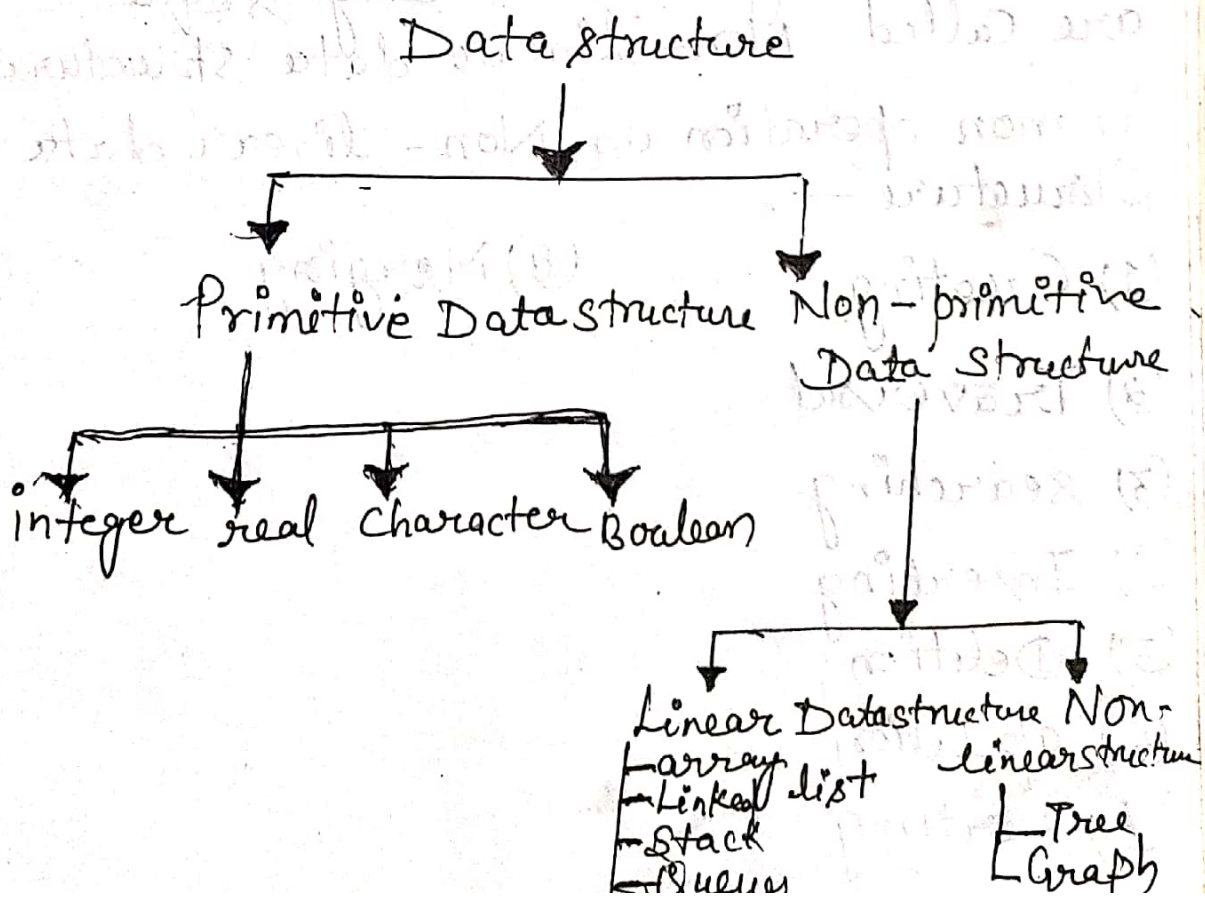
Data structure

It is important How data are stored in memory and How they are processed. Data structure invalves the study of Organization of Data in main memory and their processing.

Logical or mathematical model of the organisation of Data elements in Computer memory, where some specific method exist to access and process the data elements.

Data are the basic building block of the program, therefare, they should efficiently process the elements.



Linear Data Structure :-

Those data structure, In which data elements Organized in some sequence are Called Linear data structure.

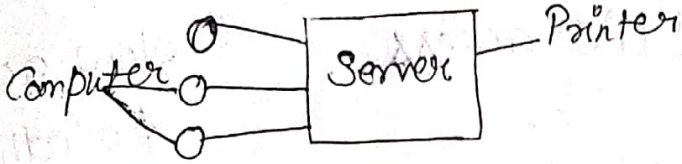
In this Category various operations are only possible in a sequence, that is we Cannot insert the element directly into some location with out traversing the previous element of the data structure.

Non-linear data structure :-

Those data structure in which data elements are organized in any Orbitrary ~~are organised in any~~ order or they are not organised in any sequence are called Non-linear data structures.

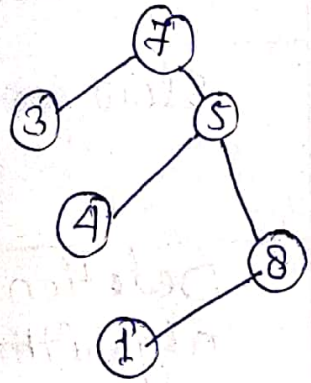
Common operation on Non-linear data structure —

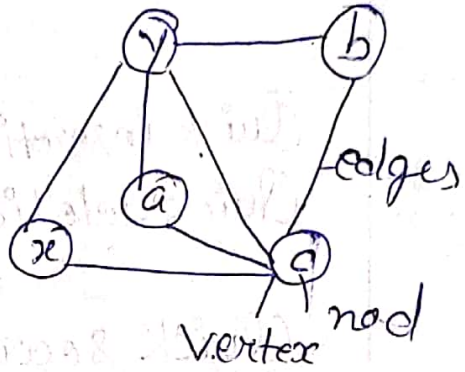
(1) Creating
(2) Traversal
(3) searching
(4) Inserting
(5) Deletion
(6) apdation
(7) Sarting
(8) Merging



यहाँ queue का use है;

Binary search tree





Uses of Data Structure

(1) DBMS
(2) Compiler design
(3) Network analysis
(4) Numerical analysis
(5) AI (Artificial Intelligence) Robat
(6) Simulation
(7) OS
(8) graphics

Advantages of Data Structure:-

Data Structure	Advantage	Disadvantage
1. Array	Quick insertion, Very fast access if index is known	slow search, slow deletion, fixed size

2. Stack	LIFO access	Slow access to other items.
3. Queue	FIFO access	Slow access to other items
4. Linked list	Quick insertion, Quick deletion	Slow search
5. B.T (Binary tree)	Quick search, insertion, Deletion	Deletion algorithm with Complex
6. Hashtable	Very fast access if key is known fast insertion	Slow deletion, access slow if key is not known, insufficient memory use, slow deletion
7. Heap	fast inserting, deletion, access to largest item	Slow access to other items
8. Graph	Models real world situation	Some algorithm are slow in access

* Program to traverse an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n, i, a[20];
clrscr();
printf("Enter the length of array");
scanf("%d", &n);
printf("Enter the elements \n");
for(i=0; i<=n-1; i++)
{
scanf("%d\n", &a[i]);
}
printf("Traversing of the array\n");
for(i=0; i<=n-1; i++)
{printf("\n%d", a[i]);
}
getch();
}
```

programming approaches:- मुख्यत: ये 4 प्रकार के है जब तक की object oriented programming approach को काम में नहीं लाया गया जिनमे

(1) structural
(2) procedural
(3) bottom up
(4) Top down

*Structural programming procedural programming का ही एक भाग है इस प्रकार की approach में programmer पूरे program के ढाँचे को दो-2 भागो में बाट देता है जिन्हे module कहा जाता है, इस प्रकार के module में similar (एक जैसे) function को लिखा जाता है ~~इस प्रकार के module~~ अर्थात् memory में ऐसे module को लोड़ किया जा सकता है जब उसकी जरूरत हो यकीनन दूसरे program के द्वारा भी पुन: उस module को इस्तेमाल किया जा सकता है module पूरा होने पर सबसे पहले उसकी टेस्टिंग की जाती है सफल हो जाने पर और त्रुटियाँ न होने पर उसे अन्य ~~module~~ module से जोड़कर पूरे program का structure तैयार कर लिया जाता है।

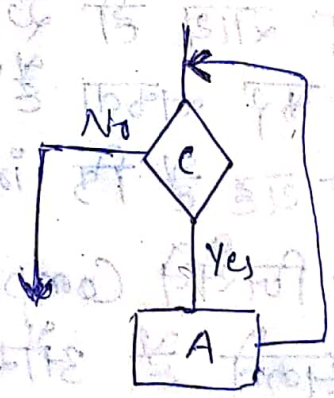
Program heirarchical model का इस्तेमाल करते हुए चलता प्रतीत होता है जिसे "for", "repeat" एवं "while", आदि looping (Condition पूरी न होने तक execute होने वाले statement)

Constraints का उपयोग होता है।

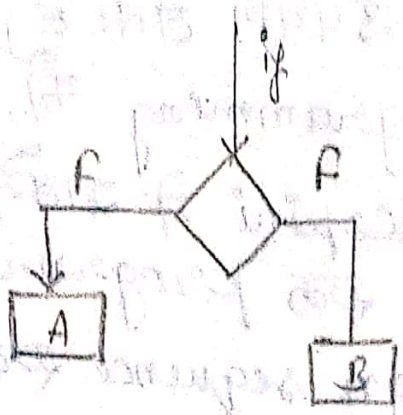
Structured programming को Corrado Bohm एवं Guiseppe Jacopini ने सबसे पहले बताया था। उन्होंने बताया कि एक program को केवल तीन structure design sequence एवं loops से बनाया जाता है।

(1) Sequence structure:- इस प्रकार की programming में केवल एक entry एवं एक ही exit statement जिससे program खत्म होता है रखा जाता है यही ~~कारण है कि इसे~~

(2) Loop ittreration:- यह एक निश्चित program statement से बना होता है जो कि विशेष परिस्थितियो के घटित होने पर ही काम करते है।



(3) Desicion making :- इस प्रकार की structural programming में Condition होती है जिसके सही या गलत होने पर अलग-2 प्रकार के निर्देश दिए जाते है।



Procedural programming approach :- यह Concept Procedure के Call पर निर्भर करता है procedure जिसे आप method भी कहते है इन methods में गणना करने के लिए निश्चित statement होते है और ये procedure कभी भी Call हो सकते है program के execution के बीच कभी भी Call हो सकती है program के एक method दूसरे method को Call कर सकते है और साथ ही एक method खुद भी खुद को Call कर सकता है, procedure का मुख्य काम ही यह है कि input लेना उसकी गणना करना जिसेसे Computation या operation कह सकते है और output देना इनमें input लेने और output देने तक के नियम बने होते है, input को arguments के रूप में और output को return value के रूप में प्रदर्शित किया जाता है। ALGOL तथा C Procedural Programming Approach के examples

main points :-

(1) ये top down approach को follow करती है

(2) data को इतनी अहमियत नहीं दी जाती जितनी Procedure को,

(3) function के द्वारा global data का use किया जाता है एवं उसे कभी भी बदला जा सकता है बिना किसी Procedure को बताए जो उसका use कर रहा है,

(4) data की एकरूपता Constistency नहीं रहती

Bottom-up & Top down approach :-

Bottom up design में programmer program के आधारभूत (Fundamental) तत्वों को बनाने पर ध्यान केंद्रित करता है ऐसे Building Blocks जो कि problem को solve कर सके जैसे कि Example programmer एक ऐसा function लिखने की सोच सकता है जो ये बताता है कि कोई Number दूसरे से devide हो सकता है या नहीं और ये function future में Program built के समय उपयोगी हो सकता है अंत में programmer इन building Blocks को एक साथ Collect करके final program का निर्माण करता है ये bottom up approach कहलाती है,

Top down :-

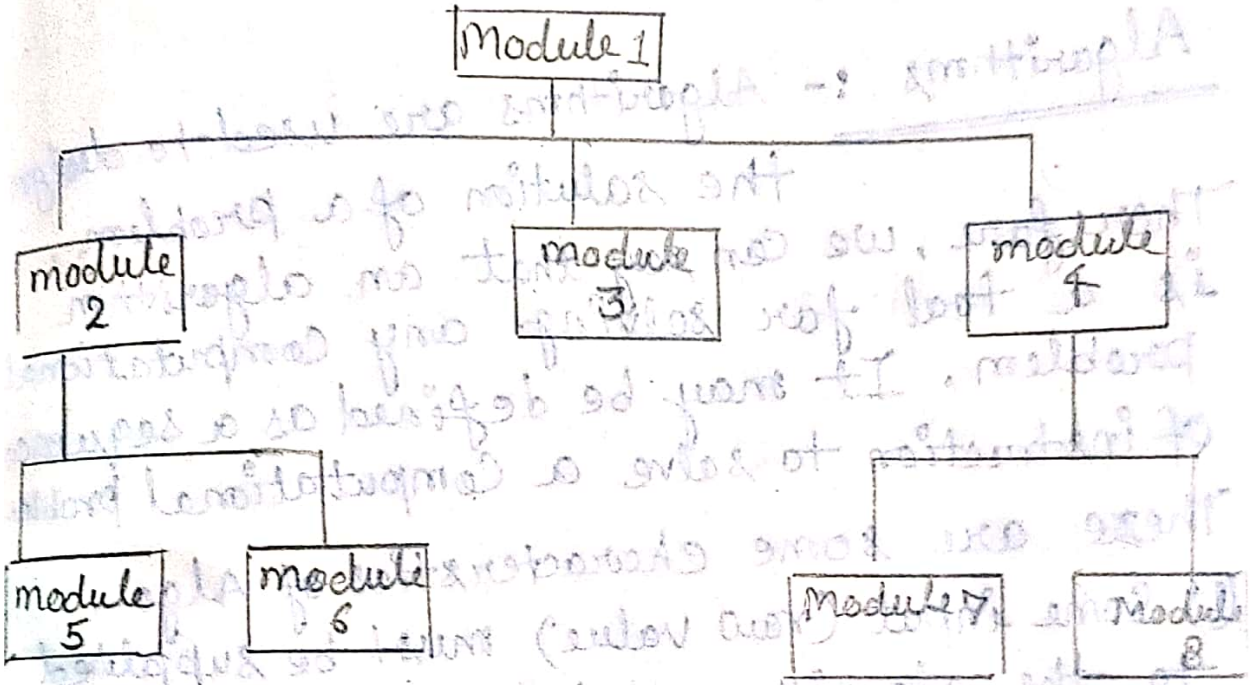
top down approach / design में structure पर ध्यान केंद्रित किया जाता है जैसे एक समस्या दूसरी छोटी - छोटी समस्याओं से जुड़ी रहती है वैसे उनका निर्माण करना है इस प्रकार पर ध्यान केंद्रित नहीं किया जाता। इस प्रकार की approach किसी भी problem को छोटी 2 problem में तोड़ दिया जाता है हर छोटी problem को solve करने के लिए function लिखा जाता है और प्रत्येक stage (स्तर) पर यह देखा जाता है की क्या problem solve हो गई। इस प्रकार हरेक स्तर पर उक्त Procedure होते होते प्रत्येक design level Complete हो जाता है।

Time Complexity

The number of ~~pro~~ instruction which a program executes during its running time is called its time complexity in Computer Science.

Top down Approach

The top down approach towards program design start with specification of the function to be performed by a program and then breaks it down to progressive by subsidiary function



Top down Approach

Bottom up approach

Bottom up approach is the reverse of top down approach. The top down approach has the following advantage :-

1) It imitates the Human Tendency to solve a problem by outlining the broad Concept first and then subsequently going into the details.

(2) The details of a module can be worked out with no (or minimum) change of

the previously outlined concept regarding its function.

(3) The programmer never loses sight of the assumptions made at the previous level. The development of the modules can take place parallel.

Algorithms :- Algorithms are used to design the solution of a problem. Therefore, we can say that an algorithm is a tool for solving any computational problem. It may be defined as a sequence of instruction to solve a computational problem.

There are some characteristics of Algo-

(1) Some input (raw value) must be supplied to the algorithm externally to solve the computational problem.

(2) The algorithm process the data by applying some operations on them, and then the desired output is generated.

(3) They must reach there concluding state that is they must terminate after executing number of instruction.

Algorithm performance :-

The performance evalution of an algorithm is obtained by totalling the number of

Occurences of each operation when running the algorithm.

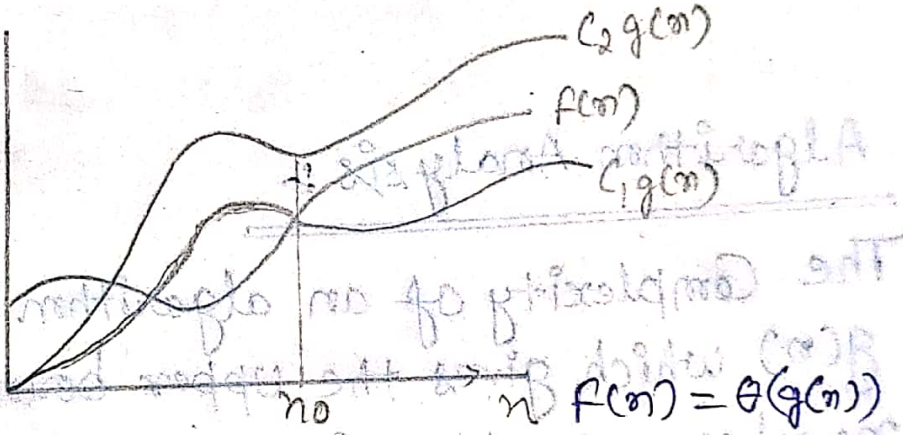
The following Notations are commonly used in performance analysis to characterise the complexity of an algorithm.

θ Notation :-

This notation bound a function to within constant factor

$$f(n) = \theta(g(n))$$

$$n_0 = c_1 g(n) = c_2 g(n) = \text{positive Constant}$$

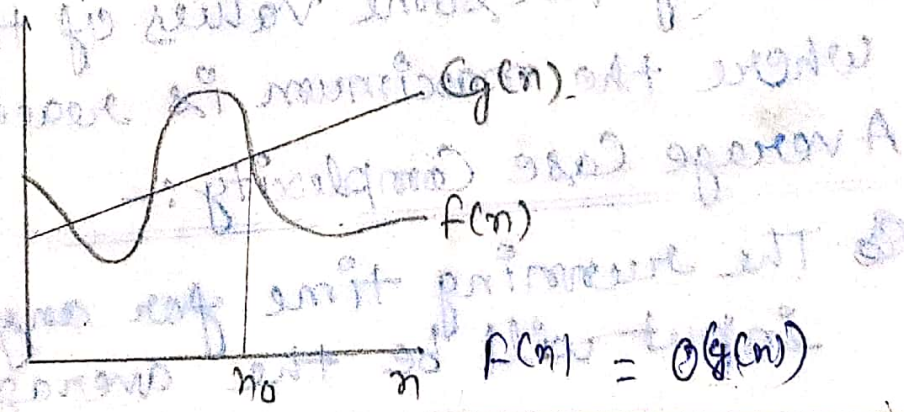


O Notation (UPPER BOUND)

This notation given an upper bound for a function to within a constant factor.

$$f(n) = O(g(n))$$

$$n_0 = cg(n) = \text{positive Constant}$$

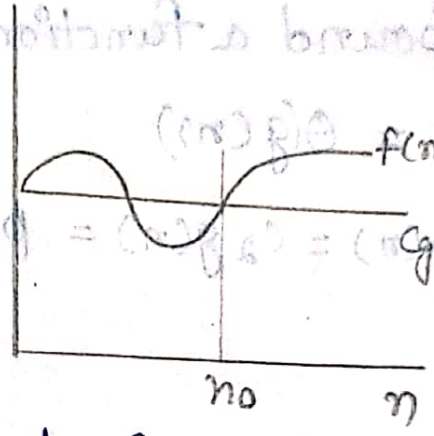


Ω Notation (Lower Bound)

This Notation given a lower bound for a function to within a constant factor.

$$f(n) = \Omega(g(n))$$

Note $g(n)$ = Positive Constant



Algorithm Analysis :-

The Complexity of an algorithm is a function $g(n)$ which gives the upper bound of the number of operations performed by an algorithm when the input size is n.

Worst Case Complexity :-

The running time for any given size input will be lower than the upper bound except possibly for some values of the input where the maximum is reached.

Average Case Complexity :-

The running time for any given size input will be the average number

of operations over all problem instances for a given size.

Variables :-

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

Two types of Variable :-

1) Local Variable :- Normally each program module contains its own list of variable which can be accessed only by the given program module for example - Exchange(x,y)
temp=x; x=y
Return

2) Global Variable :-

Variable that can be accessed by all program module are called global variable

(i) Static variable :- By a static variable, we mean a variable where length is defined befare the program is executed and cannot change i.e. remains same throughout

(ii) Dynamic variable :- We can have only as much memory as we have requested in declaration of variable arrays and other objects that we included having the size of all of them fixed before the execution of the program.

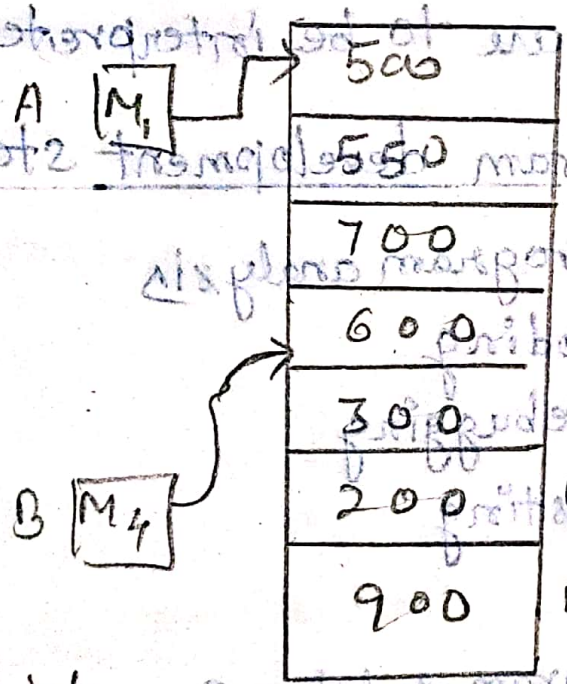
Difference between Static and Dynamic Variables

Summary

Static Variable	Dynamic Variable
1. Length of the static Variable is defined before the program execution and Cannot be changed during the execution of program	Length of the Dynamic Variable is Cannot defined before the program execution but Cannot be changed during the execution of program.
2. Static Variable are automatically initialised to Zero	Dynamic Variable are not automatically initialised
3. The value of static Variable is preserved even after it exit from the function in which it is declared	The value of Dynamic Variable is not preserved even after it exit from the function in which it is declared.
4. Static Variables are implemented as normal Variables	Dynamic variables are purely implemented through pointer

Constant :- A Constant is a data object with a name which is bound to a value permanently during its life time.

Concept of pointer Variable :- Pointers is a data type which points to a memory location containing a data value. In other word, pointer may be consider as a variable which points to a memory location i.e. pointer variable Contains memory address.



Pointer Variable — Computer variable

A & B

DATA TYPE :- A formal description of the set of values that a variable of a given types may take or to which a data

of that must belong.

Data type defination specifies two things :-

(1) It must specifies the ammount of storage that must be set aside for objects declared with that types.

for example :-

A variable of type integer must have enough space to should the largest possible integer value

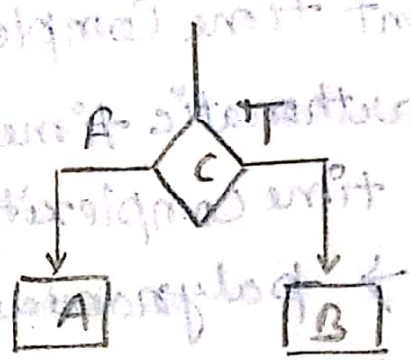
2. It specifies how data represented by strips of bits are to be interpreted.

Program development steps

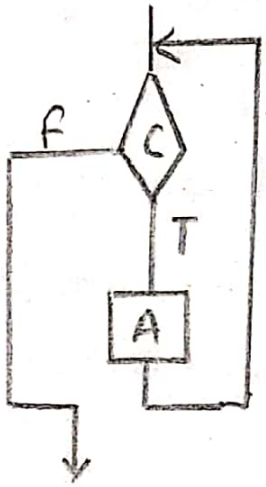
(i) step (1) :- program analysis
step (2) :- Coding
step (3) :- debugging
step (4) :- testing

Program solving Concept

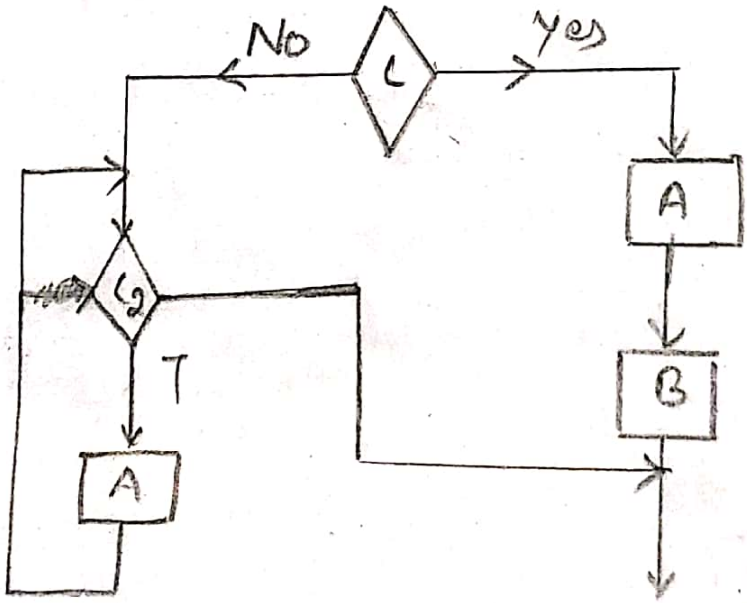
(i) Sequence structure



(iii) Loop structure



(iv) Nested structure (Modular programming)



Different Complexities

(i) $O(1)$ → Constant time Complexity

(ii) $O(\log n)$ → logarithmatic time Complexity

(iii) $O(n)$ → Linearre time complexity

(iv) $O(n^K)$ for $K > 1$ → polynomial time Complexity

(v) $O(K^n)$ for $K > 1$ → exponcial time complexity

UNIT- 02

Single dimensional Array

A linear array is a list of a finite number n of homogeneous data elements (i.e. data elements of the same type) such that:

(a) The elements of the array are referenced respectively by an index set consisting of n consecutive numbers.

(b) The elements of the array are stored respectively in successive memory locations.

The number n of elements is called the length or size of the array.

In general, the length or the number of data elements of the array can be obtained from the index set by the formula

$$\text{Length} = UB - LB + 1$$

where UB is the largest index, called the upper bound, and LB is the smallest index, called the lower bound, of the array.

Arrays As ADT

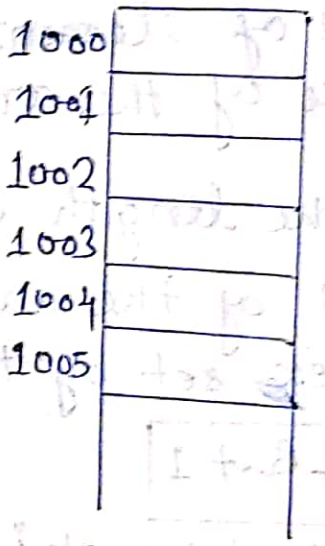
An array is a fundamental Abstract Data Type (ADT). Due to its versatility, it can be changed as per requirement. An

is a pre-defined set which has a sequence of cells where we can store different types of elements.

Representation of Linear Arrays in Memory

Let A be a linear array in the memory of the Computer. Recall that the memory of the Computer is simply a sequence of addressed locations.

LOC(A[K]) = address of the element A[K] of the array A



Computer memory

As previously noted, the elements of A are stored in successive memory cells. Accordingly the Computer does not need to keep track only of the address of every element of A, but needs to keep track only of the address of the first element of A, denoted by, Base(A)

and called the base address of A. Using this address Base(A), the Computer calculates the address of an element of A by the following formulas.

$$LOC(A[K]) = Base(A) + w(K - Lower\ Bound)$$

Q. A(5:50) where 5 is Lower Bound and 50 is Upper bound. Base(A) = 300 and w = 4 words per memory cell for A. Consider the array A. K = 15

$$= 300 + 4(15 - 5)$$
$$= 300 + 4 \times 10$$
$$= 300 + 40$$
$$= 340$$

TRAVERSING

Let A be a Collection of data elements stored in the memory of the Computer. Suppose we want to print the Content of each element of A or suppose we want to Count the number of element of A with a given a property. This Can be accomplised by traversing A, that is, by accessing and processing (frequently and visiting) each element of exactly once.

Algorithm for Traversing

1. I ← LB

2. for I = LB + UB
   PRINT A[I]
3. STOP

INSERTING

"Inserting" refers to the operation of adding another element to the Collection A.

ALGORITHM for Inserting

* At the end of the array

1. IF UB = MAX then write array is overflow & stop
2. Read Data
3. UB → UB + 1
   A[UB] = DATA
4. STOP

* At the begining of the array

1. If UB = MAX then write array is overflow & stop
2. READ DATA
3. K ← UB
4. Repeat step (5) while K ≥ LB
5. A(K+1) ← A(K)
   K ← K - 1
6. A(LB) ← DATA

7. STOP

* At the particular location of array

1. If UB = MAX then array is overflow & stop
2. Read DATA
3. Read LOC
4. K ← UB
5. Repeat (6) while K ≥ LOC
6. A(K+1) ← A(K)
7. K ← K-1
8. A(LOC) ← DATA
9. STOP

DELETING

"DELETING" refers to the operation of removing one of the element from A.

Algorithm for Deleting

* Kth element deletion

:= assignment operator

Algo: Delete(A, K, N, DATA)

A a array, K = address of element,
N = number of element

1. Set DATA := A(K)
2. Repeat for J = K to N-1
   A[J] = A(J+1)
   (Moves the (J+1)th element upward)
3. Set N := N-1

[Reset the number N of element in the array A]

4. END

Two Dimensional Array

A two-dimensional mxn array A is a collection of mxn data elements such that each element is specified by a pair of integer (such as J,K) called subscript,

Two dimensional arrays are sometimes called matrix array.

The length of a given number can be obtained from the formula

$$\text{Length} = \text{upper bound} - \text{lower bound} + 1$$

Array Declaration

data type arrayname[rows][column]

for example:- int A[3][4];

Array initialization:-

int A[2][3] = {0, 0, 0, 1, 1, 1};

Storage representation

Row-major representation / Row wise order

We can consider a two-dimensional array as a one-dimensional array since it has elements with a single dimension. As a result of this a two dimensional array can be assumed as a single column with many rows

and mapped sequentially. Such a representation is called a row major representation.

*Column major representation

We can also represt a two dimensional array as one single row of column and map it sequentially. Such a representation is called a Column major representation

Representation of two-Dimensional array in Memory

The programming language will store the array A either (1) Column by Column, is what is called column major order.
or either (2) row by row, is what is called Row major order

(a) $A[2][3] = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

$A[0][0] = a$
$A[0][1] = b$
$A[0][2] = c$
$A[1][0] = d$
$A[1][1] = e$
$A[1][2] = f$

a	b	c	d	e	f

(a) row major order

(b) Column major order

$A[2][3] = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

A[0][0] = a

A[1][0] = d

A[0][1] = b

A[1][1] = e

A[0][2] = c

A[1][2] = x

a	d	b	e	c	x

Column major order

The Computer keeps track of Base(A) — the address of the first element A[1,1] of A — and Computers the address Loc(A[J,K]) of A[J,K] using the formula

(Column-major order)

$$Loc(A[J,K]) = Base(A) + w[M(K-1) + (J-1)]$$

Row-major order

$$Loc(A[J,K]) = Base(A) + w[N(J-1) + (K-1)]$$

Again, w denotes the number of words per memory location for the array.

या

@ Column major :-

$$Loc(A[J,K]) = Base(A) + w[M(J-n) + (I-m)]$$

Row wise

$$Loc(A[J,K]) = Base(A) + w[N(I-m) + (J-n)]$$

where M X N — Column
L Row

n, m → lower bound of the index set

Question:- Left result is a 50×4 array of integer Suppose base(Result) = 1000 and each element require 4 memory cell Calculate the of the element result

(1) result[25,3], Row wise or Row major order.

= 1000 + 4[4(25-1)) + (3-1)]

= 1000 + 4[4 X 24 + 2]

= 1000 + 4[96 + 2]

= 1000 + 4 X 98

= 1000 + 392

= 1392

(2) 1000 + 4[50(2-1) + (8-1)]

= 1000 + 4[50 X 1 + 7]

= 1000 + 4[50 + 7]

= 1000 + 4 X 57

= 1000 + 228

= 1228

Operations on two dimensional array

(a) Addition;
(b) Subtraction;
(c) Multiplication;
(d) Transpose;
(e) Symetry of matrix
(f) Determinant of matrix

Program for Addition of two Matrix

```
main()
{int a[3][3];
 int b[3][3];
 int c[3][3];
 int i, j;
 clrscr();
 printf("enter the elements of 3x3 matrix a\n");
 for(i=0; i<3; i++)
 {printf("\n");
 for(j=0; j<3; j++)
{ printf("a[%d][%d]", =i,j);
 scanf("%d", &a[i][j]);
} }
 printf("enter the elements of 3x3 matrix
          b\n");
 for(i=0; i<3; i++)
 {printf("\n")
   for(j=0; j<3; j++)
{ printf("b[%d][%d]", =i,j);
 Scanf("%d", &b[i][j]);
} }
 for(i=0; i<3; i++)
{printf("\n");
 for(j=0; j<3; j++)
{ c[i][j] = a[i][j] + b[i][j];
}
 printf("\n the addition of two matrix is\n");
```

```
for(i=0; i<3; i++)
{printf("\n");
for(j=0; j<3; j++)
{printf("\t%d", c[i][j]);}
}
```

$$a = \begin{bmatrix} 4 & 1 & 3 \\ 2 & 6 & 9 \\ 3 & 5 & 8 \end{bmatrix} \qquad b = \begin{bmatrix} 9 & 2 & 7 \\ 6 & 4 & 3 \\ 5 & 7 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} 13 & 3 & 10 \\ 8 & 10 & 12 \\ 8 & 12 & 9 \end{bmatrix}$$

// PROGRAM for Substraction of two Matrix

```
#include<stdio.h>
#include<conio.h>
main()
{ int a[3][3];
  int b[3][3];
  int c[3][3];
 int i, j;
 clrscr();
printf("enter the elements of 3X3 matrix a\n");
 for(i=0; i<3; i++)
{ printf("\n");
 for(j=0; j<3; j++)
{printf("a[%d][%d]", = i j);
 scanf("%d", &a[i][j]);
}}
```

```
printf("enter the elements of 3X3 matrix b\n");
for (i=0; i<3; i++)
{printf("\n");
for(j=0; j<3 ; j++)
{ printf("b[%d][%d]",i,j);
scanf("%d", &b[i][j]);
} }
for (i=0; i<3; i++)
{ printf("\n");
for(j=0; j<3; j++)
{
c[i][j] = a[i][j] - b[i][j]
}
printf("\n The subtraction of two matrix is \n");
for (i=0; i<3; i++)
{printf("\n");
 for(j=0; j<3 ; j++)
{printf("\t %d ", c[i][j]);}
}
```

// PROGRAM for Multiplication of two matrix

```
#include <stdio.h>
#include<conio.h>
  main()
{int a[3][3];
  int b[3][3];
  int c[3][3];
  int i, j;
```

```
clrscr();
printf("enter the elements of 3x3 matrix a\n");
for(i=0; i<3 ; i++)
{printf("\n");
  for(j=0; j<3; j++)
{printf("a[%d][%d]", =ij);
 scanf("%d", &a[i][j]);
}}
printf("enter the elements of 3x3 matrix b\n");
for(i=0; i<3; i++)
{ printf("\n");
 for(j=0; j<3; j++)
{ printf("b[%d][%d]", =ij);
  scanf("%d", &b[i][j]);
}}
printf("\n the multiplication of two matrix
    is\n");
for(i=0; i<3; i++)
{ for(j=0; j<3; j++)
 c[i][j] = 0
  for(k=0; k<3; k++)
 { c[i][j] = c[i][j] + a[i][k] * b[k][j];}
}
printf("\n the multiplication of two matrix
      is\n");
 for(i=0; i<3; i++)
{printf("\n");
```

```
for(j=0; j<3; j++)
{printf("\t %d", c[i][j]);} }
```

Three Dimensional Array

000	001	002
010	011	012
020	021	022

A_0

100	101	102
110	111	112
120	121	122

A_1

200	201	202
210	211	212
220	221	222

A_2

A[1][1][1]

Page Row Column

क्रमि- कमशि page वाइ में लिखते है, A[1][1][1]

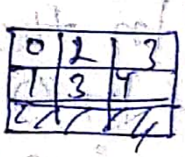
Row Column Page

Declaration:-

```
int A[3][4][2];
```

Initialization:-

```
int B[2][3][2]
{{{0,1};{2,3};{3,4};}
 {{5,6};{7,8};{9,10};}
};
int X[3][3][3] = {{ 11,12,15};
                  {14;15,16};
                  {17,18,19};
                 };
```



```
{{21, 22, 23};
 {24, 27, 26};
 {27, 28, 29};
};

{ {31, 32, 33};
 {34, 35, 36};
 {37, 38, 39};
};
```

/* PROGRAM : TRAVERSING AN ARRAY */

```
#include<stdio.h>
#include<conio.h>
main()
{ int data[5]={3,5,7,9,11};
   int i;
   clrscr();
printf("\n Elements of array 'data' are");
for(i=0;i<=4;i++)
{ printf("\n %d", data[i]);
 }
}
```

Output:-

Elements of array 'data' are

3
5
7
9
11

/* INSERTION of an elements of array at the end */

```
#include<stdio.h>
#include<conio.h>
main()
{ int a[10] = {3,5,6,8,9};
 int i, j;
 clrscr();
 printf("\n At present elements of array 'a' are");
 for(i=0; i<=4, i++)
 { printf("\n a[%d] = %d", i, a[i]);
 }
 printf("\n \n Enter an element to be inserted
 in this array :-");
 scanf("%d", &a[i]);
 printf("\n The updated array is");
 for(j=0; j<=1; j++)
 print("\n a[%d] = %d", j, a[j]);
print("\n \n Now, total no. of elements in this
array are %d", j);
}
```

Output :-

At present elements of array 'a' are

a[0] = 3

a[1] = 5

a[2] = 6

a[3] = 8

a[4] = 9

Enter an element to be inserted in this array:

The updated array is
a[0] = 3
a[1] = 5
a[2] = 6
a[3] = 8
a[4] = 9
a[5] = 12
Now, total no. of elements in this array are 6.

/* Insertion of an elements at Kth Location */

```
# include<stdio.h>
# include<conio.h>
main()
{ int data[10] = {3, 5, 6, 8, 10};
int i, K, j, n, t, x, z = 0;
clrscr();
printf("Initially array elements are");
for(i=0; i<=4; i++)
{ printf("\n\t data[%d] is %d", i, data[i]);
}
printf("\n Total no. of elements in array data
  is %d, i);
printf("\n\n Enter the position in array where
you want insert a new element 0, 1, 2, 3 - - -)");
Scanf("%d", &K);
for(j= i-1; j>=K; j--)
{ data[j+1] = data[j];
  printf("\n\t now data[%d] Contains %d", j+i, data[j]);
}
printf("\n\t now data[%d] Contains %d", K, z);
```

```
for(j=k-1; j>=0; j--)
printf("\n\t new data[%d] contains %d", j, data[j]);
printf("\n\n Enter the new element to be inserted");
scanf("%d", &n);
data[k]=n;
t = it 1;
printf("\n New array after insertion of new no.
'%d' at array position '%d' is", n, k);
for(x=0; x<=t-1, x++)
  { printf("\n\n\t data [%d] = %d", x, data[x]);
  }
}
```

Output:-

Initially array elements are

data[0] is 3

data[1] is 5

data[2] is 6

data[3] is 8

data[4] is 10

Total no. of elements in array data is 5.

Enter the position in array where you want insert a new element 0, 1, 2, 3, - - -) 3

new data [5] Contains 10

" [4] " 8

" [3] " 0

" [2] " 6

" [1] " 5

" [0] " 3

Enter the new element to be inserted: 11

New array after insertion of new number '2' at array position '3' is data[0] = 3

data[1] = 5
" [2] = 6
" [3] = 11
" [4] = 8
" [5] = 10

/* PROGRAM for deletion of an element from end of array */

```
#include<stdio.h>
#include<conio.h>
main()
{ int data a[10] = {3, 5, 6, 8, 9}
  int i, j, item;
  clrscr();
  printf("\n At present elements of array 'a' are").
  for (i = 0; i<=4; i++)
  { printf("\n a[%d]", i, a[i]); }
  printf("\n\n Delete the last element of array
    i.e. %d", a[i-1]);
  item = a[i-1];
  printf("\n\n The updated array is");
  for (j=0; j<=i-2; j++)
  printf("\n a[%d] = %d", j, a[j]);
  printf("\n\n Now, total no. of elements in
  this array are %d", j);
}
```

Output :-

At present elements of array 'a' are

a[0] = 3

a[1] = 5

a[2] = 6

a[3] = 8

a[4] = 9

Delete the last element of array i.e. 9

The updated array is

a[0] = 3

a[1] = 5

a[2] = 6

a[3] = 8

Now, total no. of elements in the array are 4

/* deletion of an elements at kth location */

```
#include<stdio.h>
#include<conio.h>
main()
{ int data[10] = {3,5,6,8,10};
int i, k, j, n, t, x, z = 0, item = 0;
clrscr();
printf("Initially array elements are");
for( i=0; i<=4; i++)
{ printf("\n\t data[%d] is %d", i, data[i]);
}
printf("\n Total no. of elements in array.
```

```
data is %d", i);
printf("\n\n Enter the position of element you
want to delete(0,1,2,3,---)");
scanf("%d", &K);
item = data[K];
for(j=K; j<i-1; j++)
{ data[j] = data[j+1];
}
printf("\n Now array after deletion of number '%d'
at array position '%d' is", item, K);
for(x=0; x<= i-2 ; x++)
 {printf("\n\t data[%d] = %d", x, data[x]);}
printf("\n Now, total no. of elements in this are %d",
 i-1);
```

Output :-

Initially array elements are

data[0] is 3
data[1] is 5
data[2] is 6
data[3] is 8
data[4] is 10

Total no. of elements ~~you want to de~~ in array is 5.

Enter the position of element you want to delete(0,1,2,3---)1

New array after deletion of number '5' at array position '1' is

data[0] is 3
data[1] is 6
data[2] is 8

data[3] is 10

Now, total no. of elements in this are 4.

/* TRANSPOSE OF MATRIX */

```
#include<stdio.h>
#include<conio.h>
main()
{ int a[10][10];
  int trans[10][10];
  int i, j, r, c;
  clrscr();
  printf("\n Enter the number of row of matrix A=");
  scanf("%d", &c);
  printf("\n Enter the number of Columns of
          Matrix A=");
  scanf("%d", &c);
  if (r>10) || (c>10)
  { printf("\n You have entered wrong data");
    exit(0);
  }for(i=0; i<r; i++)
  { for(j=0; j<c; j++)
    scanf("%d", &a[i][j]);
  }
for(i=0; i<r; i++)
{ for(j=0; j<c; j++)
  trans[i][j] = a[j][i];
}
```

```
printf("\n the transpose of matrix A is:\n");
for(i=0; i<r; i++);
{printf("\n");
 for(j=0; j<3; j++)
  printf("%5d", trans[i][j]);
 }
}
```

Output:-

Enter number of rows of Matrix A = 3
Enter number of Column of Matrix A = 3
Enter the elements of Matrix A

2
4
5
2
3
1
8
1
7
4

The transpose of matrix A is

2 3 1
4 1 7
5 8 4

/* SYMMETRY OF MATRIX */

```
#include<stdio.h>
#include<conio.h>
main()
{
```

```
int a[3][3];
int trans[3][3];
int i,j, flag=0;
clrscr();
printf("Enter the elements of 3*3 matrix a[i][j]");
for(i=0; i<3; i++)
{
printf("\n");
for(j=0; j<c; j++)
{ printf("a[%d][%d]=",i,j);
scanf("%d",&a[i][j]);
}
}for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{ trans[i][j] = a[j][i];
}
}for(i=0; i<3; i++);
{ for(j=0; j<3; j++)
{ if(trans[i][j] = a[j][i])
{ flag=1;
break;
}
}
}
if(flag==0)
printf("\n The matrices is symmetry\n");
```

```
else
printf("\n The matrices is Not Symmetry\n");
}
```

Output :-

Enter the elements of 3*3 matrix a
Enter the elements of 3*3 matrix a
a[0][0] = 2
a[0][1] = 3
a[0][2] = 5
a[1][0] = 3
a[1][1] = 4
a[1][2] = 6
a[2][0] = 5
a[2][1] = 6
a[2][2] = 7

UNIT - 3

LINKED LIST

Two types of memory allocation :-

(i) Static memory allocation,

(ii) Dynamic memory allocation :-

There are many functions in C used for allocating and managing memory :

(i) Malloc() function may be used to allocate a block of memory. A block of memory with a particular size is reserved and a pointer is returned which is of type void meaning that function can be assigned to any type of pointer

(ii) Calloc() function is used ~~to move a reserved~~ there is a need for multiple blocks of storage, of same size and then set all bytes to zero

(iii) Realloc() function is used to move a reserved block of memory to another allocation of different dimensions. when the memory allocated by using Calloc or malloc is insufficient or in excess, the size of the memory which has already ~~can~~ been allocated can be changed using this function

(iv) free() is used to release the previously used block of memory. ~~In a block of memory when~~

Syntax of malloc()

ptr = (data type *) malloc (no of elements * size of each elements);
(pointer)

Syntax of Calloc()

ptr = (data type*) Calloc (no. of elements, size of each elements);

Syntax of free()

free (Ptr);

Syntax of realloc()

realloc (ptr, new, size);

Example:-

(1) int * ptr;
ptr = (int *) malloc (10 * size of int);

(2) int * ptr;
ptr = (int*) Calloc (10, 2);

Self referencing structure → अपनी जैसी दूसरी Node का address बताता है।

Types of Memory allocation in C

① Compile time or static memory allocation using array.
② Run time or Dynamic memory allocation using pointers.

Difference between static memory or Dynamic memory allocation.

Static memory	Dynamic memory
① Memory is allocated at the start of the program.	Memory is allocated at the run time.

(2.) Memory is allocated is fixed.	Memory is allocated is Dynamic
(3.) Memory is allocation sufficient use of memory	Memory is allocation insufficient use of memory. This memory is expended

Malloc():-

(1) This function is allocates the block of memory in bytes.

(2) This fun. is like a request to the RAM at the system to allocate the memory if the request is granted it return a pointer to the first block of that block of memory and the type of memory pointer it returns is void and if request is not granted then it return a Null.

(3) This function is available in the header file <stdlib.h>, <alloc.h>

Calloc()

(1) It is work to the similar malloc it need two arguments

(2) This function is available in the header file <stdlib.h>, <alloc.h>

(3) Calloc में zero value होती है जबकि malloc

में garbage value होती है;
(4) multiple memory allocation होती है, The works of Calloc as Malloc is allocates the memory to program.

(3) Free()

(1) De-Allocate the Previously allocated memory ~~to the~~ using malloc.
(2) free function is used to return the allocated memory to the System RAM

(4) Realloc()

(1) This function is used to resize the size of memory block which is already allocated.
(2) यह <Stdlib.h> में होता है;
(3) Note that before using the realloc function the memory block to be resized should be allocated using the following statement

ptr = malloc (size)

Difference between Calloc() and Malloc() function

S.No	Calloc()	Malloc()
1.	Syntax :- ptr = (data type *) Calloc (no of element, size of each element)	Syntax :- ptr = (data type *) malloc (no of element * size of each element)
2.	Allocates multiple block of memory and block contain equal size	Allocates a contigious single block of memory with specified size in parameter.

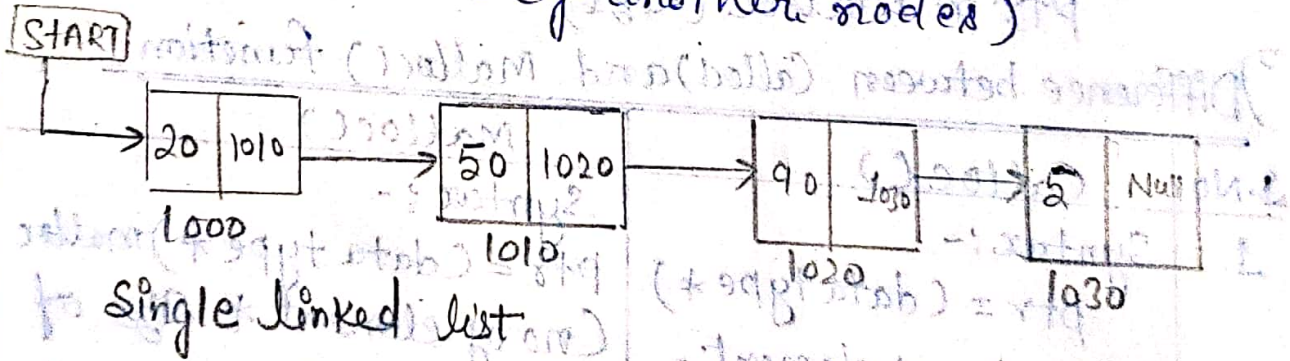
(3) Initialize all bytes in the allocated block to zero	Allocated memory location are not initialized
(4) Calloc is more expensive because is set to zero but it more Convenient than malloc function.	There is not initialized the time of initializa-tion more than Calloc

LINKED LIST

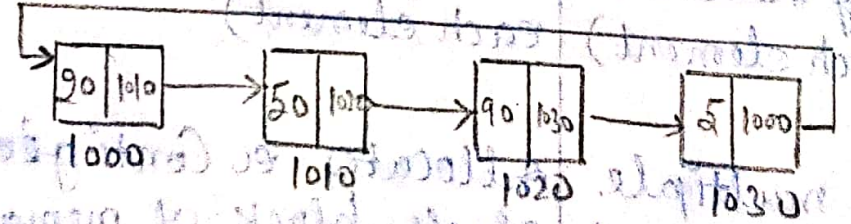
Linked list is a linear data structure. Linked list is a linear collection of data elements called nodes. The linear order is represented by means of pointer. Each nodes divided into two parts.

(1) information;

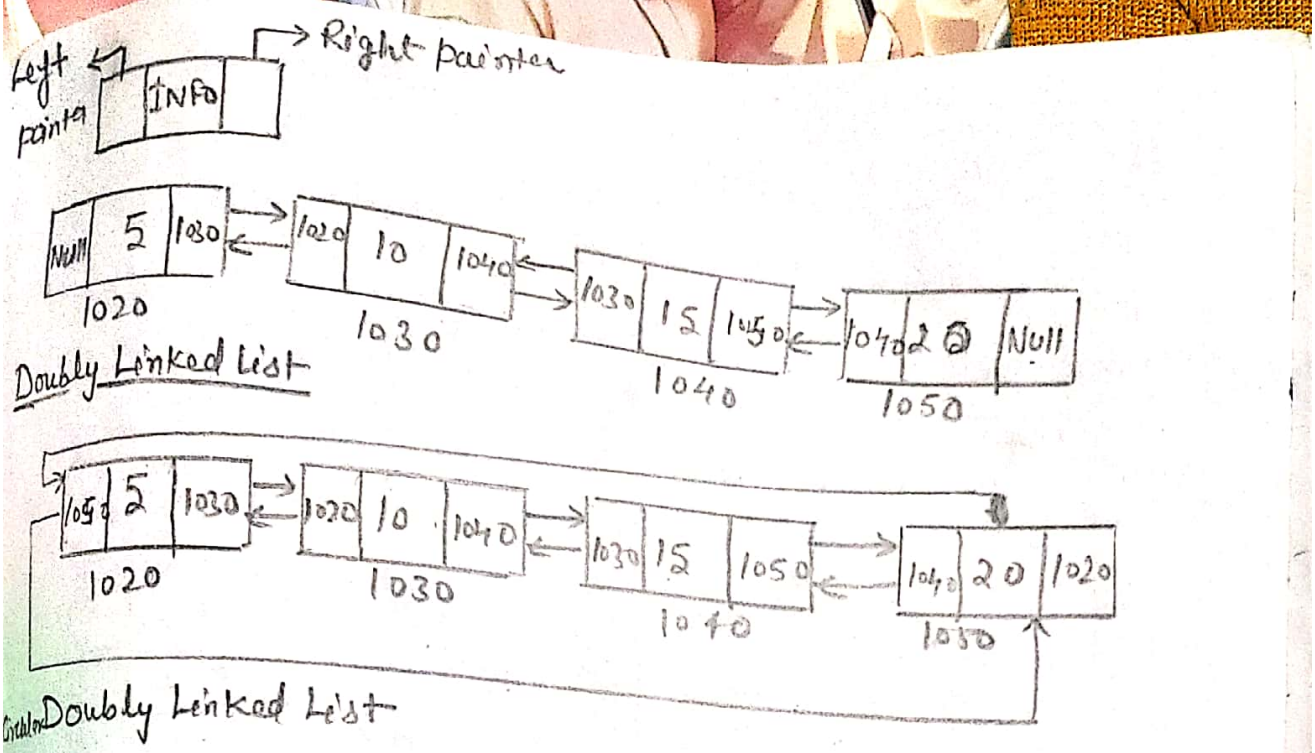
(2) Link (address of another nodes)



Single linked list



Circular linked list



Two methods are for allocate the memory of Linked List (Representation)

(i) Array like
(ii) Pointer based

START :- इसमें यह ये बताएगा कि दूसरा node कहा है,

NULL :- कौन-सा Node लास्ट है,

AVAIL :- कौन-सा पहला Node ऐसा है जिसमें Information नहीं है,

for example :- START = 9

INFO[9] = N

LINK[9] = 3, INFO[3] = A

LINK[3] = 6, INFO[6] = □

LINK[6] = 11, INFO[11] = E

LINK[11] = 7, INFO[7] = X

LINK[7] = 10, INFO[10] = I

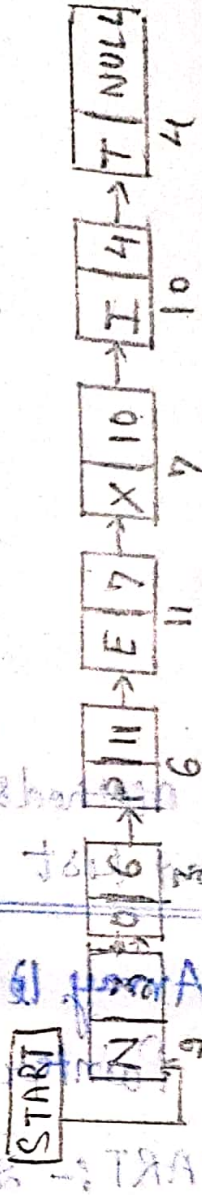
LINK[10] = 4, INFO[4] = T

LINK[4] = 0 NULL

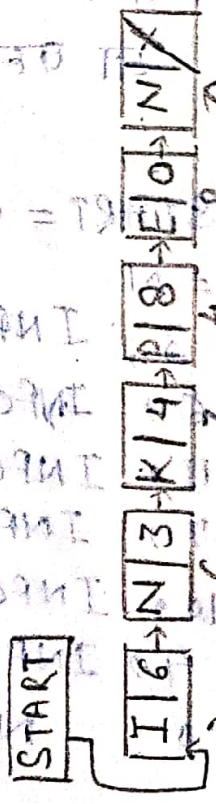
START = 9

	INFO	LINK
1		
2		
3 →	0	6
4 →	ϕ	0
5		
6 →	−1	11
7 →	X	10
8		
9 →	N	3
10 →	I	4
11 →	E	7
12		

In memory



	INFO	LINK
0	N	X
1 →	−	5
START 2 →	I	6
3	K	4
4	P	8
5	−	9
6	N	3
AVAIL 7	−	X
8	E	0
9	−	7



Implementation of

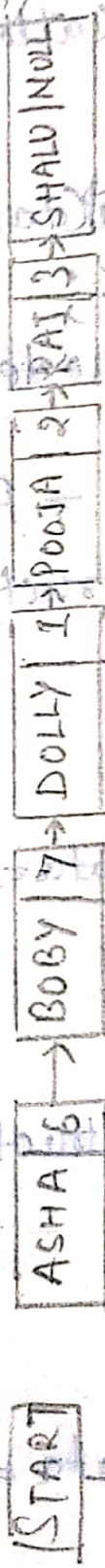
Using array Using pointer.

The following list of name is assigned to linear array INFO. Assigned value for an array link and variable start so that the list of Alphabatical name is formed.

INFO[0] = POOJA
INFO[1] = MANU
INFO[2] = RAJ
INFO[3] = SALU
INFO[4] = NEHA
INFO[5] = ASHA
INFO[6] = BOBY
INFO[7] = DOLLY

ALPHABALICAL

ASHA
BOBY
DOLLY
MANU
NEHA
POOJA
RAJ
SHALU

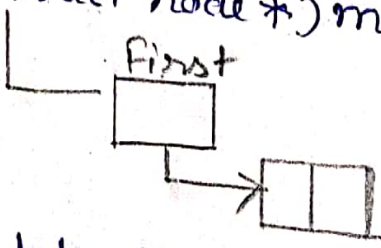


	INFO	LINK
0	POOJA	2
1	MANU	4
2	RAJ	3
3	SHALU	-1
4	NEHA	0
5	ASHA	6
6	BOBY	7
7	DOLLY	1

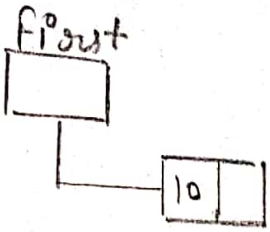
START

Implementation of Linked List pointer based

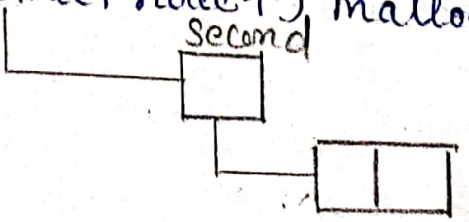
```
struct node
{ int info;
  struct info;
  struct node *, Link;
}node;
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
main()
{clrscr();
 struct node{data type}
 {int data;
  Struct node * link;
 };
Struct node. * First, * Second, * third.
First = (struct node *) malloc (size of struct node);
```



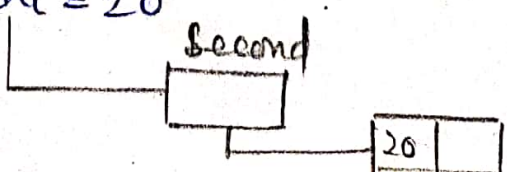
First ⟹ data=10 →



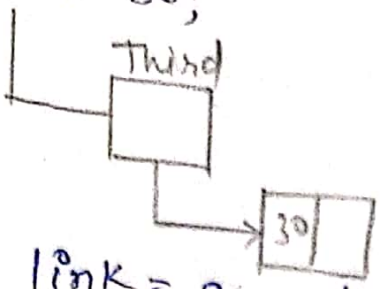

```
Second = (Struct node *) malloc (size of struct node);
```

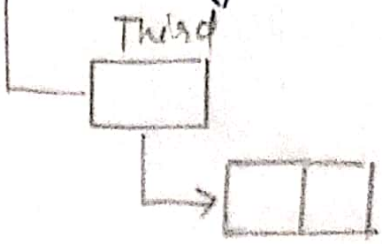


Second ⟹ data = 20



Third = (struct node *) malloc (size of struct node);
Third => data = 30;





first -> link = second
second -> link = NULL
Printf ("%d", first -> data);
printf("%d", second -> data);
printf("%d", third -> data);
}

{ -> डेस marathon painter में लगाते हैं।