PROGRAMMING LANGUAGES

A Brief History

Programming Language :

Programming Language is A notation for describing Algorithm and data.

Program :

A Sentence of a programming Language

Lets Start From YEAR "1954"

- **FORTRAN (FORmula TRANslator)**
  - → created in 1954 by John Backus
  - → First High level Language
  - → Using the first Compiler ever developed
  - → Refered to as Scientic Language
  - → machine Independent.

- **FORTRAN II**

→ In 1958 introduces Subroutines, Functions, Loops and a primitive For loop

- **IAL (International Algebraic Logic)**

→ It started as a project later Renamed ALGOL 58

→ The theoretical definition of the language is published

→ No compiler

- **LISP (LIST Processing)**

→ Created in 1958 and released in 1960 by John McCarthy of MIT.

→ LISP was intended for writing Artificial intelligence programs.

Features

- Atoms and Lists data structure
- Functional programming Style - all Component's Computation is performed by applying functions to arguments. Variable declarations are rarely used.
- A Reliance on Recursion - a strong reliance on recursion has allowed LISP to be Successful in many areas. including Artificial Intelligence.

- **Garbage Collection:** Lisp has built in garbage collection, So programmers do not need to explicitly free dynamically allocated memory.

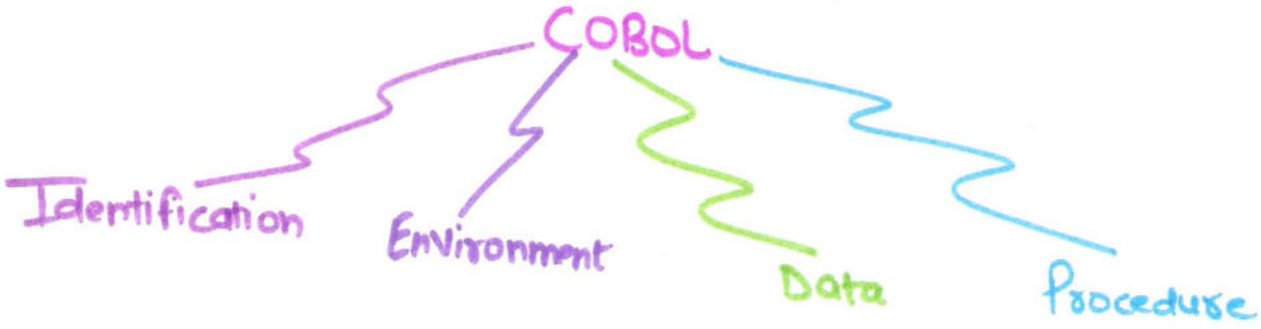
- **COBOL (Common Business Oriented Language)**

→ Created in May 1959 by the Short Range Committee of the U.S. Department of (DoD).

→ The CODASYL (COnference on DATA SYstems Languages) worked from May 1959 to April 1960.

→ ANSI Standard included (COBOL-68 (1968), COBOL-74 (1974), COBOL-85 (1985), and COBOL-2002 (2002)

→ Object Oriented Version of COBOL is introduced in 1997 i.e COBOL-97



→ Introduced the RECORD data Structure.

- **ALGOL (ALGOrithmic Language)**

→ Released in 1960, major Releases in 1960 and 1968.

→ It is first Block-Structured Language

→ Considered to be the first Second generation Computer lang.

→ Machine Independent

→ It introduced Concept like

→ Block Structure Code
(marked by BEGIN and END)

→ Scope of Variables
(Scope of local variables inside blocks)

→ BNF (Backus Naur Form)
Notation for defining Syntax.

→ dynamic Arrays

→ Reserved words

→ IF THEN ELSE, FOR, WHILE loops
the := Symbol for assignment
SWITCH with GOTO
User defined data Type.

• SNOBOL (StroNg Oriented symBOLic Language)

→ Created in 1962

→ Inted for "Strings"

→ First language to use Associative Arrays, indexed by any type of key.

→ Had feature of Pattern-matching, Concatenation,

and alternation.

→ It allowed running code stored in Strings.

→ Data Types:- integer, real, array, pattern, and user defined types.

- **BASIC** (Beginner's All-purpose Symbolic Instruction Code)

→ Designed as a teaching language in 1963 by John George Kemeny and Thomas Eugene Kurtz of Dartmouth College.

→ Intended to make it easy to learn programming.

PL/I (Programming Language One)

→ Created in 1964

→ Intended to combine the features of FORTRAN with COBOL, Plus additional facilities for Systems progrming.

→ Also borrows from ALGOL 60.

→ Originally called NPL (New programming language).

→ Introduces storage classes (automatic, Static, Controlled and based), exception processing (ON Condition

→ Select when otherwise Conditional Structure and Several

variation of DO Loop.

→ Numerous data types.

- **Pascal** (Named for French religious fanatic and mathematician Blaise Pascal)

→ Created in 1970

→ Intended to replace BASIC for teaching language

→ Quickly developed as a general purpose language.

→ Programs Compiled to a platform-independent intermediate P-code.

→ Compiler for Pascal was written in Pascal.

- **C Language**

→ developed from 1969-1972 by Dennis Ritchie.

→ used in System programming for UNIX

- **ANSI C** The American National Standards Institute (ANSI) formed a technical Sub committee, X3J11, to create a standard for the C language and its run-time libraries.

- **Ada**

→ Released in **1983 (ADA 83)**, with major releases in 1995 (ADA 95) and 2005 (ADA 2005)

→ Created by U.S. Department of Defense **(DoD)**

→ Intended for **embedded systems** and later intended for all military Computing purposes.

- **Perl (Practical Extracting and Report Language)**

→ Created by Larry Wall in 1987

→ Intended to replace the **Unix Shell, Sed, Awk.**

Python:

→ Created in **1991** by **Guido van Rossum.**

→ A scripting Language with dynamic type. intended to replace **Perl.**

Characteristics of a Good Programming Language

These are various factors, why the programmers prefer one Language over another. and some of very good Characteristics of a Good programming Language are.

1.) **Clarity, Simplicity, and Unity :** A programming Language Provides both a conceptual framework for algorithm planning and means of expressing them. It should provide a clear, simple, and unified set of Concepts that can be used as primitives in developing algorithms.

It should have → • Minimum No. of different concepts.
• with Rules for their combination being • Simple and Regular

This attribute is called **Conceptual integrity.**

2.) **Orthogonality :-** It is one of the most important feature of PL Orthogonality is the property that means "**Changing A does not change B**".

If I take Real world example of an Orthogonal System

would be a **Radio**, where changing the **Station** does not change the **Volume** and vice-versa.

When the features of a language are orthogonal, language is **easier to learn** and programs are easier to write because only few exceptions and special cases to be remembered.

3) Support for Abstraction:- There is always found that a substantial gap remaining between the abstract data structure and operations that characterize the solution to a problem and their particular data structure and operations built in to a language.

4) Programming Environment:-

An appropriate programming environment adds an extra utility and makes language to be implemented easily like

The availability of
- Reliable
- efficient
- well-documentation

Speeding up Creation & Testing by
- Special Editors
- Testing packages

Facility
- Maintaining & Modifying — Multiple version of program / software

5) Ease of Program Verification :- Reusability

The reusability of programs written in a language is always a central Concern.

A programs is checked by various testing techniques like

Formal verification Method

Desk checking

Input - output test checking

We verify the program by many more techniques.

A language that makes program verification difficult may be far more troublesome to use.

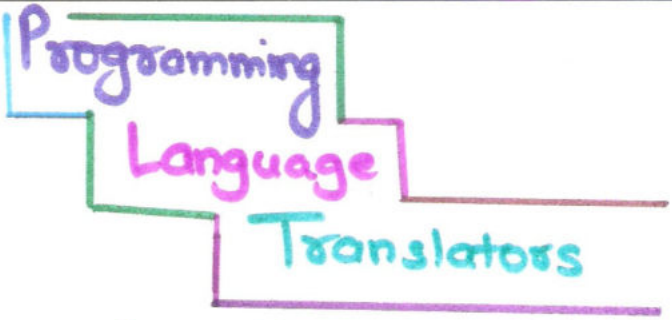
Simplicity of Semantics and syntactic structure is a primary aspect that tends to simplify program verification.

6) Portability of Programs:-

Programming Language should be portable means it should be easy to transfer a program from which they are developed to the other computer.

A program whose definition is independent of features of a particular machine forms can only Support Portability.

Eg:- Ada, FORTRAN, C, C++, C#, Java.



Assembler Compiler Interpreters

Introduction To Translators :-

To Execute a computer program written in high level language must be translated in to machine understandable language ie machine language or Machine code.

A Translator is basically computer program that performs the translation of a programm written in a given **Programming Language** into functionality equivalent program in a different Computer **Language**, without losing the functional or logical Structure of the original code.

These are 3 types of Programming Language Translators

- → **Assembler**
- → **Compilers**
- → **Interpreters**

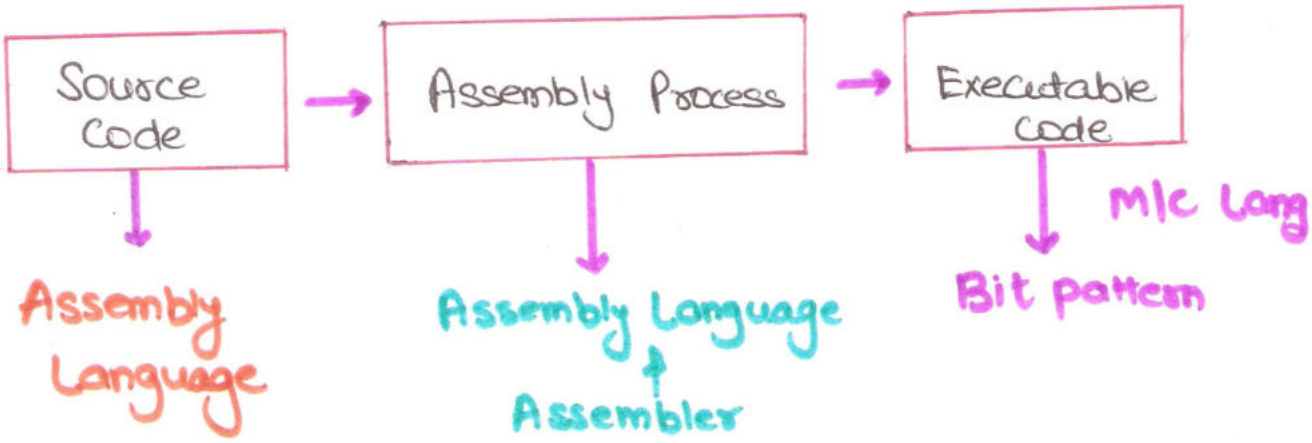
Source Code :- is the input to translator

Executable Code :- is the code that is output from Translator.

Assembler : Programming Language Translator

An Assembly Language which is basically Mnemonics like GIO, HALT, JUMP, NOT Code which is translated to Machine Language by programming Language Translator ie Assembler.

So Assembler is a program that takes Assembly and Converts them into pattern of bits that the Computer's processor can use to perform its basic operations. This pattern of bits is basically Machine language



Examples of Assemblers are

- NASM
- MASM

Compiler :- Programming Language Translator

A compiler is a program that translates a high level language into machine code.

For Example

The Turbo Pascal Compiler Translates a program written in Pascal into machine code that can be run on a PC.

Advantages of a compiler :-

1. Fast in Execution
2. The object | Executable code produced by a compiler can be distributed or executed without having to have the Compiler present
3. The object program can be used whenever required without the need of re-compilation.

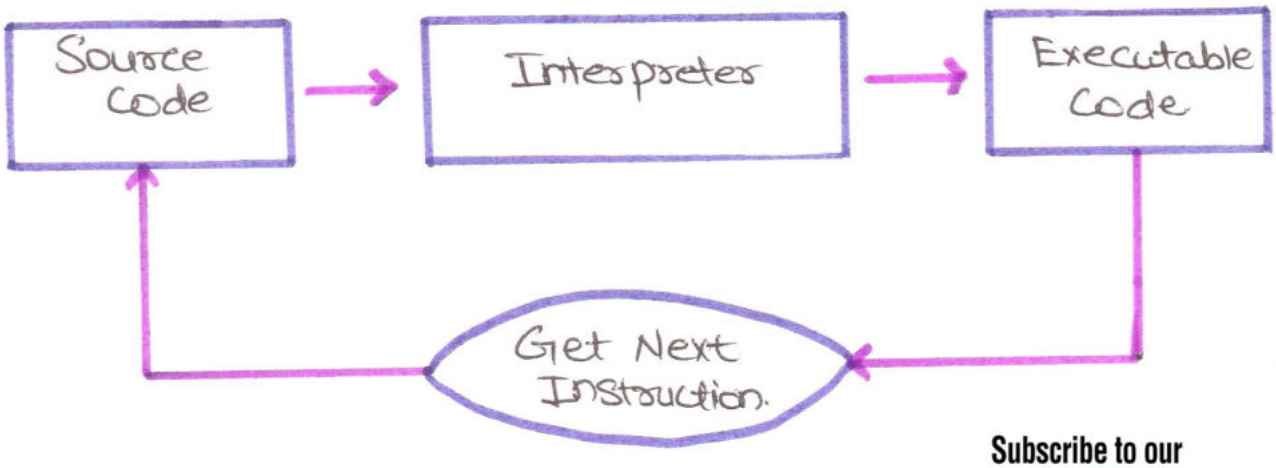
Disadvantages of a Compiler

1. Debugging a program is much harder
2. When error is found, the whole program has to be re-compiled

Interpreter :- Programming Language Translator

An **Interpreter** is also a program that translates high-level source code into **Executable code**.

How the difference between a compiler and an interpreter is that **An interpreter translates one line at a time and then executes it.** No object code is produced, and so the program has to be interpreted each time it is to be run.



Advantages of an Interpreter

1. Good at locating errors in the program
2. Debugging is easier since the interpreter stops when it encounters an error.
3. If an error is deducted there is no need to retranslate the whole program.
4. It uses less memory as only few lines has to be into the memory because No Object Code

Disadvantages of an Interpreter :-

1. Rather slow : Speed is biggest disadvantage
2. No object code is produced, so a translation has to be done every time the program is running
3. For the program to run, the Interpreter must be present.

Interpreter has to Analyse each line of code (byte code) into machine code before it can be executed.

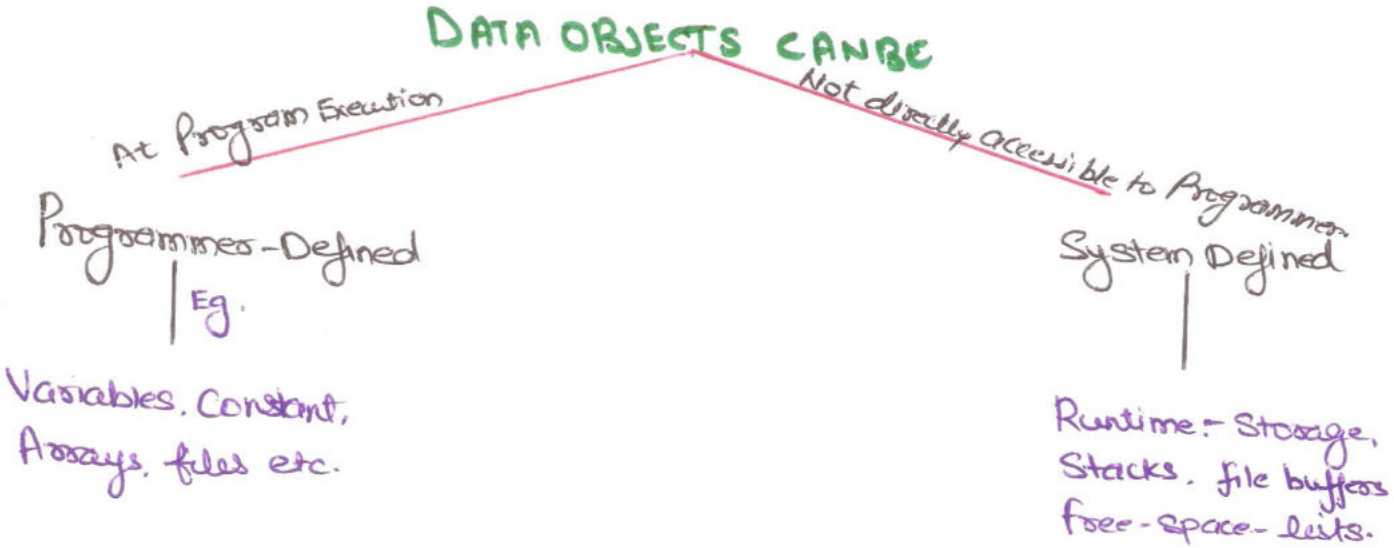
PROGRAMMING LANGUAGE

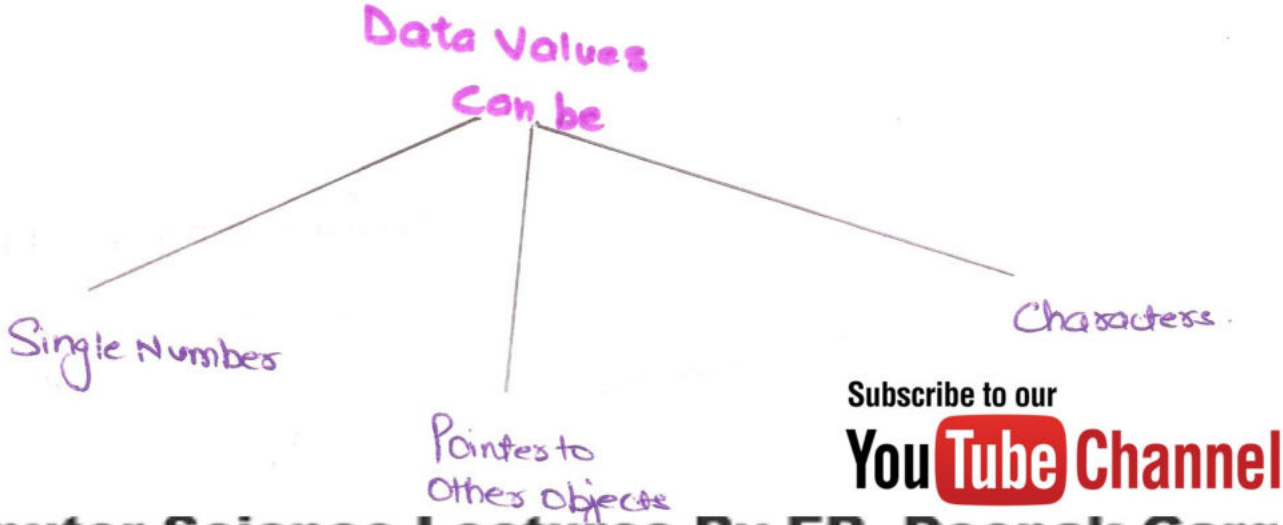
Data Object

DEF 1:- A Data object represents a container for data values, a place where data values may be stored and later retrieved.

DEF 2: A runtime grouping of one or more pieces of data in a virtual computer.

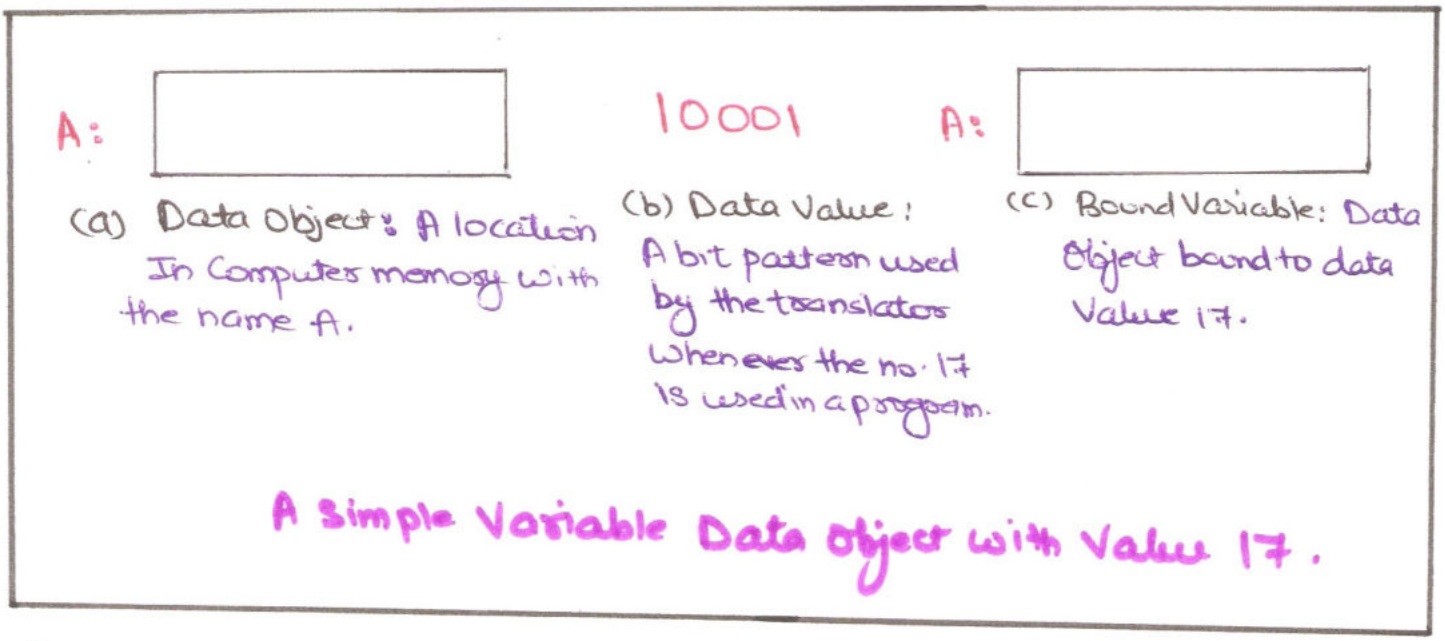
DEF 3: A location in memory with an assigned name in the actual computer.





A Data object is usually represented as storage in computer memory and a data value is represented by a pattern of bits.

So we can represent the relation between Data object and Data value



A data object is Elementary if it Contains a data value that is always manipulated as a unit.

A data object is data structure if it is an aggregate of the data objects.

Binding & Attributes of Data object:- Binding is an association of Data values & entity.

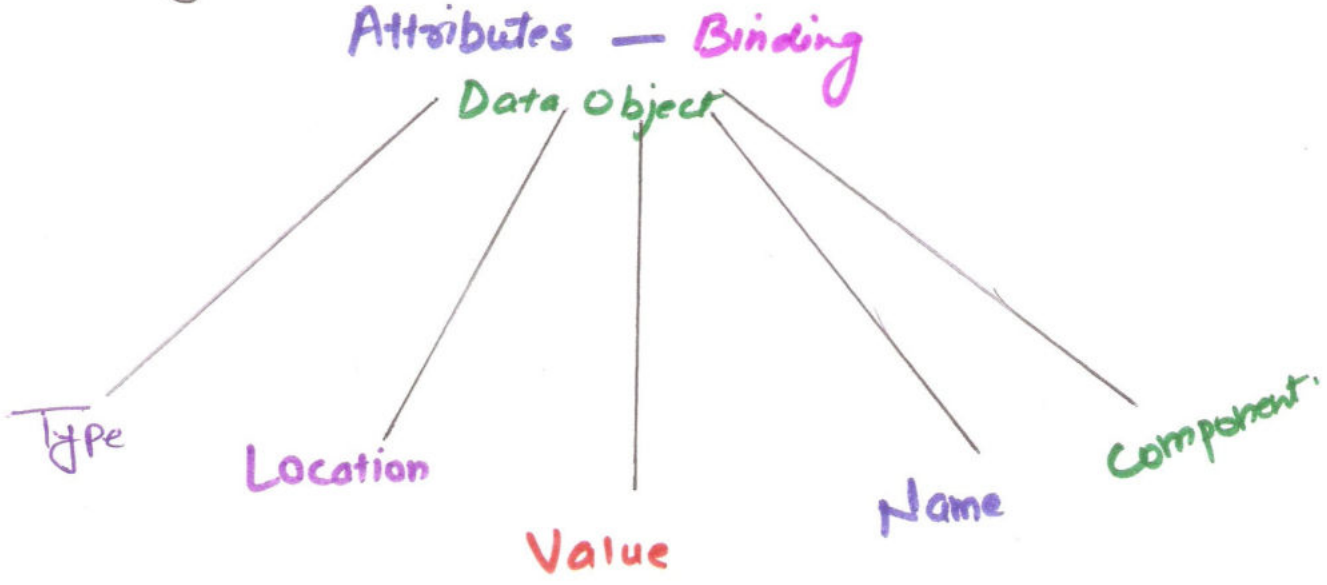
- **Type:** This associates the data object with the set of data values that the object may take.

2) **Location** : This associates the binding of a storage location in memory where the data object is represented. Only storage management routines can only change the data object in the virtual computer.

3) **Value** : This binding is usually the result of an assignment operation.

4) **Name** : The binding to one or more names by which the object may be referenced during program execution is usually setup by declaration and modified by subprogram calls and returns.

6) **Component** :- The binding of a data object to one or more data objects of which it is a component is often represented by a pointer value. And may be modified by a change in pointer.



ELEMENTARY DATA TYPES

Variables and Constants :-

Variables :- A variable is a quadruple which is composed of a name, a set of attributes, a reference and a value.

A Simple Variables is an elementary data object with a name and binding of data object to value may change during its life time.

This data objects are basically defined and named by programmer explicitly.

Attribute of a variable

Lets take eg. of a variable in ALGOL Language

```
y := 9;
```

We Can Say that it has four attributes

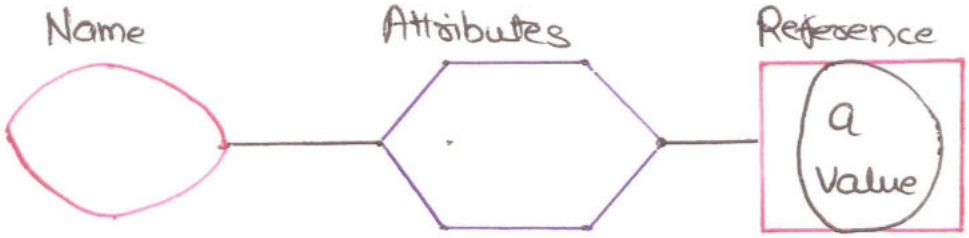
(i) The name of the box : y

(ii) The name or description of its Current Contents. ie 9 we Can also Say Square of 3.

(iii) The box or storage location(s) which hold(s) the Value.

(iv) The Contents of the box or 9.

The name of the box and its storage location are fixed, but the contents and its name may vary overtime.



The four Components of a Variables

Lets Take Another Example in C Language :-

int N; → It declares a simple data object N of type integer.
N = 27; → Data value 27 is assigned to variable N.

1) Declares the variable Name N of type integer
2) Life time of N is execution end
3) Data object bound to N during end of execution time.
4) Value 27 is assigned and may be changed during life of N.
5) Hidden from the programmer are other bindings made by virtual Computer like Creating activation Record, Storage for this activation Record in run-time Stack etc.

ELEMENTARY DATATYPE
Variables and Constants

CONSTANT : A data object with a name that is bound to a value (or values) permanently during its life time.

The Constant value can only be a number, string or identifies which denotes Constant

A Constant definition in Pascal introduces an identifies as a Synonym for the Constant value.

* Pascal uses the reserved word const to begin a constant declaration

```
Const PI = 3.1415;
```

** In ALGOL 68 we can define Constant by

```
real root2 = 1.4142135;
```

This was much acceptable that time.

** In Ada, provides a uniform notation for setting Constants to initial values and for initializing Variables:

```
X: Constant INTEGER := 17;
```

** In C language :- Const is used to initialize the Constant value

```
Const int MAX=80;
```

The Constant MAX is a programmer-defined Constant because the programmer explicitly defines the name for value 30.

In C, these is macro definitions which is used for Control the execution of program: and can be used for Declaring constant

Eg

```
#define MAX 30
```

It is a Compile-time operation that causes all references to MAX in program to be Changed to the Constant 30.

In this 30 has two names, the programmer defined MAX and literal name 30. both of which may be used to refer to a data object Containing the value 30.

* #define MAX 30 is a command, which the translator uses to equate MAX with the value 30, where as the **Const** attribute in C is a translator directive Stating that MAX will always Contain the value 30.

DATA TYPES

A data type is a set of objects and a set of operations on those objects which create, build-up, destroy, modify and pick apart instances of the objects.

OR

A data object is a class of data objects together with a set of operations for creating and manipulating them.

A programming Languages necessarily deals more commonly with data types such as the class of arrays, integers, or file and the operations provided for manipulating arrays, integers or files.

Eg:-

In LISP major data type is the binary tree (called an S-expression) and basic operations are CAR, CDR, and CONS.

FORTRAN 77	ALGOL	Pascal	Ada
INTEGER	integer	integer	INTEGER
REAL	real	real	FLOAT
LOGICAL	Boolean	Boolean	BOOLEAN
CHARACTER		Char	CHARACTER
DOUBLE PRECISION			
COMPLEX			

The basic elements of a specification of a data types

Attributes:
Distinguish data objects of that types

Values
that data objects of that type may have.

Operations
Possible manipulations of data objects of that type.

Eg:- Array data type

Attribute :-
- Numbers of dimensions
- the Subscript range for each dimension and
- the data type of Components

Value :
- It would be sets of numbers that form valid values for array components.
-

Operations :
- It may include Subscripting to select individual array components
- Create Arrays.
- Change their Shape
- Performing arithmetic on pairs of arrays.

The basic elements of the implementation of a data types :-

1. **Storage representation** : It is used to represent the data objects of the data type in the Storage of the Computer during program execution.

2. **Algorithms or procedures** : The manner in which the operations defined for the data type are represented in terms of particular algorithms or procedures that manipulates the choosen storage representation of the data objects.

Specification of Elementary Data Types

An Elementary data object Contains a single data value and class of such data objects over which various operations are defined is termed an Elementary data type.

Some Elementary Data Types :- integer, real, character, Boolean, enumeration and pointer. and specification may differ significantly between two languages.

Attributes :- Basic attributes of any data object, such as data type and name, are usually invariant during its lifetime.

Some attributes may be stored in a descriptor as a part of the data object during program execution. Others may be used only to determine the storage representation of the data object.

The value of an attribute of a data object is different from the value that the data object contains.

Values :- The type of a data object determines the set of possible values that it may contain.

Eg :- C defines the following four classes of integer types int, short, long and char because most hardware implements multiple precision integer arithmetic (eg. 16 bit and 32 bit integers or 32 bit and 64 integers)
we can use 'short' for shortest value of the integer word length.

long uses the longest value implemented by the Hardware
int uses the most efficient value that the hardware implements.

In C, characters are stored as 8 bit integers in the type char, which is subtype of integer.

Operations: The set of operations defined by a language is basically rules that how data object of that data type may be manipulated.

→ If the operations are Premitive operations, means specified as part of language.

programmer-defined operations, in the form of subprograms or method declarations as part of class definitions.

Eg:-

$+ : \text{integer} \times \text{integer} \rightarrow \text{integer}$

(a) Integer addition is an operation that takes two integer data objects as an arguments and produces an integer data object as a result.

(b) $\text{SQRT} : \text{real} \rightarrow \text{real}$

A Square-root operation, SQRT, on Real number data object is specified.

(Post of operation)

An Algorithm that specifies how to compute the results for any given set of arguments is a common method for specifying the action of an operation

In C we have concept of function prototype which is Signature of an operation, the number, order and data types of the arguments in the domain of an operation are given as well as the order and the data type of the resulting range.

Binary operation:- Two arguments with single result

Monolic operation:- Single argument with single result.

Implementation of Elementary Data types.

Implementation of an elementary data type Consists of

→ Storage Representation for data objects
→ Values of that type
→ Set of Algorithms or procedures that define the operations of the type in terms of manipulations of the storage representation.

Storage Representation :-

1.) Hardware Influence : Computer hardware influence the Storage of Elementary Data Type.

In this case Computer hardware executes the program.
If the hardware storage representation are used, then the basic operations on data of that type may be implemented using hardware provided operations.

2.) Software influenced :
If we donot use hardware Storage representation, then the operation must be Software Simulated and Same operation will execute much less efficient.

Two Methods to treat Attributes :-

→ To be determined by the Compiler and not stored in descriptors during execution or not stored in run-time storage representation. It is usually a method in C language.

→ Stored in a descriptor as part of the data object at run time in LISP, Prolog Language.

The storage representation is usually described in terms of

→ Size of the block of memory required (the number of memory words, bytes, or bits needed)

→ Layout of attributes and data values with in the block.

Implementation of operations:-

Each operation defined for data objects of a given type may be implemented in one of three main ways:-

(1) **Directly as a hardware operation** :- If simple data types are stored using the hardware representation, then the primitive operations are implemented using the arithmetic operations built in to the hardware.

2) **As a Subprogram or procedure** : A square root for an example, this operation is not provided directly as a hardware operation. So it is Software Simulated. implemented as a procedure or function.

3) **As an inline Code Sequence** : It is software implementation of the code & its operation. Instead of using a subprogram, the operation in the subprogram are copied into the program at the point where the subprogram would otherwise have been invoked.

for example :

The absolute value of function on numbers,

abs(x) = if x<0 then −x else x

is usually implemented as an inline code sequence.

(a) Fetch value of x from memory
(b) if x>0, skip the next instruction
(c) set x = −x
(d) Store new value of x in memory

Here each line is implemented by a single hardware operation.

Declarations:-

Declarations provide information about the name and type of data objects needed during program execution.

Two ways of declaration

→ implicit declaration
→ explicit declaration

Implicit Declaration or default declaration:-

They are those declaration which is done by compiler when no explicit declaration or user defined declaration is mentioned.

Eg:-
```
$abc = 'a string';
$abc = 7;
```

In 'Perl' Compiler implicitly understand that

$abc= 'a string' is a string variable

and

$abc= 7; is an integer variable.

Explicit declaration of data object :-

```
Float A,B;
```

It is an example of 'Float A,B;' of C language. In explicit we or user explicitly defined the variable type. In this example it specifies that it is of Float type of variable which has name A & B;

A declaration basically serves to indicate the desired life time of data objects.

Declarations of Operations :-

→ Compiler needs the signatures or a prototype of a subprogram or function so it can determine the type of argument is being used and what will be the result type

* Before the calling of subprogram, Translator need to know all these informations!

Eg in C language


```
Float Sub (intx, floaty)
```

It declares Sub to have the signature

Sub: int x float → float

Purpose of Declarations :-

1. Choice of Storage Representation :-

As translator determine the best storage representation of data types that why it needs to know primarily the information of data type and attribute of a data object

2. Storage Management :-

It makes to us to use best Storage Management for dataobject by providing its information. & these information as tells the life time of a dataobject.

For Example :-

In C Language we have many options for declaration for elementary data type

(i) Simple declaration : Like float A,B;
It tells life time is only at the end of execution

*As **lifetime of every data object can be maximum to**

end of execution time*

but Simple declaration tells the Single block of memory will be allocated.

(ii) Run time Declaration :-

C language and many more language provide us the feature of Dynamic Memory Allocation by keywords **malloc** and **calloc**.

So in this special blocks of memory is allocated in memory and their life time is also different.

3.) Polymorphic Operations :-

In most language, some special symbol like + to designate any one of the several different operation which depends on type of data or argument is provided.

In this operation has same name like as we discussed + So in this case operation symbol is said to be **Overloaded** because it does not designate one specific operation.

Ada allows programmer to overload Sub programs.

ML Expands this concept with full polymorphism where function has one name but variety of implementation depending on the types of arguments.

4.) Type checking :-

Declaration is basically for Static type Checking rather than dynamic.

Type checking & Type Conversion

Type checking means checking that each operation should receive proper no. of arguments and of proper data type. like

$$A = B * J + d;$$

* and - are basically int and float datatype based operations and if any variable in this A = B*J+d; is of other than int and float then compiler will generate **type error**.

Two Ways of Type checking :-

1) **Dynamic Type checking :-** It is done at runtime.

→ It uses concept of typetag which is stored in each data objects that indicates the data type of the object.

Eg:- an integer data object contains its 'type' and 'value' attribute.

So operation only be performed after type checking sequence in which the type tag of each argument is checked. If the types are not correct then error will be generated.

→ Perl and Prolog follow basically dynamically type checking

because data type of variables A+B in this case may be changed during program execution.

→ So the type checking must be done at run time.

Advantage of dynamic types is the flexibility in program design

→ No declaration are required
→ Type may be changed during execution.
→ programmer free from most concern about data types.
→

** Disadvantage :-

(i) **Difficult to Debug** :- We need to check program execution Paths for testing and in dynamic type checking, program execution paths for an operation is never checked

(ii) **Extra storage** :- Dynamic type checking need extra storage to keep type information during execution

(iii) **Seldom Hardware Support** :- As Hardware seldom Support the dynamic type checking so we have to implement in software which Reduces execution speed.

Type checking

Static Type checking :- It is done at compile time.

Information needed at compile time is provided → by declaration
→ by language structures.

The information required includes

(i) **for Each operation :** The number, order, and datatypes of its arguments.

(ii) **For Each variables :** Name and data type of data object.

for example A+B In this types of A and B variables must not be changed.

(iii) **for Each Constant :-** Name and datatype & value.

Eg :-
```
Const int x = 28;
Const float x = 2.087;
```

In this datatype, its value and name is specified and infurther it checks value assigned should match its datatype.

Advantages :-

i) **Compiler Saves information :-** If the types of data is according to the operation

then Compiler Saves that information for checking Later Operations which further no need of Compilation.

(iii) **Checked Execution Paths:-** As Static type checking includes all operations that appears in any program Statement, all possible execution Paths are checked, and further testing for type errors is not needed.

So no type tag on data objects at run-time are not required, and no dynamic checking is needed.

Disadvantage :- It affects many aspects of languages

(i) Declarations

(ii) Data Control Structures

(iii) provision of compiling Separately Some Subprograms

Strong Typing :-

If we can detect all types of errors statically in a program, we say that language is 'strongly typed.'

→ It provides a level of security to our program.

Eg:- $f : S \longrightarrow R$

In this function f with signature S generate output R and R is not outside the Range of R data type.

If every operation is type safe then automatically language is strongly typed.

Examples of Strongly typed languages are

C, Java, C++, Ruby Rail, Smalltalk, Python.

Type Infer :- In this, like in ML, the language implementation will infer any missing type information from other declared type.

Eg:-

```
Fun area(length:int, width : int):int =
    length * width;
```

This is the standard declaration which tells length and Width of int data type and its return type is int and function name area.

but leaving any two of these declarations still leaves the function with only one interpretation.

Knowing that * can multiply together either two reals or two integers, ML interprets the following as equivalent to the previous example

```
Fun area(length, width)int  = length * width;
Fun area (length:int,width) = length * width;
Fun area ( length,width:int) = length * width;
```

Howeves :-

```
Fun area(length, width) = length * width;
```

is invalid as it is now ambiguous as to the type of arguments. They could all be int or they could be real.

Type Conversion and Coercion

Explicit type Conversion :- Routines to change from data type to another

Example:-

Pascal: the function 'round' - Converts a real type to integer

C : eg (int) x, for float x Converts the value of x to type integer.

Coercion : Implicit type of Conversion, Performed by the System.

Pascal: + integer and real, integer is converted to real

Java: Permits implicit Coercions if the operation is widening [int short → long int]

C++ : explicit Cast must be given.

Two opposite approaches to type coercion :

→ No Coercion ; any type mismatch is considered an error, Pascal, ~~Ada~~.

→ Coercion are the Rule : Only if no Conversion is possible, error is reported.

Advantages of Coercion :

It basically free the programmer from the low level concerns up to some level, as adding Two different data types i.e Real & Int

Disadvantage

As programmer Concern to some level is reduced which may be that it hides some serious errors which will not be easy to point out.

Assignment & Initialization

Assignment :- A basic operation for changing the binding of a value to the data object.

Languages like C, Lisp and many more assignment also returns a value, which is a data object containing a copy of the value assigned.

In Pascal

assignment(:=) : $integer_1 \times integer_2 \rightarrow Void$

Value of $integer_2$ is copied in $integer_1$

In C,

assignment (=) : $integer_1 \times integer_2 \rightarrow integer_3$

With this action : Set the value contained in data object $integer_1$ to be a copy of the value contained in the data object $integer_2$ and also create and return a new data object $integer_3$, containing a copy of new value of $integer_3$

Two Concepts through which we can define assignment

L-Value : Location for an object

R-Value : Content at that location.

Using L-value and R-value gives a more concise way to describe expression semantics.

Eg In case of Integer :-

$$A = B$$

In this copying the value of variable B to variable A.

** i'e assign to the lvalue of A the r-value of B

In case of Pointer

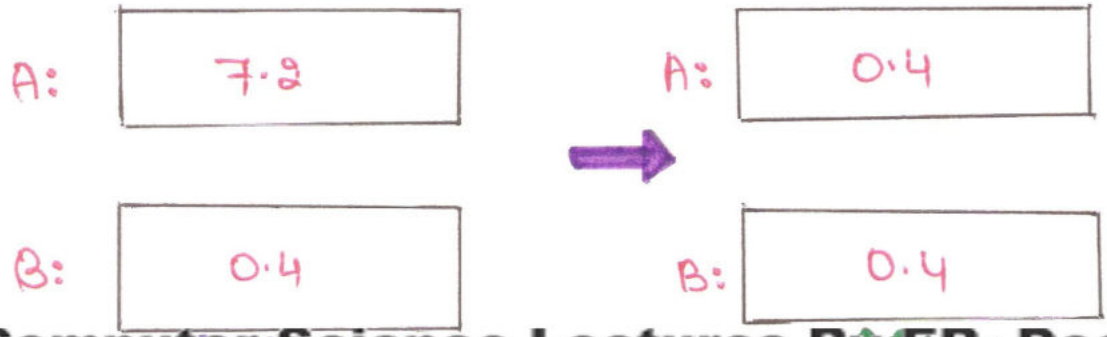
$$A = B$$

In this A & B are pointers variables. If B is a pointer then B's r-Value is the l-value of some other data object. This assignment then means,

"Make the r-value of A refers to the same data object as the r-value of B"

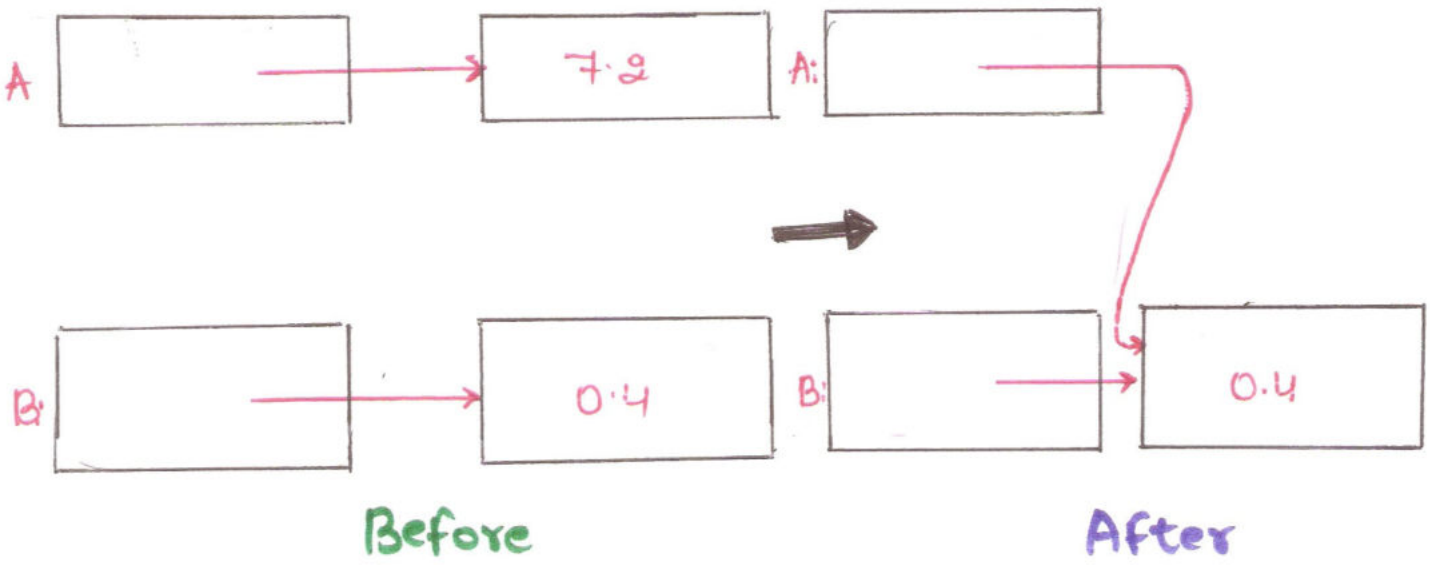
Thus, the assignment A=B means "Assign a copy of the pointer stored in variable B to variable A".

Numeric Assignment



Copy Value : (Pascal)

A := B



Pointer Assignment
In C

Two Views of Assignment:-

Copy Pointer : (SNOBOL)

A = B (Ptr to value of variable B assigned to variable A)

Initialization

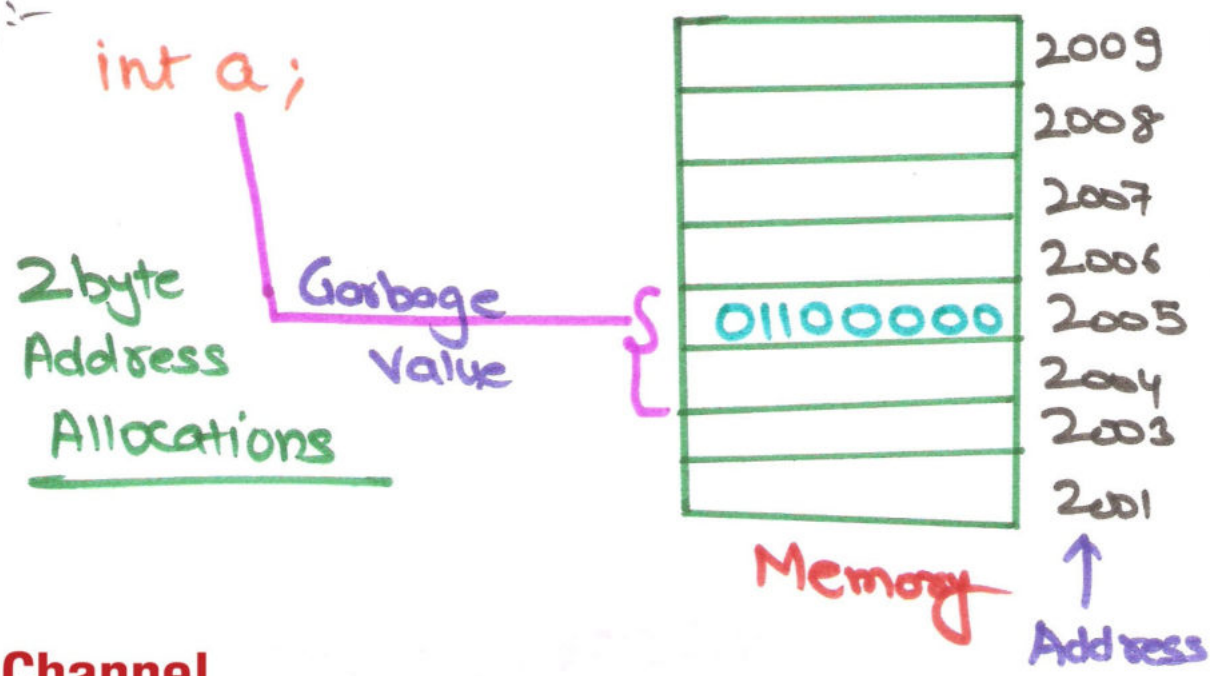
Initialization is basically a step in which we just specify the name of variable and data object but not yet assigned a value (i.e, an l-value with no corresponding r-value)

→ In this only block for storage is allocated

→ That block automatically make take some value in form of binary code.

→ It is serious programming error to create an uninitialized error as it becomes difficult to distinguish b/w original value or automatically filled value as both of them are of bit patterns.

Eg:-



Numeric Data Types

Integers :- The most primitive numeric data type is integer.

Specification :-

C has four different integers specification

int, short, long and char

but Maximal and minimal values depends upon the what bit architecture of hardware is basically and in some languages these values represented as defined constants.

Like in Pascal maxint

Types of operations on Integers

1> Arithmetic operation :-

It is of basically two types

- Binary operation
  - BinOp : integer x integer → integer
  - Eg addition(+), Substraction(-) Multiplication(*), division(/), remainder(mod)
- Unary operation
  - UnaryOp : integer → integer
  - negative(-), or identify(+), abs value

2) Relational operations:-

Signature is

Relop : integer x integer → Boolean

Where RelOp may be equal, notequal, less than, greater than, less-than-or-equal, greater-than-or-equal

Relational operation compares the values of its two arguments data values and returns a Boolean (True or false value) data object as its result.

3) Assignment Operations:-

Signature

assignment : integer x integer → integer

and

assignment : integer x integer → integer

4) Bit operations:-

In C, integers also plays the role of Boolean values Therefore additional bit operation are also defined.

Signature:-

BinOp : integer x integer → integer

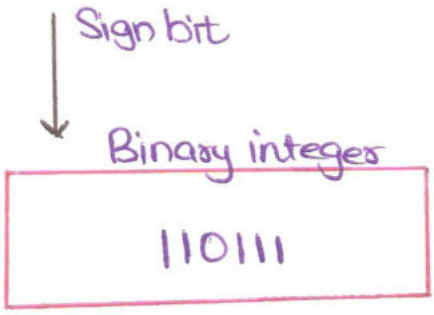
Operator (&) for and the bits together

Operator (|) for or the bits together

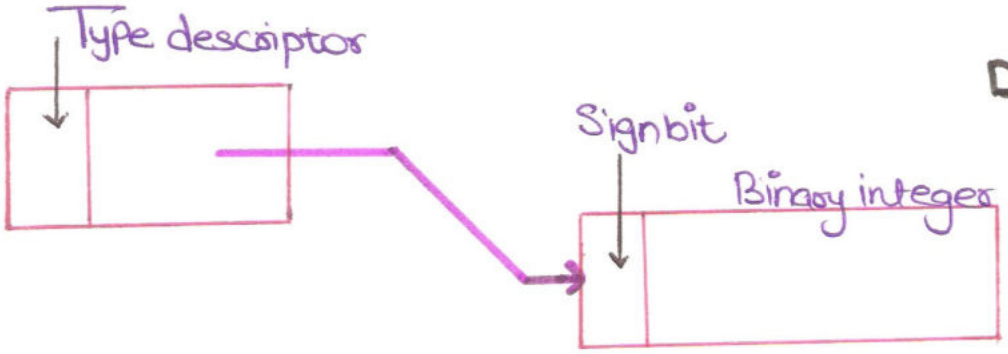
Operator (<<) for shift the bit among others.

Implementation of Integer :-

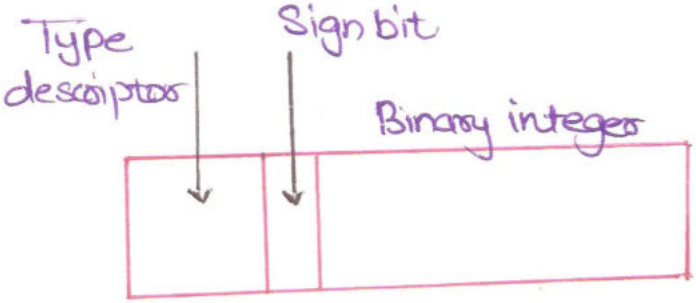
Most often using the hardware-defined integer storage representation and a set of hardware arithmetic and relational operations on integers.



NO DESCRIPTOR



DESCRIPTOR STORED IN SEPARATE WORD



DESCRIPTOR STORED IN SAME WORD

THREE STORAGE REPRESENTATIONS FOR INTEGERS

Sub Ranges of an Integer

Specification :-

A Subrange of an integer data type is a subtype of the Integer data type and consists of a sequence of integer values within some Restricted Range

Eg

Integers in Range
- 1 to 10
- -500 to 1000

Declaration in Pascal

```
A: 1..10
```


Declaration in ADA

```
A: integer range 1..10
```

Implementation :- Its implementation basically has two advantages

(i) Smaller Storage Requirement :- As a smaller Range of values, a Subrange value can usually be stored in fewer bits than a general integer value.

2) Better type checking :-

More precise type checking to be performed on the value assigned to that variables

Eg if variable Month is : **Month : 1..12**

then the assignment

Month := 0 is invalid and can be detected at compile time.

If we use assignment

Month : Month + 1

at runtime Compiler checks for range limit that should not be exceeded.

Numeric Data Types

FLOATING-POINT REAL NUMBERS :-

Specification :-
- In FORTRAN → real
- In C → float

Some precision required for floating-point numbers, in terms of the numbers of digits used in the decimal representation, may be specified by the programmer, as in Ada.

→ Similar arithmetic operations, relational and assignment operations as with integers are usually provided for real.

→ Boolean operations has restrictions

→ Equality between two Real no. is rarely achieved Due to Round off issues. because program that checks for equality to exit a loop may never Terminate.

→ Some inbuilt functions like
Sine and maximum value

Sin : real × real → real

and

max : real × real → real

Implementation :-

→ Storage representation based on hardware representation in which a storage location is divided into Mantissa (i.e Significant digit of the no.) and an exponent.

→ Any number N can be expressed as $N = m \times 2^K$ form between 0 and 1 and for some integer K.

→ A double-precision form of floating-point number is also often available ; in which an additional memory word is used to store an extended mantissa.

SCALAR DATA TYPE
ENUMERATION

An Enumerated data type is a data type whose domain values are given in a list or ordered list and whose only operations are equality and assignment.

OR

An Enumeration is an ordered list of distinct values.

OR

An Enumeration is a complete ordered listing of all items in a collection.

Pascal was first language which introduced Enumeration.

To make Enumeration facility useful, a programming language must provide a mechanism for declaring and defining the new data type and for declaring variables whose values will come from the elements of type.

It is assumed that these literals are distinct and thus equality can be directly defined.

Before an Era of Enumeration, what we had

for Example :- A variable **StudentClass** might have only four possible values representing freshers, Sophomore, Junior and senior.

Similarly, a variable **StudentSex** might have only two values

representing Male and female.

Before the Concept of Enumeration the languages like FORTRAN or COBOL Such variables is declared as integer type and distinct values are assigned.

like

Fresher = 1 , Sophomore = 2, and so on

and

Male = 0, Female = 1

Then translator manipulates values as integers.

That Creates big problem like

Sophomore = 1

&

Female = 1

as both has same values so can we apply integer based operation on it

As a point of view of programmer it should not be but accooding to Translator it can apply as they are of integer types.

Then

Languages Such as C, Pascal and Ada includes an enumeration data type that allows the programmer to define and manipulate Such variables more directly.

Specification of Enumeration :-

The programmers defines both the literal names to be used for the values and their ordering using a declaration such as in pascal

```
type months=(Jan, Feb, Mar, Apr, May, June,
             Jul, Aug, Sep, Oct, Nov, Dec);
```

in C

```
enum StudentClass { Fresh , Soph , Junior , Senior };
enum StudentSex { Male, Female };
```

In pascal, C example can be written as

```
type class = (fresh, Soph, Junior, Senior);
```

followed by declarations for variables such as

```
Studentclass : Class;
Student Sex class : Class;
```

Here type definition introduces the type name **Class**, which may be used whereever the **primitive type** name such as integer might be used.

It also introduces the **literals** of **Fresh, Soph, Junior and Senior**, which may be used whereever a language-defined literal such as "27" might be used. Thus we can write.

if StudentClass = Junior then ....

instead of the less understandable

if StudentClass = 3 then ....

which would be required if integer variables were used.

Static Compiler can find error such as

if StudentClass = Male then

As Male is part of StudentClass.

Operations which we can perform

- Relational operations (equal, less-than, greater-than, etc)
- Assignment
- Successor and predecessor

Implementation :-

→ Each value in the enumeration sequence is represented at runtime by one of integess 0, 1, 2, ... as only a small set of values is involved and the values are never negative.

→ In this integer representation is often shortened to omit the sign bit and use only enough bits for the range of values required, as with the subrange values.

→ only and maximom 2 bits are required to represent the Senior=3 in memory because $3 = 11$ (binary)

2 bits only

In C, the programmer may override this default and set any values desired for enumeration values. for example

```
enum Class { fresh=74, Soph=89, Junior=7
             Senior=28 }
```

With this storage representation for enumeration types, Relational operations such as =, >, and < may be implemented.

Scalar Data Types

Booleans

The Boolean Data Type is a data type, having two values (usually denoted true or false), intended to represent the truth values of logic and Boolean algebra.

Specification :- In Pascal and Ada, the Boolean data type is considered simply a language-defined enumeration, viz;

```
type Boolean =(false,true);
```

which both defines the names true and false for the values of the types and defines ordering **false<true**.

Common operations are

and : Boolean x Boolean → Boolean (Conjunction)

or : Boolean x Boolean → Boolean (inclusive disjunction)

not : Boolean → Boolean. (negative or Complement)

Implementation :-

Single bit of storage is provided, no descriptor designated the data type is needed. Because single bits may not be separately addressable in memory which often takes a byte or word to represent it if extended.

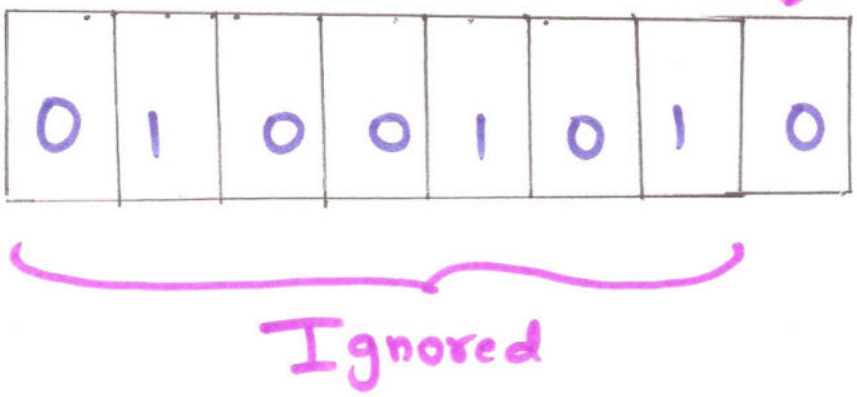
Then the values true and false might be represented

in two ways with in this storage unit :

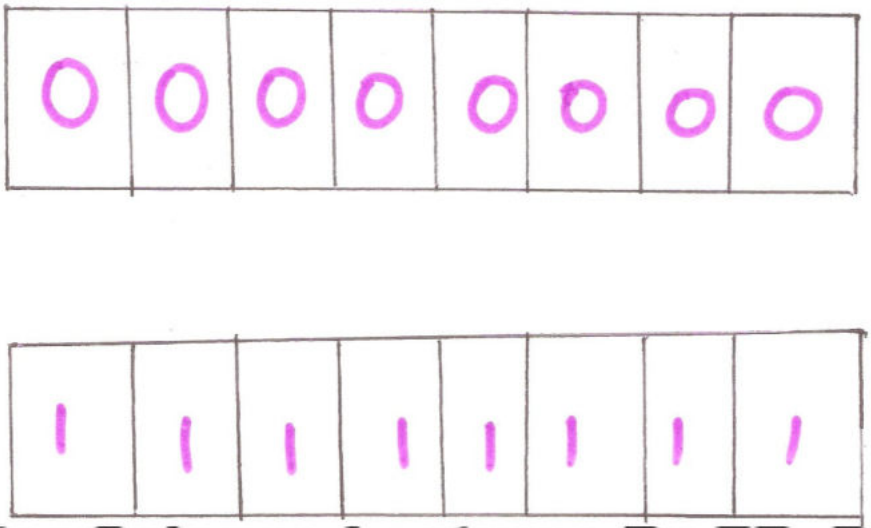
1. A particular bit is used for the value (often the sign bit of the number representation), with '0 = false', '1 = true', and the rest of the byte or word ignored, or

2. A zero value in the entire storage unit represent false, and any other non zero value represents true.

1)



2)



Scalar Data Types

Characters :-

Specification :-

A character data type provides data objects that have a single character as their value.

Set of values in character datatype depends upon hardware and operating system like ASCII character set. And ordering of the characters in this character set is called **Collating Sequence**. and ordering given by the 'relational operations'.

Character set includes
- Spaces
- digits
- Special character $, &, etc.

Operations on character data includes only

Relational operations

Assignment and

To test character for
- letter
- Digit
- Special character.

Implementation :-

Character data values are almost directly supported by the underlying hardware and operating system because

their use in Input-Out.

In C Character is declared

Char a; → one byte of storage Representation in memory

Char → Character Datatype

a → Character dataobject

ASCII value of Characters data type in C are

0 to 9 is 48 to 57 (ASCII Value)

A to Z is 65 to 90

a to z is 97 to 122

and all remaining for special characters.

```
Char a; // declaration
a = 'A'; // initializing character data object
           with A
```

Where = A = 65 = 1000001

0	1	0	0	0	0	0	1

= 65 = A In ASCII

Introduction To Syntax & Semantics

Syntax:-

Like Ordinary language English, programming Languages have Syntax.

The Syntax of a (programming) Language is a set of rules that defines what sequences of **Symbols** are considered to be valid expressions (progoams) in the language.

OR

The Syntax of a programming Language is what the program looks like.

Syntax provides significant information needed for understanding a program and provides much-needed information towards the translation of the source program into the object program.

A valid Representation of Syntax

$X = Y + Z$

Invalid Representation may be

$xy+-$

2+3x4 text will be interpreted this expression as having value 14 and not 20. That is, expression is

interpreted as if written 2+(3x4) and is not interpreted as if written (2+3) x4

We can specify either interpretation, if, we wish, by Syntax and hence guide the translator into generating the correct operations for evaluating this expression.

In a statement $X = 2.82 + 3.68$ Syntax cannot tell the type of X, on which result is depended

If X is Real then output will be 6.50 and if
If X is integer then output will be 6.

To completely describe the Syntatic Structure of programming language we need something else which can tell us the meaning of expression, Statements and program units.

Semantics :- Semantics is the meaning of an expression (program) in a programming language

In C to declare a 10 elements vector V of integer has declaration `int V{10};`

In Pascal `V: array [0,....9] of integer`

Although both creates Similar Data objects at run

time, their Syntax is very different. To understand the meaning of declaration we need to know the Semantics of both Pascal and C for Such array declaration.

Another Example

While (< Boolean_exp >) < Statement >

The Semantics of this Statement form is that when the Current value of the Boolean.exp. is true, the embedded Statement is true.

The General Problem of Describing Syntax

A Language, whether natural-like English, or artificial like C, Java, is a set of strings of characters from some alphabet. The strings of a language are called **Sentences or Statements**. So Syntax Rules of a language specify which strings of characters from the language's alphabet are in the language.

Formal descriptions of the Syntax of programming languages, for simplicity's sake, often donot include descriptions of the lowest-level Syntatic Units. These small units are called **Lexemes.**

Lexemes include its numeric literals, operators and special words, among others. We can think of programs as strings of lexemes rather than of characters.

Lexemes are partitioned into groups- for example. **The names of variables, methods, classes and so forth** in a programming language from a group called **identifiers.**

Each lexeme group is represented by a name, or token. **So, a token of a language is a category of its Lexemes.**

Consider the following Java Statement :-

```
index = 2 * count + 17;
```

Lexemes	Tokens
index	identifier
=	equal_sign
2	int_literal
*	mult_op
Count	identifier
+	plus_op
17	int_literal
;	Semicolon

2 Distinct ways of defining a language :-

A) • Language Recognizers :-

- A recognition device reads input strings of the language and decides whether the input strings belong to the language

Syntax Analyzers :- Determine whether the given program are syntatically correct.

B) Language Generators :-

→ It generates sentences of a language.

→ people prefers certain forms of Generators over recognizers because they can more easily read and

Understand them.

By Contrast, the Syntax-Checking portion of a Compiler (a language recognizer) is not as useful as language description for a programmer because it can be used only in trial-and-error mode.

To determine Correct Syntax of a particular Statement using a Compiler, the programmer can only Submit Speculated Version and note whether the Compiler accepts it.

On the other hand, it is often possible to determine whether the Syntax of a particular Statement is Correctly Comparing it with the Structure of the generator.

"Formal Methods of Describing Syntax"

ENGLISH

Alphabets: a, b, c, d, e ... z, A, B, C, ... Z

Punctuations: , . " ; ! "" - /

I
Eat
Chocolate

] WORDS: Combination of Alphabets

Sentence

I | eat | chocolate | . ✓

Eat I. Chocolate ×

He
eat
Chocolate
.

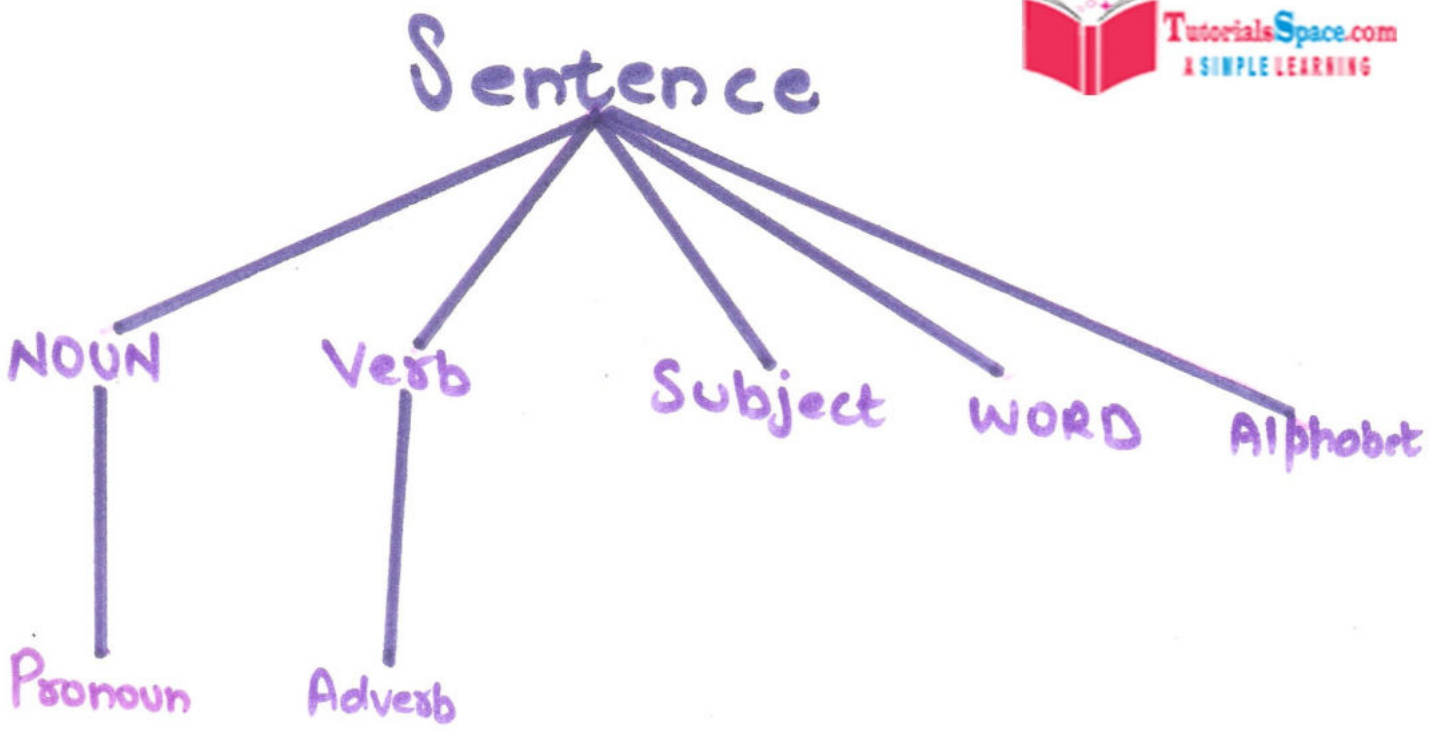
WORDS ARE CORRECT

He | eat | chocolate | . ×

Here eat should be Eats by Tenses

So

He eats chocolate. ✓



All these are connected with Grammar

Which Describes the Whole English Language Entities

Programming Language

1. Alphabets : a, b, c, d, ... Z, A, B.. Z

2. DIGITS : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3. Arithematics Symbols : *, /, +, -, Σ

4. Sepecial Symbols : →, ↑, —, $, #, @, !, ₹

Programming Language in More Technical way

→Header file : Math.h

→ Statements → Tokens

→ Expression → Lexemes

→ Identifiers → Literals

All needs a grammar which can describe their Syntax.

Grammar : Syntax and Structure of a Language.
It is used in Compiler Creation.
Describe the Syntax of a programming Language

Natural Language : A Language that has developed naturally through use.

Artificial Language: Language used to communicate with **Computers**.

Syntax : The rules governing the arrangement of words and phrases to create well-formed Sentences

Meta language : Language used to model the other Language

- **Determine** whether a Series of characters is valid
- Generate well formed statements
- **Break down** a statement into Constituent parts so it can be Converted into MIC Language.

Token: Each and every Smallest individuals units in a programming language are known as Token.

C Tokens are of Six Types

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total).
3. Constants (eg: 10, 20),
4. Strings (eg: "total", "hello").
5. Special Symbols (eg: (), {}),
6. Operators (eg: +, /, -, *)

```
int main()
{
  int x, y, total;
  x = 10, y = 20;
  total = x+y;
  Printf ("Total = %d \n", total);
}
```


Where

- main — Identifier

{, }, (,) — delimiter (Special Symbol)

- int — Keyword
- x, y, total — identifier
- main, {, }, (,), int, x, y, total — tokens

CONTEXT-FREE GRAMMARS

→ Chomsky (Liguist) described four classes of generative devices or grammars that defines four classes of languages.

→ Two of these grammars classes

Context-free ReGular

→ These grammars turned out to be useful for describing the Syntax of Programming Language.

- Regular Grammar — Describes → Tokens } of P.L.
- Context-Free Grammar — Describe → Syntax } of P.L.

Basically

Context-Free Grammar is a Language Generator which describes Syntax of Natural Language.

Backus-Naur Form

In 1960 John Backus and Peter Naur introduced formal notation method for describing Syntax of programming Language which is known As Backus-Naur Form, or Simply BNF

* BNF was basically designed for ALGOL 60

BNF and context-free Grammars was nearly identical.

BNF is a metalanguage for programming languages.

Metalanguage is a language that is used to describe another language

BNF: Symbols to describe a Syntax

::= means 'is defined as'

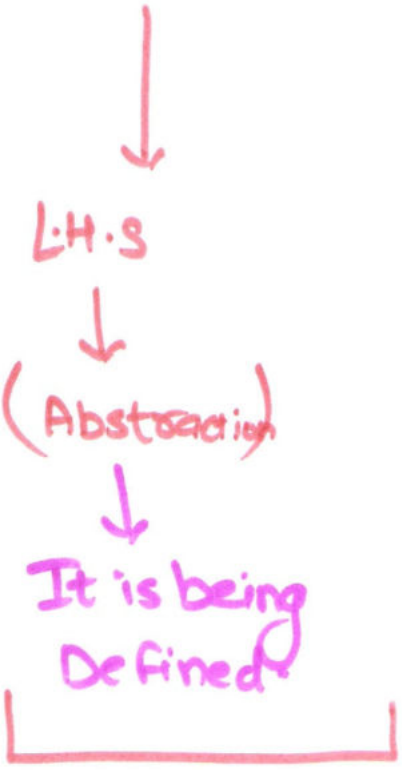
< > means 'can be described as'

| means 'Or'

BNF uses "abstractions" for Syntatic Structures.

For Eg:- Java assignment Statement

<assign> ⟶ <var> = <expression>



Terminals: Symbols that can appear in the Output of a language because of its rules, but cannot be changed by rules themselves.

Non-Terminals: Syntatic entities that defines a part of the grammar.

Non-Terminals can be **replaced** and they are string, Composition of Terminals and Non-Terminals

Eg. of Terminals

+ = { } * 1

if while Do , int

double float A , B

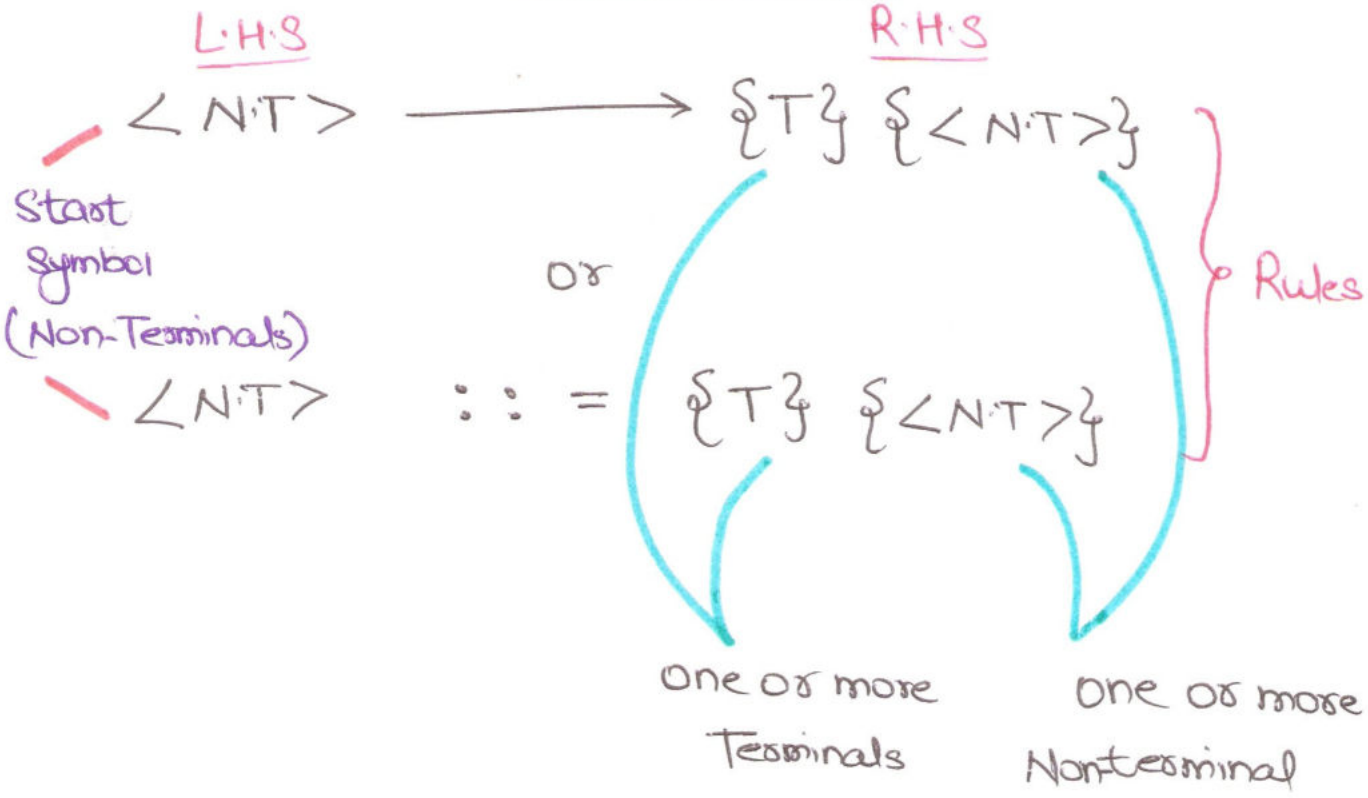
Eg. of Non-Terminals:

Sentences Expressions List of Numbers.

words statements

terms programs

How To write Rules in BNF



Example : if Loop

Structure ①

```
if ( condition )
{
   body
}
```

BNF of Structure ①

<if_loop> ::= <if_part>

<if_part> ::= if (<condition>) {<body>}

or

<if_loop> ::= if (<condition>) {<body>}

Structure ②

```
if (condition)
{  body
}
else
{
   body
}
```


BNF of Structure ②

① <if_loop> ::= <if_Part> <else_Part>

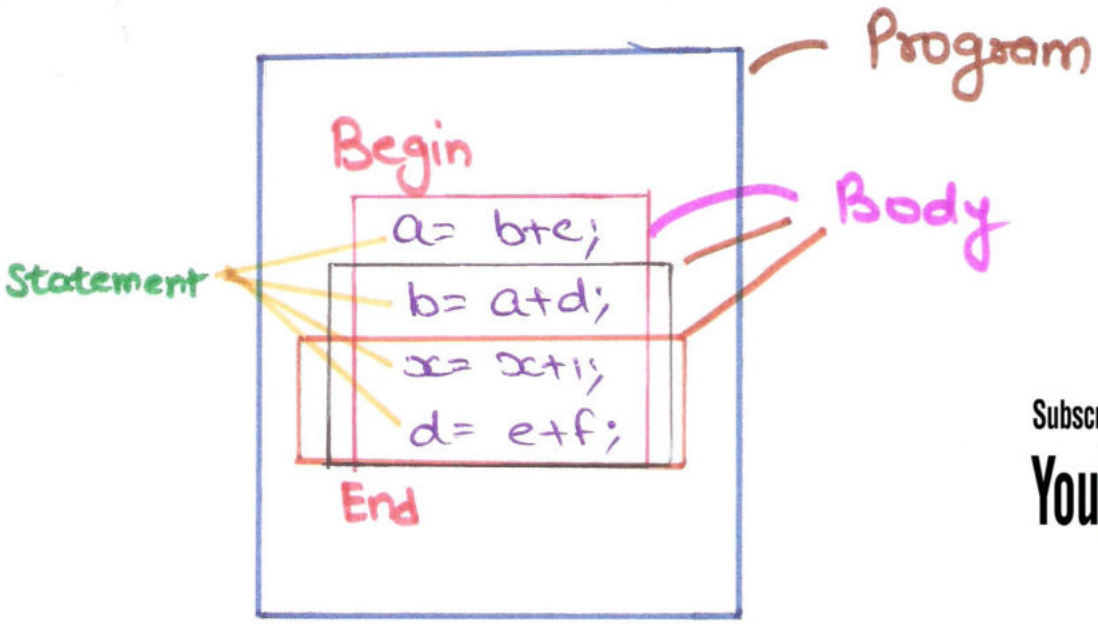
<if_Part> ::= if (<condition>) {<body>}

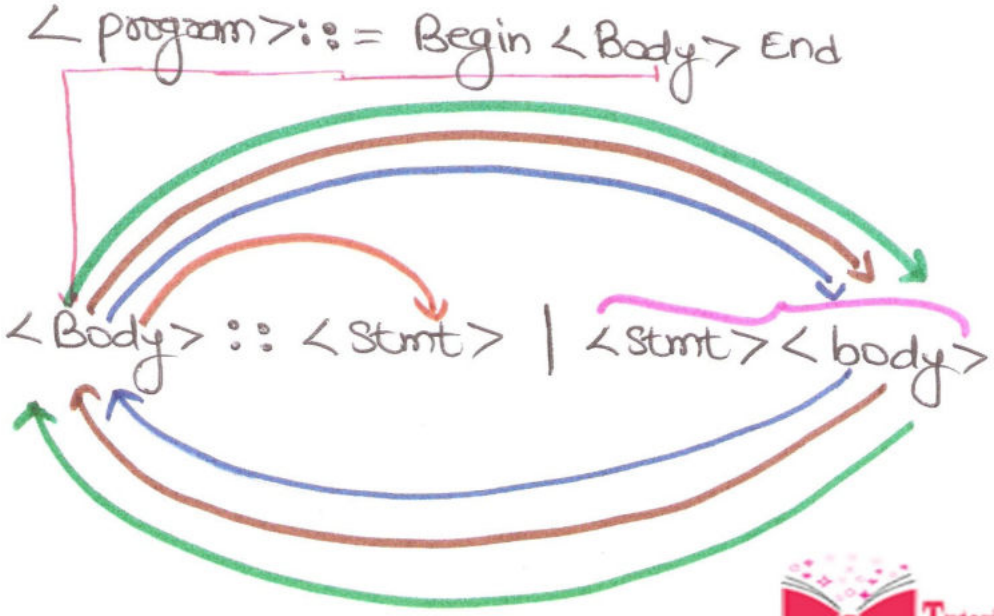
<else_Part> ::= else {<body>}

or

② <if_loop> ::= if (<condition>) {<body>} else {<body>}

Example : Recursion in BNF





Grammars and Derivations

→A Grammar is generative Device for defining languages.

→ The Sentences of the language are generated through a Sequence of applications of the rules, beginning with a Special nonterminal of the grammar called Start Symbol.

→ This Sequence of rule applications is called a Derivation.

<program> → begin <stmt_list> end

<stmt_list> → <stmt> | <stmt>; <stmt_list>

<stmt> → <var> = <expression>

<var> → A | B | C

<expression> → <var> + <var> | <var> - <var> | <var>

Subscribe to our YouTube Channel

Derivation :

<program> ⇒ begin <stmt_list> end

⇒ begin <stmt> ; <stmt_list> end

⇒ begin <var> = <expression> ; <stmt_list> end

⇒ begin A = <expression>; <stmt_list> end

⇒ begin A = <var>+<var>; <stmt_list> end

⇒ begin A = B + <var>; <stmt_list> end

⇒ begin A = B + C; <stmt_list> end

⇒ begin A = B+C; <stmt> end

⇒ begin A = B+C; <var> = <expression> end

⇒ begin A = B+C; B = <expression> end

⇒ begin A = B+C; B = <var> end

⇒ begin A = B+C; B = C end.

The symbol ⇒ is read "derives".

Each of the strings in the derivation, including <program> is called a Sentential form.

Example of another Grammar

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <id> + <expr> |
<id> * <expr> |
(<expr>) |
<id>

A = B * (A+C)

<assign> ⇒ <id> = <expr>

⇒ A = <expr>

⇒ A = <id> * <expr>

⇒ A = B * (<expr>)

⇒ A = B * (<expr>)

⇒ A = B * (<id> + <expr>)

⇒ A = B * (A + <expr>)

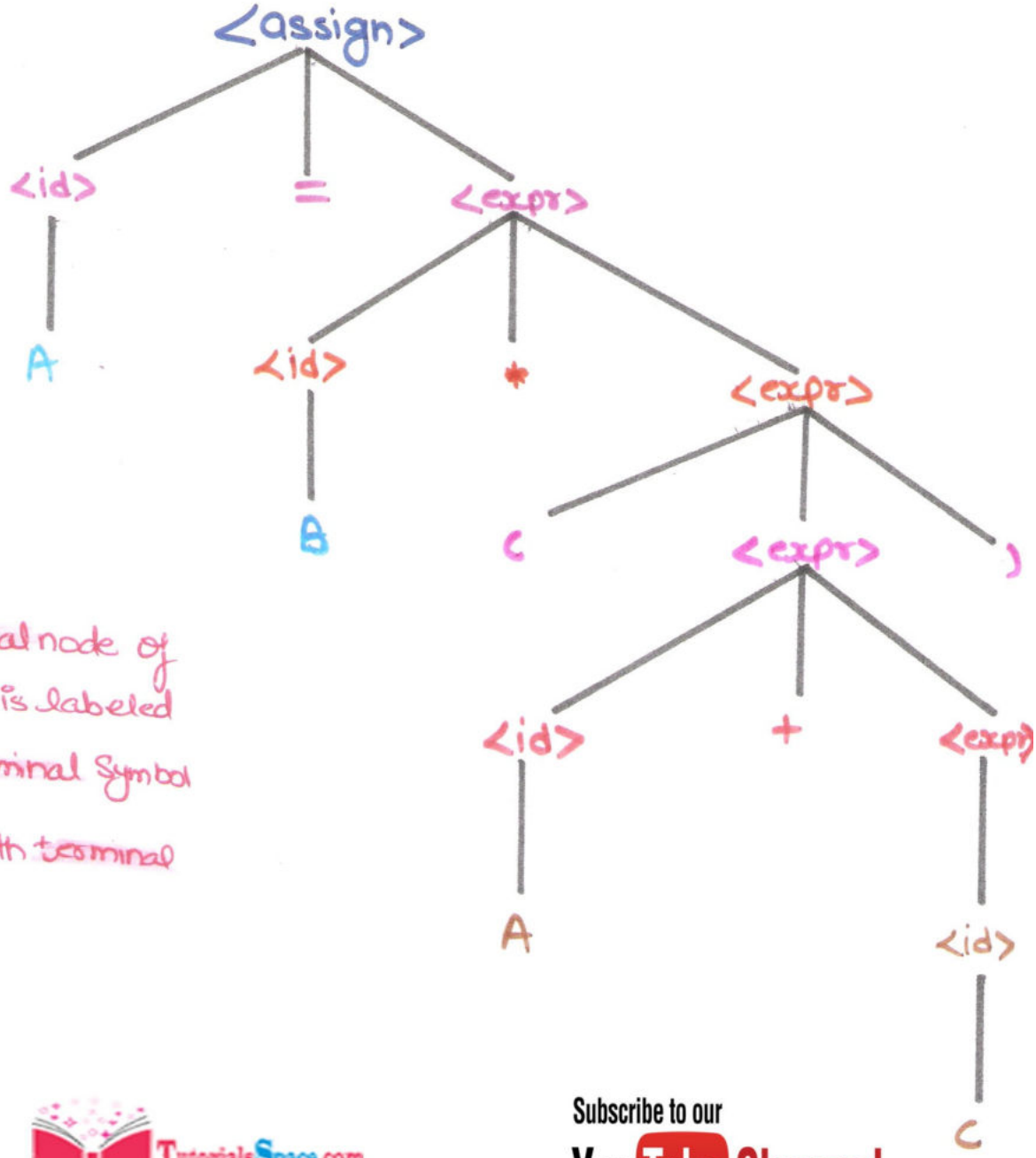
⇒ A = B * (A + <id>)

⇒ A = B * (A + C)

ParseTree:-

The Hierarchical Syntatic Structure of Sentences of the Languages is called ParseTree

A=B*(A+C)



Every internal node of a parse tree is labeled with non-terminal Symbol

every leaf with terminal

Ambiguity

A grammar that generates a sentential form for which there are two or more distinct parse trees is said to be **ambiguous.**

<assign> → <id> = <expr>

<id> → A | B | C

<expr> → <expr> + <expr>

| <expr> * <expr>

| (<expr>)

| <id>

An Ambiguous Grammer.

A = B + C * A it has two distinct Parse Tree