UNIT-II

Programming languages

A programming language is a notation with which people can communicate algorithms to computers and to one another.

Some of the aspects/features/advantages of high level language which make them preferable to machine for assembly language are the following:

1) Base of understanding:

A high level language is generally easier to read, write and prove correct than is an assembly language.

Even among high level languages, some are easier to use than others part of this to do with the operators, data structures and flow of control features provided in a language.

Subroutines and powerful operators are used and orderly data structures and the ability to create such structures are important.

2) Naturalness :

Much of the understand ability of high-level programming language come from the ease with which one can express an algorithm in that language.

FORTRAN was designed for numerical and mathematical computations.

a. COBOL for business applications and SNOBOL for string manipulation.
b. One would not write a complex numerical calculation in COBOL or SNOBOC.

3) Portability:

a. A program is said to be portable if it can be transferred from one system to another with ease.
b. Programs written in high-level languages are more portable than programs written in low-level language.
c. Ex: java

4) Efficiency of use:

High level language has facilities for defining data structures, macros, subroutines etc.

The operating system and programming environment can also be as important as the language in reducing programming time.
High level languages feature facilities reliable programming some such features are:

a) Data structures
b) Scope rules: Scope rules allow modifications to be made to piece of a program without unwittingly affecting other portion of the same program.
c) Flow of control constructs that clearly identify the looping structures in a program.
d) Subroutines allow the program to be designed in small pieces.

#Definitions of programming languages:

1) Syntax:

A program in any language can be viewed as a string of character chosen from some set or alphabet of character.
The rules that tell us whether a string is a valid program or not are called the syntax of the language.

2) Semantics:

The rules that give meaning to program are called semantic rules.
The following approaches are used for specifying semantic rules.

a) Interpretive semantics:

The interpretive semantic is abstract machine which provide rules for programs. These rules then define many of programs an abstract machine consisting of objects, their values and current state of program.

b) Translation:

The translation of assembly language into machine language is direct but we can give rules which are suitable for sentence in language therefore we could use specific machine language that could be well understood by computer system.

3) **Axiomatic Definition:**
   We can define rules that are related with data before and after the execution of program. It defines input and output meaning of the program.
   The advantage of this approach is it defines semantics of part of program rather than all program.

#The lexical and syntactic structure of a language

-lexical structure of a language determines what group of symbols are treated as identifier.

a) **Alphabets:**
   The set of symbols used in programming language is called alphabets. The FORTRAN alphabet consists of 26 characters (UPPER CASE A to Z), 10 digits (0 to 9), and 11 special symbols (=, $, @,). COBOL having 26 characters, 10 digit and 11 other characters. The machine language has only two symbols. i.e. 0 and 1

b) **Tokens:**
   Program can be divided into sequence of sub stream called as Tokens. Each token is a sequence of characters which process collectively.
   E.g. Constants =70, 20, 10.2
   Operator =+,-,*, /, %
   Keyword=if, if-else, while, go to,
   Symbol= (,),

c) **High Level Constructs:**
   The language contains expressions such as sequence of operator and operand. The operators, keywords and symbols are the building block for the high level constructs.

4) **Lexical Conventions:**
   Some languages require certain statement should be return in restricted area. The lexical convention helps to avoid syntax errors due to improper position of tokens. The blank spaces are another issue in some programming language such as FORTRAN. The blank are not accepted so that it doesn't specify blanks in program but it creates problem for specifying tokens therefore other language used blank as a tokens.

#Data Elements:

One of the basic buildings of programming language is set of data elements.

The most commonly used data elements are:

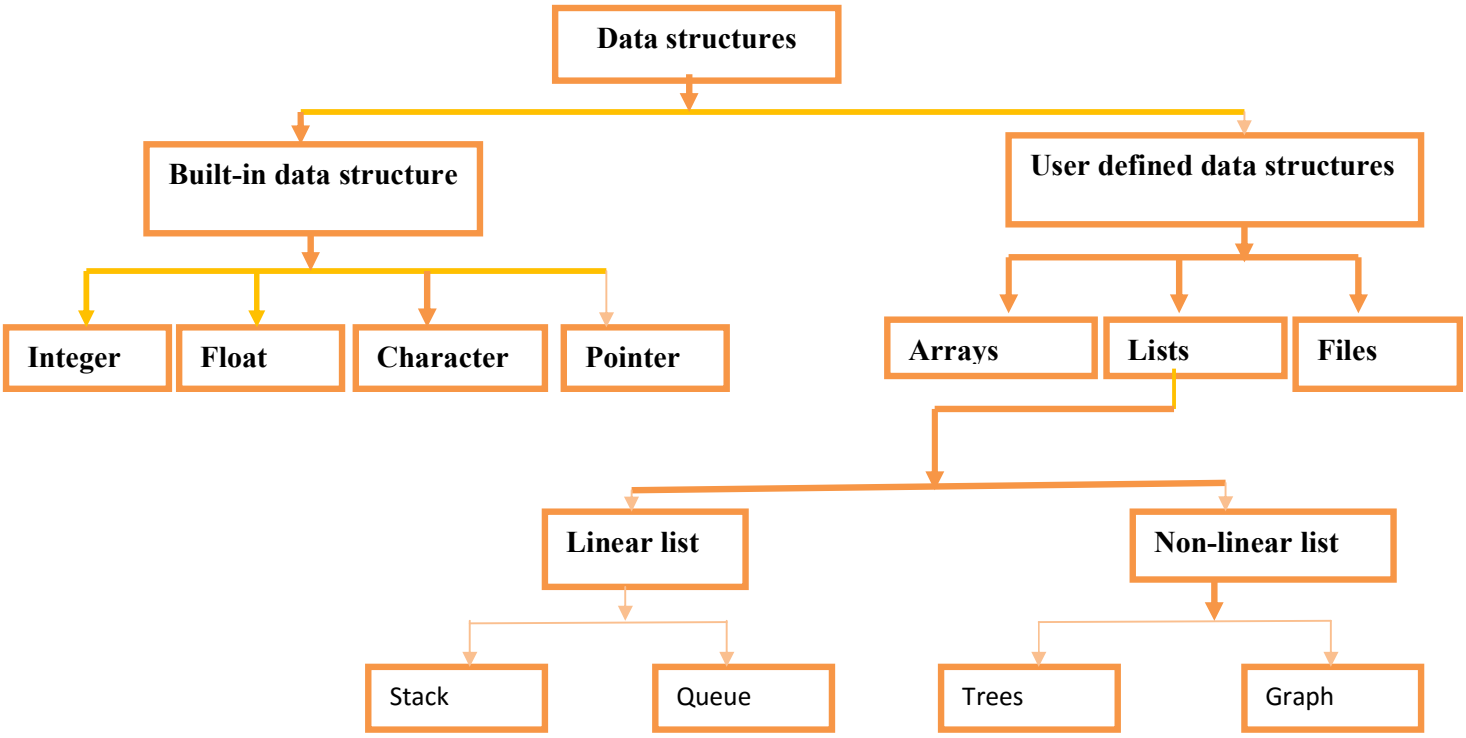
a) **Numeric data:**
   This includes integers, complex numbers, real numbers and double numbers.
b) **Logical data:**
   Most programming language has logical data which is related to logical operations.
c) **Character data:**
   The character data elements store character and string of characters.
d) **Pointers:**
   These are data elements which stores address of another data.
   E.g. p: =add r(x)
e) **Labels:**
   Some programming languages uses value as a statement in program.
   E.g. printf ("Hello");
   There are some data elements that may be defined in programming languages.
f) **Identifiers:**
   Programming language uses various data elements and it compulsory to store each data element in cells. That each cell has a name called as identifiers. The set of identifier different for each programming language.
g) **Attributes:**
   Attributes of a name determine its property. The most important attribute of a name is its type. Another attribute of name is its type. The scope of name is the block in which it is declared.

Data structure

Data structure is a way to store and organize data so that it can be used efficiently. Hence integer, float, Boolean, character etc., all are data structures. They are known as primitive data structure.

We also have some complex data structures, which are used to store large and connected data called as Abstract structure.

These are:



a) Array :

An array is series of elements. These elements are of same type. Each element can be individually accessed using an index.

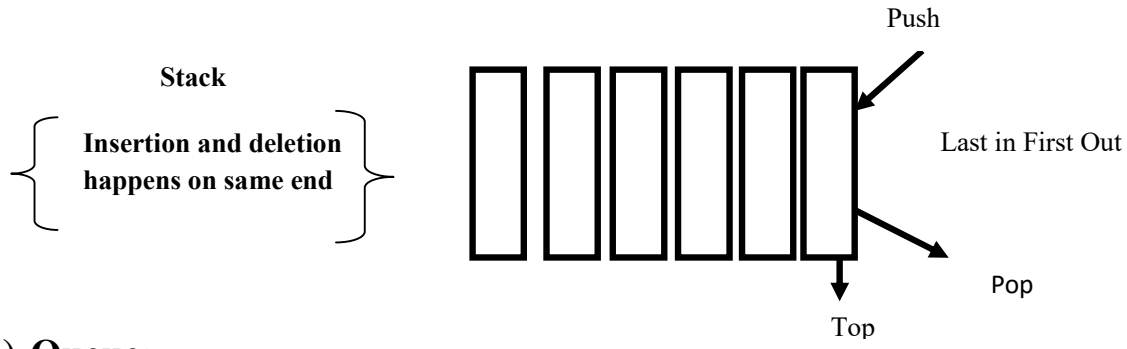
i. One dimensional array:
The one dimensional array stores the data in linear format.

ii. The data structure for one dimensional array contains following information.
   a) data type
   b) element limit
   c) no of elements
   d) lower limit
   e) upper limit

iii. Multidimensional array:
The multidimensional array store to data in one or two format i.e. rows and columns format.
E.g. A [1, 1] [1, 1]

b) **Stack**:

Stack is a linear data structure which follows a particular order, in which the operations are performed. The order may be LAST IN FIRST OUT (LIFO) or FIRST IN LAST OUT (FILO).



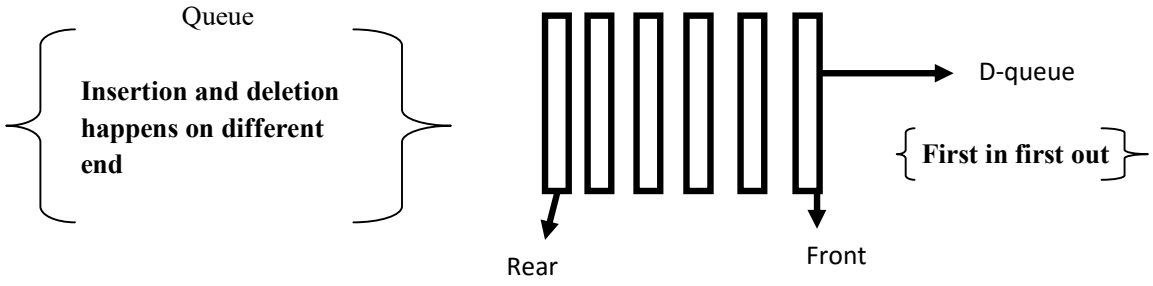
c) **Queue**:

A queue is a linear structure which follows a particular order in which the operations are performed.

The order is FIRST IN FIRST OUT (FIFO)

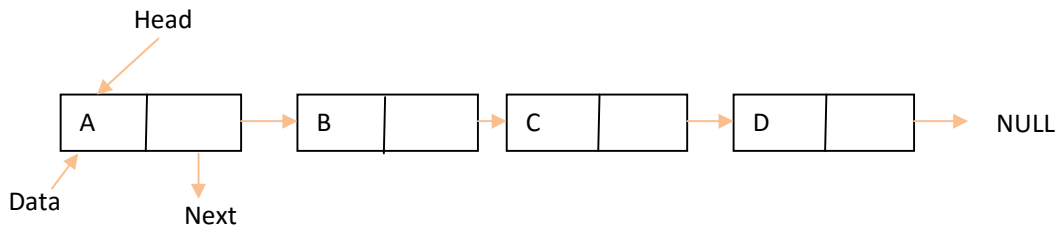
A good example of a queue is any queue of consumers for a resource where the consumer that comes first is served first.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



d) Linked-list:

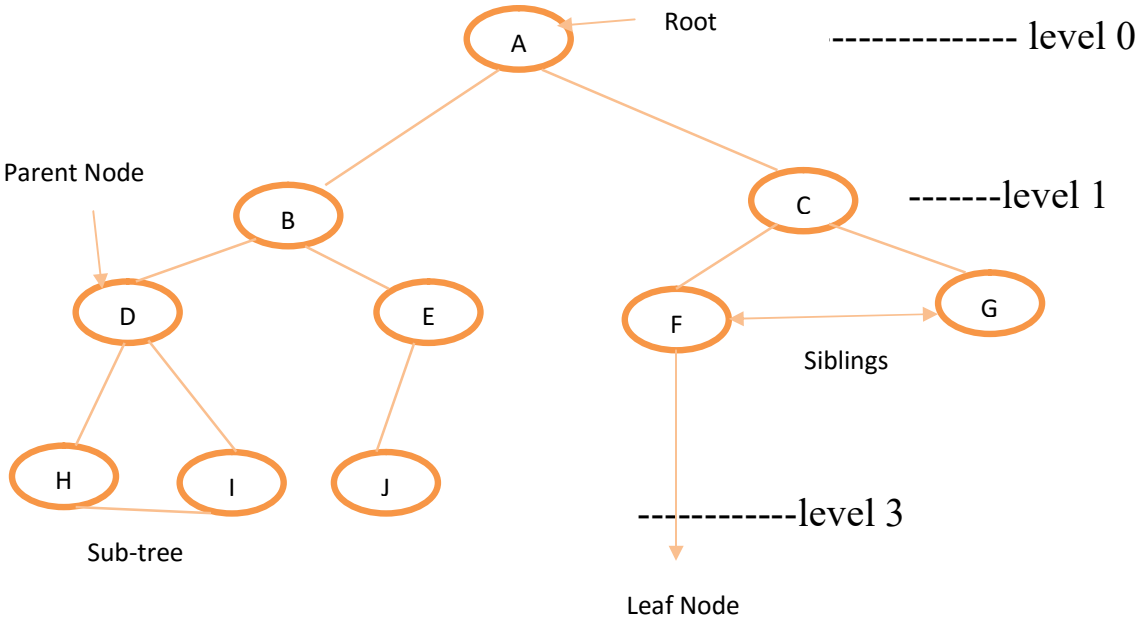
A linked list is a linear data structure, in which his elements are not stored in contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image.



A linked list consists of nodes where each node contains a data field and reference (link) to the next node in the list.

e) **Tree**:

Tree represents the nodes connected by edges. Binary tree is a special data structure used for data storage purpose. A binary tree has a special condition that each node has a maximum of two children.



#Operators:

An operator in a programming language is a symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.

a) Arithmetic operators:

The arithmetic operators are used to perform arithmetic operation. +,-,*, /, % are the arithmetic operators.

The '-'operator can be either unary (-x) or

The unary operator having postfix and prefix expressions. The postfix is used after operand and prefix is used before operand. e.g. 'c' language uses both prefix and postfix i.e. ++x and x++

b) Relational operators:

The relational operator used to specify relation between two entities. The relational operators take pair of operand and return the one logical value.

e.g. a=10 b=20

a>b=0

following are the relational operators.

>	greater than
<	Less than
>=	greater than equal to
<=	less than equal to
==	Equal to equal to
!=	not equal to

c) Logical operators:

The logical operators have argument as logical value and return it logical value.

E.g. a=10, b=20, c=30

a>b && a>c

10>20 && 10>30

0 && 0

0

Following are the LOGICAL operators:

&& logical AND

|| Logical OR

! Logical NOT

d) Assignment operators:

The assignment operator used to assign result to the variables. The different programming languages use different programming languages.

Eg: A=B is an assignment operator used in c language A: =B is an assignment operator used in ALGOL language.

e) String operator:

String operator used to perform string operations on string. Concatenation, substring formation and copy of string the operators on string.

The concatenation operation combines the pair of string to single string.

The concatenation operation combines the pair of string to single string.

Eg: “COC”, “SIT”

String= “COC” || “SIT”

String= “COCSIT”

Precedence of operators:

Suppose a+b*c

We can represent above expression in two ways (a+b)*c

A+ (b*c)

Both are correct but different from each other

∴ Some programming language provides precedence level.

Eg:

The precedence level of PL | I is

i) Unary -

ii) * |

iii) + -

iv) =

v) &&, ||,!

vi) >, <, ≥, <=

 # Assignment:

The assignment is most common operator used in programming languages.

The various programming languages uses different assignment operator

FORTRAN→ A=B

ALGOL→ A: =B

BASIC→ A ⟵ B

COBOL→ Move B to A

- A=B means put the value of B in the location of A
- The value that has been stored is on the right side of symbol.
- The value on the right size of symbol is called R value.
- The value on the left side of the symbol is called L value.
- The assignment can be, implemented as follows:

a) The compiler generates code to compute R value.
b) If data types of R and L values are different then compiler generate code to convert R value into a type suitable for L value.
c) Finally the complier generates code to store contents into L value.

Eg:
A= (c+d) + (e+f)
But assignment operator has lower precedence than plus (+)
Therefore R value is executed first then L value will be executed.
Eg: A [5] = {0, 1, 2, 3, 4}
Here, L value is location and R value is the values stored in the location.

 # Statements:

- The elementary actions of program are specified by using statements
- A statement is a command that programmer gives to the computer.
  Eg: print+ ("Hello world");
- There are two main types of statement
-

a) simple statement:

The simple statement is a statement which does not contain any embedded statement.

e.g.: Assignment statements, GOTO Statement

b) compound statement:

The compound statement is a grouping of multiple statements into single statements. It is a statement which contains embedded statement.

Eg.

i) If (condition) then statement

ii) If (condition) then statement else statement

iii) While (condition) then statement

iv) for (Initial to final value) statement

Other types of statements are:

a) computing / computation statements:

The statements apply operator to operand and computer the new value.

Eg. Assignment Statements.

b) sequence control statement:

These statements transfer control from one statement to other statement.

e.g. Function call, return and break statement

c) structural statement:

This statement contains group of simple statement into one structure.

Eg.

```
    switch (value)
{
Statements;
}
```

d) declaration statement:

These statements are used to declare variables and functions.

These statements do not contain executable code.

e.g.:

```
    int a;
    int add (int, int)
```

e) Input - output statement:

These statements are used for input and output processing.

These are string of information about how data is to be stored or write.

e.g.:

printf, scanf statements.