UNIT-I: INTRODUCTION

Data Structures, Definition, Data Structure Operations, Abstract Data Types, Complexity of Algorithms-Time and Space, Arrays, Representation of Arrays, Linear Arrays, Insertion, Deletion and Traversal of a Linear Array, Array as an Abstract Data Type, Multi-dimensional Arrays, Strings, String Operations, Storing Strings, String as an Abstract Data Type

UNIT-II: STACKS AND QUEUES

Stack, Definition, Array Representation of Stack, The Stack Abstract Data Type, Applications of Stacks: Prefix, Infix and Postfix Arithmetic Expressions, Conversion, Evaluation of Postfix Expressions, Recursion, Towers of Hanoi, Queues, Definition, Array Representation of Queue, The Queue Abstract Data Type, Circular Queues, Dequeues, Priority Queues

UNIT-III: LINKED LISTS

Pointers, Pointer Arrays, Linked Lists, Node Representation, Single Linked List, Traversing and Searching a Single Linked List, Insertion into and Deletion from a Single Linked List, Header Linked Lists, Circularly Linked Lists, Doubly Liked Lists, Linked Stacks and Queues, Polynomials, Polynomial Representation, Sparse Matrices

UNIT-IV: TREES

Introduction, Terminology, Representation of Trees, Binary Trees, Properties of Binary Trees, Binary Tree Representations, Binary Tree Traversal, Preorder, Inorder and Postorder Traversal, Threads, Thread Binary Trees, Balanced Binary Trees, Heaps, Max Heap, Insertion into and Deletion from a Max Heap, Binary Search Trees-Searching, Insertion and Deletion from a Binary Search Tree, Height of Binary Search Tree, m-way Search Trees, B-Trees

UNIT-V: GRAPHS

Graph Theory Terminology-Introduction, Definition, Graph Representation, Graph Operations, Depth First Search, Breadth First Search, Connected Components, Spanning Trees, Biconnected Components, Minimum Cost Spanning Trees, Kruskal's Algorithm, Prim's Algorithm, Shortest Paths, Transitive Closure, All-Pairs Shortest Path, Warshall's Algorithm

UNIT-VI: SEARCHING AND SORTING

Searching, Definition, Linear Search, Binary Search, Fibonacci Search, Hashing, Sorting, Definition, Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merging, Merge Sort, Iterative and Recursive Merge Sort, Shell Sort, Radix Sort, Heap Sort.

UNIT-1

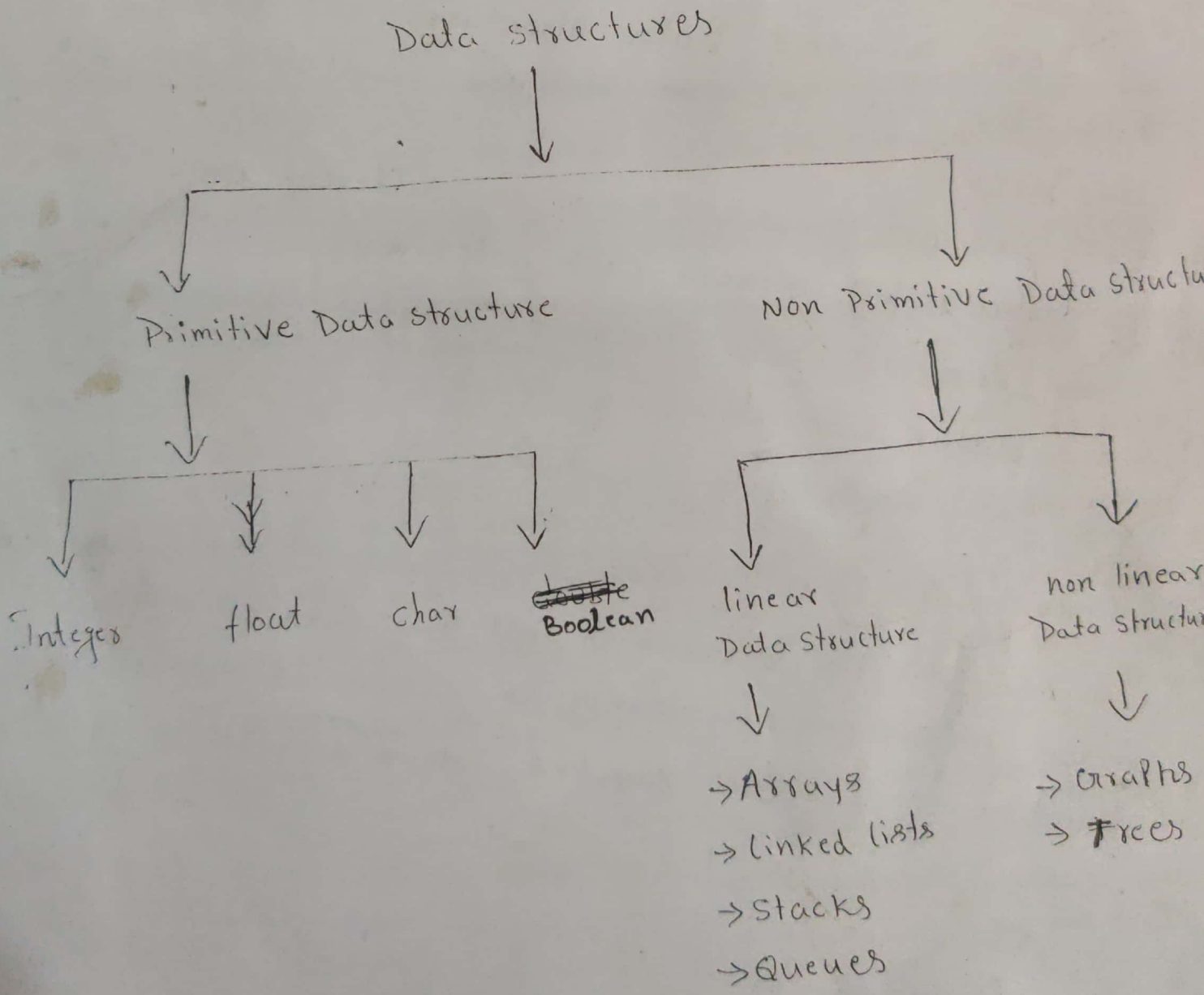
Data structure:-

The collection of data elements That are stored in memory are known as data structures.

They are considered as an efficient way of storing and organizing data inside The computer memory.

Data structures are classified into 2 types

1. Primitive Data structures
2. Non Primitive Data structures.



① Primitive Data structures :-

Primitive Data structures are the basic data structures that directly operate upon the machine instructions.

They have different representations on different computers.

Ex: Integers, float, double, char, Boolean

② Non Primitive Data structures :-

Non Primitive Data structures are more complicated data structures and are derived from Primitive data structures.

They are grouping same or different data items with relationship between each data item.

Non Primitives Data structures are divided into 2 types

(i) linear data structure

(ii) Non linear data structure.

(i) linear data structure :-

A data structures in which all the data elements are stored in sequential manner is called as linear data structure. They are divided into mainly 4 types.

(1) Arrays

(2) stack

(3) queues

(4) linked lists.

(1) Arrays :-

An array is a collection of similar data types. It is a variable that is capable of holding fixed values in contiguous memory location.

Ex:- int a[5];

a[0]	a[1]	a[2]	a[3]	a[4]
28	9	11	10	27

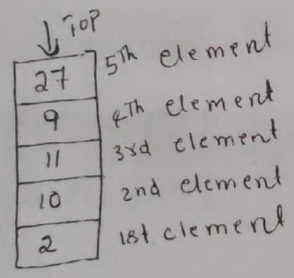
(2) stack :-

stack is a linear data structure that stores data elements in "Last In First Out" (LIFO) manner.

In stack elements can be inserted and deleted only at one end. This end is called top. The two essential operations carried on stack are.

(i) Push → This operation adds an element to a stack
(ii) POP → This operation deletes an element to a stack.

Ex:-

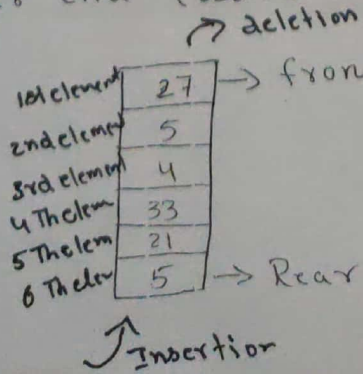


(3) Queue :-

Queue is a linear data structure. It stores data elements in "first-in-first-out" (FIFO) manner.

In queue elements can be inserted in first and deleted in first. In a queue elements are added at one end called the "rear" and deletion of element is done at the other end called the "front"

Ex:-



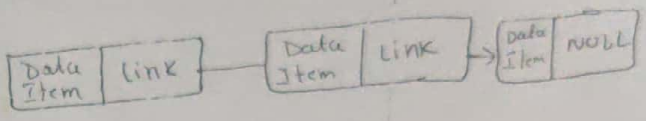
operation :- → Enqueue :- Insert
→ dequeue → deletion

(4) Linked list :-

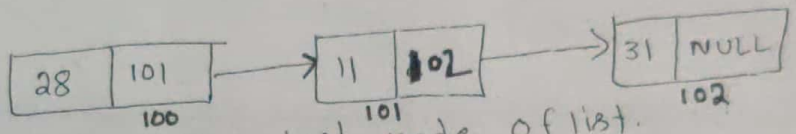
Linked list is a linear data structure that stores Similar data in memory. The elements in linked list are stored at random memory locations.

Linked list is a collection of elements called nodes, Each node contains two fields of Information. one field contains data items and the other field contains links or address of the successor list element.

A node can be represented as :-



Ex :-



Here "NULL" Indicates The last node of list.

(ii) Non-linear Data Structure :-

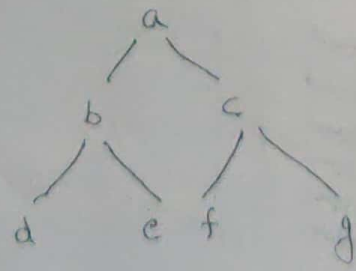
A Data Structure In which all the data elements are not stored in sequential manner is called as non-linear data structure.

They are 2 types ① Tree ② Graph.

① Tree :-

Tree is a non linear data structures that can be represented in a hierarchical manner. A tree contains nodes and each node contains name of data and link to other tree node. This type of tree is called binary tree.

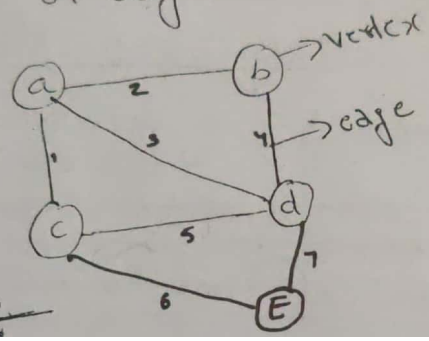
Ex :-



Graph :-

A Graph is a non-linear data structure which contains a set of ~~connected~~ vertices connected to each other via a set of edges.

Ex:-



a, b, c, d, e → vertex

1, 2, 3, 4, 5, 6, 7 → edges

type :-

Basic Operations On Data Structures:

Different operations that can be performed on the various data structures.

1. Traversing It means to access each data item exactly once so that it can be processed. For example, to print the names of all the students in a class.

2. Searching It is used to find the location of one or more data items that satisfy the given constraint. Such a data item may or may not be present in the given collection of data items. For example, to find the names of all the students who secured 100 marks in mathematics.

3.Inserting It is used to add new data items to the given list of data items. For example, to add the details of a new student who has recently joined the course.

4.Deleting It means to remove (delete) a particular data item from the given collection of data items. For example, to delete the name of a student who has left the course.

5.Sorting Data items can be arranged in some order like ascending order or descending order depending on the type of application. For example, arranging the names of students in a class in an alphabetical order, or calculating the top three winners by arranging the participants' scores in descending order and then extracting the top three.

6.Merging Lists of two sorted data items can be combined to form a single list of sorted data items.

Abstract Data type :-

An Abstract data type (ADT) is a mathematical model of a data structure that specifies the type of data stored, the operations supported on them, and the types of parameters of the operation.

An ADT specifies what each operation does but not how it does it. The ADT can be implemented using the data structures.

An ADT is describle the type of data, describe the properties of data and describe what operations you can perform the data. It does not perform the how to implementation of the data.

Ex:- Integer (type of data)

-3, -2, -1, 0, 1, 2, 3

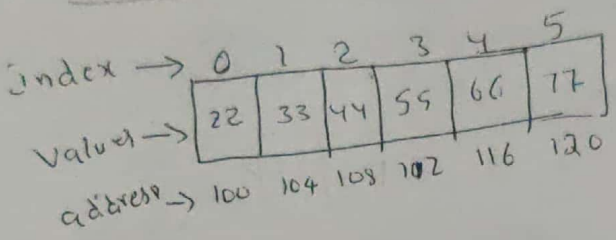
Properties:
-ve or +ve

Operations:
addition, sobstration
multiplication, division

Array as an Abstract Data type :

Array → An array is a collection of similar datatypes.
→ An array is a linear collection of items
→ Items are Indexed with Preserve the Position of item in the list
→ Array in memory consumes sequential blocks
→ It takes constant time to access any item of the array

Ex: int a[6]



Array ADT :- A set of pairs <index, value> where for each value of index there is a value from the set item. Index is a finite ordered set of one or more dimensions (1D, 2D)
array ADT follows the ① set of operations ② Data structure

① **Set of operations** :-
→ creation of array of specified size and type
→ Retrive a particular item from given index
→ store an item at given index
→ modify item at given index
→ count item stored in the array
→ Remove an item from a given index
→ Insert an item from a given array

② **Data structure** :-
→ where to store item?
→ where to store last filled Index Information?
→ where to store max size?

Data structure :-

⇒ where to store items?

- Declare a pointer to hold address of array created of specified size

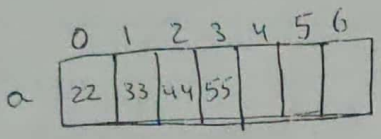
→ where to store last filled index information?

Declare a variable with name last index

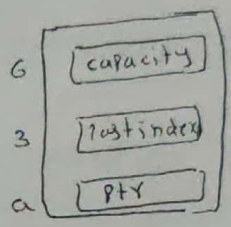
→ where to store max size information of array?

Declare a variable with name capacity.

Ex :



Ex :



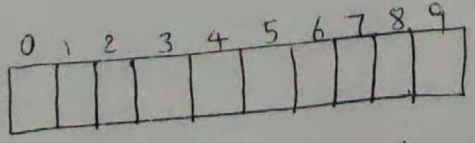

```
struct arrayADT
{
    int capacity;
    int lastindex;
    int *ptr;
};
```

(2) Set of operations :

→ Creation of array of specified size and type.

Syntax : `struct arrayADT * Create array (int capacity);`

Ex : `struct arrayADT* create array (10);`



→ It is mainly used to set the size of array ADT

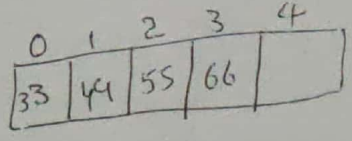
size is =10

> Retrive a particular item from given Index :-

int getitem (int index);

Retrive the values from the Array. usig getitem function.

Ex :-



"Index value = -1 Indicates empty array"

int getitem (2);

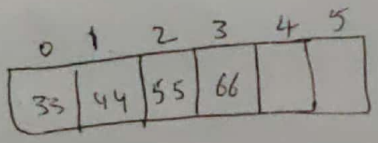
⇒ 55

> Store an Item at given Index :-

Set the values of Array ADT using setitem() function

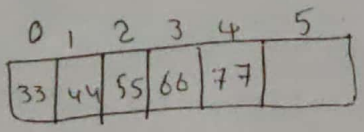
int setitem (int index, int value);

Ex :-



Suppose we want Insert value 77 In Index 7 we use Set item function.

int setitem (4, 77) Then



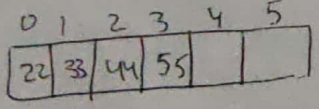
> Modify item at given Index :-

Suppos we want edit the values Then we use the edititem() All ready entered

functions

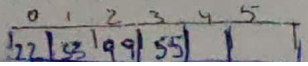
int Edititem (int index, int new value);

Ex :-



we want to change Index 2 value(44) into 99 Then

int Edititem (2, 99)

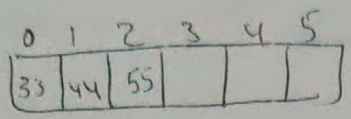


Count items stored in array :-

It displays the numbers of values entered in the array.

int countitems ();

Ex :- a



int countitem (a);

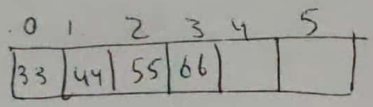
⇒ 3

Remove an item from given index :-

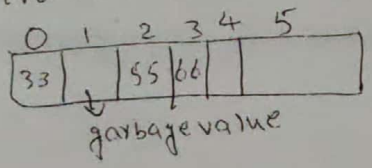
remove the values from given array.

int removeitem (int index);

Ex :-



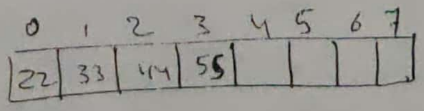
intremoveitem (1); remove the index 1 value Then print the grabage value



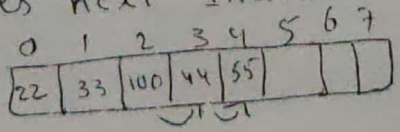
Insert an item from a given array :-

Insert the values some particular Index Theen we use the → int insertitem (int index, int newvalue);

Ex :-



Suppose we want insert the Index (2) in 100 Then change next values next Index.



Stack Abstract Data type :-

Stack is a linear data structure and is an ordered collection of homogenous data elements, where the Insertion and deletion operations takes place at one end called top of the stack. That means new element is added at top of the stack and an element is removed from the top of the stack.

In stack the Insertion and deletion operations are performed based on "LIFO" (Last In First out) Principle.

STACK Operations :-

They are mainly 5 operations are applied on stack the are ① Push ② Pop ③ Peek ④ isfull ⑤ isempty

① Push :- Insertion of an element into the stack or writ a value to the stack is called Push operation.

② Pop :- Remove of an element from the stack or read a value from the stack is called Pop operation.

③ Peek :- It is displays the top of the stack element.

④ stack overflow (isfull) :- when the stack is full and we try to Pus elements into the stack, stack overflow occurs

⑤ Stack underflow (isempty) :- when the stack is empty and we try to Pop an element from the stack. Then it is called stack

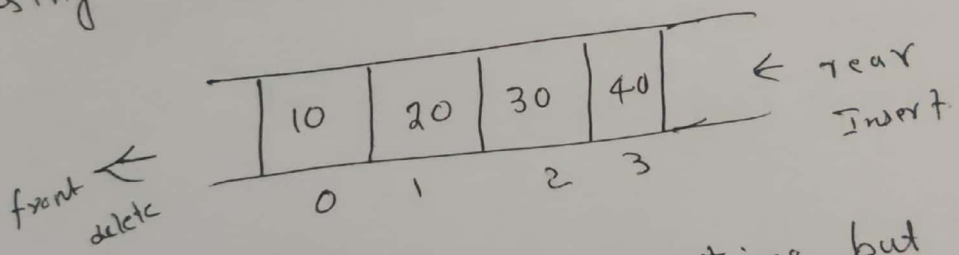
A Queue is a special type of data structure In which formally defined as ordered collection of elements That has two ends

① front (Deletions takesplace from The front end)

② rear (Insertions takesplace from The rearend)

Queue works on The basis of FIFO (First In First out) because The first element entering The Queue will be The first element to be deleted.

a Queue can be represented using the Sequential allocation (using arrays)



Queue operations:- Queue is nothing but collection of items. All The elements In The Queue are Stored sequentia The various operations on The Queue are

① Queue underflow (queue Empty)

② Queue overflow (queue full)

③ Insert of The element Into The Queue (Enqueue)

④ Delete of The element ~~In~~ from The Queue (Dequeue)

⑤ display of The Queue (traverse)

- The **List ADT Functions** is given below:

A list contains elements of the same type arranged in sequential order and following operations can be performed on the list.

- get() – Return an element from the list at any given position.
- insert() – Insert an element at any position of the list.
- remove() – Remove the first occurrence of any element from a non-empty list.
- removeAt() – Remove the element at a specified location from a non-empty list.
- replace() – Replace an element at any position by another element.
- size() – Return the number of elements in the list.
- isEmpty() – Return true if the list is empty, otherwise return false.
- isFull() – Return true if the list is full, otherwise return false.

Algorithms :-

→ Algorithm is a step by step procedure, which defines a set of Instructions to be executed in a certain order to get the desired output.

→ algorithm are generally created Independent of underlying languages, i.e. an algorithm can be implemented in more than one Programming language.

categories of algorithm :-

Search :- Algorithm to search an item in a data structure.

Sort :- Algorithm to Sort items in a certain order

insert :- Algorithm to insert item in a data structure

Update :- Algorithm to update an existing item in a data str

Delete :- Algorithm to delete an existing item from Data struc

characteristics of algorithm :-

unambiguous :- Algorithm should be clear and unambiguous.

input :- an algorithm should have 0 or more well defined in

output :- an algorithm should have 1 or more well defined outpu

Finiteness :- Algorithm must terminate after a finite numbers of step

Feasibility :- Should be feasible with the available resources.

Ex:- ① Design algorithm to add two nombers and display the result.

Step 1: START

Step 2: declare Three integers a, b, sum.

Step 3: define values a, b

Step 4: add values of a and b.

Step 5: store output of step4 to sum

Step 6: Print sum.

Step 7: STOP.

algorithms are designed using two approaches That are

(i) Top-down approach

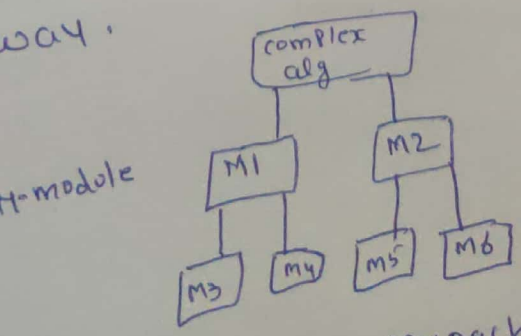
(ii) Bottom up approach.

① Top-down approach:-

→ The TOP down approach basically divides a complex Problem (or) algorithm into multiple smaller parts. (modules)

→ The process of decomposition is iterated until the desired level of module complexity is achieved.

→ The TOP-down approach is the stepwise process of breaking of large Program module into simpler and smaller modules to organise and code Program in an efficient way.



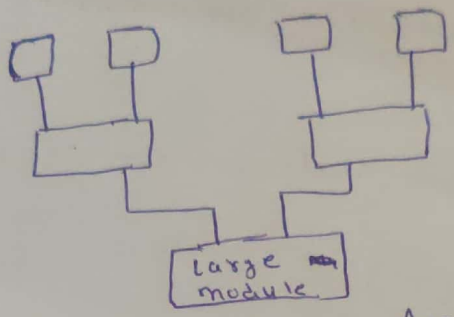
M-module

TOP-down approach

Bottom-up approach :-

→ Bottom-up approach begins with small modules and then combine them into a larger module.

→ Bottom-up approach is just the reverse of top-down approach.



Algorithm Complexity :- (Algorithm Analysis)

→ An algorithm is said to be efficient and fast if it takes less time to execute and consumes the less memory space, Easy to read, less line of code, less h/w, s/w needs.

→ Efficiency of an algorithm can be analyzed at two different types.

1. Time complexity
2. Space complexity.

(1) Time complexity :-

The time complexity of an algorithm is Total amount of Time Required by an algorithm to complete its execution.

→ Time complexity's are divided into 2 types

① Constant Time complexity

② Linear Time complexity.

① Constant Time complexity :-

If a program Requires a fixed amount of time for all input values is called constant time complexity.

Ex:- write algorithm to find the sum of N numbers (0 to n)

```
SumofNumbers (N)
{
1.  Sum = (N * (N+1)) / 2 ;
2.  Print( sum) ;
}
```

1. → 1 μsec
2. → 1 μsec

Total time : 2 μsec

② Linear Time complexity :-

If a Program requries a variable amount of Time for all input values is called linear Time complexity

Ex:- write algorithm to find the sum of N numbers (0 to N)

1, 2, 3, 4 → Varies Time 1 μsec

```
Sumof numbers (N)
{
1.  Sum = 0
2.  for (i=0 to N)
    {
3.  Sum = Sum + i
    }
4.  Print (Sum)
}
```

Ex:- N = 5

1 → 1 μsec
2 → 1 μsec
3 → n μsec
4 → 1 μsec

total = n+3 μsec

Time complexity Notations :

→ By using The value of worst case, Average case and Best case we can calculate The Time complexity.
① worst case ② average case ③ Best case.

① worst case :

→ In The worst case analysis, we calculate upper bound on running time of an algorithm.

→ we must know The case That causes maximum nomber of operations to be executed.

→ worst case is The maximum Number of steps on input data of size n.

Ex : In linear Search algorithm have 7 elements

Ex :

0	1	2	3	4	5	6
33	42	55	67	88	9	23

To find The 23 element In a list

→ worst case is Denoted by " Big O "

② Average case :

→ The running time of an algorithm is an estimate of The Running time for an "average input".

→ Average case is Denoted by " Big Θ "

Ex :

0	1	2	3	4	5	6
31	56	77	88	99	23	28

To find The 88 element In a list

(3) Best case :-

→ In the best case analysis, we calculate lower bound on running time of an algorithm.

→ We must know the case that causes minimum number of operations to be executed.

→ Best case is denoted by "Big Ω"

Ex :-

23	45	33	45	66

To find the element 23 in a list is a best case.

(2) Space Complexity :-

Whenever a solution to a Problem is written some memory is required to complete. for any algorithm memory may be used for the following.

1. variables
2. Program Instructions
3. execution.

Definition :- Space Complexity is the amount of memory and Produce the Result.

Space Required by an algorithm during to execute

Memory usage while execution.

Space complexity = Auxilary space + input space.

(1) Instruction space :-

It's the amount of memory used to save the compiled version of Instructions.

(2) Environment stack :-

→ Sometimes an algorithm (function) may be called Inside another algorithm (function). In such a situation, The current variables are pushed onto The system stack, where They wait for further execution and Then The call to The Inside algorithm is made.

Example :- if a function A() calls function B() inside it, Then all The variables of The function A() will get Stored on The system stack Temporarily, while The function B() is called and executed inside The function A().

(3) Data space :-

Amount of space used by The variables and constants.

→ Space complexity are divided Into 2 types.

(i) Constant space complexity
(ii) Linear space complexity.

(i) constant space complexity :-

constant space complexity algorithm Requires The fixed amount of space for all input values

Ex:- int add (n1, n2)
{
Sum = n1+n2
return Sum
}

Space:- n1 → 4 Bytes
n2 → 4 Bytes
Sum → 4 Bytes
Total = 12 Bytes
Auxilary space = 4 By
Total = 12+4=16 Bytes

② Linear space complexity:-

Linear space complexity algorithm Requires The Variable amount of space for all input values.

Ex:- Add (a[], n)
{
1 Sum = 0
2 for (i=0 to n)
3 {
Sum = Sum + a[i]
}
4 Print (Sum)
}

arr = N x 4 Bytes
sum = 4 Bytes
i = 4 Byte
Auxilalary space = 4 Bytes
Total : 4n + 12
n=2 = 8 + 12 = 20
n= 3 ⇒ 12 + 12 = 24
n= 4 ⇒ 16 + 12 = 28

Auxilary space :- It is extra (or) Temporary space used by algorithm during It's execution.

Types	Size
① boolean, char, unsignedchar, signed char, int_8	1 Byte
② int_16, short, unsigned short	2 Bytes
③ float, int32, int, unsigned int, unsigned long, long	4 Bytes
④ double, int_64,	8 Bytes

Arrays :- An array is a collection of similar data elements, That are stored under a common name. All elements in array are stored at continous memory locations.

In an array is identified by index or subscript enclosed in squre brackets with array name.

Ex:- Test marks of a class of students
List Employees in an organization.

Syntax :- datatype arrayname [array-size];

int a[5];

Types of Arrays:- The Arrays can be classified into 3 types

1. one-dimensional arrays (1-D)
2. Two-dimensional arrays (2-D)
3. ~~Multi~~ Large dimensional arrays (or) (multi dimensional arrays)

① one-dimensional arrays (1-D) :-

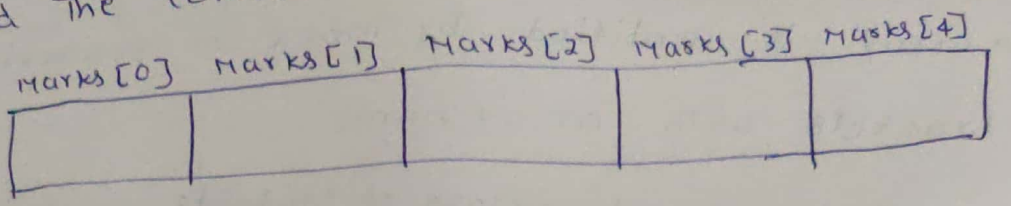
Array Declaration:- Arrays are declared in the same manner as an ordinary variables except that each array name must have the size of the array. like variables The array must be declared before they are used.

Syntax :- datatype arrayname [array-size];

From The above Syntax
The datatype specifies The type of The data, That will be contained in the array. Array name specifies the name of the array. array-size specifies The maximum number of elements That The array can hold.

Ex:- int marks[5]

Here marks is the name of the array with 5 elements of integer datatypes and the computer reserves five storage locations.



Array Initialization:- Like variables, initialization of the elements in a fixed-length array can be done when it is defined variable-length.

~~Array~~ After an array is declared, its elements must be initialized. otherwise, then will contain "garbage". An array can be initialized at either of the following stages.

1. At compile time
2. At run time.

① Compile time Initialization:-

we can initialize the elements of arrays in the same way as the ordinary variables when they are declared.

Syntax:- datatype arrayname [size] = { list of values };

(i) **Basic Inilization:-** int a[5] = { 1, 2, 7, 8, 9 };

(ii) **Initilation without size:-** int a[ ] = { 1, 2, 3, 10, 11, 12 };

(iii) **Partial Inilization :-** int a[10] = { 1, 2, 3, 4 };

char name[ ] = { 'k', 'r', 'i', 's', 'h', 'n', 'a' };

(or)

char name[ ] = "krishna";

(iv) **Initialization to all zero :-**

int a[5] = {0};

② Run time Initialization :- An array can be Explicitly Initialized at run time. This approach is usually applied for Initializing large arrays for example.

(85)

```
for   int a[5]
```

O/p screen

Enter values :- 1, 2, 7, 8, 9

Reading / Inputing values of an arrays :-

Another way to fill the array is to read the values from the keyboard. This method reading values can be done using loop.

Ex :

```
for(i=0; i<5; i++)
    scanf("%d", &a[i]);
```

from the above example first we start the Index, i at 0; since the array has 5 elements, we must load the values from index location 0 through 4.

Printing values of an arrays :-

Another common application is Printing the elements of an array. This is ~~done~~ easily done with for loop as shown below.

```
for (i=0; i<5; i++)
    Printf ("%d", a[i]);
```

Example :- change values location 2 and location 4

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int a[10], n, i, temp;
```

```
Printf ("Enter size of array");
Scanf ("%d", &n);
Printf ("Enter elements of an array");
for (i=0; i<n; i++)
Scanf ("%d", &a[i]);
Printf ("before array elements are \n");
for (i=0; i<n; i++)
Printf ("%d", a[i]);
Printf ("After changing the array elements are \n");
temp = a[2];
a[2] = a[4];
a[4] = temp;
for (i=0; i<n; i++)
Printf ("%d", a[i]);
}
```

O/P :- Enter Size of array 5

Enter element of array

	0	1	2	3	4
a	10	20	30	40	50

Before array elements are

10 20 30 40 50

After changing The array elements are

~~10 20 30 40 50~~

10 20 50 40 30

Two-dimensional Arrays:-

In one-dimensional array data are organized linearly. in only one direction. Many Applications require that the data to be stored in morethan one-dimensional one common example is table, which is an array that consists of rows and columns. It is called 2-D array.

int table[2][3] = {{0,0,0},{1,2,3}}
(2 = rowsize, 3 = column size)

Syntax: int arrayname[rowsize][columnsize];

Declaration:- 2-D arrays like 1-D arrays must be declared being as Two Pairs of Subscripts/ Index required for 2-D array.

Syntax:- Int arrayname[rowsize][columnsize];

From the above syntax datatype specifies the type of data, arrayname specifies name of the array.

Rowsize specifies the size of row.

columnsize specifies the size of column.

Ex: int table[3][3]

Here table is the name of the array with rowsize 3 and column size 3 totally This array contains stored 9 elements.

	col 0	col 1	col 2
row 0	table[0][0]	table[0][1]	table[0][2]
row 1	table[1][0]	table[1][1]	table[1][2]
row 2	table[2][0]	table[2][1]	table[2][2]

Initialization:-

Initialization of array elements can be done when the array is defined one way to Initiate. it is shown below

(i) int a[3][2] = {10, 20, 40, 3, 6, 2};

(ii) int a[3][2] = { {10, 20},
{40, 3},
{6, 8} };

(iii) Initialize the array that omits the first dimension.

int a[ ][2] = {{10,20}, {40,3}, {6,8}};

(iv) Partial Initialization :

int a[5][4] = { {10,20},
{40,3},
{6,8} };

(v) to Initialize the whole array as zero.

int a[5][4] = {0};

Reading :- Another way to fill up the values is to read elements from the keyboard. for a 2-D array, This usually requires a nested for loops. If array is an nxm array, The 1st loop varies, the row from 0 to n-1, The second loop column from 0 to m-1.

```
for (row=0; row<n ; row++)
  {
  for ( col=0; col<m ; col++)
    {
    scanf ("%d", &a[row][col]);
    }
  }
```

Printing values :- We can also display the values one by one, using two nested for loops.

```
for (row=0; row<n; row++)
  {
   for ( col=0; col<m ; col++)
   {
    Printf (" %d", a[row][col]);
   }
  }
```

Example :- Addition of two matrices.

```
#include <stdio.h>
#include<conio.h>
Void main()
{
int a[5][5], b[5][5], c[5][5];
int r1, r2, c1, c2, i, j;
Printf(" Enter row & col size of two matrices");
scanf("%d %d %d %d", &r1, &r2, &c1, &c2);
ib(r1==r2 && c1==c2)
{
Printf(" Enter first matrices elements");
for(i=0; i<r1; i++)
{
for(j=0; j<c1; j++)
 scanf("%d", &a[i][j]);
}
Printf (" Enter second matrices elements");
for( i=0; i<r2; i++)
 {
 for(j=0; j<c2; j++)
 scanf(" %d", &b[i][j]);
 }
 Printf(" Addition of The result matrices");
  for(i=0; i<r1; i++)
  {
   for(j=0; j<c1; j++)
   {
    c[i][j] = a[i][j] + b[i][j];
    Printf("%d", c[i][j]);
   }
   Printf("\n");
  }
}
```

```
else
    Printf (" Addition is not Possible");
}
```

O/P :- Enter row & col size of two matrices

2 2 2 2

Enter 1st matrix elements

1 1

1 1

Enter 2nd matrix elements

2 2

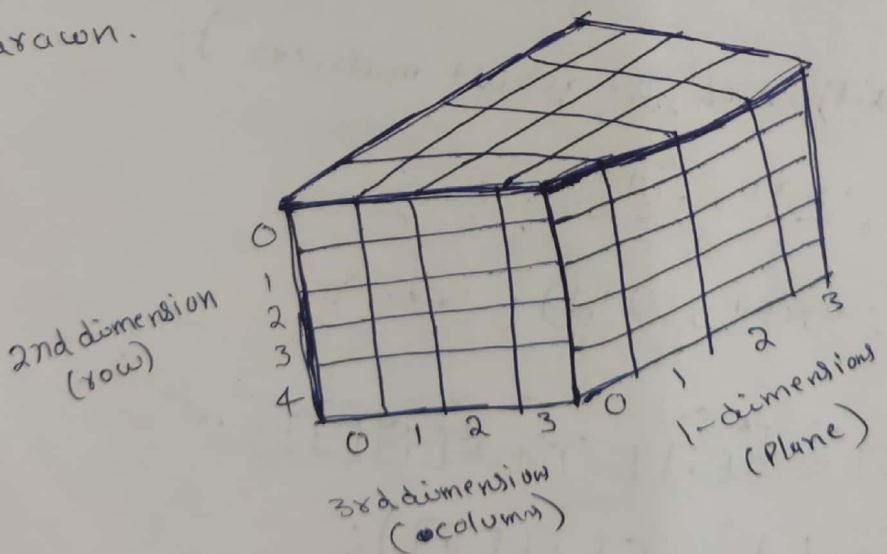
2 2

Addition of result matrix

3 3

3 3

Large ~~Multi~~ dimensional Arrays :- Multidimensional arrays can have three forms (or) more dimensions. Arrays three forms or more dimensions can be created and used, but they are difficult to drawn.



Two dimensional array to be an array of 1-D array.

3-D array is an array of array of arrays.

#include <stdio.h>

Declaration:-

Multi dimensional arrays, like 1-D arrays must be declared before being used.

Syntax:-

datatype arrayname [Plane] [row] [col];

datatype → specifies the type of the data

arrayname → specifies the name of the array.

Plane ~~size~~ → specifies the size of Plane

row → specifies the size of row

col → specifies the size of column.

Ex:- int a[3][5][4];

The definition for the array a requires 3 dimensions ~~one~~ one for row, one for column, one for the Plane

Initialization:-

Int a[2][2][2] = {1,2,3,4,5,6,7,8}

(or)

Int a[2][2][2] = {0};

int a[3][2][3] = {1,2,3,4,5};

Reading:-

```
for (i=0; i<Plane; i++)
{
  for (j=0; j<row; j++)
  {
    for (k=0; k<col; k++)
    {
      scanf("%d", &a[i][j][k]);
    }
  }
}
```

Printing:-

```
for (i=0; i<Plane; i++)
{
  for (j=0; j<row; j++)
  {
    for (k=0; k<col; k++)
    Printf("%d", a[i][j][k]);
```

Strings

A string is a sequence of characters that is treated as a single data item. string is also called as character array. string each character is stored in one byte of memory

Declaring & Initializing string variables:-

C does not support strings as a datatype. However it allows us to represent strings as character arrays. Therefore, a string variable is any valid 'c' variable name and is always declared as an array of characters. The general form of declaration of string variable is.

syntax :- datatype stringname [size];

datatype specifies the type of the data
stringname specifies the name of the string
size specifies the number of characters in the string name

Ex :- char s[10];
char name[30];

When the compiler assigns a character string to character array, it automatically supplies a null character ('\0') at the end of the string. Therefore the size should be equal to the maximum number of characters in the string plus one.

Ex :- char city[9] = "New York";
char city[9] = {'N', 'E', 'w', ' ', 'Y', 'O', 'R', 'k', '\0'};

Note :- string must be enclosed with in "double quotes"
Character must be enclose with in 'single quotes'

C also Permits to Initialize a character array with out specifiying the number of elements. In such cases, The size of the array will be determined automatically based on the number of elements Initialized.

Ex:- char name[] = {'w', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};

String Input / output functions:-

C Provides two basic ways to read & write strings

1. formatted Input / output
2. unformatted Input / output

1. formatted Input / output :-

(i) formatted Input :- we read the strings using the formatted fonction scanf. The formal specifier for string is "%s". first They skip any leading white space. once They find a character, They read until The find white space, putting each character In The array In order. when They find a strating whitespace character, They end of the string with a null character.

Ex :- char name[10];

Syntax :- scanf ("formatted specifier", stringname);

scanf ("%s", name);

→ for Example to read a string such as city, form the keyboard, we could simply write the statement as below

```
scanf ("%s", city);
```

An address operator is not required for ~~name~~ city, since it is already a pointer constant.

(ii) formatted output:- we Print (or) display the strings using The formatted output function is Printf. The format specifier for string is "%s".

Syntax :- `Printf ("%s", stringname);`

for Example to display a string such as name, we could simple write the statement as follows.

```
Printf ("%s", name);
```

Example :-

```
# include <stdio.h>
# include <conio.h>
void main()
{
 char name[20];
 Printf (" Enter a ~~string~~ name");
 scanf ("%s", name);
 printf ("%s", name);
}
```

O/P :- Enter a name: Venkatesh

Venkatesh.

② Unformatted Input / output :-

(i) Unformatted Input :- to read a character from the keyboard using the function **getchar()**. we can use this function repeatedly to read successive single character from the input and place them into a character array. The reading is terminated when the newline character ('\n') is entered and the null character is then inserted at the end of the string.

general form :-

```
char name;
name = getchar();
```

getchar() function has no parameters.

Another and more convenient method of reading a string of text containing whitespaces is to use the library function **gets** available in the <stdio.h> header file.

general form :-

```
char name[10];
gets(name);
```

(ii) unformatted output :- to display the character using the function is **putchar()**. The function putchar() requires one parameter.

```
char ch = 'a';
putchar(ch);  /* printf("%c", ch); */
```

Another way to display a string of text containing whitespaces is to use **puts()** function This is simple function with one string parameters and called as under

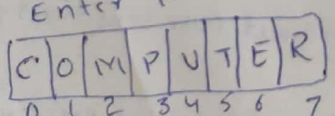
Ex :- `puts(name);`

Using Putchar() & getchar() :-

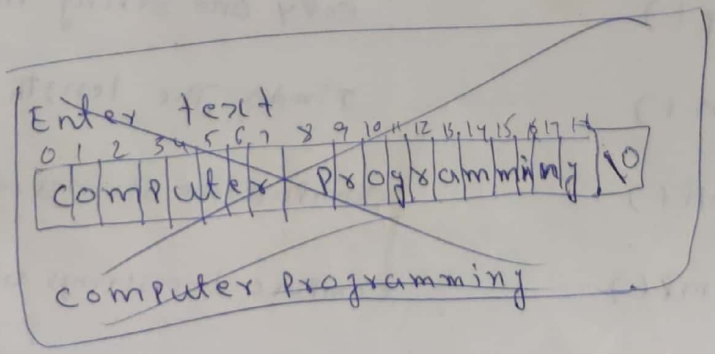
```
#include <stdio.h>
Void main()
{
char name[30], ch;
int c=0;
Printf("Enter text");
do
{
 ch = getchar();
 name[c] = ch;
 c++;
}
while (ch != '\0');
 c=0;
do
{
 ch = name[c];
 Putchar(ch);
 c++;
}
while (ch != '\0');
}
```

O/P :-

Enter text



computER



② using Putchar() & getchar

```
#include<stdio.h>
#include<conio.h>
Void main()
{
 char name;
 clrscr();
 Printf("Enter a character");
 name=getchar();
 Putchar(name);
 getch();
}
```

O/P Enter a character:

A

A

using gets() & puts() :-

```
#include <stdio.h>
Void main()
{
 char name [30];
 Printf (" Enter text");
  gets (name);
  puts (name);
  }
```

O/P :- Enter text
computer
computer.

String manipulations :- String operations :-

String Handling functions :- The C library provides a large number of string handling functions that can be used for performing string manipulations. These functions are avaible In the header file <string.h>. commonly used string handling functions.

Function	Action
Strcat()	concatenates two strings
Strcmp()	compare two strings
strcpy()	copy one string into another
strlen()	Finds the length of string
Strcmpi()	compare two strings without case sensitivity
strncmp()	compare two strings upto specified position
strncmpi()	compare two strings positions & case sensitivity

~~strcat()~~ strcat :- This function is used to join two strings.

Syntax :- strcat (string1, string2);

The strcat() function can be used to combine all contents of the two strings together.

Ex:-

string1 = | A | R | R | A | Y | S | \0 |

string2 = | S | T | R | I | N | G | S | \0 |

strcat (string1, string2);

| A | R | R | A | Y | S | S | T | R | I | N | G | S | \0 |

difference b/w strcat() and strncat() functions :-

Ex:- char st1[30] = "c programming";

char st2[30] = "classes";

char * st3;

st3 = strcat (st1, st2);

Printf ("%s", st3);

Result :- C programming classes

If you want to combine only a number of characters of the string2 with the string1 you can use the strncat() function

Syntax :- strncat (string1, string2, num of chars);

Ex:- char st1[30] = "hello"

char st2[30] = "howareyou"

char *st3;

st3 = strncat (st1, st2, 5);

Printf ("%s", st3);

Result :- hellahowar

Suppose combine 3 strings as follows.

st1 = hai st2 = how st3 = Are u

First join string1 and string2

strcat (st1, str2) will result in "hai how"

strcat (strcat (st1, st2), st3);

concatenates of 3 strings result is " hai how Are u"

strcmp() :- This function is used to compare two strings.

Syntax :- strcmp (string1, string2);

Here both string1 and string2 are character arrays. string1 and string2 are compared as per ASCII collating sequence. The strcmp() retu... an integer value according to the following rules.

) if string1 is equal to string2, value 0 is returned.

Ex :- strcmp ("Hai", "Hai");

) if string1 is greater than string2 positive value is returned

Ex :- strcmp ("cprogram", "class");

if string1 is less than string2, negative value is returned

Ex :- strcmp ("class", "c...ogram");

strcpy() :- This function is used to copy one string to another string variable. Almost it works like string assignment & operator.

Syntax :- strcpy (string1, string2);