore they were tedious and error prone. Programmers had to design their code by hand then transfer it to a computer using a punch card, punch tape or flicking switches.
- Instructions were executed directly by a computer's central processing unit (CPU) i.e. they were executed very fast.
- Memory management was done manually.
- Programs were very difficult to edit and debug.
- Used to code simple programs only.

Second-Generation Programming Languages (2GL) – Low Level Programming Languages/Assembly Languages

They were introduced to mitigate the error prone and excessively difficult nature of binary programming.

- Introduced in the 1950s
- Improved on first generation by providing human readable **source code** which must be compiled/assembled into machine code (binary instructions) before it can be executed by a CPU
- Specific to platform architecture i.e. 2GL source code is **not portable** across processors or processing environments.
- Designed to support logical structure and debugging.

By using codes resembling English, programming becomes much easier. The use of these **mnemonic codes** such as **LDA** for **load** and **STA** for **store** means the code is easier to read and write. To convert an assembly code program into object code to run on a computer requires an **Assembler** and each line of assembly can be replaced by the equivalent one line of object (machine) code:

Assembly Code		**Machine Code**
`LDA A`		000100110100
`ADD #5`	-> Assembler ->	001000000101
`STA A`		001100110100
`JMP #3`		010000000011

Such languages are sometimes still used for *kernels* and *device drivers*, i.e. the core of the operating system and for specific machine parts. More often, such languages are used in areas of intense processing, like *graphics programming*, when the code needs to be **optimized for performance**.

Almost every CPU architecture has a companion assembly language. Most commonly used are the assembly languages today like Autocoder for IBM mainframe systems, Linoreum, MACRO -11,etc.

Third-Generation Languages (3GL) – High-Level Languages

Third generation languages are the primary languages used in general purpose programming today. They each vary quite widely in terms of their particular abstractions and syntax. However, they all share great enhancements in logical structure over assembly languages.

- Introduced in the 1950s
- Designed around ease of use for the programmer (Programmer friendly)
- Driven by desire for reduction in **bugs,** increases in **code reuse**
- Based on natural language
- Often designed with structured programming in mind
- The languages are architecture independent e.g. C, Java etc.

Examples:

Most Modern General Purpose Languages such as C, C++, C#, Java, Basic, COBOL, Lisp and ML.

Fourth Generation Languages

Fourth-generation programming languages are high-level languages built around database systems. They are generally used in commercial environments.

- Improves on 3GL and their development methods with **higher abstraction** and **statement power**, to reduce errors and increase development speed by reducing programming effort. They result in a reduction in the cost of software development.
- A 4GL is designed with a **specific purpose** in mind. For example languages to query databases **(SQL),** languages to make reports (Oracle Reports) etc.
- 4GL are more oriented towards problem solving and systems engineering.

  Examples: Progress 4GL, PL/SQL, Oracle Reports, Revolution language, SAS, SPSS, SQ

Fifth Generation Languages

Improves on the previous generations by skipping algorithm writing and instead provide **constraints/conditions**.

While 4GL are designed to build specific programs, 5GL are designed to make the computer solve a given problem without the programmer. The programmer only needs to worry about **what problems needed to be solved and only inputs a set of logical constraints**, with no specified algorithm, and the **Artificial Intelligence (AI)-based compiler builds the program based on these constraints**

Examples: Prolog, OPS5, Mercury

Low-Level Languages Versus High-Level Languages

Low-level languages such as **machine language** and **assembly language** are closer to the hardware than are the high-level programming languages, which are closer to human languages. Low-level languages are converted to machine code without using a compiler or interpreter, and the resulting code runs directly on the processor. A program written in a low-level language runs **very quickly**, and with a **very small memory footprint**; an equivalent program in a high-level language will be more heavyweight. Low-level languages are **simple**, but are considered **difficult to use**, due to the numerous technical details which must be remembered.

High-level languages are closer to human languages and further from machine languages.

The main advantage of high-level languages over low-level languages is that they are **easier to read**, **write, and maintain**. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada, Algol, BASIC, COBOL, C, C++, FORTRAN, LISP, Pascal, and Prolog.

PROGRAMMING PARADIGMS

A programming paradigm is a fundamental style of computer programming, a way of building the structure and elements of computer programs. There are four main paradigms:

a) Unstructured Programming

In unstructured programs, the statements are executed in sequence (one after the other) as written. This type of programming uses the GoTo statement which allows control to be passed to any other section in the program. When a GoTo statement is executed, the sequence continues from the target of the GoTo. Thus, to understand how a program works, you have to execute it. This often makes it difficult to understand the logic of such a program.

b) Structured Programming

The approach was developed as a solution to the challenges posed by unstructured/procedural programming. Structured programming frequently employs a **top-down design model**, in which developers **break the overall program** structure into **separate subsections**. A defined function or set of similar functions is coded in a separate module or sub-module, which means that **code can be loaded into memory more efficiently** and that **modules can be reused in other programs**. After a module has been tested individually, it is then integrated with other modules into the overall program structure.

Program flow follows a simple hierarchical model that employs looping constructs such as "for," "repeat," and "while." Use of the "GoTo" statement is discouraged.

Most programs will require thousands or millions of lines of code. (Windows 2000 – over 35 millions lines of code). The importance of splitting a problem into a series of self-contained modules then becomes obvious. A module should not exceed 100 lines, and preferably short enough to fit on a single page or screen.
Examples of structured programming languages include:

- C
- Pascal
- Fortran
- Cobol
- ALGOL
- Ada
- dBASE etc.

c) Object-oriented programming (OOP)

This is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

d) Visual Programming

A visual programming language uses a visual representation (such as graphics, drawings, animation or icons, partially or completely). A visual language manipulates visual information or supports visual interaction, or allows programming with visual expressions

A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation. For example, many VPLs (known as dataflow or diagrammatic programming) are based on the idea of "boxes and arrows", where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations. An example of visual programming languages is Microsoft Visual Basic which was derived from BASIC and enables the rapid application development (RAD) of graphical user interface (GUI) applications.
Programming in VB is a combination of visually arranging components or controls on a form, specifying attributes and actions for those components, and writing additional lines of code for more functionality.

e) Internet Based Programming

This is programming oriented to the development of internet applications using languages and tools such as PHP, ASP, Perl, JavaScript, HTML, Java etc.

SOFTWARE CONSIDERATIONS

Before you can start programming in C, you will need text editor such as a plain text Notepad Editor though it does not offer code completion or debugging. Many programmers prefer and recommend using an Integrated Development Environment (IDE) instead of a text editor on which to code, compile and test their programs.
Memory requirements

Disk space required

ADVANTAGES C LANGUAGE

1. **Modularity:** modularity is one of the important characteristics of C. we can split the C program into no. of modules instead of repeating the same logic statements (sequentially). It allows reusability of modules.
2. **General purpose programming language:** C can be used to implement any kind of applications such as math's oriented, graphics, business oriented applications.
3. **Portability:** we can compile or execute C program in any operating system (UNIX, dos, windows).
4. **Powerful and efficient programming language:** C is very efficient and powerful programming language; it is best used for data structures and designing system software. Efficient in that it is a modular programming language and thus makes efficient use of memory and system resources.