INDEX

Sr.No.	Title of Topic	Page No.
1).	Data structure introduction.	3
2).	classification of data structure	7
3).	introduction to algorithm	9
4).	asymptotic analysis	15
5).	DS- pointer	18
6).	DS- structure	20
7).	DS- Array	23
8).	DS- Linked list	30
9).	DS-skip list	36.
10).	DS-stack	41
11).	DS-Queue	44
12)	DS-Tree	48.
13).	Types of Tree	58
14).	DS-Graph	62

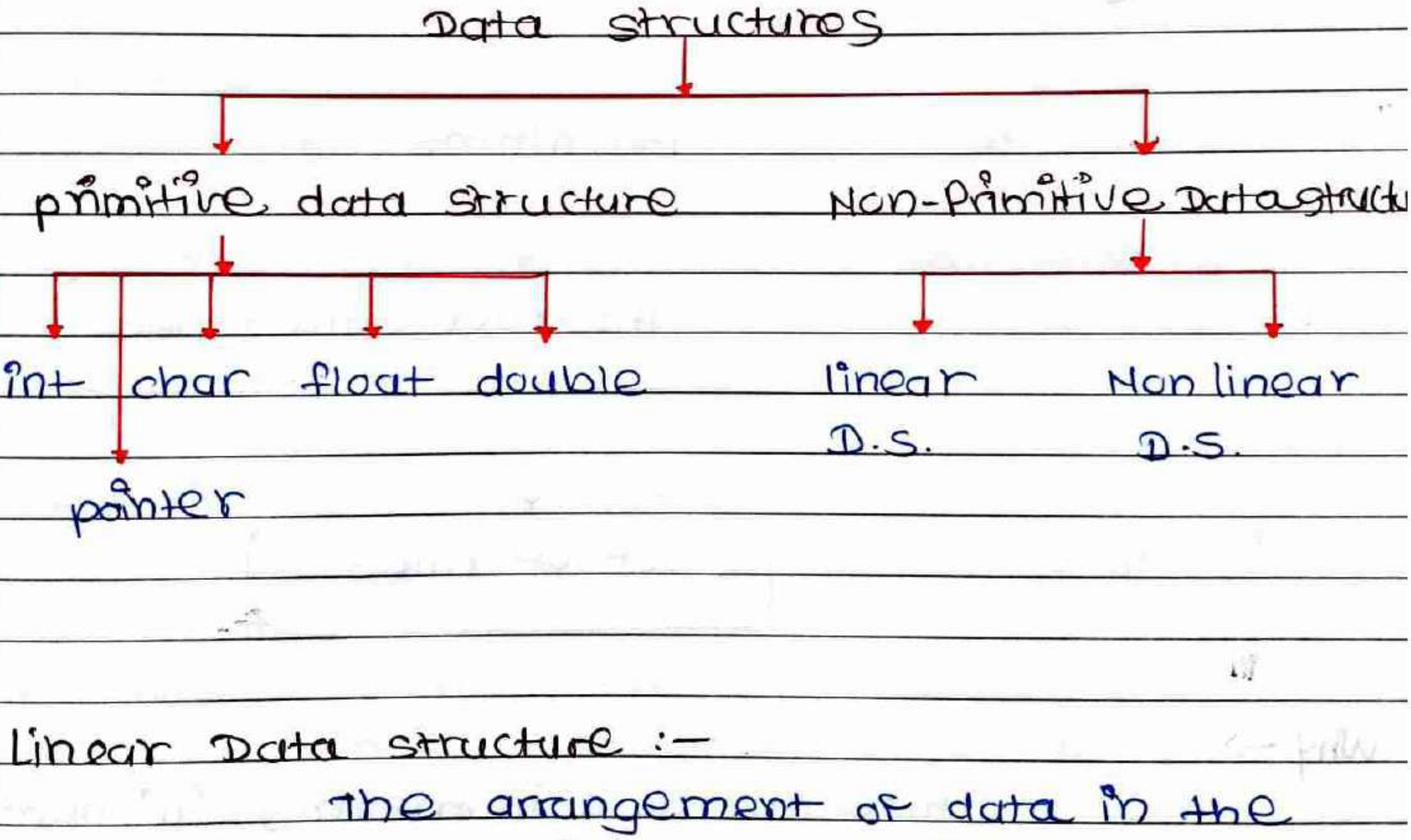
Sr. No	Topic Name	Page No.
15>.	Graph Traversal Algorithms	65
16>.	Searching	71
17).	Searching Algorithms with example	82
18).	sorting Algorithms	86.
19).	Implementations of sorting Algorithms	89.
20>.	Data structure interview questions with Answers (50 questions with Answers)	95.
21>.	DATA STRUCTURE CODING QUE.	104.
22>.	END	

What is Data structure ?

→ Data structure is a way to store and organize data so that it can be used efficiently.

As per name indicates itself that organizing the data in memory.

The data structure is not any programming language like c, c++, Java etc. It is set of algorithms that we can use in any programming language to structure data in memory.



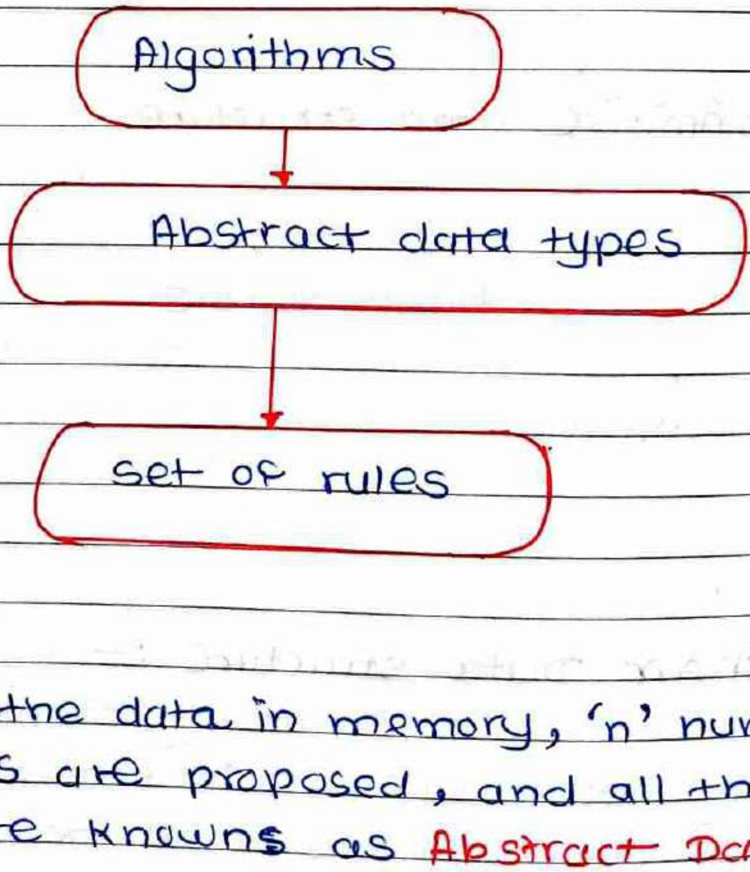
Linear Data structure :-

The arrangement of data in the sequential manner is known as linear data structur The data structure used for this purpose are Arrays, linked list, stacks and queues.

In this data structures, one element is connected to only one another element in a linear form.

Non-linear data structure :-
When one element is connected to the 'n' number of elements known as non-linear data structures.
Example :- trees and graphs.
In this case, elements are arranged in a random manner.

Algorithms and Abstract Data types ??



why →

TO structure the data in memory, 'n' number of algorithms are proposed, and all these algorithms are knowns as Abstract Data Types.

An Abstract Data Type tells what is to be done and data structure tells how is to be done ?

ADT gives us the blueprint while data structure provides the implementation part.

What is Data ?

Data can be defined as the elementary value / collection of values.
forexample :- student's name and its id are the data about student.

What is Record ?

Record can be defined as collection of various data items
example :- student entity→ name, address, course and marks can be grouped together to form record.

What is File ?

File is a collection of various records of one type of entity
example :- if there are 60 employees in class, then there will be 20 records in related file where record contains info of employee

What is Attribute and Entity ?

An entity represents class of certain objects. it contains various attributes . each attribute represents particular property of that entity.

What is need of data structures ?

As applications are getting complexed and amount of data is increasing day by day, there may arrise following problems :-

Processor speed :- As data is growing day by day to the billions of files per entity, processor may fail to deal with that amount of data.

Data Structure :- consider an inventory size of 106 items in store, if our application needs to search for a particular item, it needs to transverse 106 items every time, results in slowing down process

multiple requests :- If thousands of users are searching data simultaneously on a web server, then there are chances that to be failed to search during that process.

To solve this problems, data structures are used. Data is organized to form a data structure in a such way that all items are not required to be searched and require data can be searched instantly.

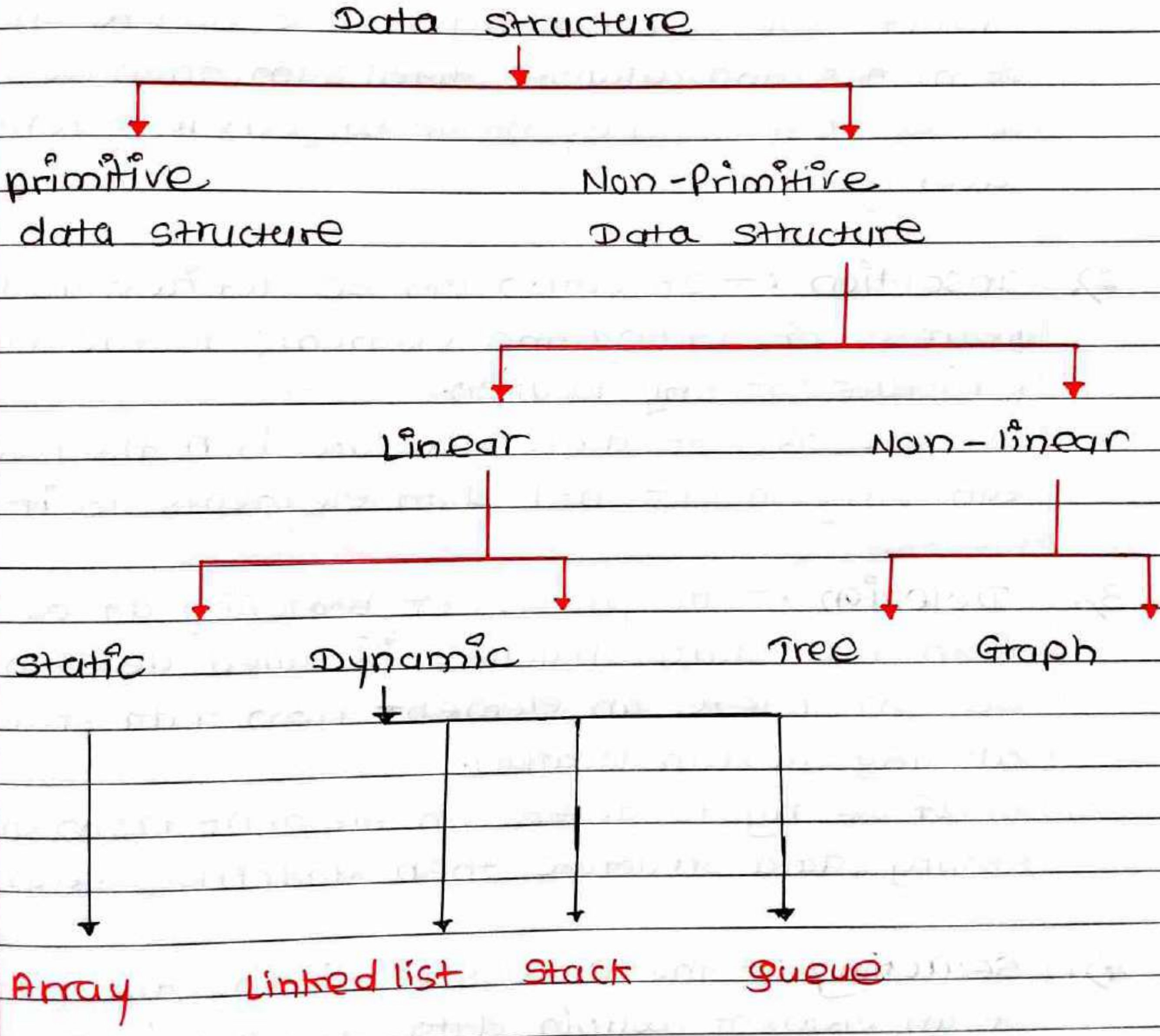
Advantages of data Structure :-

Effiency :- If the choice of a data structure for implementing a particular ADT is proper, it makes program very efficient in terms of time and space.

Reusability :- The data structure provides reusability means that multiple client programs can use the data structure.

Abstraction :- The data structure specified by the ADT also provides level of abstraction. The client cannot see interval working of data structure, so it does not have to worry about implementation.

* Data structure classification :-



Operations on data structure :-

1). Traversing :- Every data structure contains a set of data elements. Traversing data structure means visiting each element of data structure in order to perform some specific operation like searching or sorting.

Example :- If we need to calculate average of marks obtained by a student in 6 different subject, we need to traverse complete array of marks and calculate total sum, then we will devide that sum by no. of subjects ie. 6 to find average.

2). Insertion :- Insertion can be defined as the process of adding the elements to the data structure at any location.

If the size of data structure is n then we can only insert n-1 data elements to it.

3). Deletion :- The process of removing an element from the data structure is called deletion. we can delete an element from data structure at any random location.

If we try to delete an element from an empty data structure then underflow occurs.

4). searching :- The process of finding the location of an element within data structure is called searching. There are two algorithms to perform searching, linear search and Binary search.

5). Sorting :- The process of arranging the data structure in a specific order is called as sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort etc.

6). merging :- When two lists list A and list B of size M and N respectively, of similar type of elements, clubbed or joined to produce third list, list C of size (M+N), then this process is called merging.

DATA STRUCTURES AND ALGORITHM

What is Algorithm?

An algorithm is a process or a set of rules required to perform calculations or some other problem-solving operations especially by a computer.

It is not complete program or code; it is just a solution (logic) of a problem, which can be represented either as an informal description using a flowchart or pseudocode.

characteristics of an algorithm.

Input :- An algorithm has some input values. We can pass 0 or some input value to an algorithm.

Output :- We will get 1 / more output at end of an algorithm.

Unambiguity :- An algorithm should be unambigous which means that instruction in an algorithm should be clear and simple.

Finiteness :- An algorithm should have finiteness. means limited number of instructions.

Effectiveness :- An algorithm should have finite as each instruction in an algorithm affects the overall process.

Approches in Algorithm :-

1). Brute force Algorithm :- The general logic structure is applied to design an algorithm. It is also known as exhaustive search algorithm that searches all possible to provide required solution.

Such algorithms have two types :-

1). optimizing
finding all solutions of a problem and then take out the best solution is known then it will terminate if the best solution is known.

2). Sacrificing
As soon as the best solution is found, then it will stop.

Divide and conquer :- This breaks down the algorithm to solve the problem in different methods. It allows you to break down problem into different methods, and valid output is produced for the valid input. This valid output is passed to some other function.

Greedy algorithm :- It is an algorithm paradigm that makes an optimal choice on each iteration with the hope of getting best solution. It is easy to implement and has faster execution time. But there are very rare cases in which it provides the optimal solution.

The major categories of algorithms are given below:

Sort :- Algorithm developed for sorting the items in a certain order.

Search :- Algorithm developed for searching the items inside a data structure.

Delete :- Algorithm developed for deleting the existing element from the data structure.

Insert :- Algorithm developed for inserting an item inside a data structure.

Update :- Algorithm developed for updating the existing element inside a data structure.

Algorithm Analysis :-

The algorithm can be analyzed in two levels i.e. first is before creating the algorithm, and second is after creating the algorithm. There are two analysis of an algorithm.

Priori Analysis :-

Here, priori analysis is the theoretical analysis of an algorithm which is done before implementing the algorithm.

Posterion Analysis :-

Here, posterior analysis is a practical analysis of an algorithm. The practical analysis is achieved by implementing algorithm using any programming language.

Algorithm complexity :-

The performance of the algorithm can be measured in two factors :

Time complexity :-

The time complexity of an algorithm is the amount of time required to complete the execution. The time complexity of an algorithm is denoted by the big O notation.

Here big O notation is the asymptotic notation to represent time complexity. The time complexity is mainly calculated by counting the number of steps to finish execution.

```
sum = 0 ;
// suppose we have to calculate the sum of n
 numbers.
for i=1 to n
sum = sum + i ;
// when the loop ends then sum holds the sum
of n numbers.
return sum;
```

In above code, the time complexity of the loop statement will be atleast n, and if value of n increases, then time complexity also increases.

We generally consider the worst-time complexity as it is maximum time taken for any given input size.

Space complexity :-

An algorithm's space complexity is the amount of space required to solve a problem and produce an output. simillar to the time complexity, space complexity is also expressed in big O notation.

Space complexity = Auxiliary space + Input size.

The following are the types of algorithms:

Search Algorithm :-

on each day, we search for something in our day to day life.

similarly, with the case of computer, huge data is stored in a computer that whenever user asks for any data then the computer searches for that data in the memory and provides that data to the user. There are mainly two techniques available to search data in an array:

- Linear search
- Binary search

sorting Algorithms :-

sorting algorithms are used to rearrange elements in an array or a given data structure either in an ascending or desending order. The comparison operator decides the new order of the elements:

Asymptotic Analysis :-

The time required by an algorithm comes under three types :

Worst case :- It defines the input for which the algorithm takes a huge time.

Average case :- It takes average time for the program execution.

Best case :- It defines the input for which the algorithm takes the lowest time.

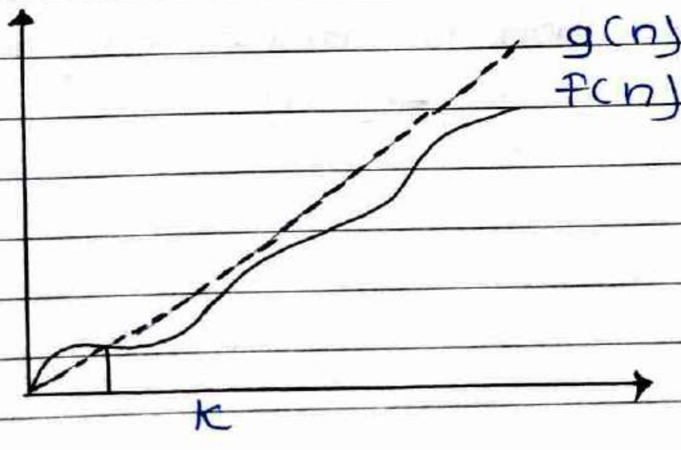
Asymptotic Notations :-

The commonly used asymptotic notations used for calculating the running time complexity of an algorithm is given below :

1). Big oh notation (O) :-

This measures the performance of an algorithm by simply providing the order of growth of the function.

This notation provides an upper bound on a function which ensures that function never grows faster than the upper bound.



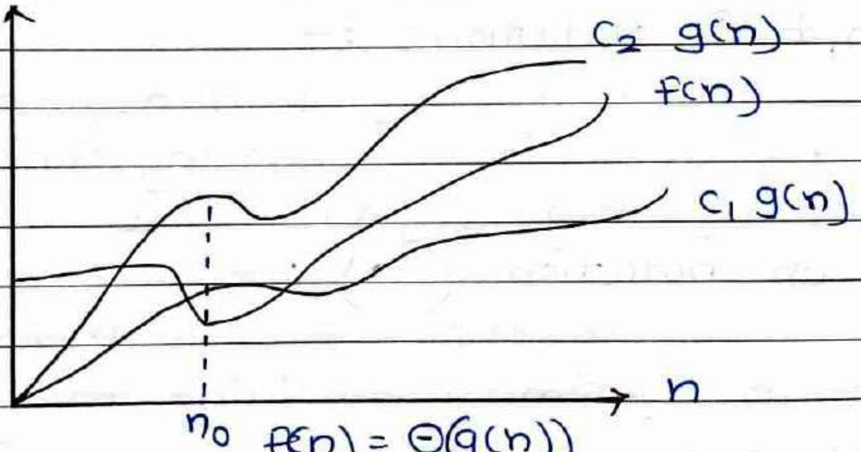
Example :- If f(n) and g(n) are two functions defined for positive integer,

then $f(n) = O\ g(n)$ as f(n) is big oh of g(n) or f(n) is on order of g(n)) if there exists constants c and no such that :

$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

2). Omega Notation (Ω) :-

It basically describes best case scenario which is opposite to big O notation. It is the formal way to represent lower bound of an algorithm's running time.



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

Example :- let f(n) and g(n) be functions of n where n is steps required to execute program:

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

The above condition is satisfied only if when:

$c1. g(n) <= f(n) <= c2.g(n)$

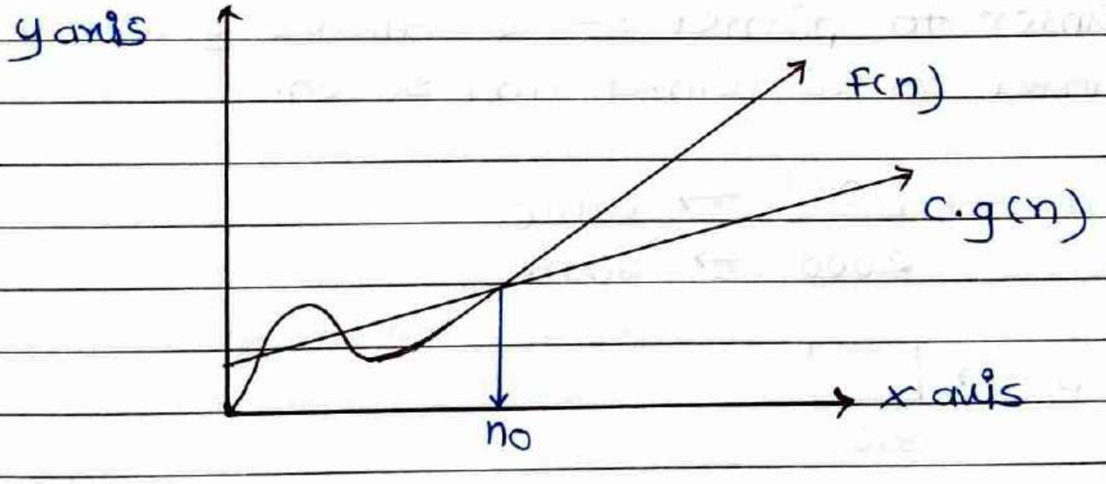
2). omega Notation (Ω)

It basically describes best-case scenario which is opposite to big-o notation It is formal way to represent lower bound to an algorithm's running time. It measures the best amount of time an algorithm can possibly take to complete or best case time complexity.

Example :- If f(n) and g(n) are two functions defined for positive integers,

then $f(n) = \Omega g(n)$ as f(n) is omega of g(n) or f(n) is on the order of g(n) if there exists constants c and no such that :

$f(n) >= c \cdot g(n)$ for all $n \geq n_0$ and $c > 0$



3). Theta Notation (Θ)

The theta notation mainly describes average case scenarios.

It represents realistic time complexity of an algorithm. Big theta is mainly used when the value of worst case and best case is same.

Pointer :-

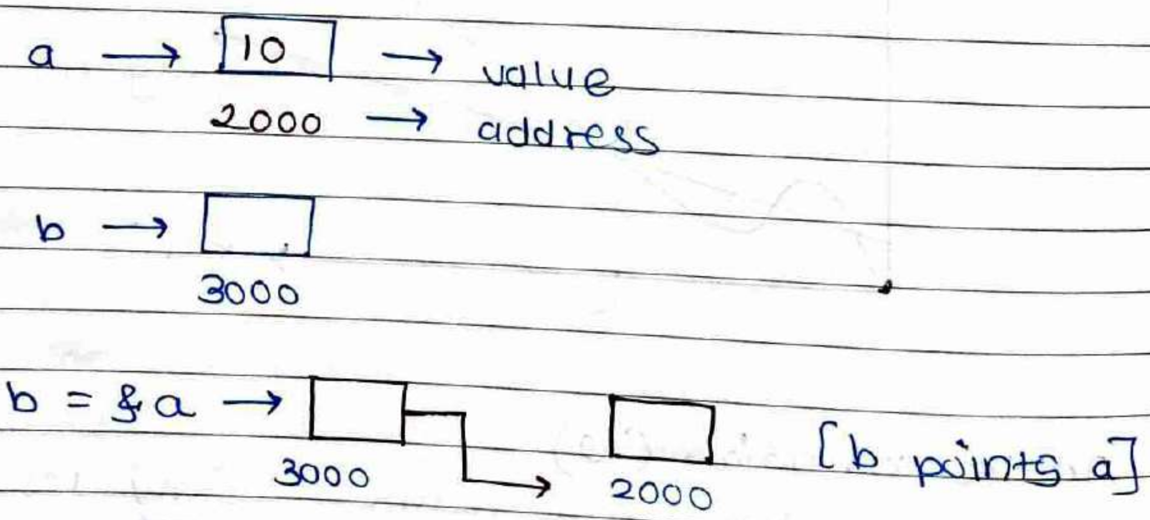
Pointer is used to points the address of the value stored anywhere in the computer memory. To obtain the value stored at location is known as dereferencing pointer.

Pointer arithmatic :-

4 arithmatic operators that can be used in pointers : ++, --, +, -

Array of pointers :- You can define array of to hold a number of pointers.

Pointer to pointer :- c allows you to have pointer on a pointer and so on.



Program
Pointer →

```
#include < stdio.h>
int main ()
```

```
{
 int a = 5 ;
 int *b ;
 b = &a ;
 printf ("value of a = %d \n", a);
 printf ("value of a = %d \n", *(&a));
 printf ("value of a = %d \n", *b);
 printf ("address of a = %u \n", &a) ;
 printf ("address of a = %d \n", b);
 printf("address of b = %u \n", &b)
 printf("value of b = address of a = %u", b);
 return 0 ;
}
```

output →

```
value of a = 5
value of a = 5
address of a = 3010494292
address of a = -1284473004
address of b = 3010494296
value of b = address of a = 3010494292.
```

Program :-
Pointer to pointer :-

```
#include < stdio.h>
int main ()
 {
  int a = 5 ;
  int *b;
  int **c ;
b = &a;
c = &b;
printf("value of a = %d \n", a);
printf("value of b = address of a = %u \n", b);
printf("value of c = address of b = %u \n", c);
printf("address of b = %u \n", c);
printf("address of c = %u \n", &c);
return 0;
}
```

output →

```
value of a = 5
value of b = address of a = 2831685116
value of c = address of b = 2831685120
address of b = 2831685120
address of c = 2831685128.
```

Structure :-

A **structure** is a composite data type that defines a grouped list of variables that are to be placed under one name in block of memory.

Program :-
structure →

```
struct structure-name
{
 data-type member 1;
 data - type member 2;
```