PPS

Unit - 1

Algorithm
Flow Chart
Basic components
Syntax & Logical error

Byte, Bit
Byte = 8 Bits
KB = 1024 Bytes
MB = 1024 KB
GB = 1024 MB

Introduction to Programming

It's the mental process of thinking instructions to give to a machine where instruction is designed an order to given to computer by a computer program
It is a set of rules that provides a way of telling a computer what operations to perform.
It provides the linguistics framework for describing computations.

Ques. Difference between machine level and assembly language

Machine Level	Assembly Language
→ It cannot be easily understand by humans	→ It is easy to write, read and maintain
→ It is written in binary form	→ It is written in english language
→ It does not require any translator	→ Assemble is used to convert assembly code into machine code.

☆☆ (Imp for exam) Difference between low level language and high level language.

Low level	High level
→ It is a machine friendly language	→ It is user friendly language
→ It takes more time to execution	→ It execute at high pace
→ Debugging hard	→ Debugging easy
→ It is not portable	→ It is portable
→ Memory efficient	→ Less memory efficient
→ It requires assemble for translation	→ It requires compiler or interpreter for translation

V.V Imp for exam ✓

RAM	ROM
→ Random Access Memory	→ Read only memory
→ It is volatile	→ It is non-volatile
→ Contains are temporary	→ Contains are permanent
→ High processing speed	→ Low processing speed
→ User difind program	→ operating systems program stored in ROM
→ Static and dynamic RAM	→ Programming ROM & EP ROM

★★ OPERATING SYSTEM (V. V. Imp topic)

(10 marks)

OS is the most important software that runs on a computer. It manages computer memory and process, as well as software and hardware. In addition it also allow to communicate with computer without knowing how to speak the computer language

Function of operating system
- File management
- Memory management
- Process management
- Handling input and output device management
- Storage management

Criteria in a good language design.

① Writability
It specifies the quality of a language that labels a programmer to use it correctly, concisely and quickly.

② Readability
Quality of a language which enables a programmer to understand the nature of computer easily and accurately.

③ Orthogonality
This parameter defines the features provided have few restrictions as much possible.

④ Liability
Quality of language that assure a program will not behave in a distraous way using execution.

(5) Maintainability
Quality of language which deals with ease of error finding and correction.

(6) Uniformity
It describes the similar features should behave familiar and look familiar

(7) Standardability
Quality of program that allows program written to be transferred from one to another without changing language structure.

(8) Implementability
Quality of language that provides a translator, interpreter and be written.

Computer along with components (IMP)
ALU, CU (IMP)

Bas knowledge honi chaiye exam mei ni aaega

- Data
- Information
- CPU
- Input devices, Output devices
- Memory Unit
- Computer Disk
- Hard Disk
- Floppy Disk

C - Language

→ Invented by Dennis Ritchie for creating system applications that directly interact with hardware device such as drivers or kernals.

C can be defined by following types

(1) As a mother language :- Because it is considered that all the modern programming language, compilers, JVM, Kernels are written in C language.

(2) C as a system programming language :-
Since a system programming language is used to creat system software. So, C can be used to do low level programming
Eg. Linux Kernel is with in C

(3) C as procedural language
Procedural language specifies a series of set for the program to solve the problem.

(4) System Programming language
Since a system programming language is used to cre system software. So, C can be used to do low level programming.
Ex. Linux is written in C.

(5) C as a procedural language
Procedural language specifies, series of step to solv the programe. It breaks the program into function data structures.

(6) C as structured programming language

Structured programming language is a subset of procedural language. Structure means to break a program into part or block.

Parameters

Time complexity → in how much time it will give output.

Space - how much space it occupied.

(V.V. imp for exam)

Compiler	Interpreter
→ Its done in one go.	→ Its done line by line
Ex: Turbo	Ex: Assembly line.

Problem Solving steps

Step 1 - Define the problem

Step 2 - Formulate the mathematical model

Step 3 - Develop an algorithm

Step 4 - Write the code of problem

Step 5 - Test the code

Define the problem

→ Example:- To calculate addition of two numbers.

Step - 1 Define the problem

A clear and concise problem statement is provided. ~~the~~ Problem definition should specify input and output.
eg. To find the average of three numbers.

Step 2 - Formulate the mathematical model
Any technical problem provided can be solved mathematically full knowledge about the problem should be provided along with the mathematical concept.
ex:- (Data 1 + Data 2 + Data 3)/(3)

Step 3 - Develop an algorithm
An algorithm is a sequence of operation to be perform. It provide the precis problem.
ex:- Problem defination → To find avg. of 3 numbe

Algorithm 1: Set sum of data value to 0
As long as the data value exist add the next data value to sum and add another data value to count.
To compute the average divide the sum by count.
Print the avg.

Step 4 - Write the code of problem
→ The algorithm developed must be converted to any programming language.

→ Compiler will convert the program code to machine language which the computer can understand.

Step 5 - Test

Testing involves checking errors in both ways that is systematically and symmetrically

Errors are called as Bugs.
When the compilers find the bugs it prevents the compling the code from programming language to machine language.

Check the program for checking set of data for testing.

** ALGORITHM (This will come in exam)
It is defined as a step by step procedure which defines, a set of instructions to be executed in a certain order to get the desired output.
Algorithm generally created independent of languages

ex:- To determine the largest number output of three numbers A, B and C

Step 1: Start
Step 2: Read three numbers A, B and C
Step 3: Find the larger number between A and B and store in max (A and B)
Step 4: Find the larger number between max (A, B, C)

Step5: Display max
Step6: Stop

Characteristics of Algorithm

→ It should have finite number of steps

→ Terminate after a finite number of steps

→ Instructions are precis and unambigous

→ Operation are done exactly in finite time

→ Output are derived from the input by applying the algorithm

Ex: Algorithm for calculating factorial value of a number

Step 1: Start
Step 2: Read the number n
Step 3: Initialize (i=1, fact=1)
Step 4: Repeat step 4 through six until i=n
Step 5: fact = fact * i
Step 6: i = i+1
Step 7: Print fact
Step 8: Stop.

Algorithm examples (Exam mei jiska algorithm aaega uska flowchart bhi aaega)

*** → Fibbonacci series
→ Even and odd
→ Prime number
→ Reverse of number
→ Palindrome

(V.Imp for exam!)

** Flowchart

(1) Tree-like structure

(2) Diagramatic rep. of algorithm that defines sequence of algorithm to be performed to get a solution.

(3) The various boxes are interconnected with the help of arrows

(4) The boxes represents operations and arrows represent sequence in which operations are implemented.

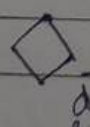
(5) Main aim of flowchart is to help programmer in understanding the logic of program.

Guidelines for representing Flowchart

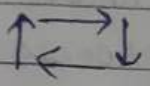
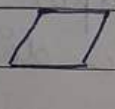
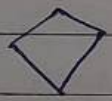
(1) It should be clear and ~~exact~~ easy to follow.

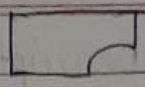
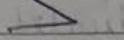
(2) It must have logical start and finish

(3) Only one flow line should enter decision symbol However two or more lines can leave decision symbol.

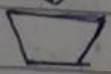


(4) Only one flow line is used with terminal symbol.

	Symbol	Symbol Name	Description
(1)		Flow lines	It is used to connect symbols. These lines indicates the sequence of steps and direction of flow of arrow.
(2)		Terminals	This symbol is used to represent beginning terminal or halt (pause)
(3)		Input output	It defines information entering or leaving the system
(4)		Processing	It is used for repres-enting arithmetic and data movement instructions.
(5)		decision	It denotes a decission to be made. This system have one entering and two exit paths.
(6)		connector	It is used to join different flow lines.
(7)		off-page connector	It is used to indicate that flowchart is continuous to next page

(8)		Document :	It is used to represent a paper document produced during process
(9)		Communication link	It is used to represent data recieved to be transmitted from an external system.
(10)		Magnetic disk	This symbol is used to represent input and output from magnetic disk.

Symbol

(11)  : Manual operation (Symbol name)

(Description) → It defines that process has to be done by developer / programmer.

* Advantages of flowchart

(1) It makes logic clear
(2) effective analysis
(3) useful in coding
(4) Proper testing and debugging

Limitation

(1) Complex
(2) Costly
- If flowchart is to be drawn for use application the time and development may get out of portion.
(3) No update
- Usually programs are regularly updated.

PSEUDOCODE → make a box always

→ It is not a real programing language but it models and look like program code
→ It uses proper english statements rather than symbols
→ To represent the processes of computer program
→ If algorithm is written in english the description may be on a highly level. It may be diff & more time to analyse the problem.
→ Therefore role of writing pseudocode is providing high level detail of algorithm which facilates analysis and eventual coding.
→ Pseudocode use some keywords which are as follows.

Input: READ, OBTAIN, PROMPT, GET,

Output: PRINT, DISPLAY & SHOW

COMPUTE: COMPUTE, CALCULATE & DETERMINE

INITIALIZE: SET, INITIALIZE

ADD ONES: INCREMENT

Adv: —

(1) Easy to develop program by pseudocode instead of algorithm.

(2) Easy language

Keywords

→ It represents reserve words. Keywords can't be used as variable name, constant name.

→ In C language there are 32 words present.

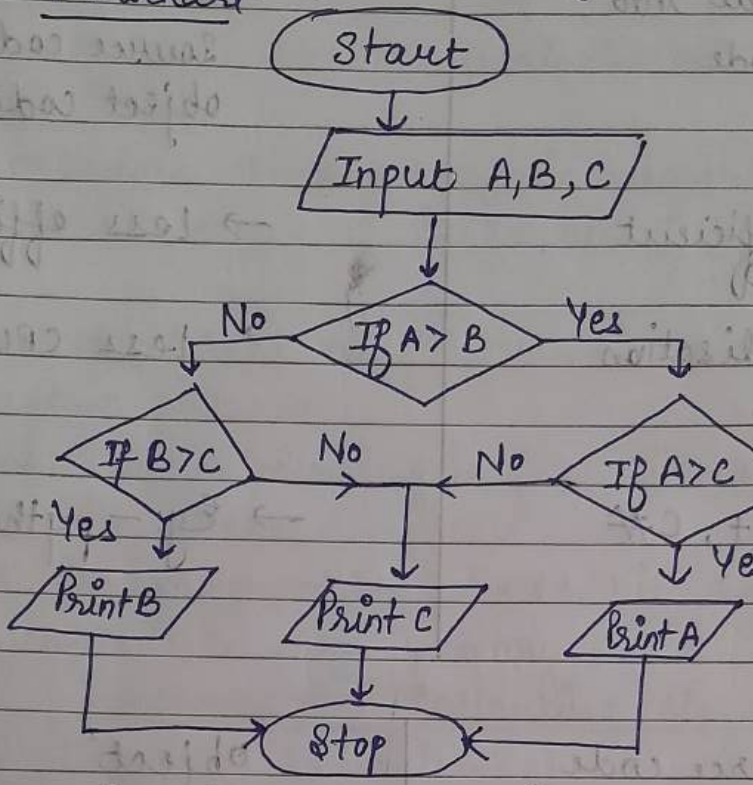
Example:- float, int, ~~tuple~~ double, union, char

Linker	Loader
1) Linker defines utility program which takes object files produced by assembler and compiler, other code to join them into a single executed file	1) It is vital component of operating system for loading programs and libraries.
2) It is used to join all modules	2) It is used to allocate address of executed file
3) Linker use an input output code produced by assembler and compiler.	3) Use an input of files produced by linker

(Imp)

★ Flowchart for finding largest of 3 No.s

Flowchart



Algorithm :-

1) Start
2) Input A, B, C
3) If (A>B) and (A>C) then print "A is greater" Else if (B>A) and (B>C) then print "B is greater" Else print "C is greater"
4) Stop

Imp topic

★ Differentiate between compiler and interpreter.

Compiler	Interpreter
→ It scales whole program in a single quote	→ It translates one statement at a time
→ Errores are shown together at end	→ Errores are shown line by line
→ Execution time is faster	→ less Execution time as compared to compiler
→ Dosnot require source code for data execution.	→ ~~It scales line by line but~~ requires source code for data execution

→ It converts Source code into object code	→ It scales line by line but don't convert Source code into object code
→ More efficient	→ Less efficient
→ CPU utilisation is more	→ Less CPU utilisation
→ Eg:- C++, C#	→ Eg:- python, MATLAB

Source code	Object
→ Source code in the C program is that place where you right code in editor and source with "C" execution which is uncompiled	→ It is output of compiler after it processes source code. It is usually defines machine code which can be understood directly by a specific cycle of CPU

Assembly language errors
- executable files
- Always save doc with .exe extension
- Also called binary files
- It is produced as output of after its processes the object code.

Array - collection of homogeneous elements

imp for exam!

Explain Memory management in Data

→ Memory management coordinates between other softwares and users.
→ It manages primary or main memory
→ It is made up of large no. of bytes (Data storage units) or words where each byte or words is assigned a certain address
→ Since main memory is fast storage & it can be directly excessed by CPU.
ex. for program to be exe. it should be first loaded in main memory.

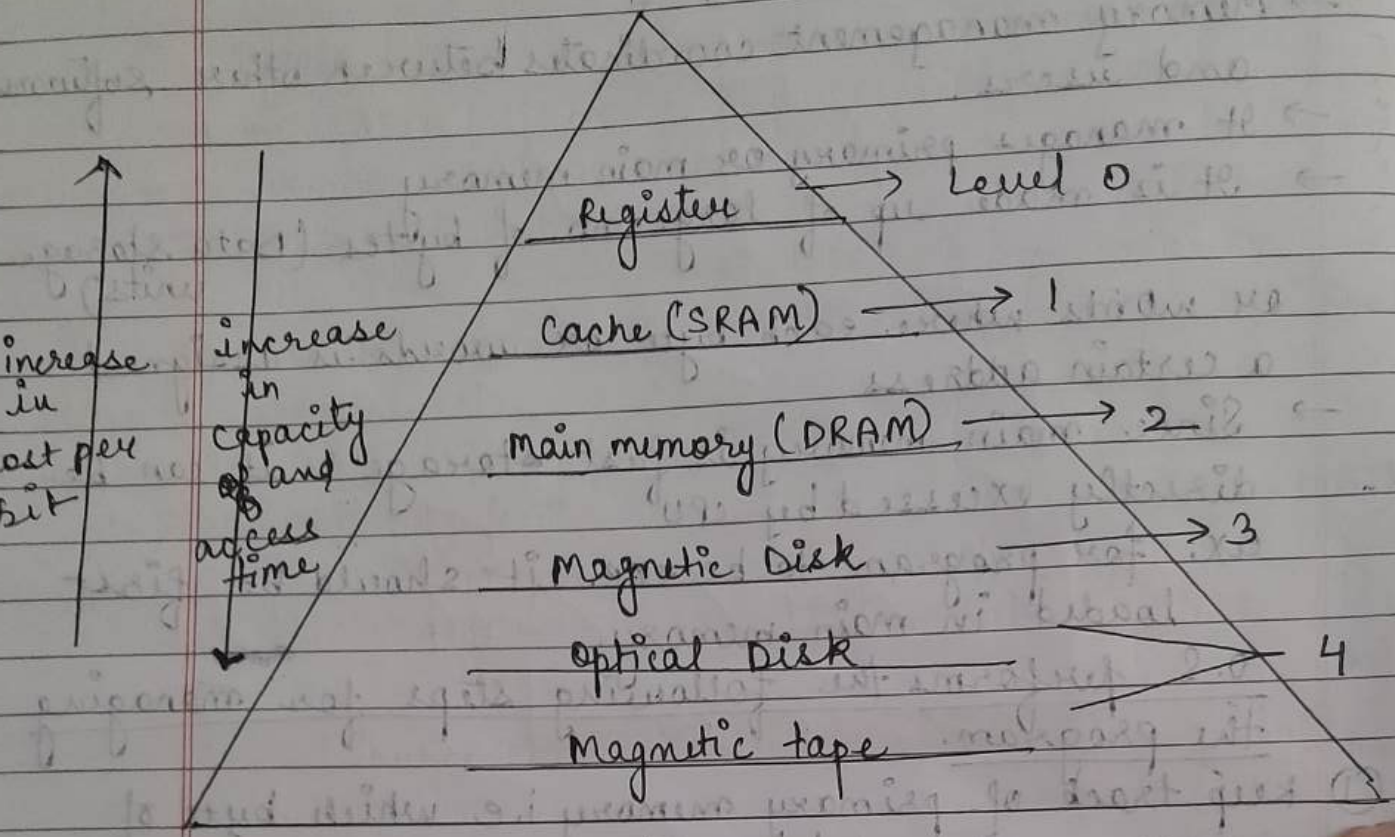
O.S performs the following steps for managing the program.

(1) Keep track of primary memory i.e which byte of memory are used by which use program.

(2) O.S decides the order in which process are granted access to memory & for how long.

(3) It allocates the memory to a process when the process request it and deallocates the memory Allocation and deallocation of a memory to process requested by temporary input, output in process.

Memory Hierarchy of CPU



** Register
→ It is small, high speed memory units located in CPU.
→ It is used to store more frequently used data and instruction
→ It size is always in bytes (16 - 64 bits < 1KB)

Cache Memory (Static Random Access Memory)
→ It is small fast low access memory stored in CPU. It stores frequently used data and instruction that have been recently accessed from main memory and designed into minimise time.
→ Size (< 16MB)

Main memory (DRAM)

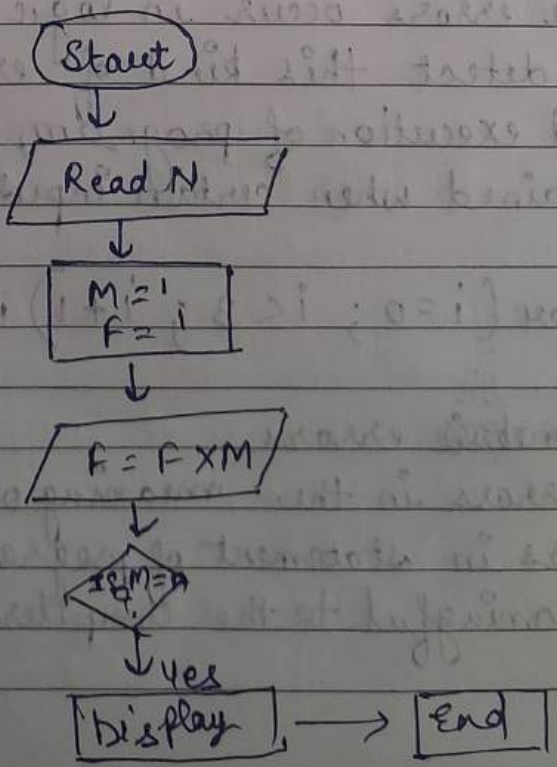
→ Also called RAM. Primary memory of computer system. It has largest storing capacity than data and instruction stored currently by CPU.
→ Size (<16 GB)

Secondary memory (>100 GB)
→ used in hard disk

V.V. Imp. (15 marks)

Flowchart algorithm for factorial

Step 1:- Start
Step 2:- Read number n
Step 3:- Initialize i=1, fact = 1
Step 4:- Repeat Step 4 through 6 until i= n
Step 5:- fact = fact * i
Step 6:- i = i+1
Step 7:- Print fact
Step 8:- Stop



PAGE No
DATE : / 202

Eg. int a, b, c ;
a + b = c ;

Run time error
Errors which occur due to program execution after successful compilation are called run-time error.

eg. divide by 0 (division error)

Datatype	Storage	Value range
Char	1 byte	-128 to 127
U Char	1 byte	0 to 255
int	2 or 4 byte	-32768 to 32767
long	4 byte	-2147483648
double	8 byte	+0 2147483648

→ Rules for creating variable in C

→ A variable must only contains, alphabets, digits and underscore
→ A variable name must start with any alphabet or an underscore only. It cannot start with a digit.
→ No whitespace is allowed within the variable name A variable name must not be any reserved word or keyword.

Unit - 2

- Int
- float
- char

] Datatype

Arithmetic operation or precedence

⇒ Evaluation of Arithmetic operator

When both operand are of integer type then the arithmetic result would be performed in an integer value

eg. int a = 3;
int b = 2;
c = a / b;
c = 1

When both the operands are of float type the arithmetic result would be of float type

eg. float a = 3.0
float b = 3.0
output float c = a / b
c = 1.0

If one operator is of integer type and other is real type (float) then mixed arithmetic will be performed.

int a = 6;
float b = 2.0;

BODMAS

output = 6.0 / 2.0
= 3.0

Ques $6 \times 2 / (2 \times 1 + 2 / 3 + 6) + 8 \times (8/4)$

$= 12 / (2 + 2/3 + 6) + 8 \times 8/4$

$= 12 / (2 + 2/3 + 6) + 16$

$= 12 / (2 + 0 + 6) + 16$

$= 6 \times 2 / (8) + 16$

$= 12 / (8) + 16$

$= 12 + 16$

$= 1 + 16$

$= 17$

→ Unary operator.
Unary operator which operates one one value

→ Prefix expression
In prefix exp. value is computed before expression is evaluated.

→ Postfix expression
Postfix - value is computed after expression is evaluated.

```
count total = 150;
      value = 20;
count = 100;
Total = (value++, count - 10);
Printf ("%d", total);
```

Statement → line by line execution

```
c = a+b;
printf(" %d", a+b);
prefix (" %d %d" a,b, a+b)
```

Imp. for exam / Conditional operator / Ternary

exp1 ? exp 2 : exp 3 ;

```
if(x>9) ? (x++) : (x--)
```

→ Conditional operator is expression that returns one if condition is true.

Operator Precedence and Associativity

Are the 2 features of operator that determines the evaluation order of sum expression in absence of brackets.

eg. 100 + 200 / 10 - 3 * 10
= 90

Conditional Branching of loops

Decision making statements

- ↳ Simple if statement
- ↳ if-else
- ↳ Nested if-else
- ↳ else if ladder
- (Imp exam topic) * ↳ switch

Decision making statement decides the order of execution of specific statements in your program.

IMP topics
↳ Switch
↳ Loop

H.W ✓ Dangling else
Nested if-else.

You can setup a condition and tell to the compiler if the statements are true return some code but if the statement is ~~not~~ true but condition is not true you can also tell the compiler to execute different code.

Nested if-else

```
if (a>b)
{
  if (a>c)
{
printf (" %d", a);
  else
printf (" %d", c);
  else
  ie (b>c)
printf (" %d", b);
  else
  printf=      (" %d", c);
  }
```

Jump for exam

Switch

→ Switch is conditional control statement which is used to select one option from several option based upon given expression value.

Switch statement is a multi(way) branch system. Switch statement requires only one arguement of integer which is checked with number case of statements.
The switch statement evaluates expression and then look for its value among the case constant statement. If the value matches with case then that particular case statement is executed.

SYNTAX

```
Switch (expression)
{
case value 1 : statement 1;
                break;
case value 2 : Statement-2;
                break;


case value_n : Statement_n;
                break;
default : default_Statement;
}
```

Else - if ladder

If else - if ladder in C programming is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions is the if - else ladder are false. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if else ladder will be executed, and the entire if else ladder will be terminated.

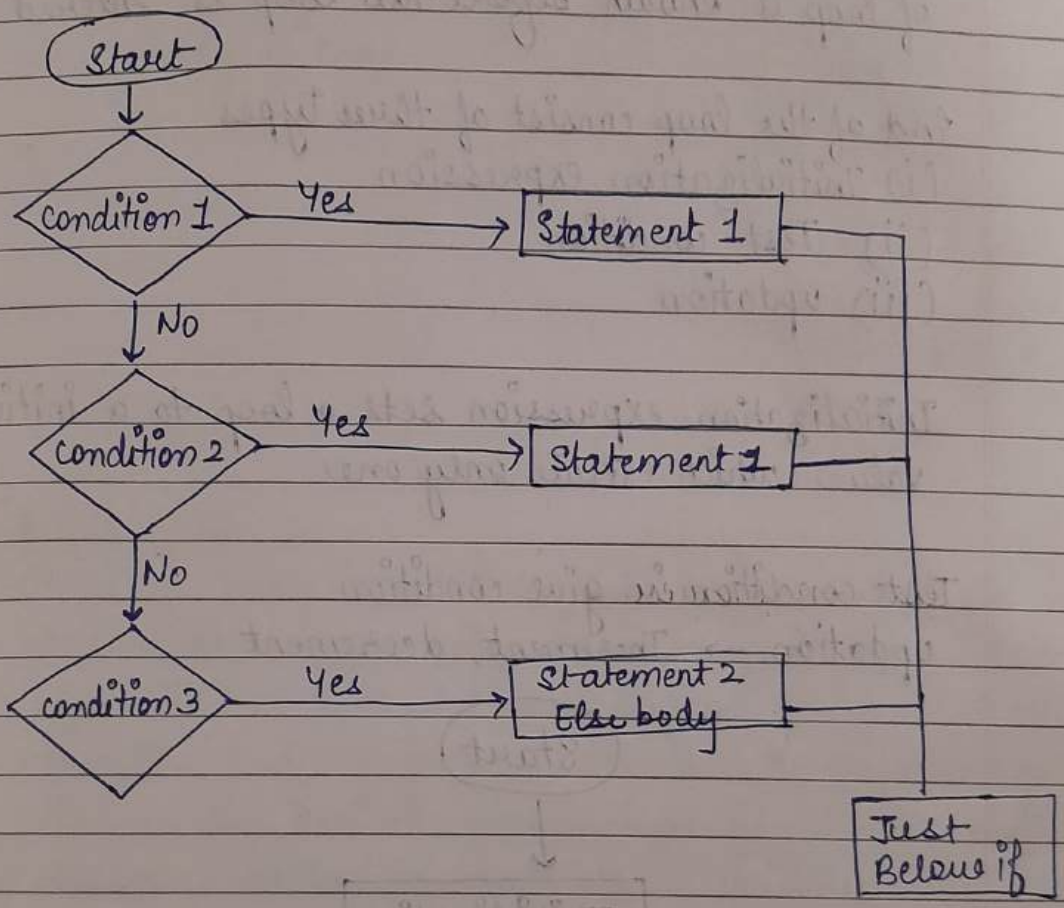
Syntax :

```
if (condition) {

}
else if (condition) {

}
```

else {

}

Flowchart



LOOPS

Loops is defined as block of statement which are executed repeatively for a given number of time.

FOR LOOP
WHILE LOOP
DO WHILE LOOP

FOR LOOP

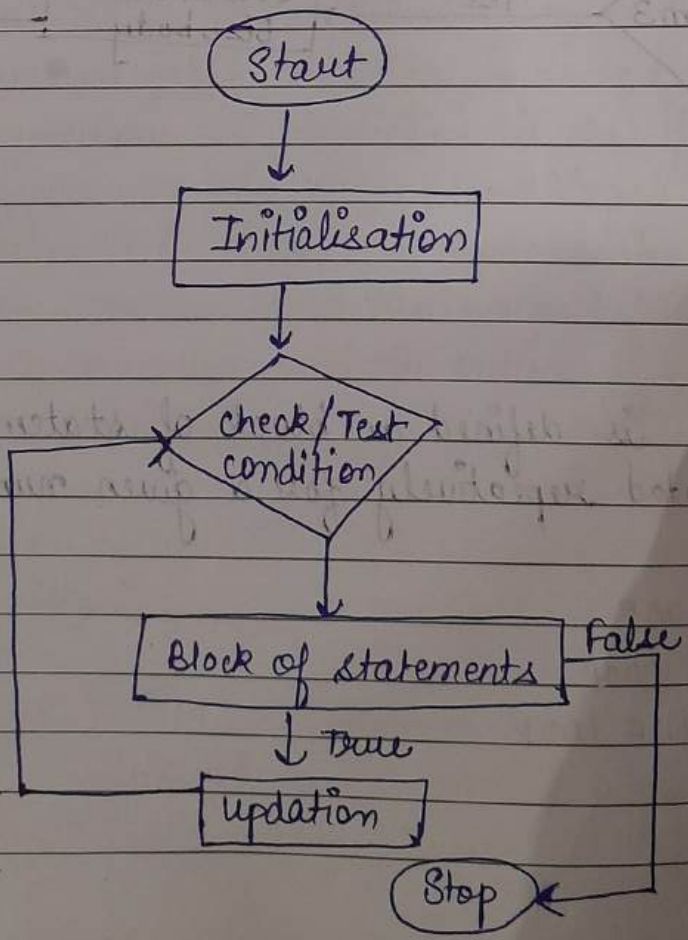
For (initialization; test condition; increment/decrementation)

For loop works well when the number of iteration of loop is known before the loop is entered

End of the loop consist of three types
(i) Initialization expression
(ii) Test-condition
(iii) updation

Initialization expression sets a loop to a initialize value which execute only one

Test condition is give condition
updation → Increment, decrement



Nested for loop

```
for (init ; test ; Updation)
  {
  for (init ; test ; Updation)
  {
    body of loop
  }
}
```

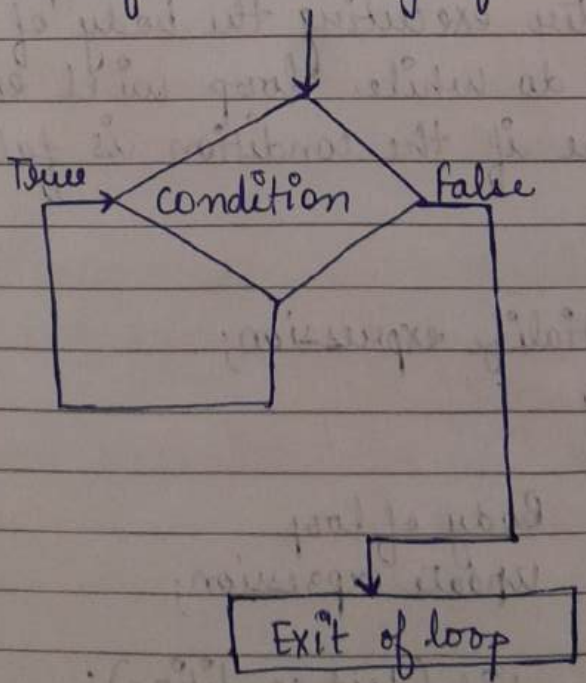
ASCII value
65 - 90 = A - Z
97 - 122 = a - z
value = decimal (int)
character = char

```
int char a;
printf("enter char");
scanf("%c", &c);
if (c >= 'a' && c <= 'z' ||
c >= 'A' && c <= 'Z')
printf("%c is an
alphabet %d", c, c);
return 0;
```

Imp ★

WHILE LOOP

It is an entry control loop statement (condition is checked before executing the body of the loop. Test condition is evaluated and if the condition is true then the body of loop is executed until test become false. On exit, the program continues with the statement after the body of the loop.



Syntax:

```
Initialize expression;
while (test condition)
{
    body of loop
    Update expression
}
```

```
int n; rev=0, rem;
printf("enter no.");
scanf("%d", &n);
while(n!=0)
{
    rem = n%10
    rev = rev *10+ rem;
    n = n/10;
}
printf("reverse is %d", rev);
}
```

DO - WHILE LOOP

Do- while is a exit control loop. Condition is checked after executing the body of the loop because in do while loop will execute atleast once if the condition is false.

Syntax:

```
initialize expression;
do
{
    Body of loop
    Update expression;
}
while(test condition);
```

check whether number is prime or not

```
int n, x = 2;
printf ("enter no. to check whether it is prime or not");
$canf ("%d", &n);
 do
{
 if (n % x == 0)
 {
  printf ("not prime");
}
    exit (0);
}
  x++;
}
  while (x <= n);
  printf (" number is prime", n);
  getch ();
}
  while (1)
     ↳ infinite loop
```

Ques. Differentiate between fundamental data type or derived datatype and premitive data type.

→ It is very basic. It is also called premitive	→ Derived data type is aggregation of fundamental datatype
→ Ex. void, float, int	→ Ex. Structure, Array, Pointers, Union.

Unconditional Control Statement

Unconditional branching is when the programmer forces for the execution of program to jump into another part of program.

→ **Break**

Terminates the execution of loop and control transfer to the statement immediately following the loop.

⇒ Break can be used in for, while, do-while, Switch

→ **Continue**

It is used to by pass the remainder to the current path through a loop. Loop does not terminate when a continue statement is encountered. Instead the remaining statement loop statement are skip and computation proceed directly to the next part of the loop.

→ Go to

(Support the goto statement branch unconditionally from one point to another)

```
Goto label;
label;
Start;
```

→ Return

Return statement terminates the execution of the function and return control to the when calling the function.

chapter - 3

Arrays

It is collection of singular types of data ~~collection~~ item stored at contiguous memory allocation.
Contiguous memory location is method single contiguous section/part of memory is located to a process or file needing it.

Arrays are the in the C programming which can show primitive data types (int, char)

Properties of array

Elements of array are stored at contiguous memory location where the first element is stored at smallest memory location
Elements of array can be randomly accessed since we calculate addressed each element of array with the given address and size of elements
eg. Address of $A[I] = B + W * (I - LB)$

I - where I represent subset of element
B - B represent base address
W - It represent storage
LB - Lower Bound (If not given then assume zero)

Ques. Given A[1300, - - - - -, 1900] as 1020 and size of each element is 2 byte. Find in address of A

BA = 1020
W = 2
I = 1700
LB = 1300
→ 1020 + 2 * (500)
= 1522

Advantages of C Array

1. Code of optimization
2. Ease of sorting
3. Ease of traversing
4. Random access

Disadvantage :- fixed size

Declaration

1. Datatype arrayname [array size]
   int A[5]
2. Arrayname

initialization

```
int a[5]
a[0] = 10;
A[5] = 0;
a[4] = 45;
for(i=0; i=25; i++)
{
printf("%d", a[2]);
}
```

Difference between 1D & 2D :

1D	2D
→ It stores single list of elements of singular data types	→ It represents "list of lists" of same data type
→ 1D represent float type data	→ 2D represent table type data (row and columns).
eg. int A[5]	int A[5][5];

Searching

→ Searching algorithm design checked for an element from any data ~~stored~~ structure where is stored.

Searching is of 2 types

① Linear search
② Binary search

Linear

→ It is also called sequential search In this the list or array is traversed sequentially and every element is checked.

→ They are less efficient than binary search

```
eg. int a[5], search, i, n;
printf ("enter no. of elements");
Scanf ("%d", &n);
printf (" enter %d integers", n);
for(i=0; i<n; i++)
Scanf ("%d", &a[i]);
for(i=0; i<n; i++)
{
if(a[i]==search)
{
printf(" %d is present at
%d");
break;
}
else
printf ("not found");
}
```

Binary

→ Binary search are also called interval search.

They are most

→ They are more efficient than linear search

```
int low, mid, high,
reader n, a[100];
printf("enter no. of elem
ents")
Scanf("%d",&n);
printF("enter%d int",n);
for(c=0; c<n; c++)
{
Scanf("%d", &a[c]);
}
low = 0;
high = n-1;
mid = ((low+high)/2);
while(low<=high)
{
mid = ((low+high)/
if(a[mid]<search)
low = mid+1;
```

$\log_2 n$ = complexity

PAGE No
DATE : / 202

```
int low, mid, high, reader, n, a[100];
printf("enter no. of elements")
scanf("%d", &n);
~~if(low <= high)~~
~~mid = (low + high)/2;~~

printf("enter %d integer", n);
for(c=0; c<n; c++)
{
scanf("%d", &a[c]);
}
  low = 0;
  high = n-1;
  mid = ((low + high)/2);
  low = mid + 1
  else if(a[mid] == search)
}
{
Printf("%d found at %d", search, c+1);
break;
}
Else   high = mid - 1;
}
```

MATRIX MULTIPLICATION

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 2 & 3 & 4 \\ 2 & 3 & 4 \end{bmatrix}$$

```
#include <stdio.h>
#include <

int a[10][10] b[10][10] mul[10][10]
r, c, i, j, k;
printf ("enter no. of rows");
for (i=0 ; i<r, i++)
{
for (j=0; f<c ; f++)
{
scanf ("%.d", & a[i][j]);
}
}
for (i=0; i<r; i++)
{ for (j=0; j<c; j++)
{ scanf ("%.d", & b[i][j]);
}
}

for (i=0, i<r, i++)
{
  for (j=0 ; j<c ; j++)
  {
    Scanf (" %.d", & [i][j]);
    }
    }
      for (i=0 ; i<r ; i++)
      {
        for (j=0, j<c, j++)
        {
          scanF (" %.d", & b[i][j];
          }
            for (i=0, i<r, i++)
```

```
{
  for (j=0, j<c, j++)
  {
    mid[i][j] = 0;
for (k=0; k<l, k++)
{
  mid[i][j] = mid[i][j] + a[i][k] + b[k][j];
  }
}
}
  for (i=0; i<n, i++)
  {
    for (j=0, j<c, j++) {
    {
      printf("%d\t", mul)[i][j];
    }
      printf("/n");
    }
      return;
    }
```

Address of A[I][J] = B + w * ((I - LR) * N + (J - LC))

where I = Row subset of an element whose address to be found
J = column subset of an element whose address to be found
w = Storage size of an element
LR = Lower limit of Row (Starting Row Index of matrix)
LC = Lower limit of column (If given otherwise)
N = No. of column given in matrix
N = UB - LB + 1

UB → Upper Bound
LB → Lower Bound

Ques B = 100
W = 1 Byte
Row Subset = 8
Col Subset = 6
arr [1, ... 10], arr [1, ..., 15]
B = 100
W = 1 Byte
Found for array [8][6] by row major

$= 100 + 1 * (8-1) * 15 + (6-1)$
$= 100 + 7 * 20$
$= 100 + 140$
$= 240$

$A[I][J] = B + w * (J - LC) * M + (I - LR))$
$= 100 + 1 * (6-1) * 10 + 8(-1)$
$= 157$

Sorting
(Bubble, Insertion, Selection, quick, merge)
Except merge all complexity $\rightarrow O(n^2)$
Avg $\rightarrow$ Merge, Quick
$O(n \log n)$

Bubble Sort

Q Swapping of two numbers

```
# include <stdio.h>
# include <conio.h>
  int main ()
  {
    int i, j, temp; a[100];
```

1 5 6 3 2
1 5 6 3 2
1 5 3 6 2
1 5 3 2 6

```
printf ("enter no. of elements");
scanf ("%d", &n);
printf ("enter %d integers", n);
for (i=0; i<n; i++)
{
  scanf ("%d", &a[i]);
}
for (i=0; i<n-1; i++)  → (for j=0; j<n-i-1; j++)
{                        {
  if (arr[j] > arr[j+1])
    temp = a[j];
    a[j] = a[j+1];
    a[j+1] = temp;
    for (i=0; i<n; i++)
    {
      printf ("%d", a[i])
    }
```

Character data type

Derived data type which is used to store collect of character or streams. Since char. data type takes 1 byte of memory ∴ char array of the number of elements in array.

Declaration Syntax

char name [size];

char name [50];

Imp. for exam ✓ String

It is a sequence of characters terminated with null char ('\0'). Also called termination of string. It is stored as array of characters

eg. char str [] = "Geeks"

Declaration → index

0	1	2	3	4	
G	E	E	K	S	

address

char string name [size];
[20]

initialization:- char str [] = "YMCA"; 'Y' 'M' 'C' 'A'

char str [20] = "YMCA University";

char str [10] = {'Y', 'M', 'C', 'A', 'U' 'n';

char str [ ] = {

Manipulation / Library

* Functions of String

(1) String Length ⇒ This function is used to find length of string.

Strlen (String_name)

```
# include <string.h>
  int main ()
{
    char str1[ ] = "YMCA";
    char str2 [ ] = " University";
    printf ("length of str1 is %d")
     strlen (str1);
      strlen (str2);
      return 0;
}
```

Imp Ques (Smart) Program to calculate, string length without using inbuilt string functions.

```
int main ()
{
char str [100]; ⟶ int i, length = 0
printf ("enter string");
scanf ("%s", str);
for (i=0; str[i] != '\0'; i++)
{
    length ++;
}
    printf ("length of string is %d", length);
    return 0;
}
}
```

Strcpy

② String copy (S1, S2)

It is used to copy the content of string S1 into S2.

```
int main()
{
    char str1[] = "YMCA";
    char str2[] = "college";
    Strcpy (str2, str1);
    printf ("%s %s", str1, str2);
    return 0;
}
```

str 2 = College
str 1 = College

③ Str cat (S1, S2)

```
int main()
{
    char str1[100] = "This is ";
    char str2[100] = "YMCA University";
    ~~puts(str1)~~ Strcat(str1, str2);
    puts(str1);
    puts(str2);
    return 0;
}
```

⇒ Put func. is used to write a line on string to output.

④ Strcmp (S1, S2)

```
int main()
{
    int result;
    char S1[] = "YMCA";
    char S2[] = "YMCA";
    char S3[] = "YMCA";
    Strcmp(S1, S2);
    Strcmp(S1, S3);
    printf("Strcmp(S1, S2) = %d", result)
    printf("Strcmp(S1, S3) = %d", result)
    return 0;
}
```

→ It compares first string to 2nd string

① If str1 = Str2
output: 0

② If 1st non-matching char in str1 lower to sky value than of str2 then value will be less than 0

③ If value is > than Output will be greater

(5) Strlower / Strupper / Strrev

```
int main;
{
  char str[] = "YMCA";
  printf("lowercase is %s", strlower(str))
  return 0;
}
```

upper
rev

V. imp for exam

Selection Sort

It is a simple alg. which contains sorted part in left hand and unsorted at right hand.

~~Algorithm maintains~~

In every itteration of selection sort minimum element from the unsorted sub-array.

→ Not suitable for larger data sites

eg.

```
int main ()
{
  int n = 10;
  int a[] = {11, 15, 19, 18, 25, 27, 29};
  int small;
  for (int i = 0; i < n-1; i++)
  {
    small = i;
    for (int j = i+1; j < n; j++)
    {
      if a[small] > a[j])
      {
        small = j;
      }
    }
  }
```

```
if (small != i)
{
    int temp = a[i];
        a[i] = a[small];
        a[small] = temp;
}
```

ALGORITHM

Step 1:- Set minimum to location zero
Step 2:- Search the min. element in list
Step 3:- Swap with value at minimum location
Step 4:- Increment min to point next iteration
Step 5:- Repeat until it sorted.

Insertion Sort (* Ques in exam → kitne array bante hai = only 1)

It is an in-place comparison base shorting algorithm. In this sublist is maintained which is always sorted.
An element which is to be inserted in the sorted sublist, hence to find is appropriate place and hence inserted there.
Array is searched seq. and unserted.

eg.
```
int main()
{
    int n = 10;
    int a[] = { any 10 value };
    for (i = 1; i < n-1; i++)
    {
        j = i;
    while (j > 0 && a[j-1] > a[j])
    {
```

→ Best case
→ Avg case
→ Wodest case

if we remove $i <= n-1$ then $i < n$

```
        temp = a[j];
        a[j] = a[j-1];
        a[j-1] = temp;
        j--;
    }
    printf("Sorted array");
    for(i=0; j<=n-1; i++)
    {
        printf("%d\n", a[i]);
    }
    return 0;
}
```

Function (fn)

Function is a set of statements that takes input do some specific compititions and produces output.

Ques Difference b/w loop and function

Loop	Function
→ Re-usability of code is restricted to specific area.	→ Reusability not restricted (u can call function code from anywhere in program)

Aspects of function

Function Declaration	Function Definition	Function call
→ Function must be declared globally in 'C' program to tell fn name, fn parameter, return type fn name (ary list);	→ It contains the actual statement which are executed It also acts as the most imp aspect to which control comes when function called → Only value can be written from the function.	→ Function can be called anywhere in program. Parameter list must not differ in fn calling and fn declaration. It has 4 aspects 1) Function without arg. and " return value 2) fn without arg. and with return value 3) fn with arg. and without return value 4) Both with
int sum (int a, int b);	return - fn name (ary_list) { function body; }	
	Syntax:	Syntax: fn name (ary_list); add ();

① Function without arg. and " return value

```
void sum ( );
void main (i)
{
Printf (Addition of 2 no);
}
void sum ( )
{
int a, b;
printf ("enter 2 no.");
scanf ("%d %d", &a, &b);
printf ("sum is %d", a+b);
}
```

② function without arg and with RV

```
int sum ( );
void main ( )
{
int total;
Printf ("addn");
total = sum ( );
Printf ("sum is %d", total);
} int sum ( )
{ int a, b;
Printf ("enter 2 no.")
; return a+b;
```

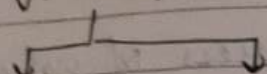
③ Fn. with arg and without return value

```
void sum (float, float);
void sum (float c, float d);
void main ( )
{
Printf ("Sum of %f", c+d);
Printf ("Addition of 2 no.");
Printf ("enter 2 no.");
Scanf ("%d %d", &a, &b);
sum (a, b);
}
```

④ Function with arg and with return value.

```
float sum (float, float);
void main ( )
{
    float a, b, total;
    printf ("enter 2 no.s");
    scanf ("%f %f", &a, &b);
    total = sum (a, b);
    printf ("sum is %f", total);
}
float = sum (float c, float d)
{
    return 0;
}
```

Types of function



→ Library are those fn which are declared in C ~~as~~ header file
Eg - Scanner function, printer function getch put float

→ User defined - Functions which are created by programmer so that programmer can use it many times. → Header file <stdio.h>

Imp for exam (VIVA)

Malloc

It is used to dynamically allocate a single large block of memory with specific size.

Eg. ptr = (int *) malloc (100 * sizeof (int));

Calloc method in C is used to dynamically

→ Contiguous, allocate the specified number of blocks of memory of the specified type. It is very much similar to malloc() but has two different points ~~and these are~~.

It initializes each block with a default value '0'.
Two parameters or arguements

Eg. ptr = (float *) calloc (25, sizeof (float));

Imp

Call by value	Call by reference
1) Copy of value is passed by fn.	1) An address of value is passed into fn.
2) Changes made inside fn is limited to fn only.	2) Change made inside fn validate outside of function also.
3) The value of actual parameters doesnot change by changing formal parameters.	3) Value of actual parameters does changing by changing the formal parameter
4) Actual and formal parameters are created at diff memory location.	4) Actual and formal parameters are located at same memory location.

⇒ Actual Parameters are those parameters which are written in function call.

⇒ Formal Para which are written in function definition

↳ continued

```
Header file
void swap (int x, int y);
int main ()
{
    int a = 10, b = 20;
    swap (a, b);
    printf ("a = %d  b = %d",
            a, b);
```

```
void swap (int * int *);
int main ()
{
int a = 10, b = 20;
Swap ( &a, &b);
printf ("a = = %d, b = %d",
a, b);
return 0;
}
```

```
return 0;
}
void swap(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
Printf("x = %d, y = %d", x, y);
}
```

```
void swap(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}
Printf("x = %d y = %d", *x, *y);
}
```

output ⇒ x = 20, y = 10
a = 20, b = 10

Ques. To calculate area and circumference of circle using call by reference.

```
#include<stdio.h>
#include<conio.h>
void area_peri(int, int*, int*);
void main()
{
  int r, p, a;
  clrscr();
  printf("Enter the radius of circle\n");
  scanf("%d", &r);
  area_peri(r, &p, &a);
  printf("Area = %d Perimeter = %d", a, p);
  getch();
}
  void area_peri(int r, int *p, int *a)
{
  *p = 2*3.14*r;
  *a = 3.14*r*r;
}
```

Recursion

When a function call itself directly or indirectly is called recursion and corresponding function is called recursive fn.

Why base condition is required in recursion?

In recursive program the solution to base case is provided and solution of bigger problem is expressed in terms of smaller problem.

Directly Recursion

A function is called direct rec. if it calls the same function

```
eg. void directrec()
    {
    // code
    direct rec();
    // code
    }
```

Indirectly Recursion

A function is called indirect rec. if it calls another function.

```
eg. void indirect rec fun1()
    {
      // code
    indirect rec fun2()
    {
      // Code
    indirect rec fun1()
      // code
    }
```

Ques. Program to calculate factorial of a number

Continued on next page

Factorial

```
#include <stdio.h>
int fact (int n);
{
  if (n==1) || (n==0)
  return 1;
else
  return n * fact(n-1);
}
  int main()
  {
    int n = 5;
    Printf ("factorial of %d
            = %d", n, fact(n));
  return 0;
}
```

Fibonacci

```
#include <stdio.h>
int fib (int n)
{
  if (n==1)
  return 0;
  if (n==2)
  return 1;
else
return fib(n-1) + fib(n-2)
}
int main ()
{
  int n = 5;
  Printf ("fibonacci series
          = %d", n)
for(int i=0; i<n; i++)
{
  Printf ("%d", fib(i));
  return 0;"
}
```

Ackerman's function → It states that not all computable functions are primitive recursive.

→ It is a function with two arguements each of which can be assigned any non-integer.
→ Ackerman's function can be defined as

Continued on next page →

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ \& } n > 0 \end{cases}$$

where m & n are non-negative integers

Ques. $A(1,2)$
↳ apply 3rd

$A(m-1, A(m, n-1))$
$A(0, A(1,1))$ — (1)
$A(1,1)$ → apply 3rd

$A(0, A(1,0))$ — (2)
$A(1,0)$ → apply 2nd
$A(m-1, 1)$
$A(0,1)$ — 1st
$A(0,1) = 2$
$A(0,2) = 3$
$A(0,2) = 3$ → apply in eqn
$A(0,2) = 3$ → Put in eqn (1)
$A(0, A(1,1))$
$A(0,3) = 4$

Merge sort(arr[ ], l, r)
If l > r

1. Find the middle point to divide array into 2 halves

middle m = (l + r) / 2

2. Call merge sort for first half

Call merge sort (arr, l, m)

3. Call mergesort for second half

Call merge sort (arr, m+1, r)

4. Merge the two halves sorted in step 2 & 3
Call merge (arr, l, m, r)

Divide and Conquer;

1) It is an process which contain 3 steps

a) Divide :- break the problem into sub problem of same type.

b) Conquer :- Recursive solve this sub problem

c) Combine :- Appropriate combine the answers

Unit - Structures

→ Structure is a keyword that creates user defind data type

→ Structures create a data type that can be used to group item of various type into single type.

Syntax of structures :-

```
struct structure name
{
    int
    char
    float
};
```

e.g

```
struct address
{
    char cityname;
    string s;
    int pin;
};
```

- Keypoints : changing the value of one data number will not affect other data number in structure.
- Total size of structure is sum of size of every data member.
- You can retrieve any member at any time.

How to declare structure variables?

→ Structure variable can be either be declared with " declaration or as a seperate declaration.

e.g
```
struct pin
{
    int x, y;
} pi;
// variable pi is declared with pin
```

```
struct pin
{
    int x, y;
}
int main()
{
    struct pin pi;
}
```

Ques How to initialize structure member?

```
{
struct pin
  int x = 0;
  int y = 0;
};
  struct pin
  {
    int x, y;
  };
  int main()
  {
    struct pin p1 = { 0, 1 };
  }
```

How to access the structure elements?

```
Struct pin
{
   int x, y;
};
   int main()
   {
   struct pin p1 = [0, 1];
        p1.x = 20; → How value is replaced
        printf("x = %d  y = %d", p1.x, p1.y)
        return 0;
   }
```

output ⇒ 20, 1

Limitations of Structure

1) 'C' structures does not allow struct data type to be treated like in built data type
2) No data hiding structures does not permit data hiding because structure member can be accessed anywhere in the scope of structure.
   (global)
3) Function inside structure is not allowed in 'C' structure

→ Array of structure

Ques → Create an array of structures of 50 students having 4 fields name, roll no, sem, ~~branch~~ branches by taking input from user

Solution:- on next page

```
struct student;
{
    int Roll no;
    char Name[50];
    int sum;
    char branch[20];
} S[50];
int main()
{
    int i;
    Printf("enter 50 students record");
    for(i=0; i<50; i++)
    {
        S[i]. = i+1;
    Printf("Roll no. = %d", S[i]. roll);
    Printf("enter name");
    Swap("%d", S[i] name);
    Printf("enter roll no.");
    Swap("%d", S[i] sum);
    Printf("enter branch");
    Swap("%d", [i] branch");
    Printf("Display info");
    for(i=0; i<0; i++)
    {
        Printf("name is");
        Puts(S[i], name);
        Printf("sum %d, S[i]);
        Puts(S[i] branch)
    }
    return 0;
}
```

Sorting Algorithm	Best	Average	Worst	Space complexity
1) Bubble sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
2) Insertion sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
3) Selection sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(1)$
4) Quick sort	$O(N \log N)$	$O(N \log N)$	$O(N^2)$	$O(N)$
5) Merge sort	$O(N \log N)$	$O(N \log N)$	$O(N \log N)$	$O(N)$
Searching algorithm				
6) Linear search	$O(1)$		$O(N)$	
7) Binary search	$O(1)$		$O(\log_2 N)$	

Quick Sort

In quick sort we picks an element as pivot and partition the given array around the picked array.

Main process is partition function.

In Partition function we select an given array and element x of array as pivot, put x at its correct position in sorted array and put all smaller elements before x and put all greater element after x. All process of partition function is done in linear time.

Pseudo Code for Recursive Quick Sort()

```
low = string index, high = Ending index
quicksort(arr[], low, high)
{
  if (low < high)
{
  pi = partion (arr, low, high);
  quicksort (arr, low, pi-1);
  quick sort(arr, pi+1, high);
 }
}
```

Pseudo Code for Partition function

```
partition (arr[], low, high)
{
  pivot = arr[high];
     i= (low-1);
 for(j=low; j <= high; j++)
{ if (arr[j] < pivot)
 {
   i++;
   swap arr[i] & arr[j]
 }
}  swap arr[i+1] & arr[high];
   return (i+1)
}
```

```
ob1. data = 10;

struct node ob2;
    ob2. prev = NULL;
    ob2. data = 20;

Struct node ob3;
  ob3. prev = NULL;
  ob3. next = NULL;
  ob3. data = 30;
// forward line
ob1. next = &ob2;
ob2. next = &ob3;
// Backward line
ob2. prev = &ob1;
ob3. prev = &ob2;

// Accessing data of ob1, ob2, ob3 by using ob1
printf("%d\t", ob1.data);
printf("%d\t", ob1. next->data);
 printf("%d\t", ob1. next -> next -> data);

// Accessing data of ob1, ob2, ob3 by using ob2
printf("%d\t", ob2. prev -> data);
print("%d\t", ob2.data);
print("%d\t", ob2 next -> data);
// Accessing data of ob1, ob2, ob3 by using ob3
print("%d\t", ob3. prev -> prev -> data);
print("%d\t", ob3. data -> prev -> data);
printf("%d\t", ob3. data);
```

Linked List

Linked list is a linear data structure. In linked list we have two parts

Data Section → Numeric value

Adder section → Hold the address

Element in the list called Node.

Size of linked list is not fixed.

And data item can be added at any location in the list.

```
Struct linked list
{
  int data;
  Struct linked list * next;
}
```

Self Reference Structure

Those structure which have one or more pointers that points to the same type of structure as their member

```
Struct node
{
  int data1;
  char data2;
Struct, node * link;
}
```

Write a Program to display self reference structure with multiple list

```
# include <stdio.h>
  Struct node
  {
    int data;
    Struct node * prev;
    Struct node * next;
  }
    int main ()
  {
  Struct node ob1;
  ob1 prev = NULL;
  ob2 next = NULL;
```

POINTERS

Pointers stores address of variable or a memory location.
To access address of a variable to a pointer, we use unary operator (&) which returns the address of variable.

```
datatype * var_name ;
int *p;

# include <stdio.h>
{
  int x ;
  printf ("%p ", &x);
  return 0;
}
// print address of x
```

Another unary operator is Asterisk (*) which is used to

(1) To declare pointer variable

```
int * p;
```

Here p is pointer to an integer type variable

(2) To access value stored in address we use unary operator asterisk (*) which return value of variable located at address specified by operand.

Ex:

```
# include <stdio.h>
  int var = 12;
  int *ptr = &var;
  printf ("value of variable = %d ", * ptr);
  printf ("address stored of var = %p ", ptr);
  * ptr = 21;
  printf (" After changing , * ptr = 2 , * ptr is %d ", * ptr)
    return 0;
}
```

10	80	30	90	40	50	70
0	1	2	3	4	5	6

low = 0, high = 5, pivot = 70

Initialize index of smaller element

Traverse element of j from low to high-1

j=0; since arr[j] < pivot

do i++ and swap (arr[i] arr[j])

i=0

arr[] = [10, 80, 30, 90, 40, 50, 70]

j=1, since arr[j] > pivot, do nothing

No change in i and arr[]

j=2, since arr[j] < povit

do i++ and swap (arr[i], arr[j])

i=1

arr[] = {10, 30, 80, 90, 40, 50, 70}

j=3; since arr[j] > pivot // do nothing

No change in i j arr[]

j=4; since arr[j] <= pivot

do i++ and swap arr[i], arr[j]

i=2

arr[] = {10, 30, 40, 90, 80, 50, 70}

j=50; since [j] <= pivot

do i++ and swap arr[i], arr[j]

i=3

{10, 30, 40, 50, 80, 90, 70}

70 is at correct place

j=high-1, 6-1=5, loop terminates

Swapping arr[i+1] and arr[high]

arr[] = {10, 30, 40, 50, 70, 90, 80}