Unit → 1

Date. 23/08/18
Page No. 1

Introduction

Introduction to programming language

language can be defined as mean of communication. language is of two type :-
(1) Natural language (2) Programming language

Natural language is use for human interaction like English, Hindi, etc while programming language are used computer purpose.
A programming language can be defined as :-

(1) A logical communication b/w user and computer.
(2) Description of algorithm and data structure
(3) A set of instruction with grammer.
(4) An artificial language that can be used to control the behaviour of computer.

Requirement & objective of the language to be characterized as the programming language.

(1) Function → A programming language is the language use to write computer program, which instruct a computer

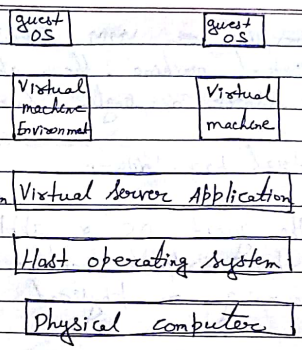
to perform some kind of computation & organize the flow of control between external devices.

(2) Target → Programming language differ from natural languages in a way that natural language are only used for interaction b/w people, while programming language allow humans to communicate instruction to machine. In some cases programming language are used by one program or machine to program another.

(3) Construct → It is a generic term that Normally refers to some particular syntax included in the language to perform some task (like a loop with the condition at the end).

(@) Virtual machine architecture
(@) Software Simulation Computer

* Architecture of Virtual machine



(1) A Virtual machine is an emulation of a Computer System.
(2) VM are based on Computer architecture and provide functionality of physical Computer.
(3) There implementation may envolve specialized hardware, software or a Combination.
(4) Starting from the bottom of the architecture is the host OS that manages the physical computer.

VMM/Hypervision

- VMM stands for Virtual machine manager.
- VMM or hypervision is a s/w that allows us to run one OS within OS.

- A hypervisor or VMM creates and runs virtual machine, they allow multiple OS to share a single hardware host.
Ex → Virtual box, VMware

- Finally the guest OS & application run on the virtual machine as if they were running on physical hardware.

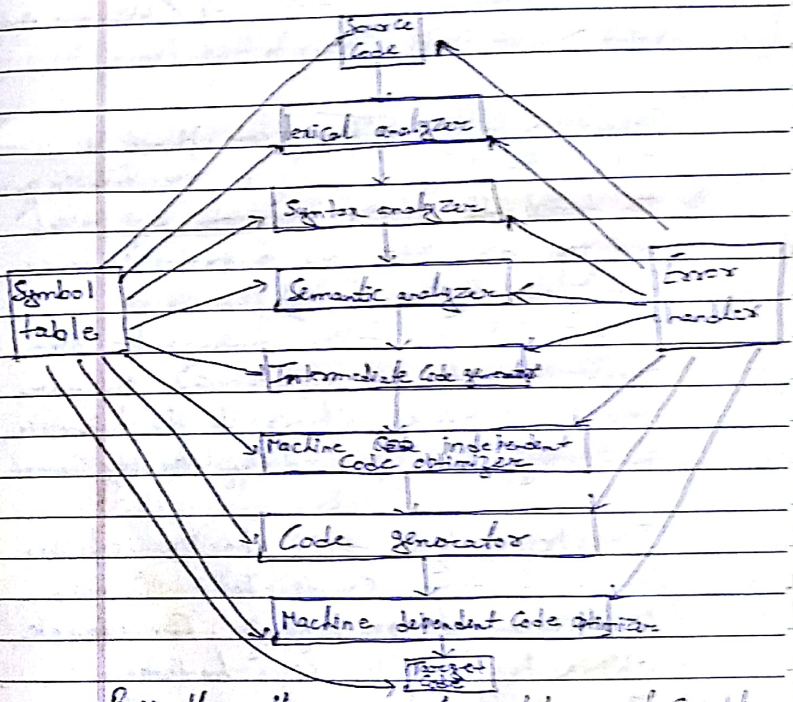
When a guest OS is running, the special purpose VM kernel manages the CPU and hardware during VM operations, creating an isolated environment in which the guest OS and applications run closed to the hardware at the highest possible performance.

Topic 1

(1) Characteristics of programming language
(2) Factors Influencing the evolution of programming language.
(3) Development in programming methodology
(4) Desirable features & designed issue.
(5) Programming language processors.
(6) Structure and operations of translation.
(7) Software Simulated Computer.
(8) Syntax & Symantics
(9) Virtual computers
(10) Binding and binding times.

* Software Simulated Computer

* Compiler phases



Basically there are two phases of Compilation
1. Analysis phase
2. Synthesis phase

Lexical Analysis → The first phase of Compiler works as a text Scanner

representation of the intermediate code and Converts this code into respective register numbers or name which are available in particular architecture depending on the target machine. Code generator translates the intermediate code into a sequence of relocatable machine code.

* Characteristics of programming language

(1) clarity, Simplicity & unity
A programming language should be very clear & simple so that user can understand language easily. unity also led to understandability. It should provide a clear, simple and unify set of concept that can be used as primitives in developing algorithm. Syntax of language should be closed to natural language.

(2) Orthogonality → The term orthogonality refers to the attributes of being able to combine various features of a language in all possible combination, with every combinations being meaningful. for example → Suppose a language provides

an expression to get a true or false.

(3) A language needs a syntax that allow the program structure to reflect the underline logical structures of a program. It should be possible to translate such a program design directly to appropriate program statement that reflect the structure of the algorithm. All the different algorithm are having different structure that are representing by the program of a language that are exactly representing by the program of a language.

(4) Support for abstraction
Abstraction hides the complexity. It gives only essential information, detailed working of an entity is not required.

(5) Program verification
check the correctness

(6) Programming Environment

(7) Portability of program

(8) Cost of use → (i) Cost of program translation
(ii) Cost of program execution
(iii) Cost of program creation, testing etc.
(iv) Cost of program maintenance

Date.
Page No. 12

(iii) Dynamic arrays
(iv) Block structure and local variables.

Late 60's

(1) SIMULA 67
* Design for Simulation application.
* Introduced some features
(i) classes for data abstraction
(ii) Coroutines for Re-entrant subprogram

(2) ALGOL 68
* Emphasised orthogonality and user defined data type.
* Not widely used.

Date.
Page No. 13

(4) Late 70's

(1) PASCAL
• It is developed by Nicklaus wirth.
• It is simplicity and emphasis of good programming style.

(2) C
• It was developed by Dennis Ritche.
• Had a great deal of flexibility.
• void pointers
• Incomplete type checking

(3) Prolog
• logic programming.
• Mostly used in AI.

* Early 80's

(1) Ada
• Specially for embedded systems.
• It contains some important features.
→Data Encapsulation with package.
→Exception handling.
→Most Early Compilers were slow and error prone.

(2) SmallTalK
* It was designed & developed by Alan Kay.
* First true object oriented language.
* Always dynamically bound.

- All classes are subclasses of object.
* It also includes software developed environment

(3) C++

- It was developed by Bjarne Stroustrup as an extension to C.
- Had a object oriented features and some additional features to improve C.
- Very powerful and flexible language.

(4) Perl

- It was developed by Larry wall.
- Takes features from C as well as Scripting language.
- Some features included
  → Regular expression handling.
  → Associative arrays.
  → Implicit data ~~typing~~ library.

late 90's

(1) Java

* It was developed by at SUN microsystem by James Gosling.
* Syntax borrows heavily from C++.
* But many features of C++ have been eliminated.
- No explicit pointers or pointer arithmetic.
- Array bound checking.

* Development in programming methodologies
It has been involved in the following ways :—

(1) Imperative language :
This paradigm of language believes on statement wise execution of the procedure.
Statement 1, ——
Statement 2, ——

In this methodology we just give instruction to the computer for the execution of statement.
for example : C, C++, Pascal, etc.

(2) Application based language
This language is also called functional language because execution of this language is function wise.
function 1
function 2
for e.g, C++, SIMULA, LISP, etc.

(3) Rule based language
This language is based on condition evaluation like if condition 1 ; expression 1
if condition 2 ; expression 2

(4) Structured Programming language
This paradigm was developed to reduce the use of GOTO statement. The main aim for the development of this paradigm is
→ Programming should be error free.
→ Programming should be easy to understand.
→ C is a structure programming language because we can use sequence, selection & iteration type of structure.

(5) Modular programming
Software programe are developed by smaller module having logical communication so that we can easily Implement abstraction in the programming modular paradigm increases the flexibility and maintainability.
for ex → C++, Java.

(6) Object oriented paradigm
In this paradigm it is based on object & classes for example

* Binding
(I) Binding is an association b/w an entity & an attribute such as between a variable and its type are values are between an operation are symbol.
(II) Binding time is the time at which a binding is created (takes place).
(3) Scope of a binding is the region of a program are time Interval in program execution during binding is active.

* Classes of Binding
* Compile time → It refers to the time duration which the statements are return in any programming language are check for errors. Compile time can also refers to the amount of time required for Compilation. It is divided in following sub categories :-
(1) Binding choosen by the programmer.
Ex → Variable names, variable type, program statement structure.
(2) Choose by the translater.
Ex location of data objects.
(3) Choose by the Linker.
related location of different object modules.

* Execution time → Many binding are performed during program execution. These includes binding of variables to their values, as well as binding of variables to a particular storage location.
Two important sub categories may be defined as :- on entry to a subprogram

(1) On entry to a subprogram or block
(2) In most languages binding are restricted to occur only at the time of entry to the subprogram or block during execution time.
For ex → In C & C++ the binding of formal to actual parameters & binding of formal to particular storage locations may occur only on entry to a subprogram.

(2) At arbitary point during execution
Some binding may occurs at any point during execution of a program. For example basic binding to variable to values through assignments.

* Translation time → Three different classes of translation time type

(3) Language Implementation type → The language this binding is done when the Compiler or Interpreter is return.
A program return is the language that uses some special features whose definitions has been fixed at language implementation type.
This binding is done when the Compiler or Interpreter is return.

(4) Language definition type → Most of the structure of programming language is fixed at the time, the language is defined.
Following binding are done at language definition type :-
(1) Alternative statement forms.
(2) Data Structure type.
(3) Array storage layouts.
Consider the simple assignment statement X = X+10 return in a language.
(1) Set of type for variable X
Variable X as a data type associated with it, such as real, integer or boolean. The set of allowable types for X is often fixed at

language definition type. For ex- real or Integer.

(2) Type of Variable X
The particular data type associated with Variable X is often fixed at translation time through an explict declaration in the program such as float X, which is the C designation for real data type.

(3) Set of possible values for Variable X
If X has data type real then its value of any point during execution is one, of a set of bit sequences real numbers.

(4) Value of Variable X
At any point during program execution, a particular value is bound to variable X. This value is determined at execution time through assignment of a value to X.

(5) Representation of the Constant 10
The Integer 10 has both representation as a Constant in the text of the program using string 10 and a

representation at execution time. Commonly, as a sequence of bits. The Choice of decimal representation in the program i.e, using 10 is usually made at language termination time whereas the choice of a particular sequence of bits to represent 10 at execution time is usually make at language Implementation time.

(6) Properties of the Operator
The choice of symbol to represent the addition operation is made at language definition type.

* Types of Binding
It is of two types :-
(1) Static binding
(2) Dynamic binding
(1) In this type associated with the variable is determined at Compile time.
(2) The binding is performed at run time is known as dynamic binding. In this type associative with variable is determined at run time. A single variable put at many different types at different point in the program.

Date.
Page No. 22

Advantages

(1) More flexibility in programming. Can you same Variables for different type.
(2) Can make operation generally.

Disadvantage

(1) It is difficult to compiler for checking error.
(2) Type checking is limited & must be run at run time.

Assignment-01 Date :- 17/08/18

(1) Explain the various characteristic of programming language.
(2) Discuss the various design use of programming language.
(3) What do you mean by Virtual Computer? Also explain System Virtual machine & process Virtual machine.
(4) Explain binding & binding time in details.
(5) Calculate the binding time for the following expression :-
X = 2 * X + 10 * Y
(6) Explain Evolution of programming language
(7) Write short notes on (I) Software Simulation Computer (II) Syntax (III) Semantics

Unit → 2

Date. 11/09/18
Page No. 23

Data object act as a container which can store any data value.
1. Data object can be created as data variable only when they are name.
2. Data object is characterige by data type
3. It can be programmable defined as system defined.

Elementary data object
It contains a single data value like Char, float, etc.

Data structure or structure data
Structure data types contain aggregate of other data object. Ex → Array, etc

```
Int N;
N = 27
```

Data type → It is a class of data objects together with a set of operations for creating & manipulating them. Every language has a set of primitive data types that are built into the language. Ex → Integer, char, float.
Data types are classified by three things :-
(1) Attributes (2) Value (3) Operation

(1) Attributes are the particular data type for any data object. Eg → int, char, float, boolean are the attributes for a particular data object of the define data type.

(2) Value → Attributes of every data type are associated with particulare value which can be assigned to them at runtime or at Compile time.

(3) Operations are the entities, with the help of which we can do the actual Implementation. It may be in form of some inbuilt function or user defines function. It is also possible through assignment & various mathematical operation.

Implementation of data types

The following are basic elements of the ~~dat~~ Implementation of data types :-

(1) Storage Implementation

It is used to represent the data objects of the data types. In the storage of the computer during program execution.

(2) Algorithms or procedures → That manipulate

the storage representation of data object for the define operations

(3) Syntactic representation → Attributes of data objects are represented by declaration type or type definition, values by Internal & operation by using built in functions or user define function.

Elementary data type

A class of data object which content single value is term as elementary data type. It is specified by Attribute, values & operation.

Implementation of elementary data type

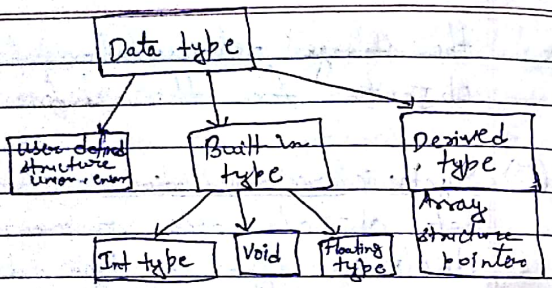
(1) Storage representation

(2) operations → Each operation defined for data objects of the given type can be implemented in one of the three ways.

(i) Directly as hardware operation.

(ii) As a procedure function.

(iii) As an inline code sequence. Ex → Square root



* Structured data types
A data object that is constructed as an aggregate of other data objects, called Components. A structure data object or data structure.
A Components may be elementary or it may be another data structure. For example ~~array~~ A Component of array may be a number or it may be a record, or character string or another array.

Specification of data types
The major attributes for specify data structure includes the following :-

(1) Number of Components
- Fixed (Compile time) Ex → Array
- Variable (Runtime) Ex → LinkedList

A data structure may be fixed size

if the no. of Components is ~~in~~ Variant during its life time or Variable Size if the number of Components changes dynamically. Variable Size data structure type usually define operation that Insert and delete Component from structures.

* Array and records are Common examples of fixed size data structure type, Stack or LinkedList, file. Example of fixed DST ~~and~~ Stack, LinkedList, set of tables are Variable size.

* Variable size dS often use a pointer data types that allows fixed size data object to be linked together explicitly by the programmer.

(2) Types of each Components
(1) Homogeneous → In this data type of each Component is same.
Ex or → array, sets, file, etc.

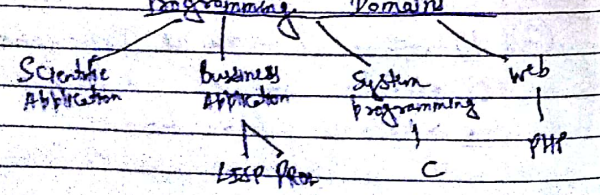
(2) Hetrogeneous → It is hetrogeneous if its Components are of different types.
Ex → Records Structure.

(3) Name used for accessing the Components
A data structure type needs a selection mechanism for Identifying individual Component of DS. For an array, the name of the integer Component, may be integer subscript or sequence subscript.
Some data structure types such as stack & files allow access to only a particular Component (Ex → top Component or Current Component) at any time but operations are provided to Change the Component i.e, Currently accessible.

* History of Programming language
In 1957 the first of the major language appeared in the form of FORTRAN

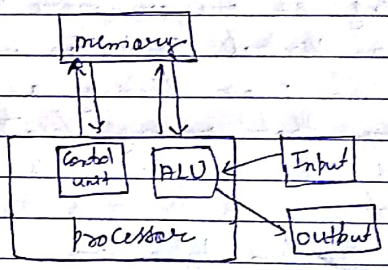
(4) Explain the language Evaluation Criteria & Characteristics that affect them.

Programming Domains



Characteristic	Readability	writability	Reliability
Simplicity	✓	✓	✓
Orthogonality	✓	✓	✓
Data types	✓	✓	✓
Syntax design	✓	✓	✓
Support for abstraction		✓	✓
Type checking			✓
Exception Handling			✓

* Von Neumann Architecture



A data processing unit for the forming of arithmetic & logical operation & Control unit.
Basic operations are LOAD to read a values from memory location.
STORE → write a value to the memory location.

Date.
Page No. 30

Elementary data Types
→ Data objects
→ Data values
→ Variables
→ Constant
→ Data Type
→ Declaration
→ Type checking

* Data objects → It refers to run time grouping of one or more pieces of data in a Virtual Computers.
Ex → If I am storing a variable x in memory that memory location has some address. This variable x is called data object.
It is of two type :-
(1) Programmer defined (2) System defined.

(1) The data objects that exist during program execution. eg :- Variable, Constant, arrays.

(2) They are generated without explict specification by the programmers.
Ex → file buffers.

Date.
Page No. 31

* Data values → It might be a single number, character or a pointer to another object. It is represented by a particular pattern of bits.
Eg :- A [1001] → data values.
data object
int A = 17 ;

* Variables → we define the variable x which is of int type. It acquire some byte in memory.
The name of this memory location is x, which is a variable.
int x ;

A data object that is defined and named by the programmer explicitly in a program is called a Variable.

* Constant → Throughout lifetime of the program the value of a Variable MAX is 30. So it is a Constant.

* Data type → It is a class of data object together with a set of operations for Creating & manipulating them.
eg :- Float a, b, c ;

modulo operation cant be performed on float data type.

eg :- Array int a[10] ;

The basic elements of specification of a data type are as follows :-

(1) Attributes - Size, Dimension

(2) Values - The valid values that array will hold.

(3) Operations - Arithmetic, Sort, Search.

Eg → Array Int a[10] ;

* Declaration →
  - Implicit → No user role. The Compiler automatically declares it. Eg:- in Perl Lang. $a = "Hello" ;
  - Explicit → It is explicitly declarated by the programmer.
    Int a = 10 ;

* Type checking

eg :- In C language, int a = 10 ;

Type checking is done at the Compile time Called as Static type checking.

eg 2 :- In Perl language

$a := "hello" ;

Type checking may be done at runtime Called as dynamic type checking.

* Type Conversion

To Convert one data type to another.

e.g :- x + y
int float

Higher byte storage is of float so int is converted to float after that addition will be performed.

① Implict type Conversion

```
Void main()
{
int a = 10; b = 3 ;
float c ;
c = a/b ;
```

② Explicit Type Conversion

```
Void main()
{
int a = 17, b = 5 ;
float c ;
(float) a/b ;
```

float int

* Elementry data types

A data object is elementry if it Contains the data value i.e, a single data value. A data object participates in various bindings during its life time

The most important attributes & binding are:-

(1) Type → This associate the data object with the set of data value set the object can have.

(2) Location → The binding of to a storage

The binding of a data object to one or more data object of which it is a component is represented by a linear value.

Specification of EDT

Some EDTs are integer, character. Their specification (i) Attributes → Basic attribute of any object such as data size & data object which are usually not changed during its lifetime.

The data type of data object determing the

set of possible values that it may combine.

+ : integer x integer → integer
SQRT : real → real
A square root operation on real number done object is specified.

* Scalar data types
Numeric data types < Int, Float
Enumeration
Boolean
Characters

Scalar data types are the types that have a single attribute for its data object.

(1) Numeric Data Types

Some form of numeric data is found in almost every programming language.

(1) Integer → C has 4 different type Integer specification Int, Short, long & Char. Operations on Integer data types include arithmetic operation.

Binop : Integer x Integer → Integer
when Binop may be addition, substraction multiplication, division. Unary operations

have the signature Unary OP : Integer → integer
Unary operations may be negation, or to find absolute value.

Relational operations
RelOP : integer × integer → Boolean
where RelOP may be equal, not equal, less than equal to etc.

Assignment
Assignment btw integer data objects may be specified as
assignment : integer × integer → void

Floating point number
A floating point real no. data type is often specified as only the single data type attribute. real as a float char & float as a C.
Although most hardwares include integer & floatingpoint data object.
There are many application :

written to n decimal places. These don't be written as integers and if written as floating point may have round off errors. A form of fixed point data values to represent such values. A

fixed point numbers represent as a digit sequence of fixed length decimal point placed at fixed between 2 digits.

(2) Enumeration → An enumeration is an ordered list of distinct value.
eg :- #include<stdio.h>
enum week{sun, mon, tue, wed, thu, fri, sat};
void main()
{
enum week day;
day = week;
printf("%d", day);
}

(3) Boolean
True - 1
false - 0
eg → void main()
{
int x1=10; x2=20; x3=30; bool b1, b2;
b1 = x1 = x2;
b2 = x1 < x2;
Cout << b1 \n;
Cout << b2 \n;

(4) Character →

eg → void main()
{
char a,b,c ;
a = 'A' ;
b = 'B' ;
if (a < b)
Printf("a comes before B");

Composite data types
An object may have multiple data attribute.
Ex → A string contains a sequence of characters.
Array, string.

Array → (Homogeneous) similar type of data is stored.

int a[10] ;

| 1 | 2 | 3 | 4 | 5 |

data type — name of the array — size

String → It is a character type of array. All the element will be character.

Pointer → They are the variables which contain the address of another variable.

int a ;
int *p ;
p = &a ;

A file is a data structure with two specific properties :-
(1) It is stored on the secondary storage device.
(2) Its lifetime may have a greatest span time then that of the program.

A sequential file is the most common type of file but many languages also provide direct access file & indexed sequential file.

* Structured data type
These are the data type which has more than one data object. It is also called data structured. It is specified by the following attributes :-
(1) No. of Component that can will fix all variables. Ex → array, structure.
(2) Variable Type for each Component
(i) Homogeneous → In this data type of each Component is same. Ex → int a[10]
Structure :
struct {
int. _
float _ ;
(3) Name use for accessing the . → A data

Structure type is a selection mechanism for identifying individual component of a data structure.

(4) Upper limit of Component → Whenever we use Static data type then we have to specify the upper limit of Component.

(5) Organization of Component → The most common organization is the simple linear sequence of Component. vector, array, records, stack, list are data structures with this organization. Array & records can be multi language also.

* Implementation of structured data type

(1) Vector or one dimensional Array → A vector is also termed as one dimensional array.

Array
int a[3]

	a[0]
	a[1]
	a[2]

The attributes of a vector are :-
1. No. of Components fixed.
2. Data type of each Component.
3. Subscript to be used.

V: a[-3, -2, -1, 0, 1, 2, 3]

(Q.) Given an array of a 100 Component subscript ranging from 0 to 99. Find the address of 49 location. Base address is 1000.

Sol:- $\alpha + I \times E$ → Size of data type (int - 2, float - 4)

↓ Base address. ↓ Component whose address is to be found.

$1000 + 49 \times 2 = 1098$

(Q.) For the array V: array of [-3, ..., 5] of Int find the address of V[3] location given base address is 1000.

Sol:- $\alpha + (I - LB) * E$

$1000 + (3 + 3) * 2$

$1000 + 10 = 1010$

(Q) In C float B[10][10], Address calculation for 2-D array.

$\alpha + (I \times n + J) * E$

↓ base address ↓ no. of rows whose address have Calculated ↓ no. of Columns ↓ Size of data types

Date.
Page No. 42

(Q.) Given an array size of a[100][107] and a[49][83] if the base address is 100 and size of data type is 2 byte.

Sol":- 100 + (49 × 107 + 83) * 2
= 100 + 10652
= 10752

* Co-routines

→ They are program component that generalise sub-routine to allow multiple entry points for suspending and resuming execution at certain locations.

→ Co-routines are execution content that exist Concurrently, but that execute one at a time and that transfer the control to each other explicitly, by name.

→ Co-routines can be used to implement such as iterators, threads, Cooperative task,

→ Because they are Concurrent (i.e., Simultaneously started but not completed). Coroutine can't share a single stack.

Statement binding analysis
A = X + 10
(1) Set of type of Variable (2) Type of Variable X
(3) Set of possible (4) Variable, (5) Representable
(6) Properties of the

Unit → 3

Date. 16/10/2018
Page No. 43

Data Control referencing Environment

The referencing Environment (statement or expression) is the set of active bindings or, the referencing Environment is the collection of all the names that are visible in the statement.
Or,
Referencing environment correspond to a collection of scope that are examine to find a binding.

Local referencing Environment

The set of association created or emptied to a sub program that represents formal parameters, local variables and sub program define only within that sub program forms the local referencing environment of that activation of the sub program.

- local variables define inside the co routine.
- Formal parameters are usually local variables
- local variable storage.

(a) Static Scope

- Allow direct access
- Provides for history sensitive sub routines.

(b) Static-dynamic

- Provides memory ~~saving~~ saving.
- Allow recursion.

Non local Referencing Environment

A set of associations for identifiers that may be used within a subprogram but that are not created on entry to it is termed as non local referencing environment of the sub program.

Non local Variables are either global or are variables that are in scope but local to another subroutine.

Global Referencing environment

If the associate created at the start of execution of the main program, then these associations form the global referencing environment of that subprogram.

Some identifiers have a predefined association i.e, defined directly in the language definition.

Scope

Scope is the textual region of a program

in which a binding is active. The word scope used as a noun is a program section of maximal size in which no binding change.

or,

The scope of a variable is the range of statement over which it is visible.

- The non local variables of a program unit are those that are visible but not declared there.
- The scope rules of a language determine how references to names are associated with variables.
- ALGOL 68 introduced the term elaboration for the process of creating bindings when entering a scope.

Scope rules

Programming languages implemented two types of scoping rule. ~~First one is static scoping~~

(1) Static scoping → In this active bindings are determined using the text of the program. The determination of scope can we make by the compiler.

(1) Most recent Scan of the program.

(2) Closed nested subroutine rule.
(3) Most Compile languages Employee static scope rules.

(2) Dynamic scoping
In this Active bindings are determine by the flow of execution at run time.
(1) dynamic scope rules are usually encounter in interpreted language.

In this the Compiler can determine when binding will be created & destroy. It is
• based on program text.
• To Connect a name reference to a Variable (by Compiler) must find the declaration.
• Search process :- Search declaration, first locally than in increasingly larger & closing scope until one is found for the given name.
• Enclosing static scope (to a specific scope) are called its static ancestor. The nearest static an ancestor is called a static parent.
• Variables can be hidden from a unit by having a "closer" Variable with the same name.

• C++ and Ada also exist to these (hidden Variables).
→ In Ada : unit.name
→ In C++ : class.name

Blocks → The classical example of static scope rules is the most closely nested rules used in block structure language.
e.g → In C & C++

```
for(———)
{
int index;
———
———
}
```

Dynamic Scope → Binding Can't always be resolved by examine the program because they are dependent on calling sequences.
Dynamic scope is :-
• based on calling sequence of program unit, not there textuale layout.
• Reference to variables are connected to declaration by searching back through the chain of subprogram called that forced execution to this point. Dynamic scope rules are usually encounter in interpreted languages.

- Such languages do not normally have type checking at compile time because type determination is not always possible when dynamic scope rules are in effect.
- Often dynamic A binding b/w names & object depend on the flow of control at run time.

```
MAIN
declaration of x
SUB 1
  --- declaration of x
  ---------
Call SUB2
  - -- --
SUB2
  ---
reference to x ---
  ---------
  ----------.
  o
```

```
Call SUB1
  ---------
MAIN Call SUB1
SUB1 Call SUB2
SUB2 use x
```

There are two approaches for accessing variable with dynamic scope.
(1) Keep a stack (2) Keep a central table

Value need to find a variable hunt down from TOP of stack.
This is equilent to searching activation record on the dynamic change. If name can't be created on runtime, table layout (the location of every slot) can be fixed at runtime. Otherwise you will need a hash function.
Every subroutine changes the table entry for its local at entry and the exist.

Parameter Passing mechanism

1. Pass by value
2. Pass by Reference
3. Pass by name
4. Pass by value-Result
5. Pass by result

In pass by value ~~reference~~ mechanism copies of the argument are created and which are stored in the temporary location of the memory.
→ The parameters are mapped to the copies of the arguments are created.
→ The change made to the parameter do not affect the arguments. Pass by value mechanism provides security to the calling program.

Example :-

```
#include<iostream>
int add(int n); /*Declaration of the function*/
int main()
{
  int number, result;
  number = 5;
cout<< "The initial value of number "<<number<<endl;
result = add(number);
cout<< "The final value of the number "<<number<<endl;
cout<< "The result is "<<result<<endl;
  return 0;
}
int add(int number)
{
  number = number + 100;
    return(number);
}
```

* Pass by reference is the second way of passing parameters to the function in this
→ The address of the argument is copied in to the parameters.
→ The changes made to the parameter effect the argument.
→ The address of the argument is passed to the function and function modifies

the values of the argument in the calling function.

* Call by name → In call by name the argument of function are not evaluating to all rather, function arguments are substituted directly into the function body using capture-avoiding substitution. If the argument is not used in the evaluation of the function, it is never evaluated; if the argument is used in several times, it is revaluated each time.

* Pass by value Result
→ The value of the arguments are copied into the formal parameters
→ The final values of the parameter are copies back out to the arguments.

```
void square(int x, int y)
{
  x* = x;
  y* = y;
}
int main()
{
```

```
int a=5;
int b=10;
Square(a,b);
Printf("a= %d, b= %d \n", a, b);
}
```

* Storage Management

Each & every programming language requires the use of particular storage techniques and the mechanism was storage management and there representation in the hardware & S/w.

Storage management is of two types :-

(1) Programming Controlled storage
(2) System " "

(1) Programming Controlled Storage → In C language storage management can be explicitly done by the programmer with the help of malloc & free, which allocates free storage for use define data structure.

Advantage

(a) Programmer can easily know when a particular OS is requiring storage or not but it is quit difficult for the system to know when a particular data elements is requiring storage or

Disadvantage

(a) Overburden on the programmer.
(b) PCSM can sometime interface with system controlled storage management.
(c) Programmer control storage management is dangerous to the programmer because it can lead to error or lost of access to availble storage.
(d) Sometimes PCSM provides less efficient use of storage.

(2) System Controlled Storage

In many high level language storage management is done implicitly with the help of various language features. Programmer has no direct controller of storage allocation.

Advantages

Programmer is not overburden because storage Management is done by Implicitly. Following points are required in storage management :-

(a) Measure elements requiring storage
(b) Measure operation requiring storage to be allocated.
(c) Storage management phases.
(d) " " techniques.

Date.
Page No. 54

* Major elements requiring Storage
Almost all the programs, runtime data elements, operation require storage during program execution. The elements of the programming language requires storage are :-
(1) Code segments for translated user programs.
(2) System runtime program.
(i) library sub-routine
(ii) Storage management utility.
(3) User define data structure.
(4) Sub program return points.
(5) Referencing Environment.
(6) Temporaries in special Evaluation.
(7) Temporaries in parameter transmission.
(8) Input/output buffers.
(9) Miscellaneous system data.
(10) Apart from data & program elements, various operations requiring storage to be allocated or freed.
Following are the major operations :-
(1) Sub program Call & return operation.
(2) Data structure creation the destruction operations.
(3) Component Insertion & deletion operation.

Date.
Page No. 55

* Storage management phases
Every system management consist of various phases. These phases are :-
(1) Initial allocation → First at the start of execution, each storage block may be either allocated or free. If block is free initially it is available for allocation dynamically. Some storage management phases technique are required to keep track which block is free & which is allocated.

(2) Recovery → Storage that are subsequently freed, must be recovered by the storage manager for reuse. It can be done with the help of :-
(i) garbage collection
(ii) stack pointer

(3) Compaction & reuse → Free storage block sometimes might not be sufficient in size to be allocated, so it is necessary to construct large block of free storage from small free block. This process of combining the many small free block into a single block to be allocated is known as Compaction. Reuse of storage

involves ~~same~~ in a same mechanism as initial allocation.

* Storage Management techniques

many different storage management techniques are known for language implementation. Three most basic storage management techniques are :-

(1) Static storage management
(2) Stack-based storage management.
(3) Heap storage management
   (i) Fixed-size element
   (ii) Variable-size element.

(1) Static Storage Management

The is the simplest form of allocation which is done during translation and remain fixed throughout the execution. Storage for code segments of user and system program is allocated statically. As static allocation don't required any run time storage software so the recovery & reuse is not required.

In static allocation, objects given as absolute address that is retained throughout the execution of program.

Following points are in static storage management.

(1) Static allocation is done at compile time.
(2) When memory is allocated remain fixed and can't be changed during execution.
(3) There is no requirement of runtime storage management software.
(4) It is incompatible with recursive sub program calls and data structure whose size is dependent on input data.
(5) When run a program, the OS allocates a part of memory for programs.

Following The allocation area are dividing into 4 parts :-
(i) Code area (ii) data area (iii) Heap (iv) Stack

(i) Code area → The code area is fixed in nature and contains the executable code of the program contain of the memory area can't change.

(ii) Data area → Data area contains global variable, arrays, etc. In the

Initialize data area variables are initialized by their respective value.

(iii) Heap → Heap area is the memory area which can be used to allocate memory at run time. The nature of heap area is dynamic.

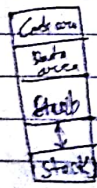
(iv) Stack → In stack area we call a function a stack frame for that function is generated in this area. This area is used by Compilers to store variable & function.

Advantages Of SSM

(1) It is efficient because no time or space is expended for storage management during the execution.

Disadvantages

It is incompatible with recursive sub program called and DS whose size is dependent on input data.



(2) Stack based Storage management

It is based on stack Data Structure that supports last in first out operation. It is needed when language permit recursion. It covers following points:-

(i) First allocated last feed

(ii)

(iii) Non applicable if storage needs to be allocated / freed at any point.

(iv) In static (last out)

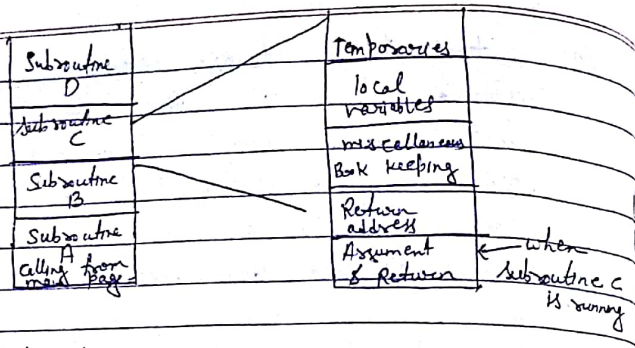
Ex → Recursive sub routine parameter. It could be useful in language without recursion because it could save space. Each sub routine invokation creates a frame as activation records.

(1) Arguments
(2) Return address
(3) local variables
(4) Book Keeping info

Stack maintain by
(1) Calling sequences
(2) Prolog
(3)

Advantages :-

Date.
Page No. 60



Heap storage

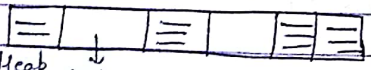
Heap is area of storage that is used for allocation for dynamic storage. The amount of dynamic storage that is req by an app depends on the data been process by program & produce that use a heap. The OS allows the use of multiple single store heap.

Heap storage management techniques

2 types :—

(1) Fixed size element
(2) Variable size element

It is also called memory pool. uses a free list block of memory. It is quite simple compaction is not the problem



Date.
Page No. 61

Variable size allocation

Because they are not allocated in the stack, the lifetime of object allocated in the heap is not confined in the subroutine where they are created.

(1) They can be assigned to parameter.
(2) They can be return as value of sub routine functions/methods.

In heap based allocation, heap/object may be allocated and deallocated at arbitary time. e.g :— objects created with C++. The heap is a region of storage in which sub block can be allocated & deallocated.

* Heap space management

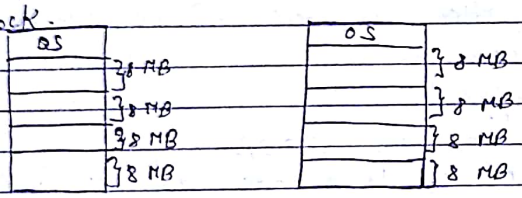
memory is allocated from a large pool of unused memory area called heap or the free store. Since the location of the allocation is not known in advance, the memory is accessed indirectly. Usually via reference. The algo used to organised the memory area at allocate & deallocated may use any of the method

(1) Fixed size block allocated.
(2) Variable size block.

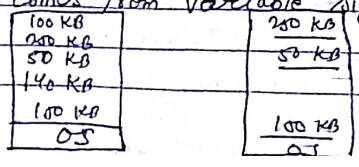
Date
Page No. 62

* Memory Fragmentation problem
The task of fulfilling allocation request is just to find unused storage block of sufficient size. In general the heap is allocated sequencly. This creates space fragmentation problem. Fragmentation problem is of two type :-
(1) Internal
(2) External

(1) Internal Fragmentation problem
If size of object to allocated is larger than the size of the available heap. It arise from fixed size heap block.



(2) External → If size of object to allocated is not larger than the size of the available heap, but the available space in the heap is scattered in such a way no contiguous shape fix.
It comes from variable size block.



Unit → 4

Date. 17/11/18
Page No. 63

Syntax & Translation

Syntax → It is the form of expression, statement & program units or syntax is a grammar of any programming language.
or, It is the way or writing the declaration or declaring the expression. It is the way of writing the statement literals & identifiers.

Semantics → It is the meaning of those expression, statement & program units, we can evaluate the expression, perform condition checking & other logical operator. Semantic is logic of any programming language. Semantic indicates meaning of the code what code will actually do.

General Syntatic Criteria
The primary purpose of syntax is to provide a notation for communication b/w programmer & programming language procedural. Following are the syntatic criteria on the basis of which we can identify good syntax.

(1) Readability → A program is called readable if the structure of the

algo & data representation by the program is apparent from an inception from the program text. Readability is enhanced by language features such as statement formats, program syntax, structured statement, use of keywords, unrestricted link identifiers, nemonics operation symbol, free field format & data declaration.

Features that increase the readability of code are of the following :-

(i) Comments
(ii) Named Constants
(iii) Clearly understood & Controlled Statement.
(iv) language orthogonality.

(2) Writability → The syntatic criteria that a program can easily be writable accomplish writability. It is inhanced by used of regular syntatic structure A detailed analysis of program is required to determine the complete control structure. In writability syntatic criteria, a syntax may be redundant, if it can communicate some informations in more than one way.

Features that help writability :-

(i) Clearly understood controlled statement
(ii) Sub program.
(iii) Orthogonality.

(3) Ease of Translation → It is easier goal that making program easy to translate into executable form.

(4) Ease of Verifiability → It is the simble concept of program correctness & program verification. Program correctness means simble that the syntax is used in program has syntatically correct and it should have proper meaning.

(5) Reliability → A program is said to reliable if it performs to its specification under all conditions. Features are type checking, readability, writability & exception handling.

There are two different ideas of reliability,

(i) Programs are less sutable to logic errors.
(ii) Programs have the ability of recover from exceptional situation. Ex → Exception handling.

- Syntactic elements
- The basic syntactic elements are :-

(1) Character set → There are several ASCII widely used ______ such as ASCII 64, each containing a different set of special characters in addition to the basic letters and digits.
Various bit encoding schemes are can be used.

5, 6, 7, 8 encoding schemes

Uppercase	A to Z
lower case	a to z
Digits	0 to 9
Special symbol	$, #, %

(2) Identifiers → It is a string of letters & digits usually beginning with a letter.
(a) Symbols such as letters, digits, dollar, _.
(b) .

(3) Operation Symbol → Most language use the special character : + & − to represent two basic arithmetic operation.

(4) Keywords → Keywords in an identifier used as a fixed part of the Syntax of

a statement. Ex →

(5) Reserved words → Keywords are that can not be assigned by programmer is reserved words most languages today used reserved word. because it can improve the error detecting capabilities of Adding new reserved word to a week of language can make old program incorrect.

(6) Comments → Comments are used for documentation and increase readability. They are not translated & executed.

(7) Blanks → Blanks or tabs are used in programming language. languages used blank as separator.
They are used for improving readability.

(8) Noise or special words → Special words are optional words are used in that are inserted in statements to improve readability.

(9) Delimeters and brackets → A delimeter is a Syntatic element used simply for mark beginning or end of statement.
Brackets are also delimeters but called "pair delimeters".

(10) Expression → Expressions are used to access data object in a program & return value. In expression operators are used to solve the expression. The type of operators are :- Arithmetic & logical

(11) Statement → Statements are syntactic components in languages. Statements are used for building block of programming. In simple statement o contains no other embedded statements.

Stages in translation

The process of Converting Source program into object code is known as translation.

(1) Analysis of Source Code :-
(a) Lexical analysis or Scanning.
(b) Syntax analysis or passing.
(c) Semantic analysis.
(d) Intermediate code generation

(2) Synthesis of object code
(i) Code optimization
(ii) Code Generation

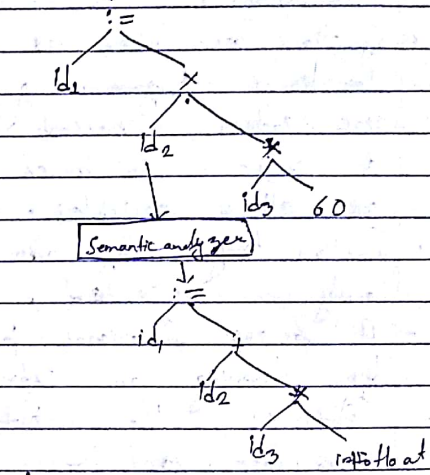
position = initial + rate * 60

source pr

LA

$id_1 = id_2 + id_3 * 60$

Syntax Analyzer



Semantic analyzer

IMC

$t_1 := \text{inttofloat}(60)$
$t_2 := id_3 * t_1$
$t_3 := id_2 + t_2$
$id_1 := t_3$

Code optimization

$t_1 = id_3 * 60$
$id_1 = id_2 + t_1$

Code Generation

Introduction to functional programming

- The functional programming must be confused with imperative programming. Neither it is the object oriented programming.
- Functional programming is all about expressions.
- Functions are used as a object in functional programming i.e, they are often passed around within a program in much the same way as other variables.
- Function programming also tends to be heavily less oriented.
- A pure functional program is structured by defining an expressions which captures the intent of the programmer. Each term of the expressions is in turn a statements of the characteristics of a problem may be encapsulated as an single expression & the evaluation of each of the term eventually.

* Properties of functional programming

(1) Function programming is based on expression rather than sequence of statements.

(2) Function program is highly abstract.
(3) The resultant code is shorter than imperative language.
(4) The resultant code is difficult to understand by layman.
(5) Code is highly reliable.
(6) Complex programming language.
(7) To access the expressions in functional programming, we have to make use of function. This programming is purely mathematical concept.
(8) Function programming focus on what rather than how.
(9) Functional programs are heavely list-oriented.

* Lambda Calculus

In mathematical logic and computer science lambda calculus also λ-calculus is a formal system designed to investigate function definition, function application & recursion. lambda calculus can be used to define what a computable function is. λ-calculus has greatly influenced functional programming languages such as lisp, ML. Following are the properties of λ-calculus :-

Date.
Page No. 72

(i) λ-calculus can be called the smallest universal programming languages. It consist of single transformation rule (variable substitution) and a single function definition scheme.

(ii) λ-calculus is universal in the sense that any computable function can be expressed and evaluated using this formalism.

(iii) λ-calculus imphisizes the use of transformation rules and doesn't care about the actual machine implementing them.

(iv) It is an approach related to software then to hardware.

More about λ-calculus

(1) In λ-calculus, every expressions stands for a function a single Input for its argument, the argument is in function with a term, with a single argument and the value of the function is another function with a single argument. Ex → two function such that $f(x) = x + 2$ could be expressed in λ-calculus.

(2) Function application is left associated.

Date.
Page No. 73

A function of two variable is expressed in λ-calculus as a function of one argument which returns a function as one argument

Ex → $f(x,y) = x - y$

while the λ calculus doesn't contain symbols for integers or additions, these can be defined as abbreviation, calculus and arithmetic can be expressed.

λ-calculus expression may contain three variables i.e, not bound Vs

* Data flow programming language

In Comp. programming PRP is main concept

Parallel processing

It parameters the data from main concept start with an .