SOFTWARE PRACTICES 1

Lecture Notes_1

INTRODUCTION TO COMPUTERS

Any programming language is implemented on a computer. Right from its inception, to the present day, all computer system (irrespective of their shape & size) performs the following 5 basic operations. It converts the raw input data into information, which is useful to the users.

- Inputting: It is the process of entering data¬ & instructions to the computer system.
- Storing: The data & instructions are stored for either initial or additional processing, as & when required.
- Processing: It requires performing arithmetic or logical operation on the saved data to convert it into useful information.
- Outputting: It is the process of producing the output data to the end user
- Controlling: The above operations have to be directed in a particular sequence to be completed. Based on these 5 operations, we can sketch the block diagram of a computer.

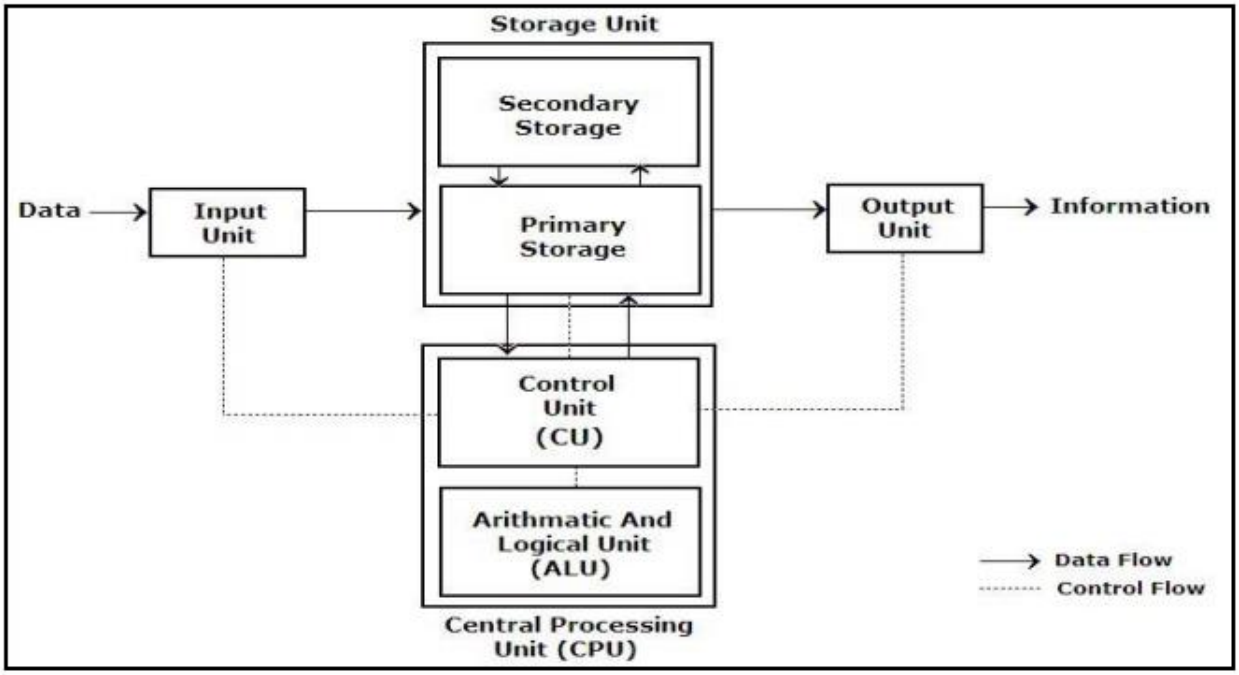

Fig 1: Block Diagram of a Computer

1. Input Unit

We need to first enter the data & instruction in the computer system, before any computation begins. This task is accomplished by the input devices. (E.g.: keyboard, mouse, scanner, digital camera etc). This device is responsible for linking the system with the external environment. The data accepted is in a human readable form. The input device converts it into a computer readable form.

2. Storage Unit
The data & instruction that are entered have to be stored in the computer. Similarly, the end results & the intermediate results also have to be stored somewhere before being passed to the output unit. The storage unit provides solution to all these issues. This storage unit is designed to save the initial data, the intermediate result & the final result.

This storage unit has 2 units:
Primary storage & Secondary storage.
i. Primary Storage: The primary storage, also called as the main memory, holds the data when the computer is currently on. As soon as the system is switched off or restarted, the information held in primary storage disappears (i.e. it is volatile in nature). Moreover, the primary storage normally has a limited storage capacity, because it is very expensive as it is made up of semiconductor devices.

ii. Secondary Storage: The secondary storage, also called as the auxiliary storage, handles the storage limitation & the volatile nature of the primary memory. It can retain information even when the system is off. It is basically used for holding the program instructions & data on which the computer is not working on currently, but needs to process them later.

3. Central Processing Unit
Together the Control Unit & the Arithmetic Logic Unit are called as the Central Processing Unit (CPU). The CPU is the brain of the computer. Like in humans, the major decisions are taken by the brain itself & other body parts function as directed by the brain. Similarly in a computer system, all the major calculations & comparisons are made inside the CPU. The CPU is responsible for activating & controlling the operation of other units of the computer system.

4. Arithmetic Logic Unit
The actual execution of the instructions (arithmetic or logical operations) takes place over here. The data & instructions stored in the primary storage are transferred as & when required. No processing is done in the primary storage. Intermediate results that are generated in ALU are temporarily transferred back to the primary storage, until needed later. Hence, data may move from the primary storage to ALU & back again to storage, many times, before the processing is done.

5. Control Unit
This unit controls the operations of all parts of the computer but does not carry out any actual data processing. It is responsible for the transfer of data and instructions among other units of the computer. It manages and coordinates all the units of the system. It also communicates with Input/Output devices for transfer of data or results from the storage units.

6. Output Unit
The job of an output unit is just the opposite of an input unit. It accepts¬ the results produced by the computer in coded form. It converts these coded results to human readable form. Finally, it displays the converted results to the outside world with the help of output devices (Eg :monitors, printers, projectors etc..).

So when we talk about a computer, we actually mean 2 things:
Hardware- This hardware is responsible for all the physical work of the computer.
Software- This software commands the hardware what to do¬ & how to do it.

Together, the hardware & software form the computer system. This software is further classified as system software & application software.
System Software System software are a set of programs, responsible for running the computer, controlling various operations of computer systems and management of computer resources. They act as an interface between the hardware of the computer & the application software.
E.g.: Operating System Application Software: Application software is a set of programs designed to solve a particular problem for users. It allows the end user to do something besides simply running the hardware. E.g.: Web Browser, Gaming Software, etc.

INTRODUCTION TO PROGRAMMING

A language that is acceptable to a computer system is called a computer language or programming language and the process of creating a sequence of instructions in such a language is called programming or coding. A program is a set of instructions, written to perform a specific task by the computer. A set of large program is called software. To develop software, one must have knowledge of a programming language. Before moving on to any programming language, it is important to know about the various types of languages used by the computer. Let us first know what the basic requirements of the programmers were & what difficulties they faced while programming in that language.

Computer Languages

Languages are a means of communication. Normally people interact with each other through a language. On the same pattern, communication with computers is carried out through a language. This language is understood both by the user and the machine. Just as every language like English, Hindi has its own grammatical rules; every computer language is also bounded by rules known as syntax of that language. The user is bound by that syntax while communicating with the computer system.

Computer languages are broadly classified as:
1. Low Level Language: The term low level highlights the fact that it is closer to a language which the machine understands.

The low level languages are classified as:
i. Machine Language: This is the language (in the form of 0's and 1's, called binary numbers) understood directly by the computer. It is machine dependent. It is difficult to learn and even more difficult to write programs.

ii. Assembly Language: This is the language where the machine codes comprising of 0'sand 1's are substituted by symbolic codes (called mnemonics) to improve their understanding. It is the first step to improve programming structure. Assembly language programming is simpler and less time consuming than machine level programming, it is easier to locate and correct errors in

assembly language than in machine language programs. It is also machine dependent. Programmers must have knowledge of the machine on which the program will run.

2. High Level Language: Low level language requires extensive knowledge of the hardware since it is machine dependent. To overcome this limitation, high level language has been evolved which uses normal English, which is easy to understand to solve any problem. High level languages are computer independent and programming becomes quite easy and simple.

Various high level languages are given below:
BASIC (Beginners All Purpose Symbolic Instruction Code): It is widely used, easy to learn general purpose language mainly used in microcomputers in earlier days.

COBOL (Common Business Oriented language): A standardized language used for commercial applications.
FORTRAN (Formula Translation): Developed for solving mathematical and scientific problems which is one of the most popular languages among scientific community.

C: Structured Programming Language used for all purpose such as scientific application, commercial application, developing games etc.

C++: Popular object oriented programming language, used for general purpose.

Programming Language Translators

As you know that high level language is machine independent and assembly language though it is machine dependent yet mnemonics that are being used to represent instructions are not directly understandable by the machine. Hence to make the machine understand the instructions provided by both the languages, programming language instructors are used. They transform the instruction prepared by programmers into a form which can be interpreted & executed by the computer.

Flowing are the various tools to achieve this purpose:

Compiler: The software that reads a program written in high level language and translates it into an equivalent program in machine language called compiler. The program written by the programmer in high level language is called source program and the program generated by the compiler after translation is called as object program.

Interpreter: it also executes instructions written in a high level language. Both complier & interpreter have the same goal i.e. to convert high level language into binary instructions, but their method of execution is different. The complier converts the entire source code into machine level program, while the interpreter takes 1 statement, translates it, executes it & then again takes the next statement.

Assembler: The software that reads a program written in assembly language and translates¬ it into an equivalent program in machine language is called as assembler.

Linker: A linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.

Characteristics of a Good Program

- Transferability: Must be able to work on any computer machine.
- Reliability: It can be relied upon to do what it is expected to do.
- Efficiency/cost saving: It must not cost more than its benefits and enables problem to be solved appropriately, quickly and efficiently.
- Simplicity: It should be as simple as possible to understand.
- Understandability/Readability: It must be readable and understandable by other programmers and end users.
- Flexibility/Adaptability / Maintainability: A good program must be flexible adaptable and maintainable in order to suit user's need. Modification must be possible and very easy.

Phases of Program Development (Programming)

The process of producing a computer program (software) may be divided into eight phases or stages:

1) Problem definition/Analysis
2) Selection or development of an algorithm
3) Designing the program
4) Coding the programming statements
5) Compiling/Compilation stage
6) Testing/Running and Debugging the program
7) Documentation.
8) Maintenance

1) Problem Definition/Analysis Stage: There is need to understand the problem that requires a solution. The need to determine the data to be processed, from of the data, volume of the data, what to be done to the data to produce the expected / required output.

2) Selection or development of an algorithm: An algorithm is the set of steps required to solve a problem written down in English language.

3) Designing the program: In order to minimize the amount of time to be spent in developing the software, the programmer makes use of flowchart. Flowchart is the pictorial representation of the algorithm developed in step 2 above. Pseudocode IPO chart (input processing output) and Hipo

chart (Hierarchical-input-processing and output) may be used in place of flowchart or to supplement flowchart.

4) Coding the statement: This involves writing the program statements. The programmer uses the program flow chart as a guide for coding the steps the computer will follow.

5) Compiling: There is need to translate the program from the source code to the machine or object code if it is not written in machine language. A computer program is fed into the computer first, then as the source program is entered, a translated equivalent (object program) is created and stored in the memory.

6) Running, Testing and Debugging: When the computer is activated to run a program, it may find it difficult to run it because many syntax errors might have been committed. Manuals are used to debug the errors. A program that is error free is tested using some test data. If the program works as intended, real required data are then loaded.

7) Documentation: This is the last stage in software development. This involves keeping written records that describe the program, explain its purposes, define the amount, types and sources of input data required to run it. List the Departments and people who use its output and trace the logic the program follows.

ELEMENTS OF STRUCTURED PROGRAMMING

Structured programming is one of the several different ways in which a programming language can be constructed. "It was originally introduced as a means of getting away from the 'spaghetti' code that was used in the early days and to provide some means by which programmers could more easily follow code written by other programmers.

In structured programming design, programs are broken into different functions these functions are also known as modules, subprogram, subroutines and procedures. Structured programming minimizes the chances of the function affecting another. It allows for clearer programs code. It made global variables to disappear and replaced by the local variables.

Each function is design to do a specific task with its own data and logic. Information can be passed from one function to another function through parameters. A function can have local data that cannot be accessed outside the function’s scope. The result of this process is that all the other different functions are synthesized in another function. This function is known as main function. Many of the high-level languages support structured programming.

Structured programming minimizes the chances of the function affecting another. It allows for clearer programs code. It made global variables to disappear and replaced by the local variables. Due to this change one can save the memory allocation space occupied by the global variable. Its organization helps in the easy understanding of programming logic, so that one can easily understand the logic behind the programs. It also helps the newcomers of any industrial technology company to understand the programs created by their senior workers of the industry.
One of the most important concepts of programming is the ability to control a program so that different lines of code are executed or that some lines of code are executed many times. The mechanisms that allow us to control the flow of execution are called **control structures**. Flowcharting is a method of documenting (charting) the flow (or paths) that a program would execute. The structured program mainly consists of three types of elements:

- Selection Statements
- Sequence Statements
- Iteration Statements

The structured program consists of well structured and separated modules. But the entry and exit in a structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

Selection: This is where you select or choose between two or more flows. The choice is decided by asking some sort of question. The answer determines the path (or which lines of code) will be executed.

Sequence: Very boring. Simply do one instruction then the next and the next. Just do them in a given sequence or in the order listed. Most lines of code are this.

Iteration: Also known as repetition, it allows some code (one to many lines) to be executed (or repeated) several times. The code might not be executed at all (repeat it zero times), executed a fixed number of times or executed indefinitely until some condition has been met. Also known as looping because the flowcharting shows the flow looping back to repeat the task.
A fourth category describes unstructured code.

Branching: An uncontrolled structure that allows the flow of execution to jump to a different part of the program. This category is rarely used in modular structured programming.

All high-level programming languages have control structures. All languages have the first three categories of control structures / elements (sequence, selection, and iteration). Most have the if…then …else structure (which belongs to the selection category) and the while structure

(which belongs to the iteration category). After these two basic structures, there are usually language variations.

The concept of **structured programming** started in the late 1960's with an article by Edsger Dijkstra. He proposed a "go to less" method of planning programming logic that eliminated the need for the branching category of control structures. The topic was debated for about 20 years. But ultimately – "By the end of the 20th century nearly all computer scientists were convinced that it is useful to learn and apply the concepts of structured programming.
The languages that support Structured programming approach are:

- C
- C++
- Java
- C#
- Pascal

Structured Programming in Everyday Life

Since the invention by Von Neumann of the stored program computer, computer scientists have known that a tremendous power of computing equipment was the ability to alter its behavior, depending on the input data. Calculating machines had, for some time, been able to perform fixed arithmetic operations on data, but the potential of machines capable of making decisions opened up many new possibilities. Machines that could make decisions were capable of sorting records, tabulating and summarizing data, searching for information, and many more advanced operations that could not even be imagined at the time.

In early programming languages, like Fortran (first invented in 1954) and various low level machine languages, the goto statement allowed the computer to deviate from the sequential execution of the program instructions. The goto statement was recognized to be a very powerful construction, and soon, programs of increasing complexity and power were developed.

However, the increasingly complex code that resulted from goto statements became harder and harder to maintain. Dijkstra, in 1966, was one of the first persons to recognize that this run away complexity of programs was due to the overuse of the goto statement (Dijkstra, E. W., "*Go To Considered Harmful*," Communications of the ACM, March 1966). In fact, it was determined shortly thereafter, that the goto statement is not needed at all. Dijkstra showed that any program construction that could be created with goto statements could be created more simply with the sequence, repetition and decision constructions that are discussed in the following sections.
This was the birth of the discipline of Structured Programming.

1. **Sequence** Execute a list of statements in order.

Example: Baking Bread

```
Add flour.
Add salt.
Add yeast.
Mix.
Add water.
Knead.
Let rise.
Bake.
```

2. Repetition Repeat a block of statements while a condition is true.

Example: Washing Dishes

```
Stack dishes by sink.
Fill sink with hot soapy water.
While moreDishes
   Get dish from counter,
   Wash dish,
   Put dish in drain rack.
End While
Wipe off counter.
Rinse out sink.
```

3. Selection Choose at most one action from several alternative conditions.

Example: Sorting Mail

```
Get mail from mailbox.
Put mail on table.
While moreMailToSort
   Get piece of mail from table.
   If pieceIsPersonal Then
      Read it.
   ElseIf pieceIsMagazine Then
      Put in magazine rack.
   ElseIf pieceIsBill Then
      Pay it,
   ElseIf pieceIsJunkMail Then
      Throw in wastebasket.
   End If
End While
```

Structured Programming in Visual Basic

Structured programming is a program written with only the structured programming constructions: (1) sequence, (2) repetition, and (3) selection.

1. **Sequence.** Lines or blocks of code are written and executed in sequential order.

   Example:

   ```
   x = 5
   y = 11
   z = x + y
   WriteLine(z)
   ```

2. **Repetition.** Repeat a block of code (Action) while a condition is true. There is no limit to the number of times that the block can be executed.

   ```
   While condition
      action
   End While
   ```

   Example:

   ```
   x = 2
   While x < 100
      WriteLine(x)
      x = x * x
   End
   ```

3. **Selection.** Execute a block of code (Action) if a condition is true. The block of code is executed at most once.

   ```
   If condition Then
      action
   End If
   ```

   Example:

   ```
   x = ReadLine()
   If x Mod 2 = 0
      WriteLine("The number is even.")
   End If
   ```

Extensions to Structured Programming

To make programs easier to read, some additional constructs were added to the basic three original structured programming constructs:

1. **Definite Repetition (For Loop)** Combine initialization, checking a condition, and incrementing a counter in a single statement called a for statement. Here is the general form:

   ```
   For i = 1 To n Step k    ' Step k is optional
       action
   Next
   ```

   Example:

   ```
   For i = 1 To 20
      WriteLine(i)
   Next i
   ```

   Example:

   ```
   For i = 20 To 1 Step -1
      WriteLine(i)
   Next
   ```

2. **If-Then-Else Statements** Execute the first action whose corresponding condition is true. Here is the general form:

   ```
   If condition1 Then
      action1
   ElseIf condition2 Then
      action2
   ElseIf condition3 Then
      action3
   Else
      defaultAction
   End If
   ```

   Example:

```
If n = 1 Then
   WriteLine("One")
ElseIf n = 2 Then
   WriteLine("Two")
ElseIf n = 3 Then
   WriteLine("Three")
Else
   WriteLine("Many")
End If
```

3. **Select Statement** Execute the action corresponding to the value of the expression.

```
Select Case value1
   Case value1
     action1
   Case value2
     action2
   Case value3
     action3
   Case Else
     defaultAction
End Select
```

Example:

```
Select Case n
   Case 1
     WriteLine("One")
   Case 2
     WriteLine("Two")
   Case 3
     WriteLine("Three")
   Case Else
     WriteLine("Many")
End Select
```

Advantages of Structured programming

- It is user friendly and easy to understand.
- Similar to English vocabulary of words and symbols.
- It is easier to learn.
- They require less time to write.
- They are easier to maintain.
- These are mainly problem oriented rather than machine based.
- Program written in a higher-level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.
- It is independent of machine on which it is used, i.e., programs developed in high level languages can be run on any computer.

Disadvantages of Structured Programming

- A high-level language has to be translated into the machine language by translator and thus a price in computer time is paid.
- The object code generated by a translator might be inefficient compared to an equivalent assembly language program.
- Data types are proceeds in many functions in a structured program. When changes occur in those data types, the corresponding change must be made to every location that acts on those data types within the program. This is really a very time-consuming task if the program is very large.
- Let us consider the case of software development in which several programmers work as a team on an application. In a structured program, each programmer is assigned to build a specific set of functions and data types. Since different programmers handle separate functions that have mutually shared data type, other programmers in the team must reflect the changes in data types done by the programmer in data type handled. Otherwise, it requires rewriting several functions.

PROGRAMMING PARADIGMS

A programming paradigm, or programming model, is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Examples of these paradigms are imperative, functional, logical, object-oriented paradigms and others.

Solving a programming problem requires choosing the right concepts. All but the smallest toy problems require different sets of concepts for different parts of the program. A programming paradigm, or programming model, is an approach to programming a computer based on a mathematical theory or a coherent set of principles. It is a way of conceptualizing what it means to perform computation and how tasks to be carried out on the computer should be structured and organized. Programming languages are used to realize programming paradigms. Examples of programming paradigms: imperative, functional, logical, object-oriented. Most popular languages are imperative and use structured programming techniques.

Structured programming techniques involve giving the code you write structures, these often involve writing code in blocks such as sequence (code executed line by line), selection (branching statements such as if..then..else, or case) and repetition (iterative statements such as for, while, repeat, loop).

Imperative paradigm This paradigm is based on the ideas of a Von Neummann architecture. A command has a measurable effect on the program and the order of commands is important. First do this and next do that. Its main characteristics are incremental change of the program state (variables) as a function of time; execution of commands in an order governed by control structures; and the use of procedures, abstractions of one or more actions, which can be called as a single command. Examples: Fortran, Algol, Basic, C, Pascal

Functional paradigm This paradigm is based on mathematics and theory of functions. The values produced are non-mutable and plays a minor role compared to imperative program. All computations are done by applying functions with no side effects. Functions are firsts class citizens. Evaluate an expression and use the resulting value for something. Example: Haskell, Clojure Check for Functional paradigm.

Logical paradigm The logic paradigm fits well when applied in problem domains that deals with the extraction of knowledge from basic facts and relations. Is based on axioms, inference rules, and queries. Program execution becomes a systematic search in a set of facts, making use of a set of inference rules. Answer a question via search for a solution. Examples: Prolog and List.

Object-oriented paradigm Data as well as operations are encapsulated in objects. Information hiding is used to protect internal properties of an object. Objects interact by means of message passing. In most object-oriented languages objects are grouped in classes and classes are organized in inheritance hierarchies. Send messages between objects to simulate the temporal evolution of a set of real-world phenomena. Examples: C++, Java.

Other Paradigms Other paradigms include Visual paradigm, Constraint based paradigm, Aspect oriented paradigm and Event-oriented paradigm.