

ALGORITHMS

Algorithms means a step by step process.
It should begin with start/Begin and end with stop/end.

start/begin
→ to start brief at multiraps no start

Step 1
Step 2

Step 3
Stop/End

end/for set state
→ end, last sign

Program can be solved in three ways:-

→ Algorithm or box / flow chart or flow chart : Appt2
→ Flowchart
→

(1) A simple Algorithm:-

Start

Step 1: Print "Welcome"

Stop

<Appt2> short life

<Appt2> short life

() round box

3

(2) Write an algorithm to add two numbers.

Start

Step 1: Read two input values i.e., a and b →

Step 2: Add the two values c = a+b.

Step 3: Print the result i.e., c

Stop.

(3) Write an algorithm to find area and perimeter of a rectangle.

Start

Step 1: Read two input values i.e., l and b.

Step 2: Calculate area of a rectangle i.e., A = l * b.

Step 3: Calculate perimeter of a rectangle i.e., P = 2(l+b)

Step 4: Print the result i.e., A and P.

Stop.

→ to start brief at multiraps no start

(4) Write a program to find area and perimeter of a rectangle.

#include <stdio.h>

void main()

int l, b, A, P;

A = l * b;

P = 2(l+b);

printf ("enter l and b")

```

    scanf ("%d %d", &a, &b); INITIALIZATION
    printf ("%d", A);
    printf ("%d", P);
}

```

incorrect qnt, yet gets a correct output

{
 for loops & while loops after input handle it
 (ax^2+bx+c) term.

(5) Write an algorithm to find roots of a quadratic equation

Start

Step1: Read ^{the values} a, b, c .

Step2: calculate root1 i.e., $\text{root1} = (-b + \sqrt{b^2 - 4ac}) / 2a$

Step3: calculate root2 i.e., $\text{root2} = (-b - \sqrt{b^2 - 4ac}) / 2a$.

Step4: Print the result i.e., root1 and root2 through a

Stop.

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

Void main()

{

int a,b,c,x,y;

~~x =~~ $(-b + \sqrt{b^2 - 4ac}) / 2a$; ^{calculator out bits at multiplication no std lib}

~~y =~~ $(-b - \sqrt{b^2 - 4ac}) / 2a$; ^{calculator out bits at multiplication no std lib}

printf("enter a, b, c"); ^{calculator out bits at multiplication no std lib}

scanf ("%d %d", &a, &b, &c); ^{scanf "scanf" : input}

printf ("root1=%d", x); ^{calculator out bits at multiplication no std lib}

printf ("root2=%d", y); ^{calculator out bits at multiplication no std lib}

} ^{calculator out bits at multiplication no std lib}

printf ("root1=%d", x); ^{calculator out bits at multiplication no std lib}

printf ("root2=%d", y); ^{calculator out bits at multiplication no std lib}

} ^{calculator out bits at multiplication no std lib}

printf ("root1=%d", x); ^{calculator out bits at multiplication no std lib}

printf ("root2=%d", y); ^{calculator out bits at multiplication no std lib}

} ^{calculator out bits at multiplication no std lib}

(6) Write an algorithm to swap or interchange two values

o To interchange two words but at memory level o

Start

Step1: Read two input values i.e., a and b

Step2: Take $c=a$.

Step3: Take $b=c$.

Step4: Take $a=c$.

Step5: Print the results i.e., a and b .

Stop.

```

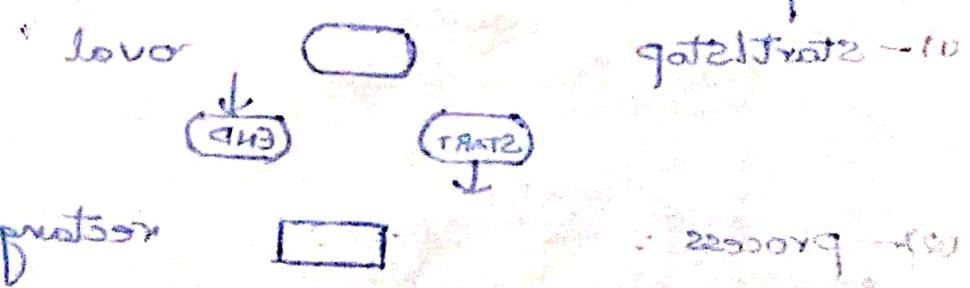
#include <stdio.h>

Void main()
{
    int a,b,c;
    printf("enter a and b");
    scanf ("%d %d",&a,&b);
    c=a;
    a=b;
    b=c;
    printf("%d",a);
    printf("%d",b);
}

```

(or)
 start : good process starts
 Step 1: Read two input values i.e., a and b.
 Step 2: Take $a = a + b$
 Step 3: $b = a - b$.
 Step 4: $a = a - b$
 Step 5: Print the result i.e., c and b.
 Stop.

include <stdio.h> // to print size of int
 void main() {
 int a,b;
 printf("enter a and b");
 a=b;
 b=a-b;
 printf("%d %d",a,b);
 }



Algorithm :- It is defined as finite set of steps that can be followed for solving a definite no. of problems.

The steps in an algorithm can be divided into 3 categories

(i) Sequence (ii) Selection (iii) Iteration (repetition)

(based on condition)

(iii) Sequence :- The steps described in an algorithm are performed successfully one by one without skipping any step.

(ii) Selection :- In case, the operation is unsuccessful, then the sequence of algorithm should be changed in such a way that the system will select the statement.

Ex:- if (condition)

then step1
otherwise step2

while using loop

initialise

(m)
for

incrementation

: nqstz

condition:

if (condition)

do stet : go to

else : go to

end if

end loop

end for

end program

(iii) Iteration :- In a program, sometimes it is necessary to perform the same action for a no. of times.

The statements written in an iteration block is executed for a given no. of times based on certain condition.

There are two basic ways to describe any activity

(i) Algorithm

(ii) Flowchart

By describing the process step by step called as Algorithm.
Also called Pseudocode.

By representing various steps in the form of a diagram called as Flowchart.

Flowchart :- It is a visual representation of the sequence of steps for solving a programme.

It is a set of symbols that indicate various operations in the program.

For every process there is a corresponding symbol in the flowchart.

(i) - start/stop



oval



(ii) - process :



rectangle

It is used for ~~data~~ ~~flow~~ ~~control~~ ~~functions~~ ~~variables~~ ~~values~~ ~~operations~~ ~~processes~~ ~~data~~ ~~handling~~ and ~~assigning~~ ~~values~~ ~~to~~ ~~variables~~ ~~and~~ ~~modifying~~ ~~existing~~ ~~values~~ ~~of~~ ~~variables~~ ~~and~~ ~~more~~.

Ex:-

```
z = a + b
a = 2
```



Parallelogram.

(3) Input & Output symbol

printf, scanf, Read input values.



Validation good

(4) Decision / Test symbol



Rhombus

Decision "if" process not

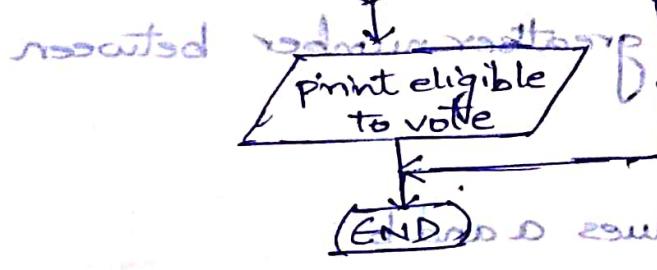
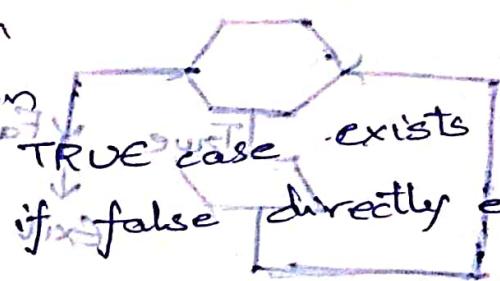
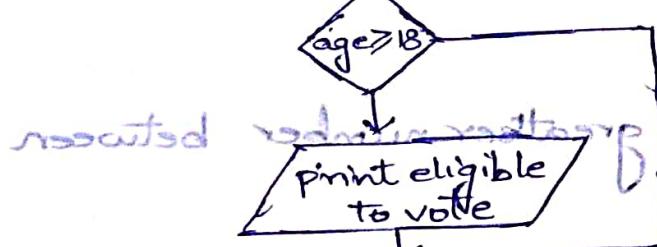
- single alternate

Decision
method :- If true to fi

- Two alternate Decision

Decision
method :- If true to fi

only TRUE case exists
if false directly end.



brief of multiple no stirke
- print out

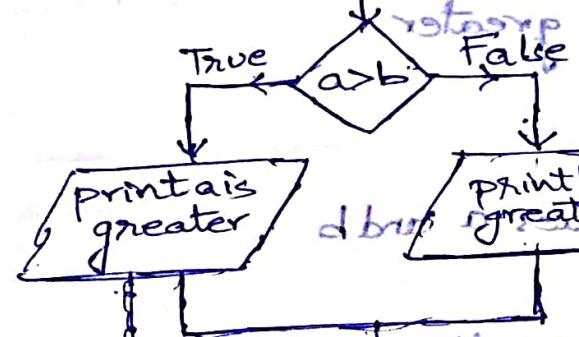
first

(END) so output first : 1952

Two alternate :- Two alternates exist if :- 1952

entry retsorp z i o ting nolt

retsrp z i d ting seiranto



output first :- 1952

gate
(no)
first

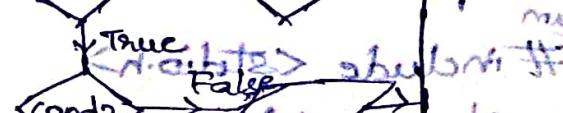
d b r i p i g r e a t e r ,

d < 0 fi do :- 1952

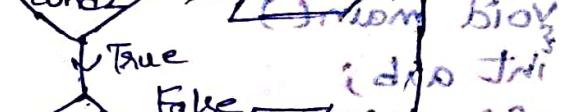
Multiple alternative decisions :- 1952



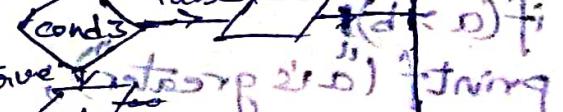
morph



bio



do fi



is

(5) connectorsymbol circle (A)
 connects two parts of a single flow chart from one to another page.



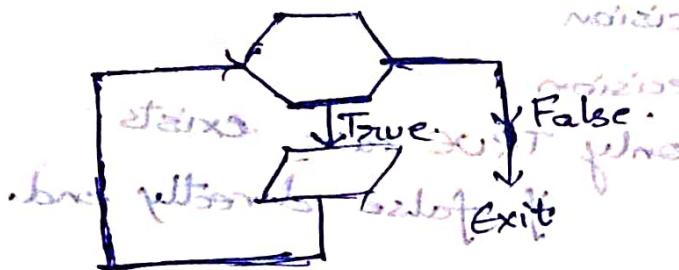
(6) Loop symbols:

for repeating 'n' no. of times

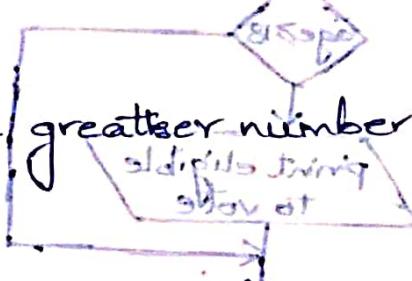


for repeating two & three times

for test / condition



if at any one condition, it is not false, then it goes to infinity.



Write an algorithm to find greater number between two numbers.

Start

Step 1: Read two input values a and b

Step 2: If $a \geq b$ then print a is greater
 otherwise print b is greater

Stop
(or)

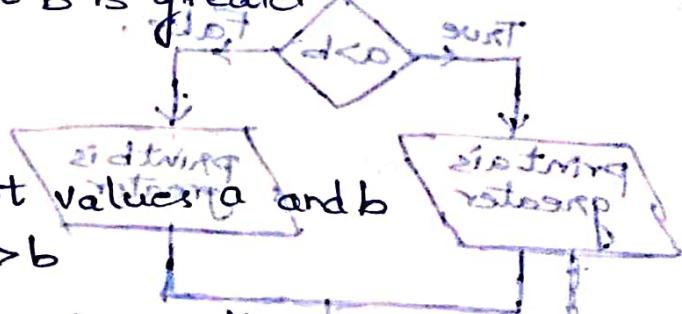
Start

Step 1:- Read two input values a and b

Step 2:- Check if $a > b$

Step 3:- Then print a is greater

Step 4:- Otherwise print b is greater



Program

```
#include <stdio.h>
void main()
{
    int a, b;
    if(a > b)
        printf("a is greater");
    else
        printf("b is greater");
}
```

printf("b is greater");

}

Start

Step 1:- Read two input values a and b .
Step 2:- if ($a > b$) then go to step 3
otherwise go to step 4.

Step 3:- Print a is greater and go to stop.

Step 4:- Print b is greater and go to stop.

= Write an algorithm to check the given year is it

leap year or not.

Start

Step 1:- Read year.

Step 2:- If (year % 4 == 0) then go to step 3
otherwise go to step 4.

Step 3:- Print it is a leap year and go to stop.

Step 4:- Print.

Stop.

```
#include<std.h>
```

```
void main()
```

```
{
```

```
int a;
```

```
if (a % 4 == 0)
```

```
scanf
```

```
("Enter the year");
```

scanf ("%d", &a);

```
if (a % 4 == 0)
```

```
printf ("it is a leap year");
```

```
else
```

```
printf ("it is a non-leap year");
```

```
}
```

(START)

"Enter the year"

(IF)

(IF)

"it is a leap year"

"it is a non-leap year"

(IF)

Write an algorithm to check the given number is even or odd number.

Start

Step 1:- Read the number and put it out here.

Step 2:- If ($\text{number} \% 2 == 0$) then go to Step 3

Otherwise go to step 4.

Step 3:- Print it is an even number and go to stop

Step 4:- Print it is an odd number and go to stop

Stop.

→ Write an algorithm to find out if the number is positive or negative.

Start

Step 1:- Read the number.

Step 2:- If ($\text{number} > 0$) then go to Step 3

Otherwise go to step 4.

Step 3:- Print the number is positive.

Step 4:- Print the number is negative.

Stop.

#include <stdio.h>

Void main()

{

int a;

printf ("Enter the number");

scanf ("%d", &a);

if ($a > 0$)

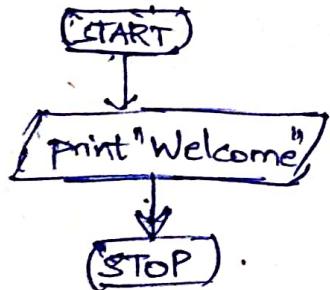
printf ("It is an even number");

else

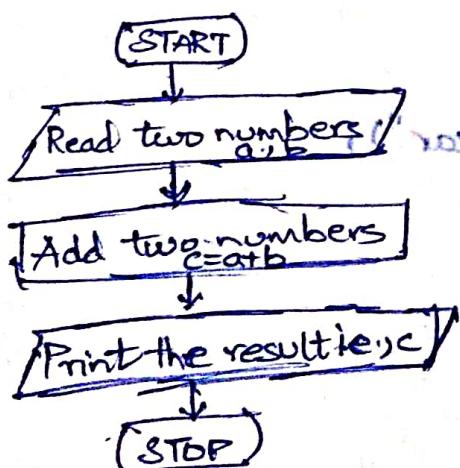
printf ("It is an odd number");

}

Write a flowchart to print a simple message "Welcome".



Draw a flowchart to add two numbers.



→ Write an algorithm to calculate the times.

Start

Step 1:- Print "Enter the number".

Step 2:- Initialize i=1.

Step 3:- Repeat

2.1 Print "Result = " + i

2.2 i = i + 1

Stop.

#include <stdio.h>

Void main()

{

int i=1;

for(; i=1; i++)

{

printf ("The result = %d", i);

}

}

→ Write an algorithm to check the given number is positive or negative.

Start

Step 1: Read the number.

Step 2: If (number > 0) then go to step 3
otherwise go to step 4.

Step 3: Print the number is positive and go to step 4.
Step 4: Print the number is negative and go to step 4.
Stop.

```
#include <stdio.h> // input output library
void main()
{
    int a;
    printf("Enter the number");
    scanf("%d", &a);
    if (a > 0)
        printf("It is positive number");
    else
        printf("It is negative number");
}
```

→ Write an algorithm to print 'Hello' message for 10 times.

Start

Step 1: Print Hello

Step 2: Initialise i=1

Step 3: Repeat the following steps until i=11
2.1 print "Hello"
2.2 i=i+1

Stop.

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

```
void main()
```

```
{
```

```
int i=1;
```

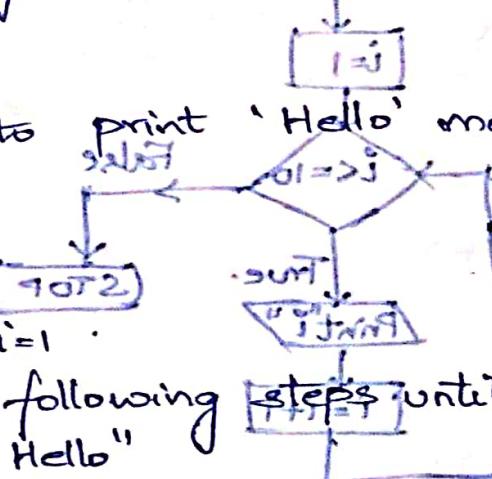
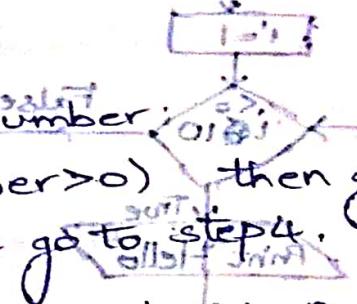
```
for(i=1; i<=10; i++)
```

```
{
```

```
printf("Hello");
```

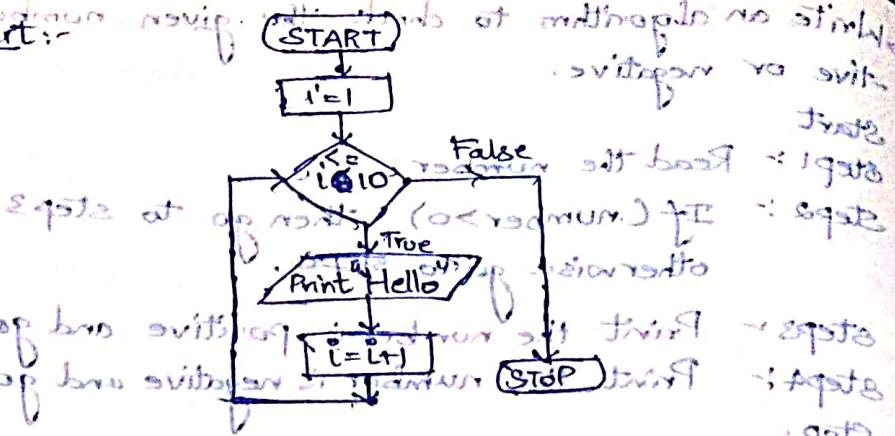
```
}
```

```
}
```



```
<stdio.h> abubri #include <iostream.h>
                {
                    i = j
                    (i + j) (0 = > i; i = j) ->
                    ("i") fwing
                }
```

Flowchart:-



→ Write an algorithm to print 1 to 10 numbers which are not multiples of 3.

Start

Step 1:- Read a number

Step 2:- Initialise i = 1

Step 3:- Repeat the following steps until i > 10

 3.1 print "i" value.

 3.2 i = i + 1

Step 4:- Print a stop

Flowchart:-

→ Write an algorithm to print 1 to 10 numbers which are not multiples of 3.

start

Step 1:- Initialise i = 1

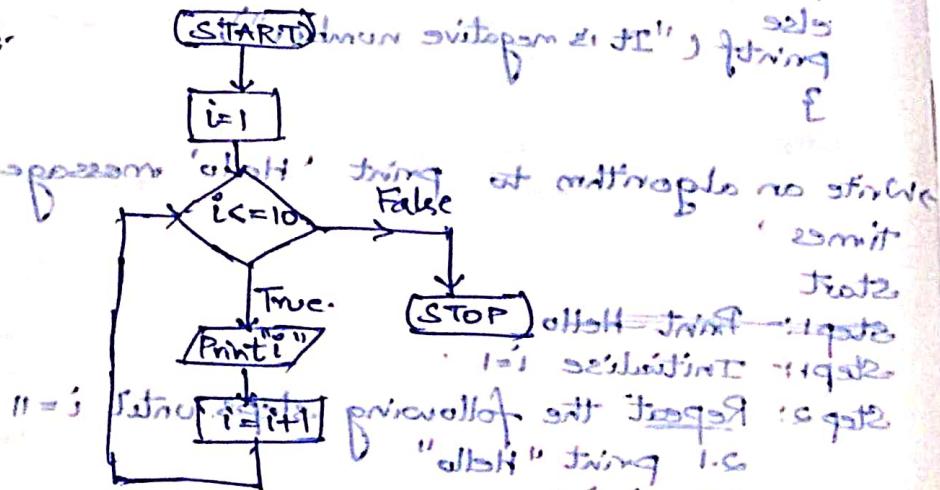
Step 2:- Repeat the following steps until i > 10

 2.1 print "i" value.

 2.2 i = i + 1

Stop

Flowchart:-



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

```
Void main()
```

```
{
```

```
int i=1;
```

```
for(i=1;i<=10;i++)
```

```
{
```

```
printf("i");
```

```
}
```

```
}.
```

<3.1> bno printing

(++i : 0 <= i <= 10) not

i ("allst") printing

i2 : bno = 20

```
# include <stdio.h>
Void main()
{
int i=1, s=0, n;
for(i=1; i <= n; i++)
{
    printf("%d", i);
    s=s+i;
}
printf("\n");
}
```

→ Write an algorithm to print sum of first n numbers.

Start

Step 1 :- Read a number n

Step 2 :- Initialise $i=1$ and $s=0$.

Step 3 :- Repeat the following step until $i=n+1$.

 3.1 ~~$i = i + 1$~~ $s = s + i$ \rightarrow sum of first i numbers is s

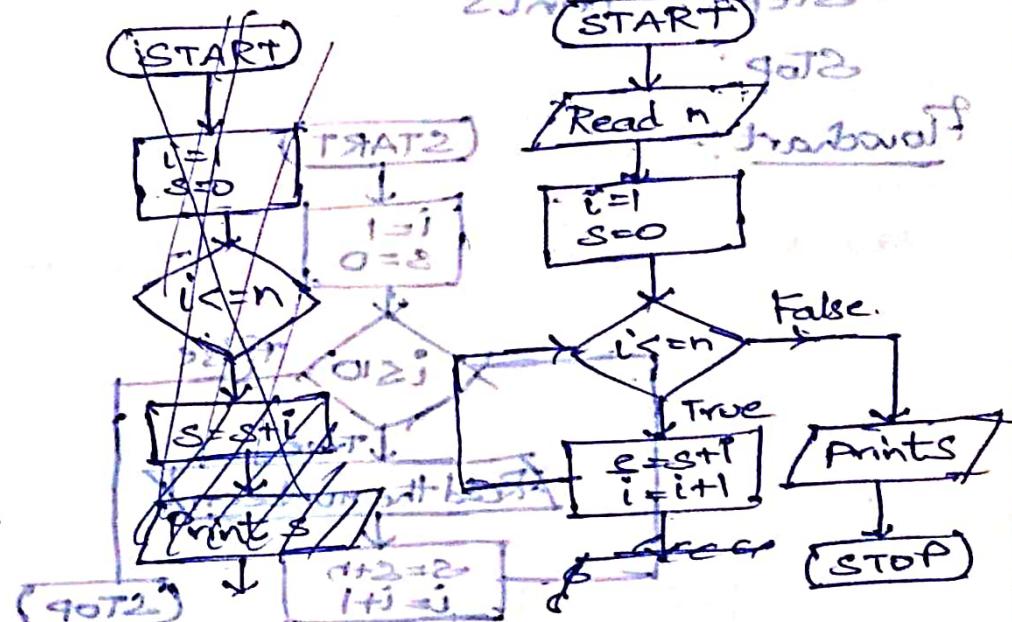
 3.2 $i = i + 1$

→ end of the loop

Step 4 :- Print s

Stop

Flowchart :-



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

```
void main()
```

```
{
```

```
int i=1, s=0, n;
```

```
for(i=1; i <= n, i++) {
```

```
    }
```

~~```
 printf
```~~

```
 s=s+i;
```

```
 }
```

```
printf("The sum of first n numbers is %d, %d", s);
```

```
}
```

( $\{2 \leq b \leq 2$  &  $i$  even or  $i$  odd for next step) fitting

$at\ b=2$

$\}$

→ Write an algorithm to find sum of any 10 different numbers.

Start

Step 1: Read 10 numbers Initialise  $i=1$  and  $s=0$ .

Step 2: Repeat the following steps until  $i=11$

2.1  ~~$i = i + 1$~~  Read a number  $n$

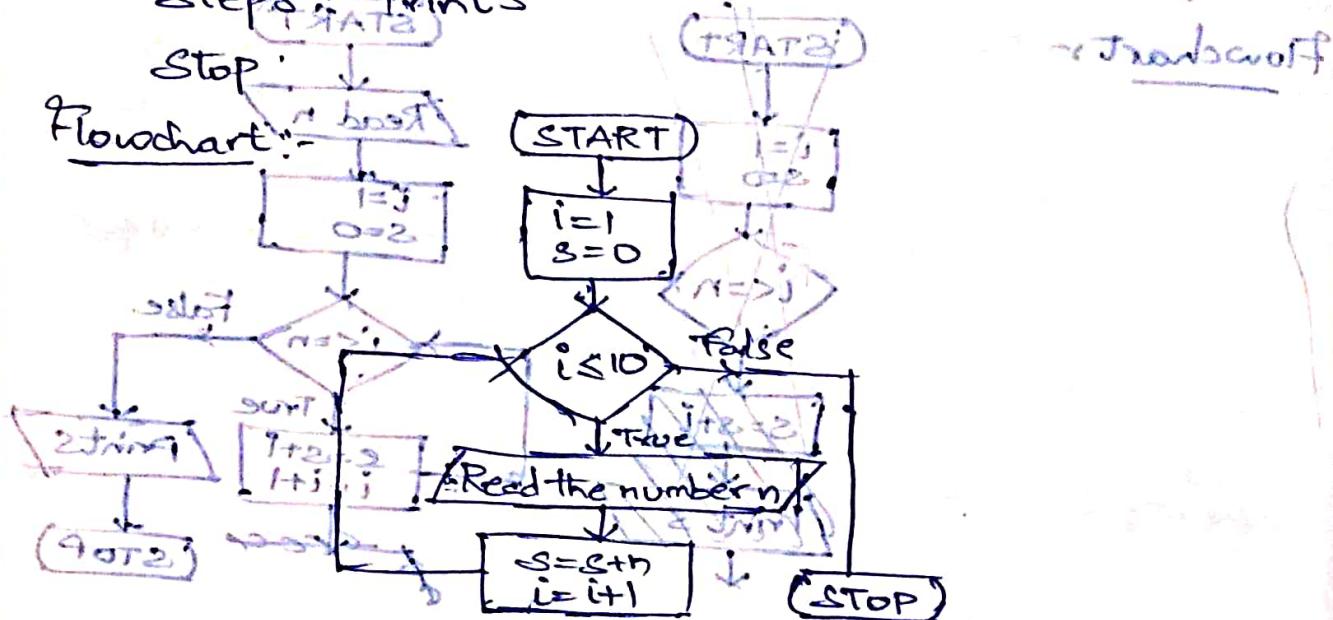
2.2  $s = s + n$

2.3  $i = i + 1$

Step 3: Prints

Stop

Flowchart:



Program :-

```
#include<stdio.h>
Void main()
{
 int i, s, n;
 for (i=1; i<=10; i++)
 {
 printf("Enter any number");
 scanf("%d", &n);
 s = s+n
 }
 printf ("The sum of any 10 numbers is %d", s);
}
```

→ Write an algorithm to convert temperature from fahrenheit to celsius and celsius to fahrenheit.

Start

Step1:- Read the temperatures of F and C. (Input)

Step2:- Read the temperature in Fahrenheit degree.

Step3:- Convert the Fahrenheit to Centigrade using

$$C = \frac{5}{9}(F - 32)$$

$$(C + 32) * \frac{5}{9} = F$$

Step4:- Print the Fahrenheit and Centigrade value.

Step5:- Read the temperature in degree Centigrade

Step6:- Convert the Centigrade to Fahrenheit

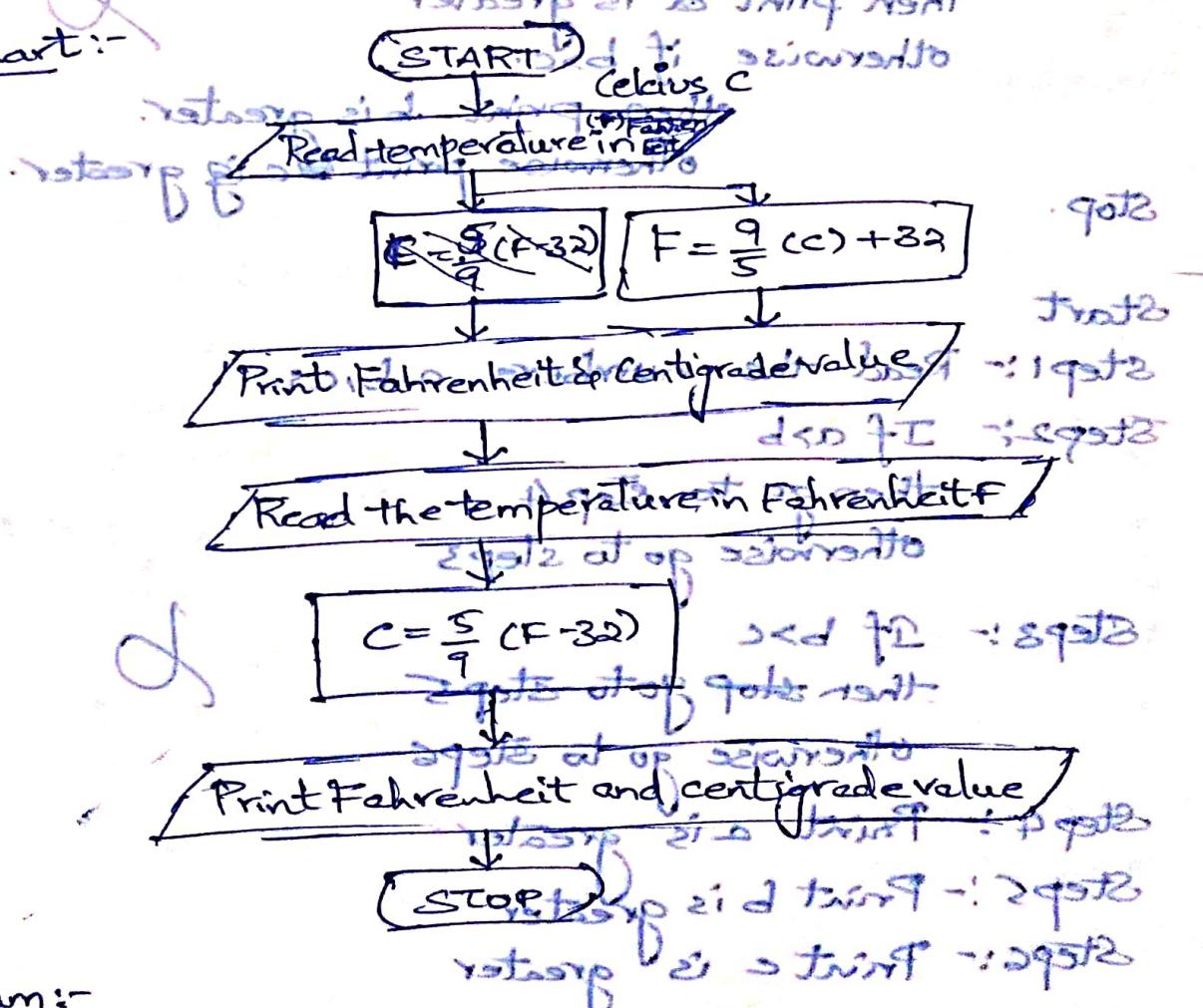
$$F = \frac{9}{5}C + 32.$$

(Output)

Step7:- Print the Fahrenheit and Centigrade value.

Stop.

Flowchart:-



Program:-

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

```
Void main()
```

```
{
```

```
int c,f;
```

```
printf("Enter the temperature in Fahrenheit");
```

scanf("%d", &f); // takes input from user  
 Enter the value of fahrenheit.  
 printf("%d", c); // prints the value of c  
 printf("Enter the temperature in Celcius");  
 scanf("%d", &c); // takes input from user  
 f = 1.8\*c + 32; // formula to convert Celcius to Fahrenheit  
 printf("f=%d", f); // prints the value of f

→ Write an algorithm to compare any three numbers  
 Start

Step 1:- Read the numbers a, b, c and print  
 Step 2:- If  $a > b$   
 then print a is greater  
 otherwise if  $b > c$   
 then print b is greater.  
 otherwise print c is greater.

Stop.

Start

Step 1:- Read the numbers a, b, c and print

Step 2:- If  $a > b$

then go to step 4 first or  
otherwise go to step 3

Step 3:- If  $b > c$

(CE-7)  $\frac{P}{F} = 5$

then stop go to step 5

otherwise go to step 6

Step 4:- Print a is greater

Step 5:- Print b is greater

Step 6:- Print c is greater

Stop.

morning

<Algorithm> solution

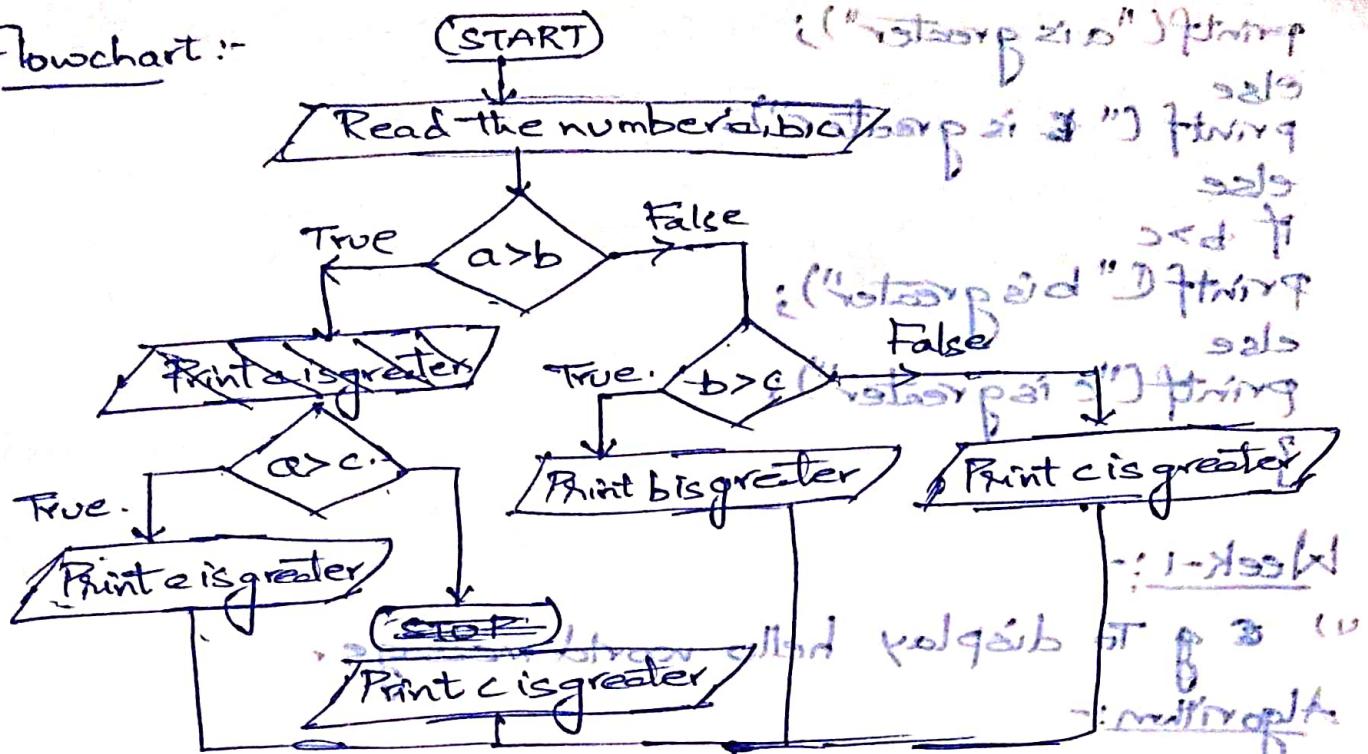
noon time

3

afternoon

: ("Fahrenheit is greater than Celsius") string

Flowchart :-



Algorithm :-

Start

Step 1 :- Read the numbers a,b,c

Step 2 :- If  $a > b$

then go to step 3

otherwise go to step 4

Step 3 :- If  $a > c$

then go to step 5

otherwise go to step 7

Step 4 :- If  $b > c$

then go to step 6

otherwise go to step 8

Step 5 :- Print a is greater.

Step 6 :- Print b is greater.

Step 7 :- Print c is greater.

Stop.

Program :-

```

#include<stdio.h>
Void main()
{
 int a,b,c;
 if a>b
 if a>c
}

```

```
printf("a is greater");
```

else

```
printf("b is greater");
```

else

```
if b > c
```

```
printf("b is greater");
```

else

```
printf("c is greater");
```

if a > b  
if a > c

Week-1:-

"e.g To display hello world message."

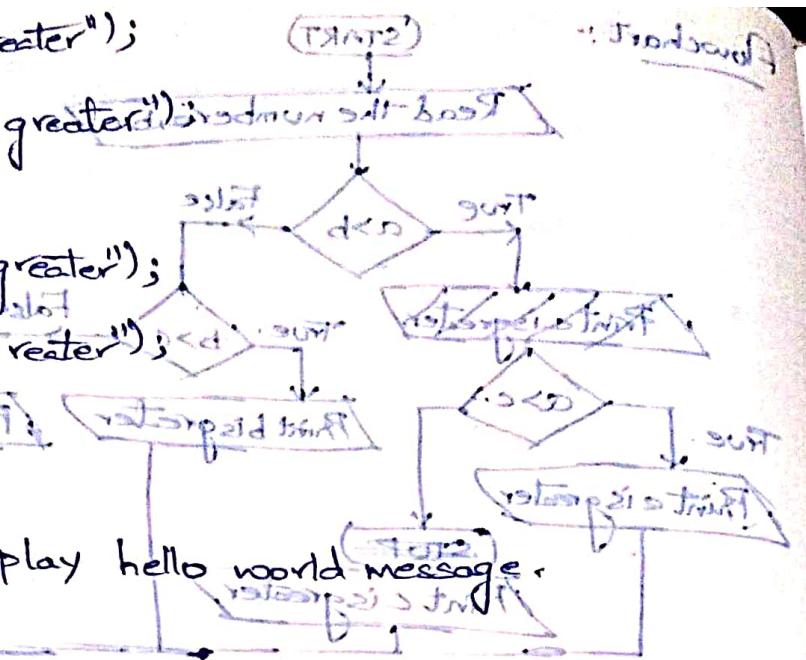
Algorithm:-

Start.

Step1:- Print Hello World.

Stop.

Flowchart:-



Flowchart:-



Program:-

```

#include<stdio.h>
void main()
{
 int a;
 char b;
 float c;
 double d;
 printf("Enter");
 scanf("%d", &a);
 printf("a");
 printf("Enter");
 scanf("%c", &b);
 printf("b");
 printf("Enter");
 scanf("%f", &c);
 printf("c");
 printf("Enter");
 scanf("%lf", &d);
 printf("d");
}

```

Program:-

```

// to print HelloWorld
#include<stdio.h>
void main()
{
 printf("HelloWorld");
}

```

(2) To scan all data types variables as input and print it as output.

Algorithm:-

Start

Step1:- Read the integer value, character value, float value and double value

Step2:- Print the corresponding integer value, character value, float value and double value.

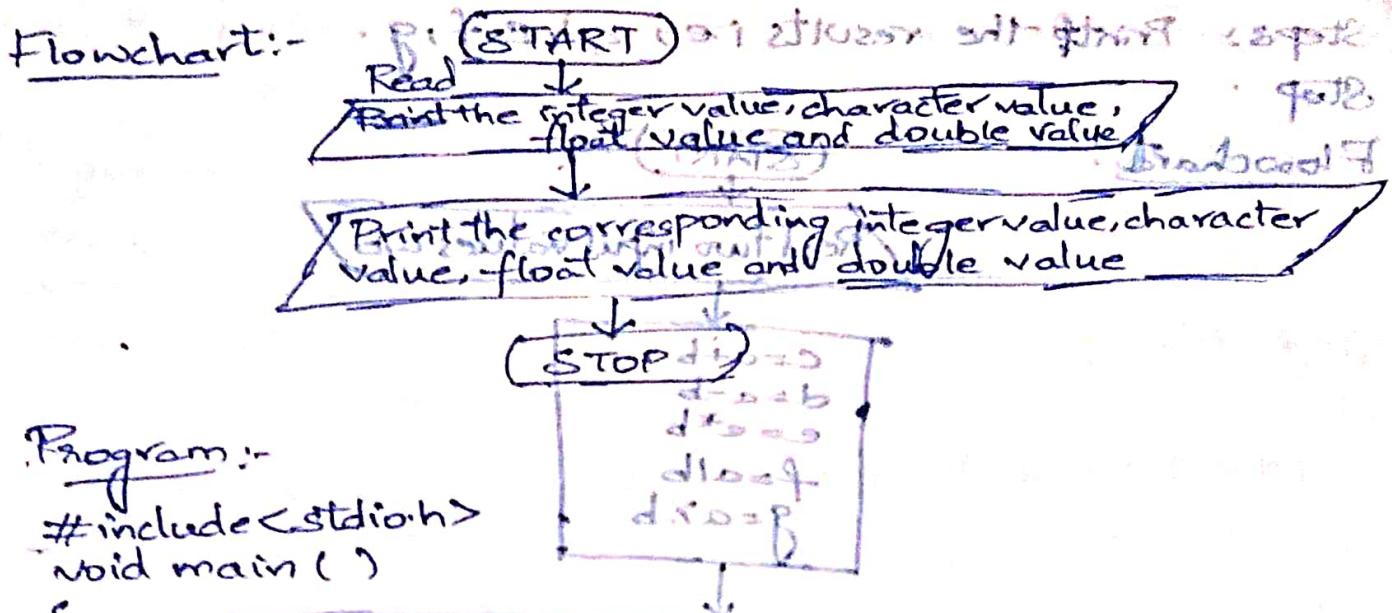
Stop.

Algorithm

Start

Step1:-

Step2:-



Program:-

```

#include<stdio.h>
void main()
{
 int a; /* Input */
 char b; /* Input */
 float c; /* Output */
 double d; /* Output */
 printf("Enter the integer value a"); /* Output */
 scanf("%d", &a); /* Input */
 printf("a=%d", a); /* Output */
 printf("Enter the character value b"); /* Output */
 scanf("%c", &b); /* Input */
 printf("b=%c", b); /* Output */
 printf("Enter the float value c"); /* Output */
 scanf("%f", &c); /* Input */
 printf("c=%f", c); /* Output */
 printf("Enter the double value d"); /* Output */
 scanf("%lf", &d); /* Input */
 printf("d=%lf", d); /* Output */
}

```

(1) Input      (2) Output      (3) Processing

Q) To perform arithmetic operations, like +, -, \*, /, % on two input variables.

Algorithm:-

Start

Step 1:- Read two input values a and b.

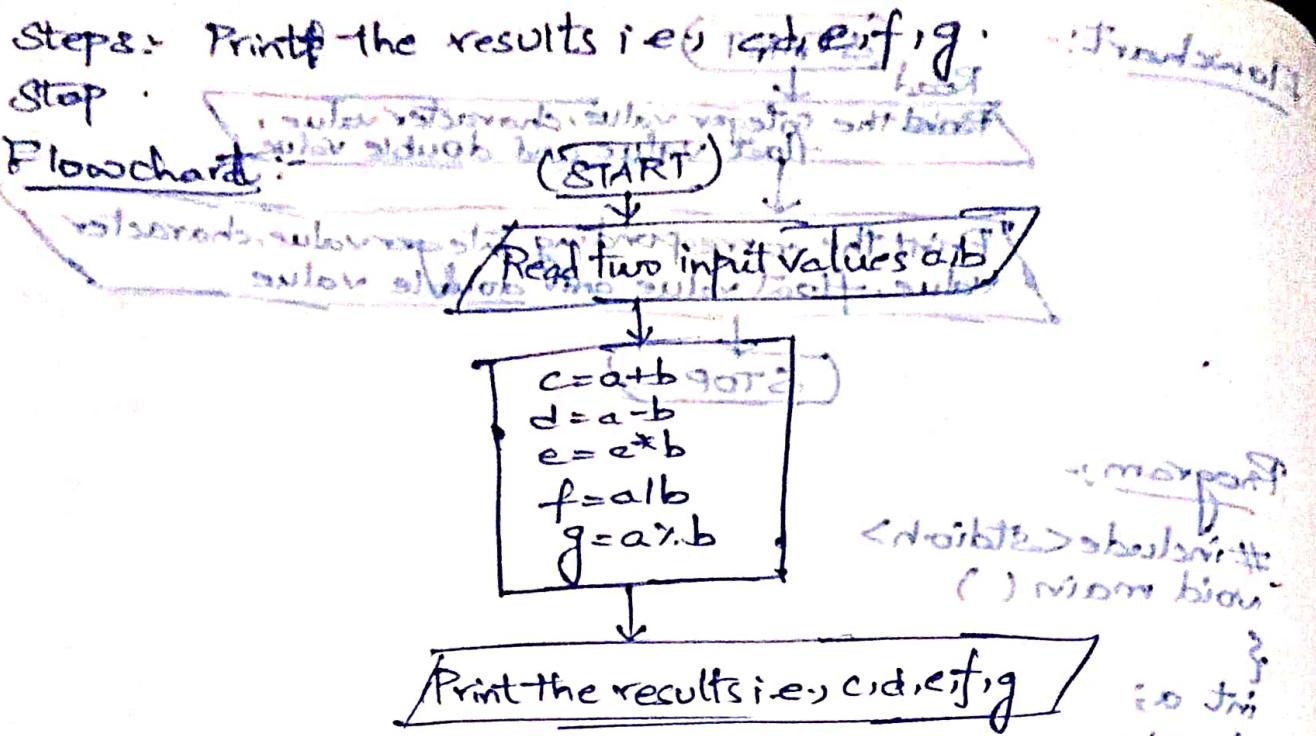
Step 2:- Add two values i.e.,  $c = a+b$ .

Subtract two values i.e.,  $d = a-b$

Multiply two values i.e.,  $e = a*b$ .

Divide two values i.e.,  $f = a/b$ .

Give the remainder i.e.,  $g = a \% b$ .



Program:-

```

#include <stdio.h>
void main()
{
 float a,b;
 c=a+b;
 d=a-b;
 e=a*b;
 f=a/b;
 g=a%b;
 printf("Enter any two numbers");
 scanf("%f %f",&a,&b);
 printf("The sum is : c=%f , difference is d=%f , product

is e=%f , ratio is f=%f and remainder is g=%f,

and c,d,e,f,g");
}

```

d is a scalar variable out float  
d+a=5.5 is a scalar out float  
d-a=2.5 is a scalar out float  
d\*a=13.5 is a scalar out float  
d/a=1.8 is a scalar out float  
d%a=1.5 is a scalar out float

(4) To perform temperature conversions from centigrade to Fahrenheit and vice versa.

$$(68+32) \times \frac{5}{9} = 7$$

Algorithm:-

Start

Step1:- Read the temperature in Fahrenheit degree.

Step2:- Convert the Fahrenheit to Centigrade using

$$C = \frac{5}{9}(F - 32)$$

Step3:- Print the Fahrenheit and Centigrade value

Step4:- Read the temperature in Centigrade degree

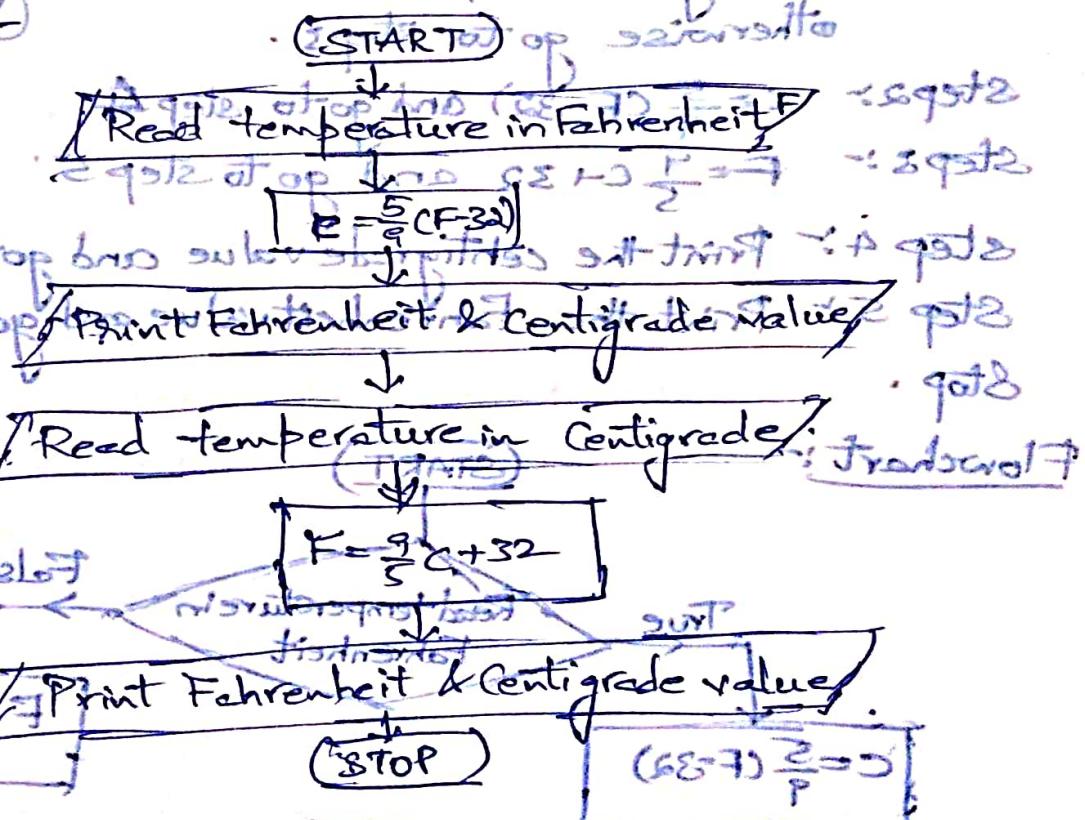
Step5:- Convert the Centigrade to Fahrenheit value

Step6:- Using  $F = \frac{9}{5}C + 32$

Step7:- Print the Fahrenheit and Centigrade value

Stop

Flowchart:-



Program:-

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

```
Void main()
```

```
{
```

```
int F,C;
```

```
printf("Enter the temperature in Fahrenheit");
```

```
scanf("%d",&F);
```

$$C = \frac{5}{9}(F - 32);$$

```
printf("C=%d",C);
```

```
printf("Enter the temperature in Centigrade");
```

~~scanf("x.d", &C);~~ read centigrade value from keyboard  
~~F =  $\frac{9}{5}C + 32;$~~   
~~printf("F=%d", F);~~  
~~};~~

Algorithm (Method) of finding out Fahrenheit value from Centigrade value  
 Start

Step 1: ~~Read C (Fahr.)~~ ( $F=32$ ) and go to next step

Step 2:  ~~$F = \frac{9}{5}C + 32$~~  and go to next step

Step 3: ~~Read the If Read the temperature in Fahrenheit~~  
 Start

Step 1: ~~If Read the temperature in Fahrenheit = 32~~  
 then go to step 2.  
 otherwise go to step 3.

(X) Next step

Step 2:  ~~$C = \frac{5}{9}(F - 32)$~~  and go to step 4.

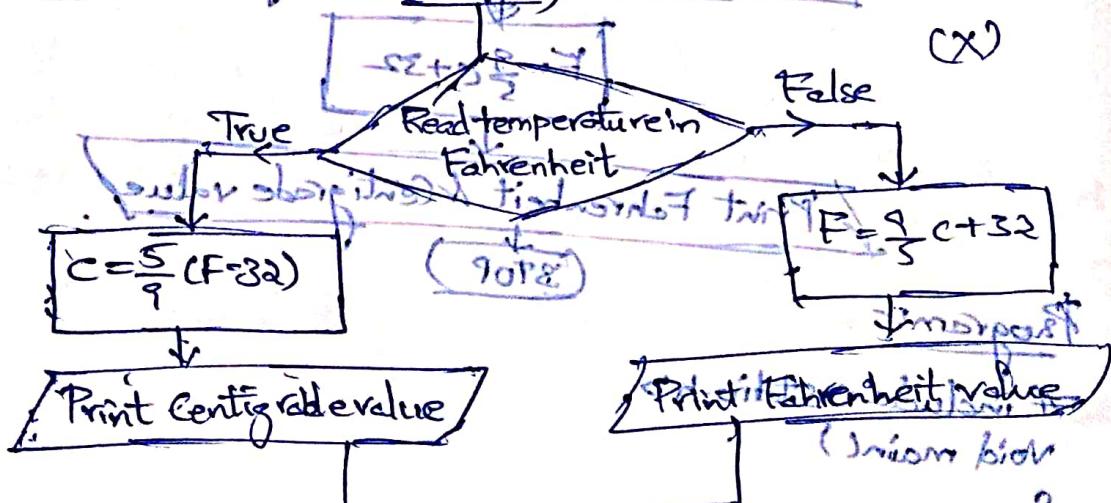
Step 3:  ~~$F = \frac{9}{5}C + 32$~~  and go to step 5.

Step 4: Print the centigrade value and go to stop.

Step 5: Print the Fahrenheit value and go to stop.

Stop.

Flowchart:



i ("Fahrenheit ni enter")  
 i ("3.7 fah")  
 i ("7.21 b.v.") fah  
 i ((58-7)\*2/5 = 3  
 i ("Lx=3") fah  
 i ("shorpitve ni enter") fah

Program:  

```

#include <stdio.h>
void main()
{
 int c,f;
 if
 printf("Enter");
 scanf("%d", &c);
 f = c * 9 / 5 + 32;
 printf("F=%d", f);
 else
 printf("F=%d", f);
}

```

Start

Step 1:

C to F & F

#include

void main()

int c,f;

char x;

printf("R

scanf("I

#include

void main()

int a,b;

printf

scanf

print

printf

scanf

if (c

{

printf

b = (

print

3

else

Program to convert temperature between Fahrenheit & Celsius

```
#include <stdio.h>
void main()
{
 int c,F;
 if(c=5/9*(F-32), "cannot convert") {
 F = 9/5*c + 32;
 printf("Enter temperature in Fahrenheit", "(c*(F-32)/9) = 0");
 scanf("%d", &F);
 printf("C = %d", C);
 } else
 {
 scanf("%d", &c);
 printf("F = %d", F);
 }
}
```

Start  
Step :-

°C to F & F to °C acc. to user's choice :-

```
#include
void main()
{
 int c,f;
 char x;
 printf("Read temperature and character x=c or f");
 scanf("%d%d", &c, &x);
 #include
 void main()
 {
 int a,b,x;
 printf("Enter temperature");
 scanf("%d", &a);
 printf("1. Celsius to Fahrenheit\n2. Fahrenheit to Celsius");
 printf("Enter user's choice");
 scanf("%d", &x);
 if(x==1)
 {
 printf("Entered temperature is in degree celsius");
 b = (9*c/5) + 32;
 printf("Temperature in fahrenheit is %d", b);
 } else if(x==2)
 }
```

Perform arithmetic operations  
users choice:

<digit> shubham  
(Innum kia)

{  
printf("Entered temperatures is %f in Fahrenheit", f);  
 $\theta = \frac{5}{9}(b - 32) + 32$ ,  $(58 + 5 \times 32) / 9 = 7$   
printf("Temperature in celsius is %f", f);  
}  
(X)  
else.  
printf("re-enter choice");  
}.

; (58, "b.x") free  
(7, "b.x = 7") fitting  
,

fitting  
"1.952

selections made at 0.00 2.00 7.00 7.00

shubham  
(Innum kia)  
(fis jni  
; x=0.00

i("Please enter two numbers between 0 and 1000") fitting  
(x1, "b.x") free

shubham  
(Innum kia)  
(x1=x2 jni  
("Please enter two numbers between 0 and 1000") fitting

(x2, "b.x") free

i("Please enter two numbers between 0 and 1000") fitting  
("x1 < x2 or x2 > x1") fitting  
("x1 < x2 or x2 > x1") fitting

("x1 < x2 or x2 > x1") fitting

(x2, "b.x") free

(1=x2) fi

i("x1 is less than or equal to x2 or x2 is less than or equal to x1") fitting

$(58 + (2 \times 32)) = 100$

i("After entering two numbers in range") fitting

{  
(x2 = x2) fi sets

Write an algorithm to find sum of the following series

$$1+x+x^2+x^3+\dots +x^{10}.$$

(A. objective) shubham

Start.

Step1:- Read ~~value~~  $x$ .

(Initial value)

Step2:- Initialise  $i=1, s=1$ .

Step3:- Repeat the following steps until  $i \geq 10$ .

$$s = s + \text{pow}(x, i)$$

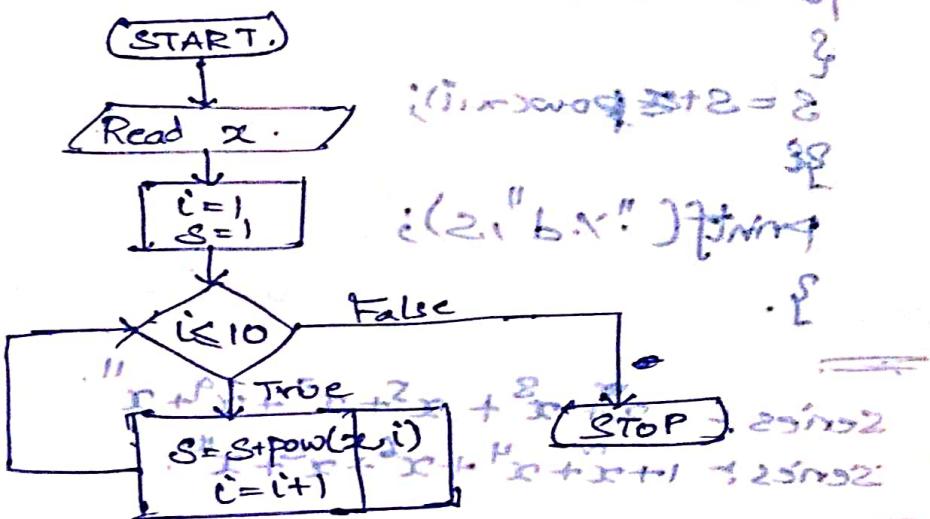
$$i = i + 1$$

Step4:- Print the result ~~s~~.

(Output value)

Stop.

Flowchart :-



→ draw script for output brief of algorithm no struck

Step1:- Read  $x$ .

(Input)

Step2:- Initialise  $i=0$ , ~~value~~  $s=0$  at least  $\approx 1992$

Step3:- Repeat the following steps until  $i \geq 10$ .

$$s = s + \text{pow}(x, i)$$

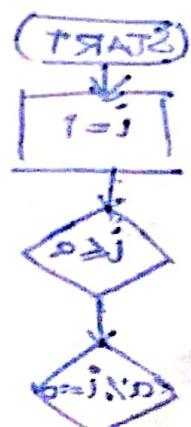
$$i = i + 1, (0 \leq i \leq 10) \text{ fi}$$

Step4:- Print ~~value~~,  $i$ . ~~using next~~

Stop.

$i+1 = j$  ~~for loop~~

pt2



→ flow chart

Program:- write a program to find sum of first n natural numbers.

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

```
void main()
```

```
{
```

```
int i;
```

```
i=0; sum=0; i++;
```

```
for(i=0; i<10; i++)
```

```
{
```

```
sum = sum + i;
```

```
printf("Sum of first 10 natural numbers is %d", sum);
```

```
for(i=0; i<10; i++)
```

```
{
```

```
s = s + pow(i, 2);
```

```
}
```

```

graph TD
 Start((START)) --> Init[i=1]
 Init --> Cond{i <= n}
 Cond --> Calc[s = s + i2]
 Calc --> Inc[i = i + 1]
 Inc --> Cond
 Cond --> Print["Print s"]
 Print --> Stop

```

```
printf("%d", s);
```

```
}
```

$$\text{Series: } \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \frac{x^8}{8!} + \frac{x^9}{9!} + \frac{x^{10}}{10!}$$

- Write an algorithm to find factors of a given number.

Start

Step 1:- Read the number  $a$ .  $a = 12$

Step 2:- Initialize  $i = 1$ .  $i = 1$

Step 3:- Repeat the following steps until  $i = a$  or  $i > a$ .

3.1:- if ( $a \% i == 0$ ) .  $i = i$

then print  $i$ , ~~if i < a~~  $i = i + 1$

else  $i = i + 1$

Stop.

Flowchart:-

```

graph TD
 Start((START)) --> Init[i=1]
 Init --> Cond{i <= a}
 Cond --> Factor["a % i == 0"]
 Factor --> Print["Print i"]
 Factor --> Inc[i = i + 1]
 Inc --> Cond
 Cond --> Stop

```

```

graph TD
 Start((START)) --> Init[i=1]
 Init --> Prime{Is i prime?}
 Prime --> Print["Print i"]
 Print --> Inc[i = i + 1]
 Inc --> Prime
 Prime --> Stop

```

Program:-

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

```
void main()
```

```
{
```

```
int i;
```

```
printf("Prime numbers between 1 and 10 are:
```

```
for(i=1; i<10; i++)
```

```
{
```

```
int flag=0;
```

```
for(j=2; j<i; j++)
```

```
{
```

```
if(i%j==0)
```

```
{
```

```
flag=1;
```

```
}
```

```
if(flag==0)
```

```
{
```

```
printf("%d ", i);
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

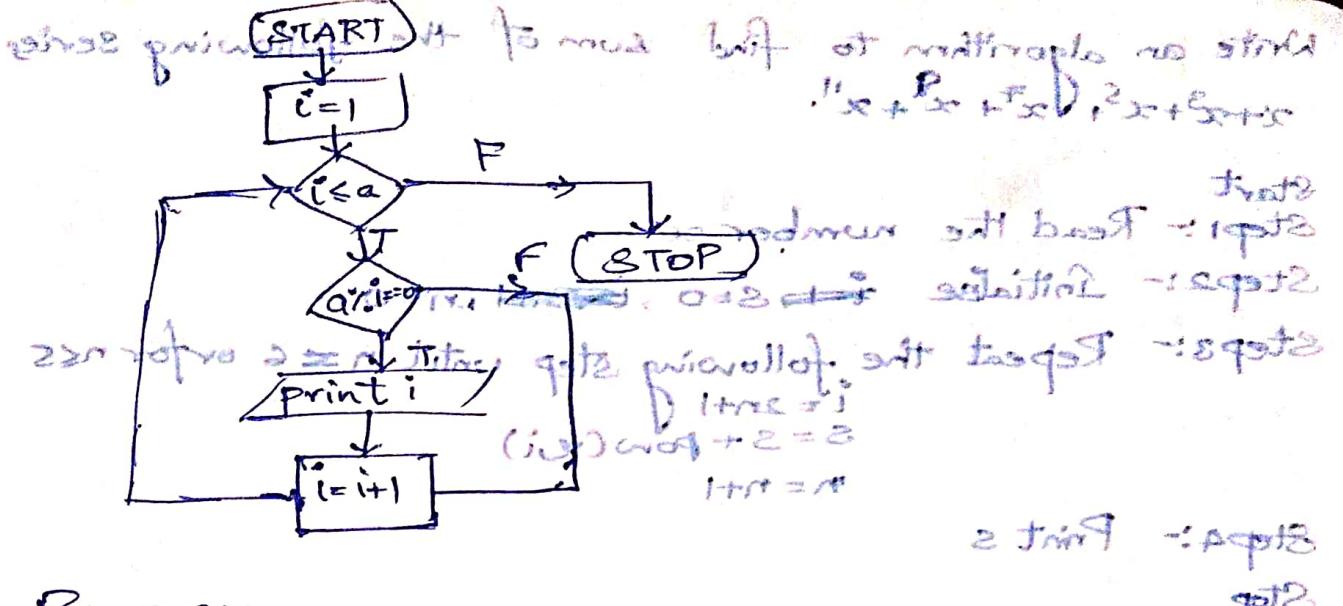
```
}
```

```
}
```

```
}
```

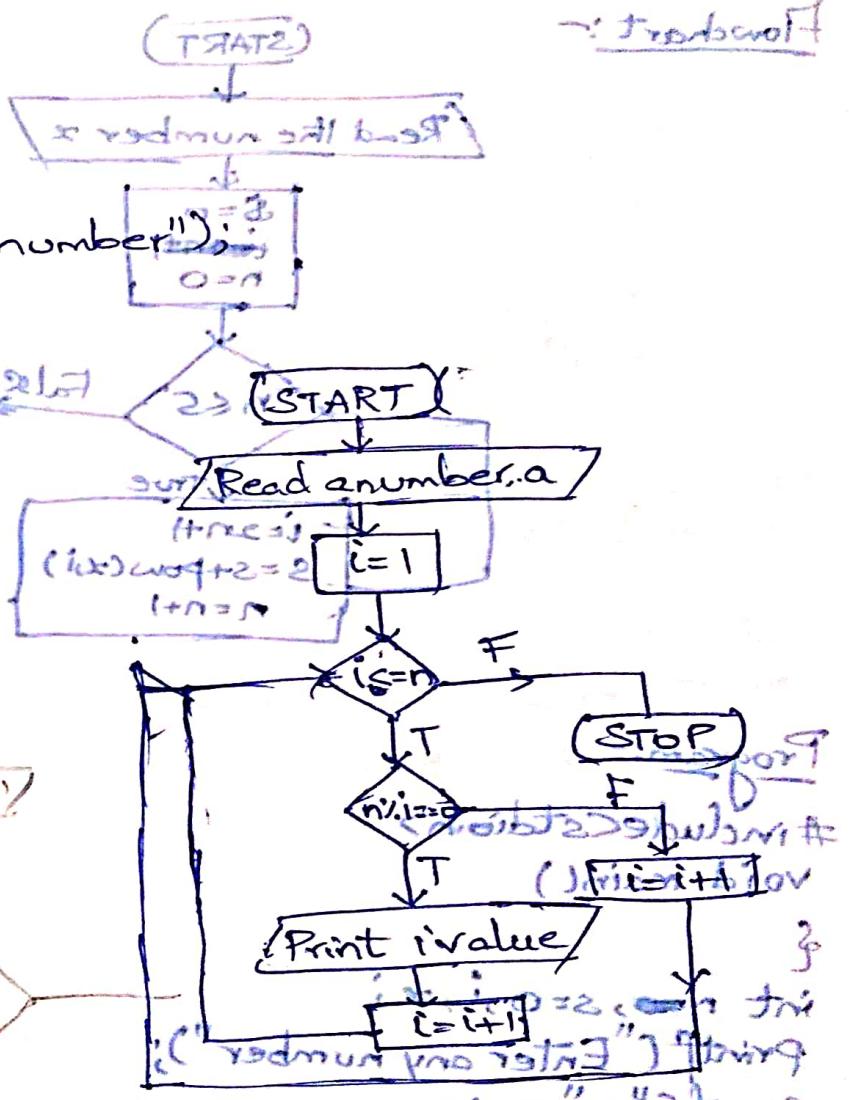
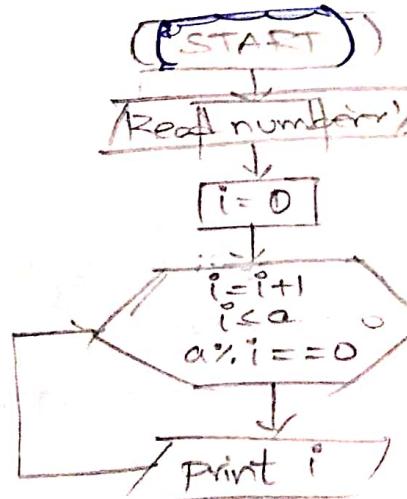
```
}
```

```
}
```



Program:-

```
#include <stdio.h>
Void main()
{
 int i,a;
 printf("Enter any number");
 scanf ("%d", &a);
 for(i=1, i <= a, i++)
 {
 if (a % i == 0)
 printf ("%d", i);
 }
}
```



$(a \% b = 1) \text{ if } n \neq 1$

$(i+1) \text{ mod } 2 = 1$

Write an algorithm to find sum of the following series

$$x+x^3+x^5, \sqrt{x^7+x^9+x^{11}}$$

Start

Step 1:- Read the number  $x$

Step 2:- Initialise  $s=0$ ,  $n=0$

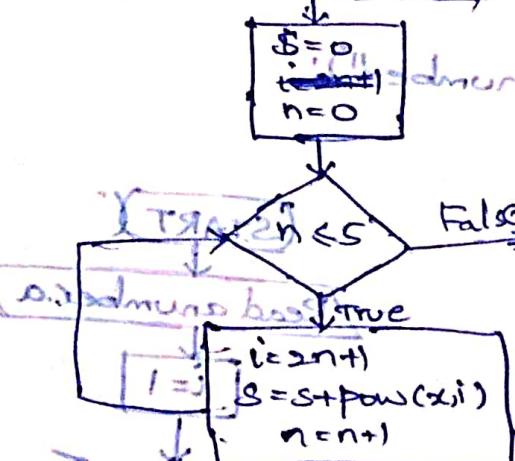
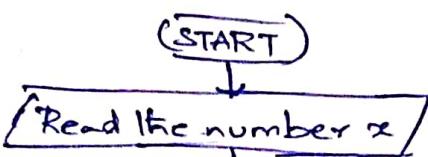
Step 3:- Repeat the following step until  $n \leq 5$  or  $n > 5$

$$\begin{aligned} i &= 2n+1 \\ s &= s + \text{pow}(x, i) \\ n &= n+1 \end{aligned}$$

Step 4:- Print  $s$

Stop

Flowchart :-



<--> *(n <= 5) true*

*(n <= 5) false*

*(n <= 5) true*

*(n <= 5) false*

*(n <= 5) true*

*(n <= 5) false*

Program :-

```
#include<cslibio.h>
void main()
{
 int n=0, s=0, i, x;
 printf("Enter any number");
 scanf("%d", &x);
 for (n=0, n<=5, n++)
 {
 i=2n+1;
 s= s + pow(x, i);
 }
 printf("%d", s);
}
```

Write an algorithm to find sum of the following series

Start

Step 1:- R

Step 2:-

Step 3:-

Step 4:-

Step 5:-

Flowchart

Flowchart

Prog

#

no

s

int

pr

s

+

g

,

Write an algorithm to find sum of following series  

$$1+x^2+x^4+x^6+\dots+x^{10}$$

**Start**

Step 1:- Read the number  $x$  (Input,  $x=2$ )  
 Step 2:- Initialize  $s=0$ ,  $n=0$  value of  $i$  till  $i = 10$   $\Rightarrow s=0$

Step 3:- Repeat the following step 2 until  $n = 6$  or for loop  
 (Initial)  $s=0$ ,  $i=2$ ,  $n=0$

(If)  $i \leq 10$        $s=s+pow(x,i)$

$i=i+2$

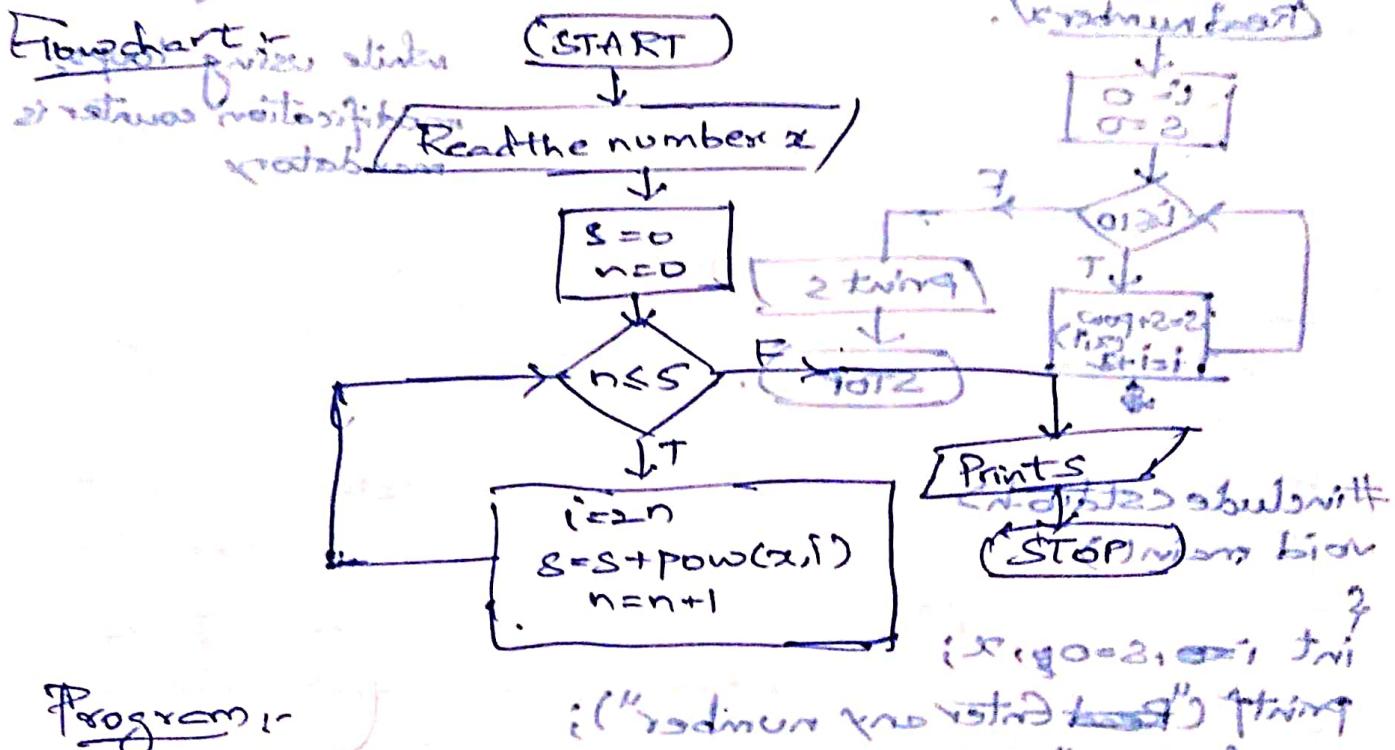
$n=n+1$

return result from loop

Step 4:- Print  $s$

(Final result)  $s=102$

Step 5:- Stop



Program :-

```

#include <stdio.h>
void main()
{
 int n, s=0, i, x;
 printf("Enter any number");
 scanf("%d", &x);
 for(n=0, n≤5, n++)
 {
 i=2n;
 s=s+pow(x,i);
 }
 printf("The sum is %d", s);
}

```

$\{ (" \text{read number and store in } s") \text{ thing}$   
 $\{ (" \text{scanf } b.x") \text{ from } s$   
 $\{ (s+i=i, o=i, o=i) \text{ for }$

$\{ (i=2n+2=8$   
 $\{ (2, b.x) \text{ thing}$

$\{ (2, b.x) \text{ thing}$

$$1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10}$$

Start

Step 1:- Read the number  $x$ .

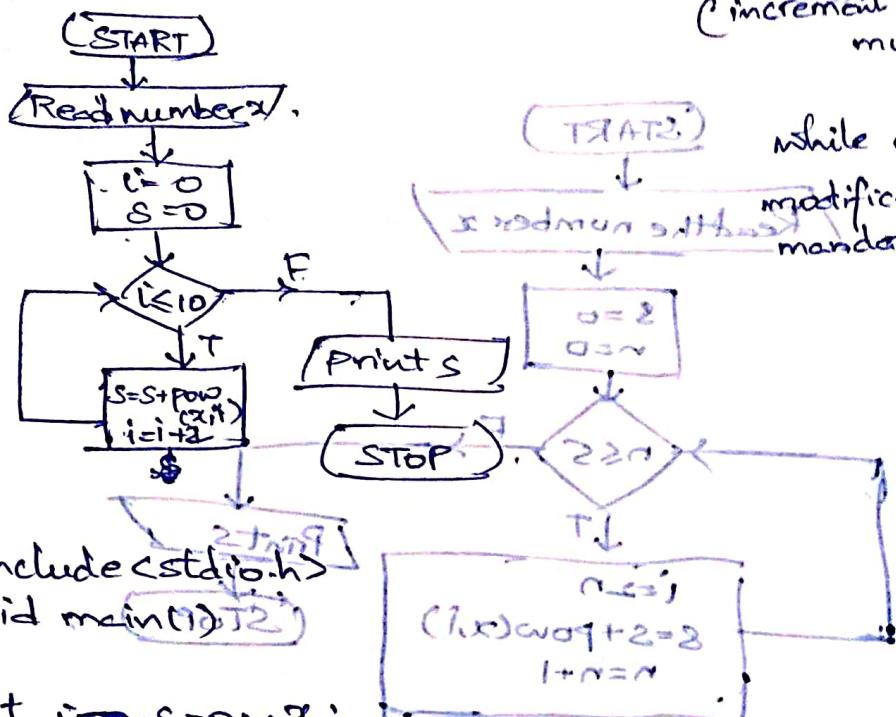
Step 2:- Initialise  $s=0, i=0$

Step 3:- Repeat the following steps for  $i \leq 10$

$s = s + pow(x, i)$  all of this is for loop  
 $i = i+2$

Step 4:- Print  $s$

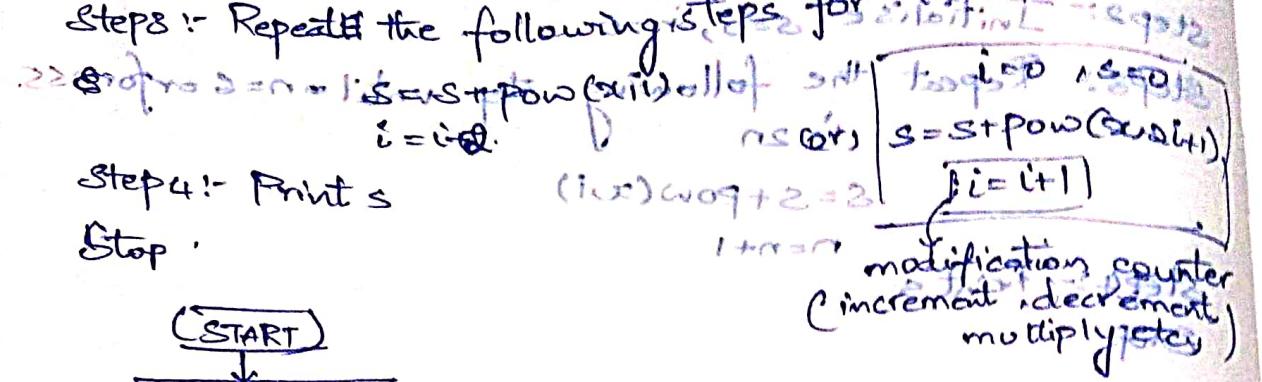
Stop



```

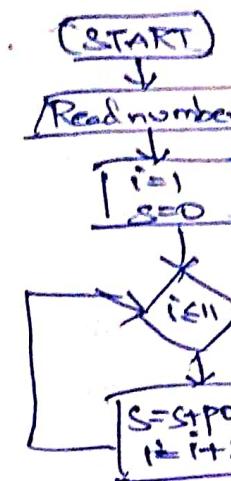
#include <csdio.h>
void main()
{
 int i=0, s=0, x;
 printf("Enter any number");
 scanf("%d", &x);
 for(i=0, i<=10, i=i+2)
 {
 s = s + pow(x,i);
 }
 printf("%d", s);
}

```



Start  
 Step 1:- Read  
 Step 2:- Initialise  
 Step 3:- Repeat

Step 4:- Print  
 Stop



```

#include <csdio.h>
void main()
{
 int i=1, s=0, x;
 printf("Enter any number");
 scanf("%d", &x);
 for(i=1, i<=n, i=i+1)
 {
 s = s + pow(x,i);
 }
 printf("%d", s);
}

```

↳ main part  
 ↳ while loop part  
 ↳ increment part

↳ redun part  
 ↳ for part  
 ↳ for part

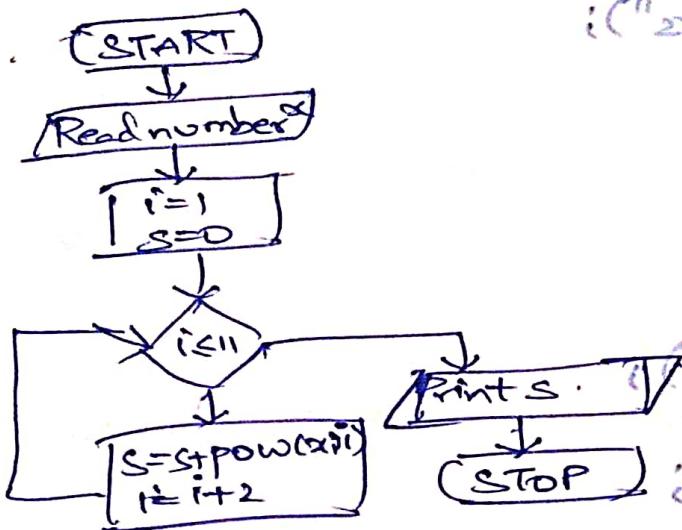
↳ i=1  
 $(1/x)^{wog+2}=2$

↳ (2, "b") fitting

$x^3 + x^5 + x^7 + x^9 + x^{11}$ . Initialize s=0  
 Start  
 Step 1:- Read the number x.  
 Step 2:- Initialise  $s=0$ ,  $i=1$   
 Step 3:- Repeat the following steps for  $i \leq 11$ .  

$$s = s + \text{pow}(x, i)$$
  
 $i = i + 2$

Step 4:- Prints  
Stop.



```

#include <stdio.h>
void main()
{
 int i, s=0, x;
 printf("Enter any number");
 scanf("%d", &x);
 for (i=0, i<=11, i=i+2)
 {
 s = s + pow(x,i);
 }
 printf("%d", s);
}

```

$\rightarrow$   $(d=0) \rightarrow$   
 $(d=2) \rightarrow$   
 $(d=4) \rightarrow$   
 $(d=6) \rightarrow$   
 $(d=8) \rightarrow$   
 $(d=10) \rightarrow$

Perform arithmetic operations. "+, -, \*, /" + on user choice.

Start

Step 1:- Read two numbers & choice

Step 2:- If choice is 1 print sum of both numbers = 19352

#include<stdio.h>

Void main()

{

int a,b,c,x;

printf("Enter two numbers");

scanf("%d %d", &a, &b);

- printf (" 1. Addition in  
2. Subtraction in  
3. Multiply in  
4. Divide in  
5. Give remainder");

scanf

printf("Enter your choice");

scanf("%d", &x);

if (x==1)

{

printf("Adding");

c=a+b;

printf ("%d", c);

{

else if (x==2)

{

printf("Subtracting");

c=a-b;

printf ("%d", c);

{

elseif (x==3)

{

printf("Multiplying");

c=a\*b;

printf ("%d", c);

{

else if (x==4)

{

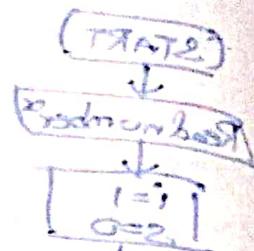
else printf("Division not possible");

(i.e.) a%b=2

i.e. 5

2+3=5

50%2



printf ("c=a+b;  
printf ("c=%d";  
else if (c  
{  
printf ("c=a%b;  
printf ("c=%d";  
else  
printf ("c=a/b;  
printf ("c=%d";  
else  
printf ("c=a%b;  
printf ("c=%d");

→ Write prime  
start  
step1  
step2  
step3

ste

flow

```

printf("Dividing");
c=a/b;
printf("%d", c);
else if (x==5)
{
 printf("Remainder");
 c=a%b;
 printf("%d", c);
}
else
 printf("re-enter choice");
}

```

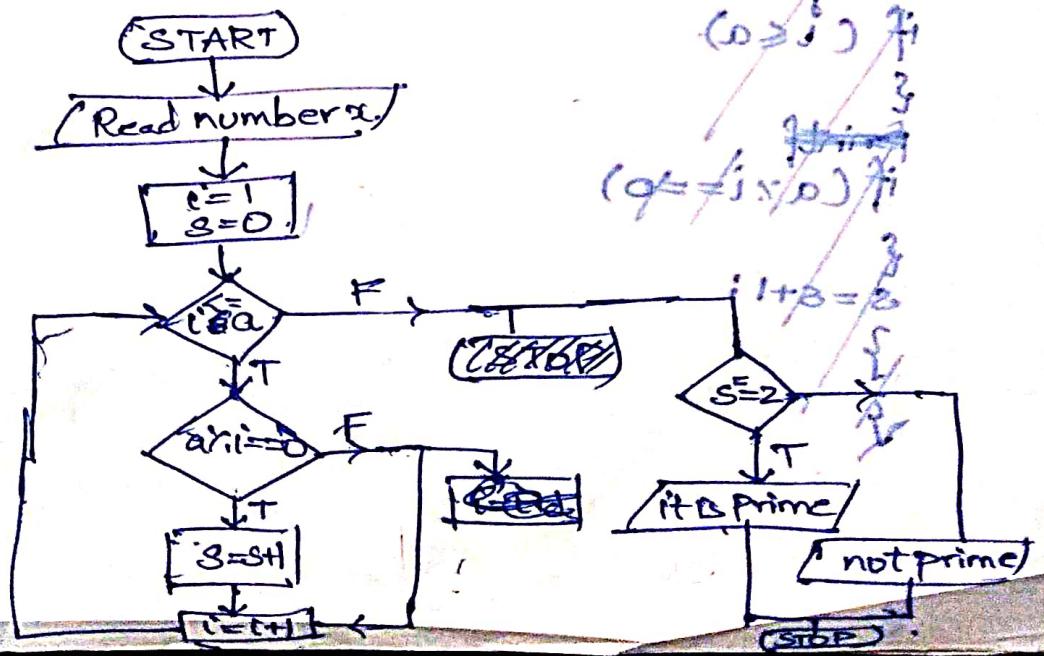
(if true)  
 (odd, b=1) false  
 (+i, s+1, i) not  
 (o==i, o) fi

Write an algorithm to check the given number is prime or not.      Input:- number    Output:- message.

Start  
 Step 1:- Read the number 'a'  
 Step 2:- Initialise  $i=1$ ,  $s=0$   
 Step 3:- if  ~~$a \neq 1$~~   $\rightarrow$  Repeat the following steps until  $i \geq \sqrt{a}$   
 3.1:- if  $a \% i == 0$   
 then  $s = s + 1$ .  
 3.2:-  $i = i + 1$   
 Step 4:- if  $s = 2$ .  
 then printf("it is a prime number")  
 otherwise printf("it is not a prime number")

Stop.

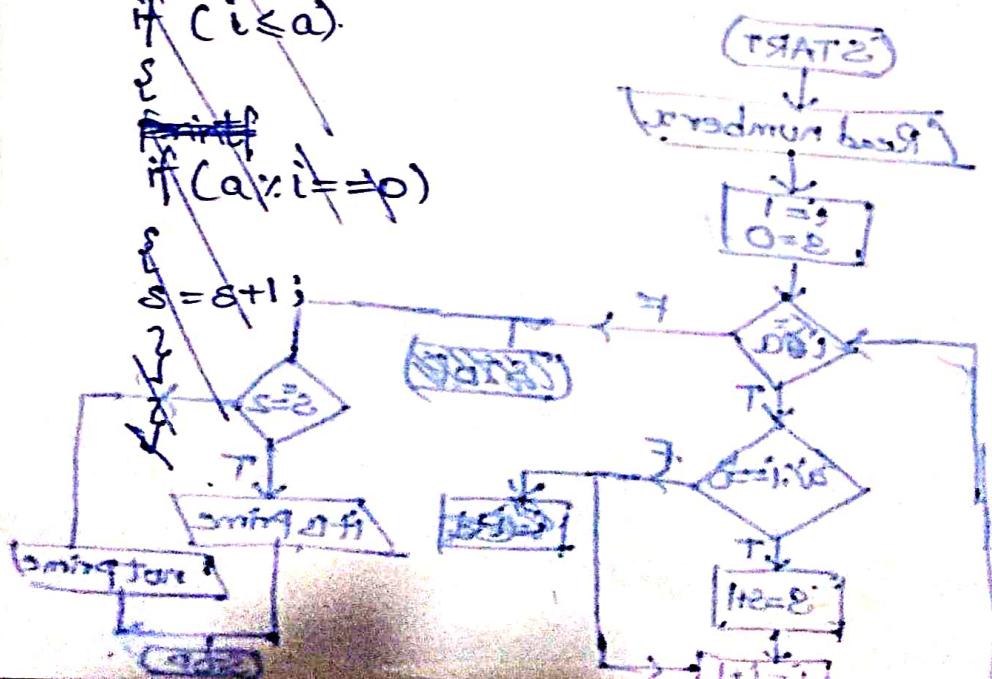
Flowchart.



Program :-

```
#include <stdio.h>
Void main()
{
 int i,s,a;
 printf ("Enter any number");
 scanf ("%d",&a);
 for (i=1,i<=a,i++)
 {
 if (a%i==0)
 s=s+i;
 }
 if (s==a)
 printf ("it is prime");
 else
 printf ("it is not prime");
}
```

~~# include <stdio.h>~~  
~~Void main()~~  
~~{~~  
 ~~int i,s,a;~~  
 ~~s=0;~~  
 ~~for (i=1;i<=a;i++)~~  
 ~~{~~  
 ~~if (a%i==0)~~  
 ~~s=s+i;~~  
 ~~}~~  
 ~~if (s==a)~~  
 ~~printf ("it is prime");~~  
 ~~else~~  
 ~~printf ("it is not prime");~~  
~~}~~



→ Write  
 ↓ to  
 Start  
 Step 1  
 Step 2  
 ↓  
 Step 3  
 ↓  
 Else

Pro  
 #  
 {  
 }  
 -  
 -

con

Write a program to print even numbers from 1 to 100.  
if given 1 to n then read 'n'

Start

Step 1:- Initialise  $i=1$

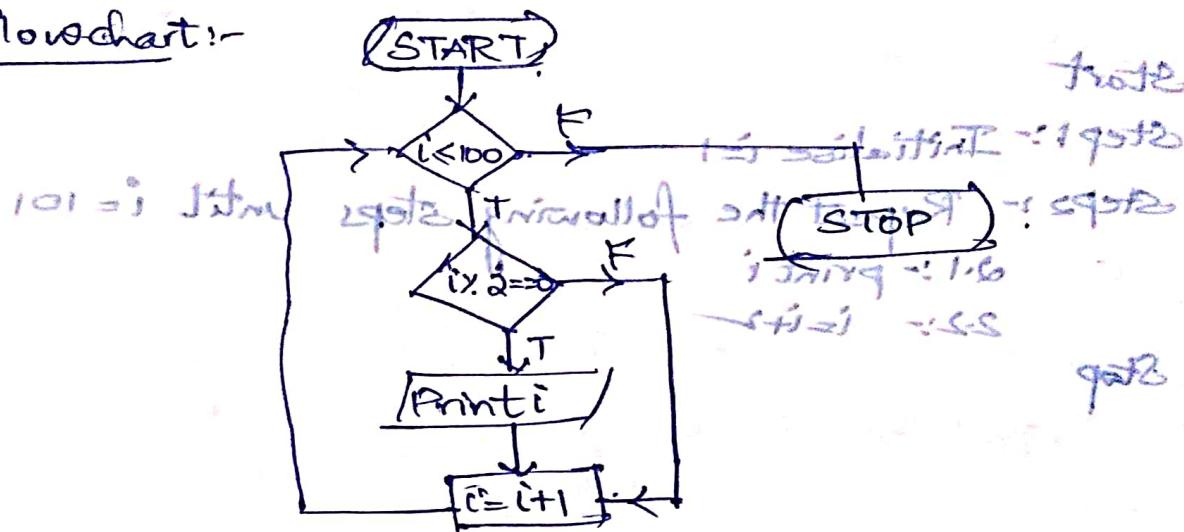
Step 2:- Repeat the following steps until  $i=101$ .

2.1:- if  $i \times 2 == 0$ , then print  $i$

2.2:-  $i = i + 1$ .

Step.

Flowchart:-



Program:-

```
#include <stdio.h>
void main()
{
 int i;
 for(i=1; i<100; i++)
 if(i % 2 == 0)
 printf("%d", i);
}
```

Ques Start

① initialise.  $i=2$ .

② Repeat the following until  $i=102$ .

2.1 print  $i$

2.2  $i = i + 2$

Stop

import stdio  
for odd numbers  
n base not, not, n+if fi

Start

Step 1:- Initialise i=1

Step 2:- Repeat the following steps until  $i=10$

2.1:- if ( $i \% 2 \neq 0$ )

2.2:- then print i

2.3:-  $i = i + 1$

Stop

(or). Start

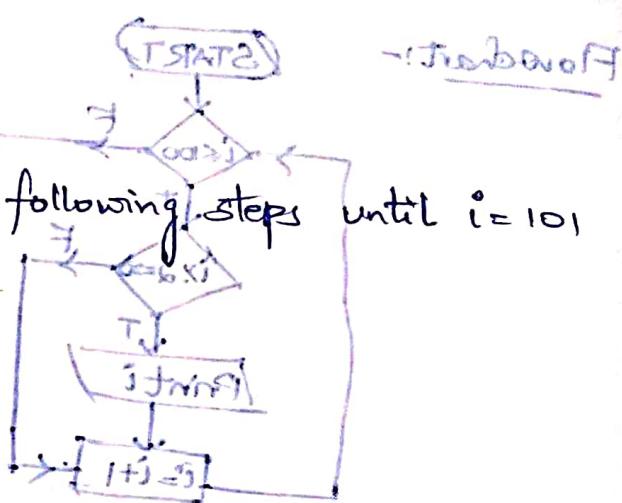
Step 1:- Initialise i=1

Step 2:- (Repeat the following steps until  $i=10$ )

2.1:- print i

2.2:-  $i = i + 2$

Stop



→ Write an

Start

Step 1:- Re

Step 2:- In

Step 3:- R

D = 3, a

3

Stop.

Flow chart

Program:

```
#include <iostream.h>
void main()
{
 int i;
 scanf("%d", &i);
 for (i = 1; i <= 10; i++)
 {
 if (i % 2 != 0)
 cout << i << endl;
 }
}
```

import  
std::cout  
std::endl  
(i % 2)

$i++$  ( $i \% 2 \neq 0$ );  $i = i + 2$ )  
 $i = s * j$  if

$i = "bx"$  if

$s = i$  . selection (1) or

interval of odd loops (2)

$i$  printing, ie  
 $s + i = j$  so

→ Write an algorithm to print 'n' multiplication table.

Start

Step 1: Read a number 'n'

Step 2: Initialise i=1.

Step 3: Repeat the following steps until  $i=20$

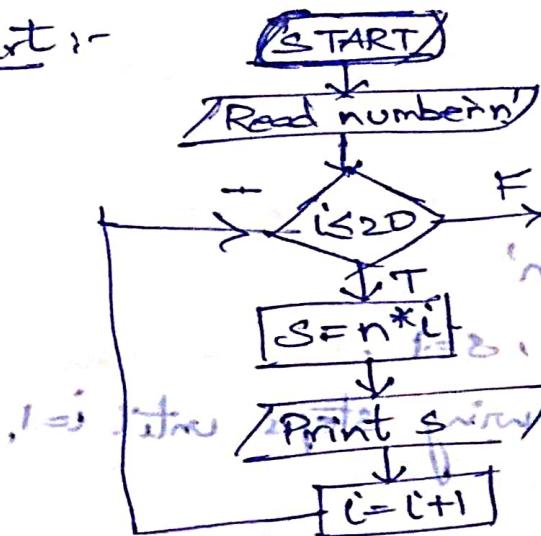
$S = n * i$  ~~Print S~~  $i = i + 1$

Print S

$i = i + 1$

Stop.

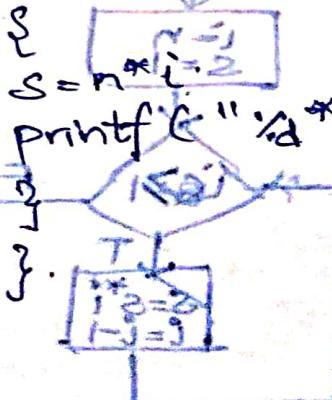
Flowchart:-



Program:-

```
#include <stdio.h>
void main()
{
 int i;
 scanf("%d", &n);
 for(i=1; i<=20, i++)
 {
 s = n * i;
 printf("%d * %d = %d", n, i, s);
 }
}
```

```
#include <stdio.h>
void main()
{
 int i;
 scanf("%d", &n);
 for(i=1, i<=20, i++)
 {
 s = n * i;
 printf("%d * %d = %d", n, i, s);
 }
}
```



→ Write an algorithm to find factorial of a given number.

Start.

Step 1 :- Read a number  $n$ .

Step 2 :- Initialise  $i = n$ ,  $s = 1$ .

Step 3 :- Repeat the following steps until  $i = 1$ .

3.1 :-

$$s = s * i$$

( $i = i - 1$ )

3.2 :-

$i = i - 1$

Step 4 :- Print 's'

Stop.

Start

Step 1 :- Read a number 'n'

Step 2 :- Initialise  $i = n$ ,  $s = 1$ .

Step 3 :- Repeat the following steps until  $i = 1$ .

3.1 :-  $s = s * i$ .

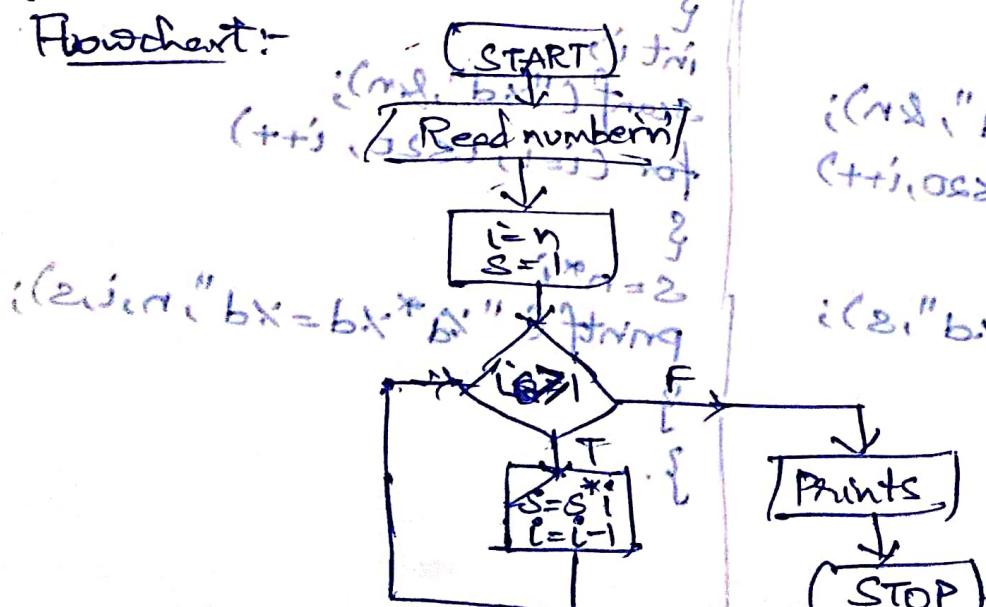
( $i = i - 1$ )

3.2 :-  $i = i - 1$ .

Step 4 :- Prints

Stop.

Flowchart :-



if (readnum > 1) {  
 fact = 1;

for (int i = 2; i <= readnum; i++) {  
 fact = fact \* i;  
 }

cout << "Factorial of " << readnum << " is " << fact << endl;

}

Program :-

```
#include <iostream.h>
void main()
{
 int i, s;
 scanf("%d", &n);
 for (i = n; i > 1; i--)
 {
 s = s * i;
 }
 printf("%d", s);
}
```

code for

Start

Step 1 :- R

Step 2 :- S

Step 3 :-

End

After 2nd

noising :

```

#include <stdio.h>
void main()
{
 int i, s;
 scanf("%d", &n);
 for(i=n, i>=1, i--) {
 s = s * i;
 }
 printf("%d", s);
}

```

~~<limits.h> stdio.h~~  
 (In main know  
 $i=1, i=2, i=3$   
 $(++i, 2 \geq i, i \geq 1)$  ref  
 $x^i = (j^k) = k$   
 $i^k (i \times i \times i) + 2 = 3$   
 $(2, "bx")$  fitting  
}

— code for  $1 + x^1 + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!}$ : ~~2: 2nd part~~

Start

Step 1:- Read the number no. given by user ref :- tri (1)

Step 2:- Initialise ~~i=1, s=0, k=1~~ ~~but~~ ~~int~~ ~~k=1~~ or int

Step 3:- Repeat the following steps until  $i=6$

~~variables~~ ~~s, i, power(x, i)~~ variables ref :- vars (5)

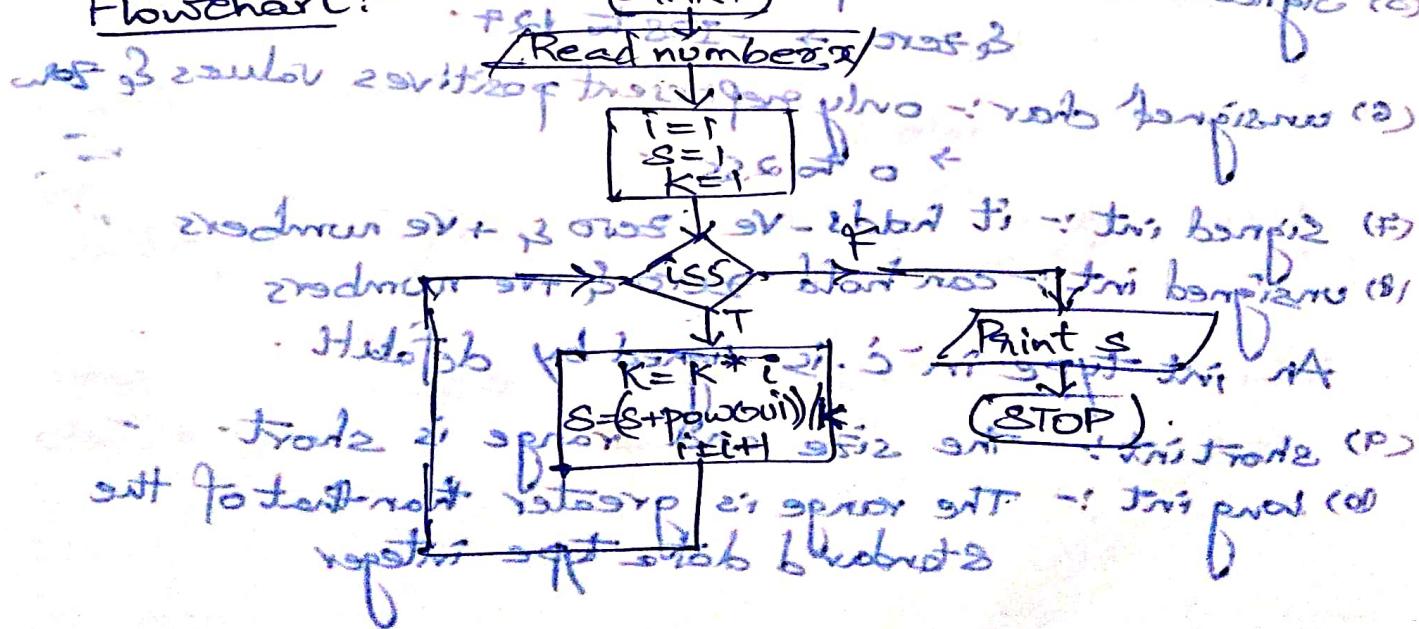
After execution  $k = K * i$  becomes value of  $\frac{1}{i!}$  ref (3)  
 initialising variable  $s = (s + power(x, i)) / k$  print ref

After execution  $i = i + 1$  becomes value of  $i+1$  ref (4)

Step 4:- Print the result  $s$  ref tri print

Step 5:- ~~2: 3rd part~~ ~~if~~ ~~for~~ ~~no~~ ~~works~~ ~~break~~ (2)

Flowchart:-



## Program :-

```
#include<stdio.h>
Void main()
{
 int p, q = 1, k = 1;
 for (i = 0, i <= 5, i++)
 {
 k = k * i;
 s = (s + (pow(x, i))) / k;
 }
 printf("%d", s);
}
```

(1) ~~long double~~

<limits> sub with  
( ) memo for

3. 8.3 tri

{(n8, "bx") free

s(--i, i <= i, i = i) not

; \* 3 = 3

{(2, "bx") fitting

{

Data types in C :- S:  $\frac{2^8}{12} + \frac{1^8}{11} + \frac{5^8}{12} + \frac{7^8}{18} + 3^8 + 1$  not obes

(1) int :- for declaring an integer variable

Zero, positive and negative values

no decimal values

(2) char :- for declaring character type variables

(3) float :- to store decimal numbers (numbers with floating point value) with single precision.

(4) double :- to store decimal numbers (numbers with floating point value) with double precision.

(5) signed char :- can represent both +ve & -ve values

& zero  $\rightarrow -128$  to  $127$  ~~minimum to maximum~~

(6) unsigned char :- only represents positive values & zero



(7) signed int :- it holds -ve, zero & +ve numbers

(8) unsigned int :- can hold zero & +ve numbers

An int type in C is signed by default.

(9) short int :- the size ~~large~~ range is short.

(10) long int :- The range is greater than that of the standard data type integer

(11) long dou

zbro

Data types

to use

These in

characters

Keyword

char

unsigned

char

signed

char

int

unsigned

int

signed

int

short int

long int

unsigned

long

unsigned

short

float

etc

doubt

long

doubt

(ii) long double :- it is more precise than double  
 Datatypes are used to define a variable before to use in a program, + ~~and~~ ~~also~~ ~~it~~ ~~is~~ ~~not~~ ~~used~~ ~~in~~ ~~any~~ ~~other~~ ~~language~~  
 These include integers, floating point numbers, characters = strings etc.

| Keyword        | bytes   | range                                         | control string |
|----------------|---------|-----------------------------------------------|----------------|
| char           | 1byte   | -128 to 127 or 255                            | % C            |
| unsigned char  | 1byte   | 0 to 255                                      | % C (or) % H H |
| signed char    | 1byte   | -128 to 127                                   | % C (or) % H H |
| int            | 2bytes  | -32,768 to 32,767                             | % d            |
| unsigned int   | 2bytes  | 0 to 65,535                                   | % u            |
| signed int     | 2bytes  | -32,767 to +32,767                            | % i (or) % d   |
| short int      | 2bytes  | -32,768 to 32,767                             | % d            |
| long int       | 4bytes  | -2,147,483,648 to 2,147,483,647               | ld             |
| unsigned long  | 4bytes  | 0 to 4,294,964,295                            | % lu           |
| unsigned short |         | 0 to 65,535                                   | % hu           |
| float          | 4bytes  | 1.2E-38 to 3.4E+38<br>(6 decimal places)      | n.f            |
| double         | 8bytes  | 2.3E-308 to 1.7E+308<br>(15 decimal places)   | % lf           |
| long double    | 16bytes | 3.4E-4932 to 1.1E+4932<br>(19 decimal places) | % LF           |

## Operators

→ Performs operations on variables/operands

- (i) Arithmetic operators +, -, \*, /, %
- (ii) Relational operators (comparison operators) <, >, ==, !=, <=, >=

(iii) Logical operators:-

- (i) logical 'and' &&
- (ii) logical 'or' ||
- (iii) logical 'not' !

(we can join conditions)

| True condition → 1 | False condition → 0 |
|--------------------|---------------------|
| a & b              | a    b              |
| 0 & 0              | 0    0              |
| 1 & 0              | 1    0              |
| 0 & 1              | 0    1              |
| 1 & 1              | 1    1              |

Assignment operator :- Short hand assignment operator

$$i = i + 2 \quad \text{operator} = \downarrow$$

↓  
operator of assignment  
of var, exp, func -> operator =

Same variable should be on 2 sides

$$i = i^2 \quad \text{operator} = \downarrow$$

↓  
operator of assignment  
of var, exp, func -> operator =

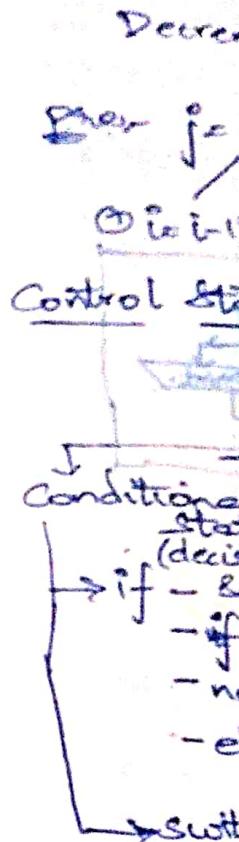
Increment & Decrement Operators:-

Increment < Pre ++ i or 0

Post i++ or 0  
Ex:- int i=2; i++ ; print(i);

i = ++j  
print(j);  
j = ++i — 0 (i = 4)

Post:- j = i++ or 0 (i = 1)  
j = i++



Conditions  
if statement

(i) Simple

if n  
is

(2) if



Decrement

pre      post

for  $j = -i$        $j = i - 1$

  |      |  
  |      |  
  ①  $i = i - 1$     ②  $j = j - 1$

post :-  $j = i - 1$

  |  
  ①  $j = i - 1$   
  ②  $i = i - 1$

Control statements :- to control execution of program

("Volume End") flowing

Control statements

Conditional (Selection) statements (nesting)  
(decision making)

if - Simple if  
- if else  
- nested if  
- else if ladder

switch.

Iterative/loop, Repetitive statements (nesting)  
for  
while  
Dowhile

Jump statements  
break  
Continue  
goto

Conditional / Selection statements (nesting)

if statement :-

(1) Simple if :- Syntax :-

```

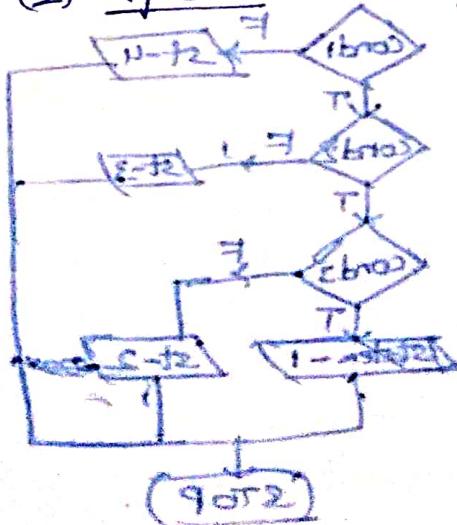
if (condition)
{
 statements 1;
 statement 2;
}

```

(2) Block statements  
(no. of statements placed within braces)

if we don't use control statements, the execution is sequential

(2) if else :- Syntax :-



```

if (condition)
{
 statement 1;
 statement 2;
}
else
{
 statement 3;
 statement 4;
}

```

Ex:-  $a=2, b=3$

if( $a < b$ )

{

printf("a is smaller");

}

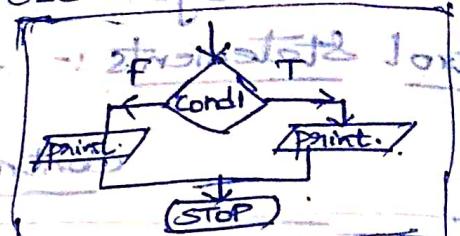
else

incorporate to next two lines of code

printf("b is smaller");

}.

if  $a < b$   
True  
returns '1'  
else returns '0'  
 $i = j$



### (ii) Nested if:-

multiple conditions  
surround one if within another if block

if (condition 1)

{

· if (condition 2)

{

· if (condition 3)

{

statement 1;

else

{

statements 2; if statements 3

{

if statements 3;

{

Statement 3; (condition 3 or condition based on condition 3)

{

second

if

else

{

third

if

else

{

fourth

if

else

{

fifth

if

else

{

sixth

if

else

{

Statement 4;

.

(if block 1)

(if block 2)

{

else

{

if block 3

{

else

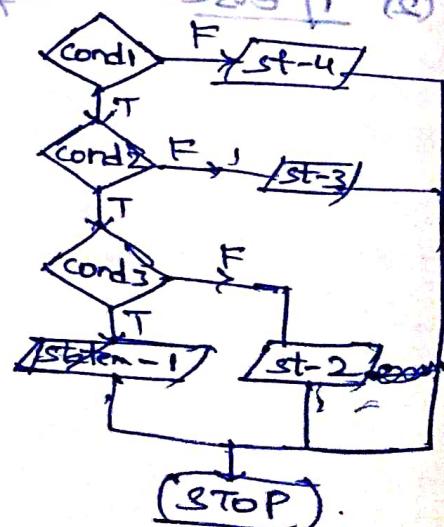
{

if block 4

{

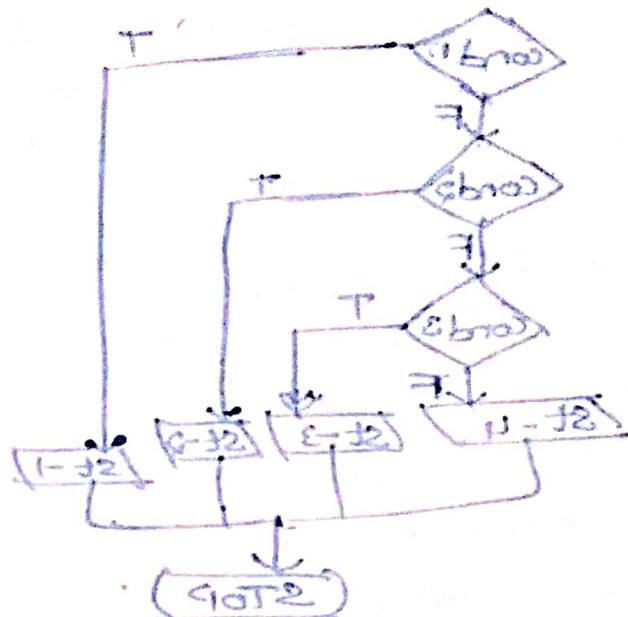
else

{



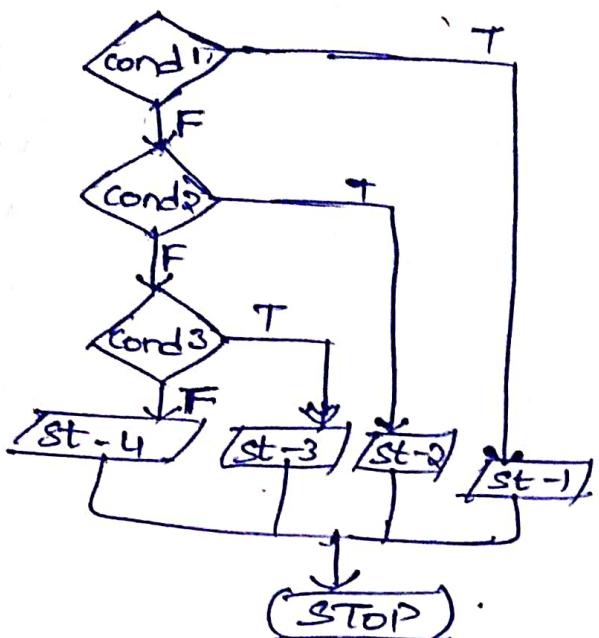
Ex:- Comparing three numbers a, b, c

```
if(a>b) {if statements}
{
if(b>c) {if statements}
{
printf("a is bigger");
}
{
if(c>a) {if statements}
else
{
printf("c is bigger");
}
}
else
{
if(b>c)
{
printf("b is bigger");
}
else
{
printf("c is bigger");
}
}
```



(4) else if ladder id - syntax :- `if (condition1) {  
 statement1;  
}  
} else if (condition2) {  
 ("napped id") thing  
}  
} else if (condition3) {  
 ("napped id") thing  
}  
} else {  
 ("napped id") thing  
}  
} else if (condition4) {  
 ("napped id") thing  
}`

Flowchart :-



• based on following data.

|       |    |                               |
|-------|----|-------------------------------|
| > 90  | A  | (ss)arts) Native              |
| 80-90 | B. | {                             |
| 70-80 | C. | { (translates: English words) |
| 40-70 | D  | { (translates2                |
| < 40  | F. | { (bad)                       |

#

```
#include <stdio.h> { (bad) #include <stdio.h> .
void main() { "ss)arts go Below" } } void main(b)
{
 scanf("%d", &a);
 if (a > 90) { 'good' result, bottom = (a+b+c+d+e) / 5; }
 else if (a > 80 && a < 90) { bottom = (a+b+c+d+e) / 5; }
 else if (a > 70 && a < 80) { bottom = (a+b+c+d+e) / 5; }
 else if (a > 40 && a < 70) { bottom = (a+b+c+d+e) / 5; }
 else { ("ss)arts Below") } }
 printf("%d", E);
}
```

• else { for loop or do { (translates too) } for loop }

```
(ss)arts result
(ss)arts di "bx") from
(ss)arts) it's ok
{
```

```
{ d++ = 3 ; '+' seen
(ss)arts) from
```

Switch Statement menu driven program is a switch statement which has no default.

Syntax:- `switch (choice)`

```

 {
 case value1: statement1;
 statement2;
 break;
 case value2: statement3;
 statement4;
 break;
 default: printf("invalid choice");
 }

```

If one case is executed, then 'break' comes out of the switch if remaining cases not executed.

for character constants → in single code. '+'.

for integer → no need. (OP> & OP< ) if double codes.

for float → no need. (OP> & OP< ) if double codes.

For int choice;

```

scanf("%d", &choice);
switch(choice)
{
 case 1: c=a+b;
 printf("%d", c);
 break;
 case 2: c=a-b;
 printf("%d", c);
 break;
 default: printf("invalid choice");
}

```

default is last statement, so no need of break.

```

char choice;
scanf("%d", &choice);
switch(choice)
{
 case '+': c=a+b;
 printf("%d", c);
 break;
}

```

case '-' : c  
default :  
3

Loops :-

while :-

Ex:-

-----+-----

cos

sin

Prog

#in

int

s

int

se

fo

&

S

case - :  $c = a - b;$

~~printf("x.d", c);~~

~~break;~~

default: ~~printf("Invalid choice");~~

}

Loops :-

while  
do while  
for.

Initialisation

$i + j = j$

$i + j = i$

$i + j = i$

while :- syntax :- Initialisation

$\dots + \frac{x^2}{2!} - \frac{x^4}{4!} + \dots$  while (condition) 2 trinf  $\rightarrow$  p95t2

statement;

• Updation counter;

$j$

Ex :-  $\sum_{i=0}^{\infty} i! \cdot s^i = 1, 0=2, \phi=1, i=1..$  initial and 2 q5t2  
.while(  $i \leq j = 10$ )  
{  
printf("Hello");

$\dots + \frac{x^5}{5!} - \frac{x^6}{6!} + \dots$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Program :-

#include <stdio.h>

int main()

{  
int x, n, k=1, i, l=1;  
scanf("%d", &x);

for (i=0, i < n, i=i+2)

{

s = (s + pow(x, i)) / k

<1.0t2> abcdw#  
( )new t1l

{s=t, i=1, o=2, i, l=1} trinf

i((i+j), n > i, o=i) rot

; } ((i+x)(o+i)+2)=2

i(+\*j + 2) = 1

{s+i = 1}

{s+i = 1}

; (2, "b") trinf

# include  
int main

{

float k

float l

{

s = (

(subtraction)

K = (

(division)

t = (

g

PV

2

①

②

→

→

→

→

→

Start

Step 1:- Read  $x, n$ .

Step 2:- Initialise  $k=1, i=0, s=0, l=t=2$ .

Step 3:- Repeat the following until  $i \leq n$  do {

$$s = (s + (\text{pow}(x, i))) / k$$

$$k = k * l * t$$

$$l = l + 2$$

$$t = t + 2$$

$$i = i + 2$$

Step 4:- Print  $s$ .

Stop.

Start

Step 1:- Read  $x, n$ .

Step 2:- Initialise  $k=1, i=0, s=0, l=t=2$ .

Step 3:- Repeat the following until  $i \leq n$  do {

$$s = (s + (\text{pow}(x, i))) / k$$

$$k = k * l * t$$

$$l = l + 2$$

$$t = t + 2$$

$$i = i + 2$$

Step 4:- Print  $s$ .

Stop.

Program

#include <stdio.h>

int main()

{

float k=1, i, s=0, l=1, t=2;

for ( i=0, i<n, i+=2);

{

$s = (s + \text{pow}(x, i)) / k;$

$k = k * (l * t);$

$l = l + 2;$

$t = t + 2;$

}

}

printf(" %d ", s);

}

$(d-d=0) : \text{ans}$

$((0, "bx")) \text{ printing}$

$(1, "bx")$

$t=2$

start

start do

-: 29021.

not

initialization -: exit -: start

(calibration)  $\frac{x^1}{1!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

(transistor)  $\frac{x^1}{1!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

(iteration)  $\frac{x^1}{1!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$

#include <iostream>  
 int main() {  
 float K = 1, P, S = 0; // K=1, P=0; S=0  
 for (int i = 0; i < n; i++) { // i=0, i=1, i=2, ...  
 S = (S + pow(x[i])) / K; // S = (S + x[0]^1) / 1  
 K = -K \* (1 + t); // K = -1 \* (1 + 1) = -2  
 t = t + 2; // t = 1 + 2 = 3  
 cout << S; // Union b101  
 }
 }

~~1.  $S = (0 + 1^0)/1 = 1$~~   
 ~~$1 + (1^2)/(-2) = \frac{1}{2} + \frac{1}{2} = 0.5$~~   
 ~~$\frac{1}{2} + (1^4)/(-24) = \frac{1}{2} + \frac{1}{24} = 0.54167$~~   
 ~~$(1^6)/(-24 \times 5 \times 6) = \frac{-1}{720} + 0.54167 = 0.54167$~~   
 ~~$0.54167 \times 10^{-3} + 0.54167 = 0.54167$~~   
 ~~$= 0.001389 + 0.54167 = 0.543$~~   
 ~~$(\text{"Jugni bilovni"}) \text{firing}$~~

~~2.  $1 - \frac{1}{6} + \frac{1}{120} - \frac{1}{5040} = \frac{5040 + 42 - 840 - 1}{5040} = \frac{4201}{5040} = 0.8414.$~~

~~$\rightarrow 0 + (1^1)/1 = 1$~~   
 ~~$\rightarrow 1 - (1^3)/6 = 1 - \frac{1}{6} = 0.8334$~~   
 ~~$\rightarrow 0.8334 + (1^5)/6 \times 4 \times 5 = 0.8334 + \frac{1.1667}{120} = 0.8334 + 0.011667 = 0.8414$~~   
 ~~$\rightarrow 0.8414 + (1^7)/6 \times 4 \times 5 \times 6 \times 7 = 0.8414 + 0.0001984 = 0.8414$~~   
 ~~$(0.8414, "Jugni bilovni") \text{firing}$~~

$$\Rightarrow 0.8334 + \left(\frac{1}{120}\right) = 0.008334 + 0.8334 \text{ (from } t_n)$$

$$\Rightarrow 0.8417 - \left(\frac{1}{5040}\right) = 0.8417 - 0.001984 \text{ (from } t_n)$$

$$= 0.8415$$

```
printf("%d",
} else
printf("%d",
}
```

Write a program to extract last two digits (individual and display them (input must be minimum three digits)).

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

```
Void main()
```

```
{
```

```
int a;
```

```
scanf("%d", &a);
```

```
if (a < 1000 && a > 99)
```

```
{
```

```
printf("%d", a % 100);
```

```
printf("%d", a % 10);
```

```
C = a % 100;
```

```
F1 = f1112.0 + 01xP8E.1e30 =
```

```
printf("%d", d);
```

```
printf("%d", d % 10);
```

```
d = c / 10.
```

```
printf("%d", d % 10);
```

```
printf("%d", d % 10);
```

```
F1 = f1112.0 + 01xP8E.1e30 =
```

```
else F1 = f1112.0 + P8E100.0 =
```

```
{
```

```
printf("invalid input");
```

```
}
```

$$\frac{1}{N/100.0} = \frac{1-0.8334-0.001984}{200.0} = \frac{1}{2} - \frac{1}{100} + \frac{1}{200} - 1 \quad . \quad (5)$$

```
{
```

```
int a,c,d;
```

```
scanf("%d", &a);
```

```
if (a < 1000 && a > 99)
```

```
F1 = f1112.0 + P8E100.0 =
```

```
C = a % 100;
```

```
D = C / 10
```

```
printf("%d\n", d % 10);
```

$$1 = f1112.0 + 0. \quad \leftarrow$$

$$f1112.0 = \frac{1}{2} = f1112.0 - 1 \quad \leftarrow$$

$$f1112.0 = \frac{1}{100} = \frac{1}{100} + f1112.0 \quad \leftarrow$$

$$f1112.0 = \frac{1}{200} = \frac{1}{200} + f1112.0 \quad \leftarrow$$

$$f1112.0 = \frac{1}{2} - 1 = f1112.0 + 12f1.1 \quad \leftarrow$$

```
printf("invalid input");
```

else  
 printf("invalid input");  
}

```
#include <stdio.h>
void main()
{
 int a,c,d;(0)
 scanf ("%d",&a);
 while(a>99)
 {
 d=a%10;
 a=a/10;
 printf ("%d",c);
 }
}
```

```
#include <stdio.h>
void main()
{
 int n,s,c=0;(0)
 scanf ("%d",&n);(0)*2)=2
 while(n>0)
 {
 s=n;((2,"bx"))
 n=s;
```

smashbang; si fi weiter geht an morpheme struktur

for no  
<chibz> abstrakt

( ) neom bior

(0=2,r,-strix)

((0), "bx") free2

(0<n>) winter

~~0=0~~=0

(0<n>) winter

(01K&)=r

(01B&)=n

((r+(01\*2)=2

{} bring

(0=2) fi

; ("smashbang si fi") bring

Write a program to reverse the given

```
#include <stdio.h>
void main()
{
 int a, r, s = 0;
 scanf ("%d", &a);
 while (a > 0) {
 r = a % 10;
 a = a / 10;
 s = (s * 10) + r;
 }
 printf ("%d", s);
}
```

( " 12345 " )  
12345  
—  
5

3. Lecture 8

3.

→ to reverse  
for memo

Write a

using

```
#include <stdio.h>
void main()
int a;
scanf ("%d", &a);
printf ("%d", a);
```

( " 12345 " )  
12345  
—  
5  
10) 3 (0  
0  
(0, " 5 " )  
(PP<0) Split  
.  
(0<0) Split

→ Write a program to check whether it is palindrome or not

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

```
{
 int a, r, s = 0;
 scanf ("%d", &a);
 while (a > 0)
 {
 r = a % 10;
 a = a / 10;
 s = (s * 10) + r;
 }
 if (s == a)
 {
 printf (" it is palindrome ");
 }
 else
 {
 printf (" not a palindrome ");
 }
}
```

( " 123 " )  
123  
—  
3  
10) 2 (0  
0  
(0, " 1 " )  
(PP<0) Split  
.  
(0<0) Split

switch  
{  
case

case  
case  
case

if  
l  
c

3. Write a program to reverse words of a string.

3.

→ To reverse letters of a word.  
for memory operations - O(100)

<1.0111> abulnitt  
( ) Niam Bro

Write a C program to perform arithmetic operations using switch statement.

#include <stdio.h>

void main()

int a,b,n;  
scanf("%d %d", &a, &b);

printf("1. Addition\n2. Subtraction\n3. Multiplication\n4. Division\n5. Remainder");

scanf("%d", &n);

switch (choice)

{

case 1: printf("The sum is %d", a+b);

break;

case 2: printf("The difference is %d", a-b);

break;

case 3: printf("The product is %d", a\*b);

-> distributive law

case 4: printf("the ratio is %d", a/b);

(a) ratio

case 5: printf("the remainder is %d", a%b);

o secon

default: printf("invalid choice");

1 secon

}

fflush(stdin); → for two inputs, clear : flush

1 or 2 also no flush enter int, void.

choice ← getche();

↓ (integer (int2char))

predefined string handling function

return

0

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

→ C program to check even or odd using switch

```
#include <stdio.h>
Void main()
{
 int n;
 scanf("%d", &n);
 if (n % 2 == 0)
 printf("it is even");
 else
 printf("it is odd");
}
```

→ Write a C program to check even or odd

Nested switch:-

```
scanf("%d", &n);
switch (n)
{
 case 0: printf("zero"); break;
 case 1: printf("odd"); break;
 default: if (n % 2 == 0)
 printf("even");
 else
 printf("odd");
}
```

→ Write a C program to check even or odd

A-Z  
=  
→ Write a C program to check even or odd

```
#include <stdio.h>
Void main()
{
 char ch;
 scanf("%c", &ch);
 if (ch >='A' & ch <='Z')
 printf("%c is Capital letter", ch);
 else if (ch >='a' & ch <='z')
 printf("%c is Small letter", ch);
 else
 printf("%c is not an alphabet", ch);
}
```

→ Write a C program to check even or odd

$$A - Z = 65 \text{ to } 90 \quad a - z = 97 \text{ to } 122 \quad (\text{Hanging on zeros})$$

= Write a program to check the given character is vowel or consonant or digit or special character.

```
#include<stdio.h>
void main()
{
 char x;
 scanf("%c", &x);
 if ((x >='a' && x <='z') || (x >='A' && x <='Z'))
 {
 if (x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u' || x == 'A' || x == 'E' || x == 'I' || x == 'O' || x == 'U')
 printf("%c is a vowel", x);
 else
 printf("%c is a consonant", x);
 }
 else if (x >='0' && x <='9')
 printf("%c is a digit", x);
 else
 printf("%c is a special character", x);
}
```

= Write a program using `switch`.

```
#include<stdio.h>
void main()
{
 int a,b,c,d,e,n;
 scanf("%d%d%d%d%d", &a, &b, &c, &d, &e);
 n = (a+b+c+d+e)/5;
 K = n * 10;
 switch(K)
 {
 case 0:
 case 1:
 case 2:
 case 3:
 case 4:
 case 5:
 case 6:
 case 7:
 case 8:
 case 9:
 break;
 case 10:
 case 11:
 case 12:
 case 13:
 case 14:
 case 15:
 case 16:
 case 17:
 case 18:
 case 19:
 break;
 case 20:
 case 21:
 case 22:
 case 23:
 case 24:
 case 25:
 case 26:
 case 27:
 case 28:
 case 29:
 break;
 case 30:
 case 31:
 case 32:
 case 33:
 case 34:
 case 35:
 case 36:
 case 37:
 case 38:
 case 39:
 break;
 case 40:
 case 41:
 case 42:
 case 43:
 case 44:
 case 45:
 case 46:
 case 47:
 case 48:
 case 49:
 break;
 case 50:
 case 51:
 case 52:
 case 53:
 case 54:
 case 55:
 case 56:
 case 57:
 case 58:
 case 59:
 break;
 case 60:
 case 61:
 case 62:
 case 63:
 case 64:
 case 65:
 case 66:
 case 67:
 case 68:
 case 69:
 break;
 case 70:
 case 71:
 case 72:
 case 73:
 case 74:
 case 75:
 case 76:
 case 77:
 case 78:
 case 79:
 break;
 case 80:
 case 81:
 case 82:
 case 83:
 case 84:
 case 85:
 case 86:
 case 87:
 case 88:
 case 89:
 break;
 case 90:
 case 91:
 case 92:
 case 93:
 case 94:
 case 95:
 case 96:
 case 97:
 case 98:
 case 99:
 break;
 default:
 break;
 }
}
```

case 10 : printf("A"); break; // OPT 2a ==> A

case 9 : printf("B"); break; // starts with comparing to value  
break is better than goto and therefore  
admits shadow (more bugs)

case 8 : printf("C"); break;

case 7 : printf("D"); break;

case 6 : printf("E"); break;

case 5 : printf("F"); break;

case 4 : printf("G"); break;

default : printf("H"); break;

(or)

case 10 : break;

case 9 : printf("A"); break;

case 8 : printf("B"); break;

Example : Iterative Statements : (Loops / Repetitive) - initialisation  
- while - for - do while

while : initialisation - variable is used - moving - static  
P - while (condition) called loop variable  
{ statements;  
    Updation counter;  
}

Print Hello for 100 times. i.e. "bvbvbvbvbvb" five times.

#include <stdio.h>

Void main()

{ int i;

i = 1; while (i <= 5) { printf("%c", i); i++; } }

→ Write a

#include

void m

{

int i;

i = 1;

while

{

if (

{

printf

{

i = i +

{

3.

Flowchart

To print

#include

void m

{

int i;

while

{

printf

{

```

i=1;
while(i<=100)
{
 printf ("Hello");
 i++;
}

```

```

while (i<100)
{
 printf ("Hello");
 i=i+1;
}

```

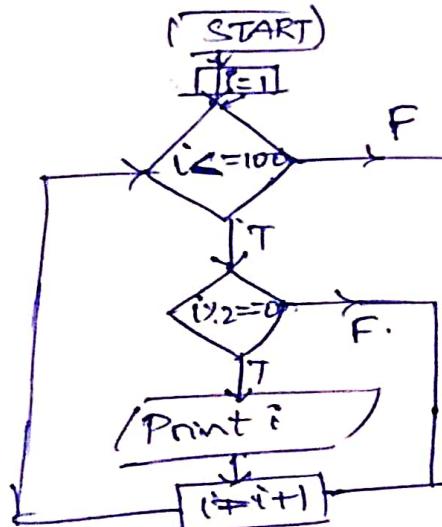
→ Write a program to print even numbers using while.

```

#include <stdio.h>
void main()
{
 int i;
 i=1;
 while(i<=100) (Or)
 {
 if (i%2==0)
 {
 printf("%d", i);
 }
 i=i+1;
 }
}

```

Flowchart :-



To print 1-10 numbers (while).

```

#include <stdio.h>
void main()
{
 int i;
 while (i<=10)
 {
 printf ("%d");
 i=i+1;
 }
}

```

To print series  $x + x^2 + x^3 + \dots + x^{10}$ . (while)

```
#include <stdio.h>
void main()
{
 int i=0, s=0, x;
 scanf ("%d", &x);
 while (i <= 10)
 {
 s = s + pow(x, i);
 i = i + 1;
 }
 printf ("%d", s);
}
```

$x + x^2 + x^3 + \dots + x^{10}$

```
#include <stdio.h>
void main()
{
 int i=0, s=0, x;
 scanf ("%d", &x);
 while (i <= 10)
 {
 s = s + pow(x, i);
 i = i + 2;
 }
 printf ("%d", s);
}
```

$x^2 + x^4 + x^6 + \dots + x^{10}$ .

```
#include <stdio.h>
void main()
{
 int i=0, s=0, x;
 scanf ("%d", &x);
 while (i <= 10)
 {
 s = s + pow(x, i);
 i = i + 2;
 }
 printf ("%d", s);
}
```

→ To print even terms of series

Flowchart:

```

graph TD
 Start([START]) --> Input[/Input/]
 Input --> Init[i=0]
 Init --> Decision{Decision}
 Decision -- T --> Sum[s = s + x]
 Sum --> Decision
 Decision -- F --> Output[/Output/]
 Output --> End([END])

```

→ To print factors of a given number (while).

```
#include <stdio.h>
void main()
{
 int r=1, a;
 scanf("%d", &a);
 while (r <= a) {
 if (a % r == 0)
 printf("%d", r);
 r++;
 }
}
```

(ans "bx") press  
(a=>i) slider  
open =  
(a,i,n) bx = bx \* bx^n ) thing

→ To check the given number is prime or not (while)

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

```
{
 int i=1, s=0, a;
 scanf("%d", &a);
 while (i <= a) {
 if (a % i == 0)
 s=s+1;
 i++;
 }
 if (s==2)
 printf("prime");
 else
 printf("not prime");
}
```

Slider  $\frac{x}{n} + \dots + \frac{x}{n} + \frac{1}{n} = 1$

(a,b) slider  
(n,m) box

(n,i=1, i=2) if tri  
(n,i,"bx") press  
(n=>i) slider

$i^*k=n$   
 $i*(i+1)(i+2)/2$

→ To print n multiplication table (while) ("bx") thing

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

```

int i=1;
scanf("%d",&n);
while (i<=10)
{
 s = n * i;
 printf("%d * %d = %d", n, i, s);
}

```

→ To find factorial of a given number (while)

```

#include <stdio.h>
void main()
{
 int i,s;
 scanf("%d",&n);
 while (i>=1)
 {
 s = s * i;
 i = i - 1;
 }
 printf("%d",s);
}

```

→  $1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$  (while)

```

#include <stdio.h>
void main()
{
 int i=1,s=1,k=1,n;
 scanf("%d",&n);
 while (i<=n)
 {
 k = k * i;
 s = (s + pow(x,i)) / k;
 }
 printf("%d",s);
}

```

Do while  
initialisation  
do  
{  
 statement  
update  
}while (condition)

For loop  
for (initialisation;  
{  
 statement  
}  
flowchart

→ write  
ffind  
void  
{  
int r  
int  
scanf  
for  
{  
printf  
}

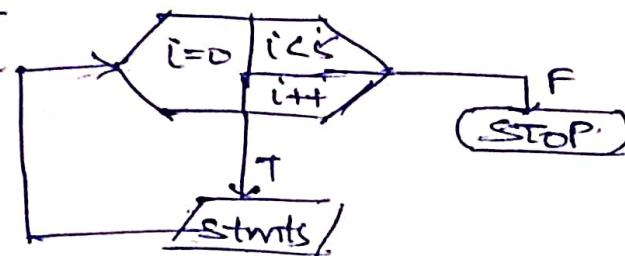
→ W  
#  
Ve

Do while  
initialisation  
do  
{  
statement  
update  
}while (condition);

For loop

for (initialisation; condition; Updation)  
{  
statements;  
}

flowchart:-

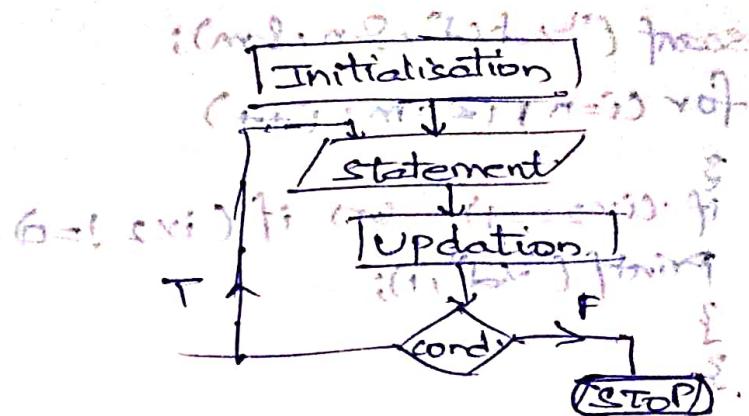


→ Write a program to print nth table.

```
#include <stdio.h>
void main()
{
int n;
int i;
scanf ("%d", &n);
for (i=1; i<=10; i++)
{
printf ("%d * %d = %d", n, i, n*i);
i++;
}
```

→ Write a program to print odd nos. b/w specified range, using for loop

```
#include <stdio.h>
Void main()
{
int i, n, m;
```



ex: while (start < end) {  
 /\* statements \*/  
}

(loop, start < end) true  
 (start) fi

{ start = 5  
 /\* statements \*/  
}

{ start = 10  
 /\* statements \*/  
}

{ start = 15  
 /\* statements \*/  
}

{ start = 20  
 /\* statements \*/  
}

{ start = 25  
 /\* statements \*/  
}

(start < 30) true  
 (start) fi

{ start = 30  
 /\* statements \*/  
}

{ start = 35  
 /\* statements \*/  
}

{ start = 40  
 /\* statements \*/  
}

{ start = 45  
 /\* statements \*/  
}

{ start = 50  
 /\* statements \*/  
}

{ start = 55  
 /\* statements \*/  
}

{ start = 60  
 /\* statements \*/  
}

{ start = 65  
 /\* statements \*/  
}

{ start = 70  
 /\* statements \*/  
}

{ start = 75  
 /\* statements \*/  
}

{ start = 80  
 /\* statements \*/  
}

{ start = 85  
 /\* statements \*/  
}

{ start = 90  
 /\* statements \*/  
}

{ start = 95  
 /\* statements \*/  
}

{ start = 100  
 /\* statements \*/  
}

```

scanf ("%d %d", &n, &m);
for (i=n; i<=m; i++)
{
 if (i%2==1) { // odd
 printf ("%d", i);
 }
}

```

→ #include <stdio.h>

```

Void main()
{
 int i, n, m;
 Scanf ("%d %d", &n, &m);
 if (n>m)
 {
 n=n+m;
 m=n-m;
 n=n-m;
 }
 for (i=n; i<=m; i++)
 {
 if (i%2==0)
 printf ("%d", i);
 }
}

```

→ #include <stdio.h>

```

Void main()
{
 int i, c=0;
 Scanf ("%d", &n);
 for (i=1; i<=n; i++)
 if (*scanf ("%c", &c) == 'b') // if b is found
 c++;
}

```

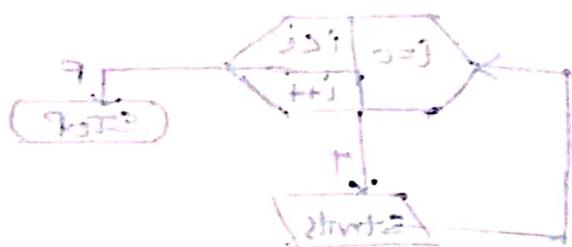
printf ("%d", i);

c++;

}

}

if (c==2)



(condition) skipping

loop

iteration

- iteration

<+> b <+> b  
(incon. b)

i (in. "bx") press  
(i+j; or => j+i; j) ref.

i (in. "bx") press

printf ("pri  
else  
printf ("  
3.  
3.

→ #include  
void main()

int i, j;

for (i=5;  
j=

for (k=

Printf (

for (j=

printf (

3  
printf (

3  
3.

→ #inclu

void

{

int a

Scanf

Scanf

while

S

= Q

Q = a

S = S

{

Print

if C

Print

else

Print

3.

```

printf("prime");
else
printf("not prime");
}.

```

↳ nested if condition

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

```

```

int i,j,k;
for(i=5;i>=1;i--)
{
 for(k=5;k>=1;k--)
 printf("%d",k);
 for(j=5;j>=1;j++)
 printf("%d",j);
 printf("\n");
}
printf("n");
}.

```

↳ Nested for loop → string  
 ↳ (i,j,k) iteration  
 ↳ (i=5 to 1) first row  
 ↳ (k=5 to 1) nested loop  
 ↳ (j=5 to 1) nested loop  
 ↳ (i=4 to 1) second row  
 ↳ (k=5 to 1) nested loop  
 ↳ (j=5 to 1) nested loop  
 ↳ (i=3 to 1) third row  
 ↳ (k=5 to 1) nested loop  
 ↳ (j=5 to 1) nested loop  
 ↳ (i=2 to 1) fourth row  
 ↳ (k=5 to 1) nested loop  
 ↳ (j=5 to 1) nested loop  
 ↳ (i=1 to 1) fifth row  
 ↳ (k=5 to 1) nested loop  
 ↳ (j=5 to 1) nested loop

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

```
void main()
```

```
{
```

```
int a,n,r,s=0;
scanf("%d", &a);
scanf("%d", &n);
while(a>0)
```

```
{
```

```
r=a%10;
```

```
a=a/10;
```

```
s=s+pow(r,3);
```

```
}
```

```
printf("%d", s);
```

```
if(s==a)
```

```
printf("The given integer is Armstrong");
```

```
else
```

```
printf("The given integer is not Armstrong");
```

```
.
```

↳ extracted if condition

↳ (condition) sider

→ Write a syntax for 'while':-

initialisation;  
while (condition)  
{  
statements;  
update;  
}

→ Write a program to print even numbers in the range 1-100 using for, while, do while?

```
#include<stdio.h>
Void main()
{
 int i=1;
 for(i=1, i<=100, i++) (or)
 {
 if (i%2==0)
 printf("%d", i);
 i=i+1;
 }
 i=1;
 while(i<=100)
 {
 if (i%2==0)
 printf("%d", i);
 i=i+1; (or) i+=1;
 }
 i=1;
 do
 {
 if (i%2==0)
 printf("%d", i);
 i=i+1;
 } while(i<=100);
}
```

(using for)  
("using for") thing  
<initiated variable  
(i), now i=1

(-> i++ or i+=1) for  
i ("i") thing  
int i=1; infinite loop  
for( ; i<=100;) (or) for(; ; ) foring  
{  
 i=i+1;  
}  
i ("i") thing  
{  
}

<initiated variable  
(i), now i=1

(i=2)(i=10) for  
(or, "bx") for2  
(and, "bx") for2  
(6<>) enter  
{  
}

(or, i=8  
(or) i=0  
((8-1) and i+2 = 2  
{  
}

((2, "bx") thing  
(or == 2) fi.  
(or == 2) fi.  
})

("for statement ei repetitii norip art") thing  
sel3

("for statement for ei repetitii norip art") thing  
sel3

Jump Statement  
Condition;  
Switch  
Iteration  
→ break :-  
for in  
break  
→ continue

for

for  
{  
if (i  
cont  
pri  
{  
}

break

→ goto  
if (n  
go to  
else  
goto  
Even  
odd

Jump Statements :- break, continue, goto.

Condition :- based on if/for/while.

Switch :- only one case executed.

Iteration :- executed many times.

→ break :- to stop the program in middle of loop.

exit in 'switch' and also for?

break & continue (only in loops).

→ continue it is to continue with next iteration. If  $i=1, 2, 3, 4, 5$ .

if 3rd iteration should be executed,

'continue' continues the next iteration

i++

$i=3$

continue

$s=1$

$\Sigma=1$

$i=2$

$s=2$

$\Sigma=2$

$i=3$

$s=3$

$\Sigma=3$

$i=4$

$s=4$

$\Sigma=4$

$i=5$

$s=5$

$\Sigma=5$

} for  $i=3$ , these statements are not executed.

for  $(i=1; i \leq 5; i++)$

Break

Output :- 1 2

1  
2  
3  
4  
5 → output

for  $(i=1; i \leq 5; i++)$

Continue

Output :- 1 2 4 5 (Inclusion)

{ if ( $i==3$ )

continue;

printf ("%d", i);

}

2 3 5 → good value

break :- to stop the process

<condition> should

be true

(++i; i >= 3; i++)

(++i; i >= 3; i++)

(++i; i >= 3; i++)

i ("for") thing

→ goto label :-

if ( $n \% 2 == 0$ ) → label

goto Even;

else

goto odd;

Even : printf("even"); both will be

odd : printf("odd"); executed.

goto label; control is transferred  
to ("label") thing  
then it goes to the definition  
of label.

Syntax: `goto label;`

↳ user-defined names based on conditions

label definition `label: starts`

exit from loop by `label`

Nested for loop:  
for (`i=1; i<=3; i++`) `int i=1; i<=3; i++`  
  {    `i=1; i<=3; i++`  
    for (`j=1; j<=3; j++`) `j=1; j<=3; j++`  
     {    `j=1; j<=3; j++`  
        printf (" i=%d, j=%d ", i, j); `i=1; j=1`  
     } `i=1; j=1`  
    } `i=1; j=1`  
  } `i=1; j=1`

for each value of 'i', inner loop is repeated 3 times  
So total iterations =  $3 \times 3 = 9$

Output: `i=1 j=1` `i=1 j=2` `i=1 j=3`  
`i=2 j=1` `i=2 j=2` `i=2 j=3`  
`i=3 j=1` `i=3 j=2` `i=3 j=3`

• Matrix: `i=1 j=1` `i=1 j=2` `i=1 j=3`  
`i=2 j=1` `i=2 j=2` `i=2 j=3`  
`i=3 j=1` `i=3 j=2` `i=3 j=3`

Program :-

1  
1 2  
1 2 3

```
#include <stdio.h>
void main()
{
 int
 for (i=1; i<=3; i++)
 {
 for (j=1; j<=i; j++)
 {
 printf ("%d", j);
 }
 }
}
```

`printf ("x\n");` `i=1; i<=3; i++`  
with `i` at stop `i=1; i<=3; i++`  
  `i=1; i<=3; i++`

outer loop :- rows  
inner loop :- columns `i=1; i<=3; i++`

↳ labels at op

`i=1; i<=3; i++` `i=1; i<=3; i++`

`i=1; i<=3; i++`

so printed `i="rows")` `i="rows")`  
, `rows` `i="cols")` `i="cols")`

```
#include <stdio.h>
void main()
{
 int i=1, j=1;
 for (i=1; i<=3; i++)
 {
 for (j=1; j<=i; j++)
 printf("%d", j);
 }
}
```

printf("\n");  $\Rightarrow$  this is for outer loop.

3. printing diagonal elements in downward direction  
 3.  $\Rightarrow$  from row 3 to row 1 (forward) downward

Program: 1 2 3 4 /< down-trail > transpose A + 3\*3  
 1 2 3 4 (row 4)  
 1 2 3 (row 3)  
 1 2 (row 2)  
 1 (row 1)

1 2 3 4 5 6 7 8 9 (5 columns)  $\Rightarrow$  transpose

#include <stdio.h> for (j=1; j<=5; j++)

void main() { printing digit - matrix pattern }

3

```
int i, j;
for (i=5; i>=1; i++)
{
 for (j=1; j<=i; j++)
 printf("%d", j);
 printf("\n");
}

```

#include <stdio.h>

void main()

{

```
int i, j;
for (i=1, i<=5; i++)
{
 for (j=1
 printf("%d %d", i, i);
 printf("\n");
}

```

## Pascal triangle :-

|         |       |       |   |
|---------|-------|-------|---|
|         | 1     | 1     |   |
|         |       | 2     | 1 |
|         |       | 3     | 3 |
|         | 1     | 4     | 6 |
| 4 3 2 1 | 12345 | 11111 | 1 |
| 3 2 1   | 1234  | 2222  |   |
| 2 1     | 123   | 333   |   |
| 1       | 12    | 44    |   |
|         | 1     | 4     |   |

<Arithmatic operators  
(Division bin)

$$i = [j : i = j] \text{ rot}$$

$$++[j = i : i = j] \text{ rot}$$

"j" string

Operators :- Arithmetic, Relational, Logical, Bitwise, Conditional (ternary), Increment & Decrement, Pre, Post, comma, Assignment, short hand Assignment, operator =.

Separate the expressions

Expressions :- combination of operators and operands

$2 + 3 * 5$  follow priority (preference)  $* > +$  (with level)

Multiplication - high priority.  $+ -$  (with level)

$2 + 15$

$$2 * 15 = 30$$

Operator precedence table :-  
2/3%4 based on associativity L-R or R-L

$((i:j)) : (k:l) = (i:k) : (j:l) \text{ rot}$

$$++[j = i : i = j] \text{ rot}$$

"j" string

$i:j$  string

$$++[j = i : i = j] \text{ rot}$$

$$i = [j] \text{ rot}$$

"j" string

"i" string

## Operator

( )

[ ]

.

$\Rightarrow$

$++-$

$+-$

$! \sim$

(Type)

\*

&

sizeof

\* / %

+ -

<< >>

< <=

> >=

= !=

&

^

|

&&

||

? :

=

+= -+

+= / =

% = &

^= .

<<= >

| <u>Operator</u> | <u>Description</u>                              | <u>Associativity</u> |
|-----------------|-------------------------------------------------|----------------------|
| ( )             | Parentheses or function call                    | left to right        |
| [ ]             | Square brackets or array subscript              | left to right        |
| .               | Dot or Member selection operator                | left to right        |
| →               | Arrow operator                                  | left to right        |
| ++ --           | Positive increment / decrement                  | left to right        |
| ++--            | Prefix increment / decrement                    | right to left        |
| + -             | Unary plus and minus                            | right to left        |
| ! ~             | not operator & bitwise complement               | right to left        |
| (type)          | Type cast                                       | right to left        |
| *               | Indirection or dereference operator             | right to left        |
| &               | Address of operator                             | right to left        |
| sizeof          | Determine size in bytes                         | right to left        |
| * / %           | Multiplication, Division & modulus              | left to right        |
| + -             | Addition and subtraction                        | left to right        |
| << >>           | Bitwise left shift & right shift                | left to right        |
| < <=            | Relational less than / less than equal to       | left to right        |
| > >=            | Relational greater than / greater than equal to | left to right        |
| == !=           | Relational equal to / not equal to              | left to right        |
| &               | Bitwise AND                                     | left to right        |
| ^               | Bitwise exclusive OR                            | left to right        |
|                 | Bitwise inclusive OR                            | left to right        |
|                 | Logical AND                                     | left to right        |
|                 | Logical OR                                      | left to right        |
| ? :             | Ternary operator                                | right to left        |
| =               | Assignment operator                             | right to left        |
| += -=           | Addition / Subtraction assignment               | right to left        |
| *= /=           | Multiplication / division assignment            | right to left        |
| % = &=          | Modulus and bitwise assignment                  | right to left        |
| ^= ^=           | Bitwise exclusive (inclusive OR) assignment     | right to left        |
| <<= >>=         | Bitwise assignment                              | right to left        |
| ,               | Comma operator                                  | left to right        |

4321  
821  
21  
1

Program:-      returns result of  $\pi$  to standard output

```
#include <stdio.h>
Void main()
{
 int i,n,j;
 float pi;
 Scanf ("%d", &n);
 for(i=n; i>=1; i--)
 {
 for(j=i, j>=1; j--)
 printf ("%d", j);
 printf ("\n");
 }
}
```

3. Input at first      at beginning of all numbers  
                          or at end of all numbers  
                          or at any place

12345      at first      at beginning of all numbers  
123      at first      at beginning of all numbers

12345      at first      at beginning of all numbers

Program:-      at first

```
#include <stdio.h>
Void main()
{
 12345 at first
 int i,j,n;
 Scanf ("%d", &n);
 for(i=n; i>=1; i--)
 {
 for(j=1; j<=i; j++)
 printf ("%d", j);
 printf ("\n");
 }
 printf ("01\n");
}
```

Program

#

%

<

=

for

1111  
2222  
333  
44  
5.

Program :-

```
#include <stdio.h>
void main()
{
 int n, i, c;
 scanf("%d", &n);
 for (i = 0; i <= n; i++)
 {
 for (j = 0; j < i; j++)
 for (c = 1; c <= i; c++)
 printf("%d", G[i]);
 printf("\n");
 }
}
```

Output :- Those things

```
#include <stdio.h>
void main()
{
 int i, n, c;
 scanf("%d", &n);
 for (i = n; i >= 0; i--)
 {
 for (c = 1; c <= i; c++)
 printf("%d", G[i]);
 printf("\n");
 }
}
```

statements  
→ d. define  
→ d. define  
→ d. define  
→ d. print

-: output part of statements  
→ statements in final definition of  
statements function (4)

that message → entering about global A as main  
main, function

program starts at main - → output  
and now prints ( main block )

local local

(equating

main function →

the main -

## ✓ FUNCTION :-

Structure of C program:- Documentation section  
Preprocess directives.  
Header file section.

Successful compilation  
→ objective code  
→ no errors  
→ I & O format

Global declaration  
void main() function  
local declaration  
Executable section

functions / Pre-defined  $\Rightarrow$  built-in = only limited no. of  
user-defined.

name()  $\Rightarrow$  function

After main function

Userdefinedfunction1()

{  
=

Userdefined function2()

{  
=

Functions are two types:-

- (1) Predefined / Built-in functions
- (2) Userdefined functions

function :- A block of code performs a specific task.

## User defined functions

four types:-

Structure :-  
- function declaration - to allocate memory  
(also called function prototype) declaring variables.  
local global

- function definition
- function call

C++ : key words:- int, float, ...

else, ...  
also called

reserved words

i, for, ...

for, printf, scanf, sqrt, pow,  
abs.

headerfiles

stdio.h = functions

conio.h =

math.h =

string.h =

function

is called

function &  
body.

if ret

statement  
body  
end

Exit

func

## function declaration:-

Syntax:

Return type    functionname(argument1, argument2);  
 ↓  
 userdefined data type & name.  
 name      scope  
 (cannot be key words)      optional.  
 return type: int, float, char returning the same no. of  
 if no return      void.  
 →      return type is optional.

Ex:-

int fact(int n);      int add(int a, int b);  
 (or)      int add(int a, char b);  
 int fact(int);      int add(int a, char b);  
 (or)      int add(int, int);  
 →      int add(int, int);      mandatory.

Diagrammatic representation of type & name

## function definition:-

is called      ( ) { }      discarded function header.  
 function body      ( ) { }  
 →      ( ) { }  
 keyword      return value;  
 →      returns only one value?  
 cannot be used for swapping.

if return type is void

return value; ( ) not needed.

then print value

Ex:- int fact (int n)

{ (S, B, T) thing

int i, fact1=1;

for(i=1; i<=n; i++)      for calling fact(s) two times (S)

fact1=fact1\*i;

return fact1;

}

(C) int or void

(data variable) names / values,

## function call:-

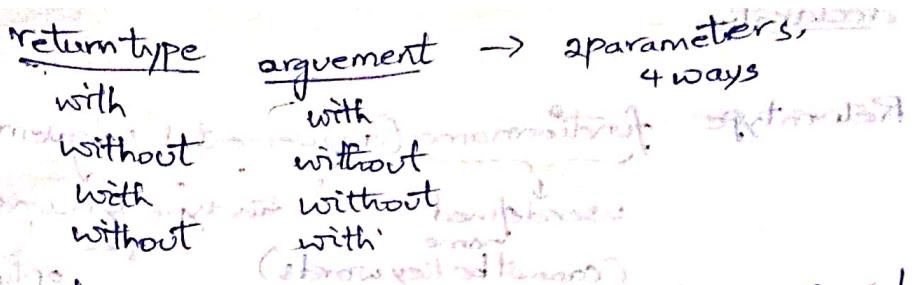
fact(5);

(or)

fact(n);

↓  
actual argument, actual parameter is void

formal (?)



Based on return type & arguments, user defined functions are four types.

### User defined function categories:-

- without return type      without arguments
- without return type      with arguments
- with return type      without arguments
- with return type      with arguments.

#### (1) without return type without arguments.

~~void add();~~

~~void add(int a, int b)~~

~~void add(void);~~

~~void add(void)~~

~~{ void sum = a + b; }~~

~~if (sum == 100) cout << "Sum is 100";~~

```
void main()
{
 void add();
 add(); // no arguments.
}
```

#### void add(); (operator)

```
int a, b, c;
scanf("%d%d", &a, &b);
c = a + b;
printf("%d", c);
```

#### (2) without return type with arguments:-

```
void main()
{
 void add(int a, int b);
 int a, b;
```

~~scanf("%d%d", &a, &b);~~ ← before here, read the values first.

```
void add(int a, int b)
{
 int c;
```

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

#### (3) with return

```
void main()
{
 void add();
 int a, b;
 scanf("%d%d", &a, &b);
 add(a, b);
}
void add()
{
 int c;
 c = a + b;
 printf("%d");
}
```

#### (4) with return

```
void main()
{
 int add();
 int a, b;
}
```

```
int add()
{
 int a, b;
 scanf("%d%d", &a, &b);
 c = a + b;
 return c;
}
```

#### (5) with return

```
void main()
{
 int add();
 int a, b;
 scanf("%d%d", &a, &b);
 D = add(a, b);
}
```

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

Value of c is displayed

3) with return type without arguments.

(a) void main()

```
{
void add (int p,int q);
int a,b;
scanf ("%d%d",&a,&b);
add (a,b);
}
```

```
void add (int p,int q)
{
int c;
c=p+q;
printf ("%d",c);
}
```

3) with return type without arguments

void main()

```
{
int add (int);
int a,b;
```

~~int add();~~ → (or) ~~printf ("%d",add());~~

int add (void)

```
{
int a,b,c;
```

scanf ("%d%d",&a,&b);

c=a+b;

return c;

}

(4) with return type with arguments

void main()

{

int add (int ,int);

~~int~~

int a,b;

scanf ("%d%d",&a,&b);

D=add (a,b);

```

int add(int a, int b)
{
 int c;
 c = a + b;
 return c;
}

```

(3) with return without argument :-

```

void main()
{
 int add();
}

```

$D = add()$

$\text{printf}("%d", D);$

}

int add(void)

{

int a, b, c;

$\text{scanf}("%d %d", &a, &b);$

$c = a + b;$

return c;

}

=

Recursion :- A function calls itself  $\rightarrow$  recursion

non-recursive function

recursive function

func()

{

func();

}

func calls  
func  
(itself)

main() (func) bbs tri

{

func();

main function

{

func(); b calls func

{

func is matter

}

}

fact(5)

fact = 1; initial value sqrt matter after (1)

for ( $i = 1; i \leq 5; i++$ ) (initial b)

fact = fact \* i;

fact(n) =

$n * \text{fact}(n-1)$

$n * \text{fact}(n-2)$

$n * \text{fact}(n-3)$

$\dots$

$$\frac{5 \times 4 \times 3 \times 2 \times 1}{4!}$$

here no need to use any loops as otherwise by calling itself, it is repeating.

factor(n) =

$$\begin{aligned} & n * \text{fact}(n-1) \\ & 5 * \text{fact}(4) \\ & 5 * 4 * \text{fact}(3) \\ & 5 * 4 * 3 * \text{fact}(2) \\ & 5 * 4 * 3 * 2 * \text{fact}(1) \\ & 5 * 10 \\ & 50 \end{aligned}$$

fact(1)

should return to  $\text{fact}(2) * \text{fact}(1)$

$\text{fact}(2) = 2 * 1 = 2$  should return to  $\text{fact}(2)$

$\text{fact}(5)$  should be returned to main

queue

DS :- Data Structure

Places all the calls in

LIFO last in first out

FIFO "first in last out"

internals function calls should be solved first

|                      |
|----------------------|
| $2 * \text{fact}(0)$ |
| $3 * \text{fact}(2)$ |
| $4 * \text{fact}(3)$ |
| $5 * \text{fact}(4)$ |
| $\text{fact}(5)$     |

solver first

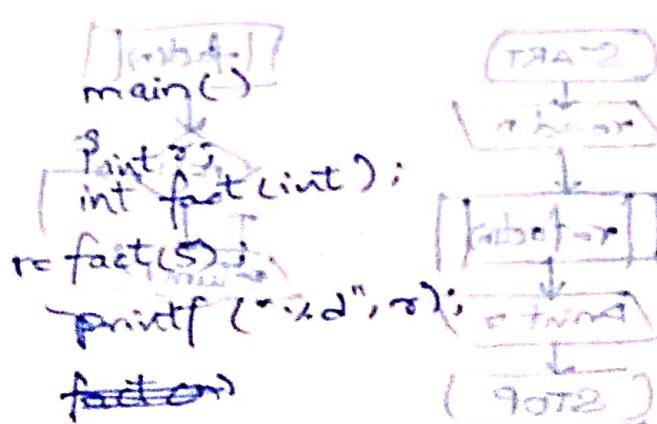
(~~2 \* 1~~) //

1 \* 1 = 1

3 \* 1 = 3

4 \* 1 = 4

5 \* 1 = 5



wait for traversal

int fact(int n)

{

if (n==1)

return 1;

else

return (n+fact(n-1));

Print gcd using recursive :-

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

```
int gcd(int a, int b);
printf("%d", gcd
scanf("%d%d", &a, &b);
printf("%d", gcd(a, b));
}
```

```
int gcd(int a, int b)
{
```

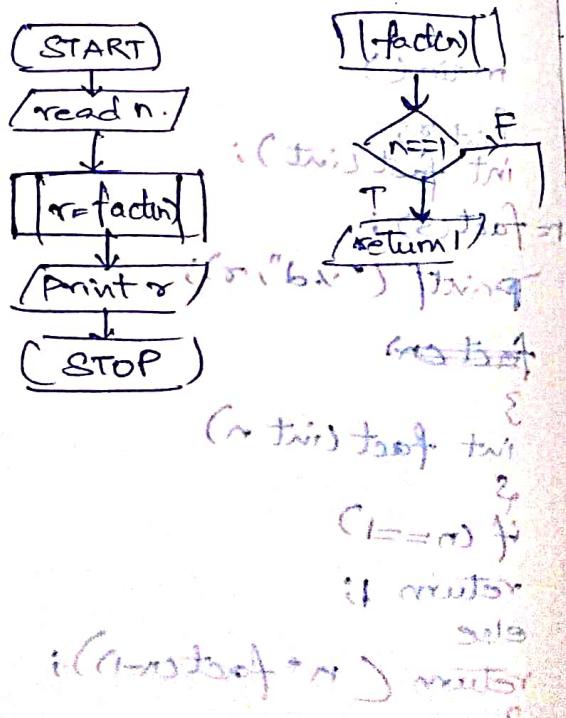
```
if (a < b) return b;
else
```

```
b = a + b; a = b - a;
b = b - a;
```

```
- if a % b == 0
 if (b == 0)
 return b;
 else
 return (gcd(b, a % b));
}
```

Also, also note that ax.b not in  
if b not 0

Flowchart for function:-



fibonacci series

0, 1

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

{

```
int i=0, j=1;
scanf("%d", &i);
for (k=0; k<
```

{

```
i=j;
j=l;

```

{

```
j=i+j;
printf("%d", i);
i=j;
j=l;

```

{

```
j=l;
}
```

.

with non-recursion

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

{

```
int n;
scanf("%d", &n);
printf ("%d",
```

{

```
int fibonaci();
int fibonaci()
{
```

{

```
int i=0, j=1;
for (k=0; k<
```

{

```
i=j;
j=l;

```

{

```
i=j;
j=l;

```

{

```
i=j;
j=l;

```

{

## Fibonacci Series

0, 1, 1, 2, 3, 5, 8, 13, 21,

#include <stdio.h>

void main()

{

int i=0, j=1, k=n;

scanf("%d", &n);

for (k=0; k<n; k++)

{

l=i+j;

printf("%d", l);

i=j;

j=l;

}

}

};

with non-recursive function:

#include <stdio.h>

void main()

{ int n; → int fibonacci(int);

scanf("%d", &n);

printf("%d", fibonacci(n));

}

int fibonacci (int n)

{

int i=0, j=1, l, K;

for (K=0; K<n; K++)

{

l=i+j;

printf("%d", l);

i=j;

j=l;

}

};

with recursive function:-

```
#include<stdio.h>
void main()
{
 int fib(int);
 int n;
 scanf("%d", &n);
 printf("\n%d\n", fib(n));
}
int fib(int n)
{
 if (n==1)
 return 0;
 if (n==2)
 return 1;
 if (n>2)
 return fib(n-1) + fib(n-2);
}
```

```
#include<stdlib.h>
void main()
{
 int fib(int);
 int n, i=1, c=0;
 scanf("%d", &n);
 for (c=0; c<n; c++)
 {
 printf("\n%d\n", fib(i));
 i++;
 }
}
int fib(int n)
{
 if (n==1)
 return 0;
```

```
<stdio.h> sub with
() main()
{
 int n, i=0;
 for (i=0; i<n; i++)
 fib(i);
 printf("\n%d\n", fib(n));
}
int fib(int n)
{
 if (n==1)
 return 0;
 if (n==2)
 return 1;
 if (n>2)
 return fib(n-1) + fib(n-2);
}
```

3 = 2+1 = fib(2) + fib(1)

4 = 3+1 = fib(3) + fib(2)

5 = 2+2+1 = fib(4) + fib(3)

6 = 3+2+1 = fib(5) + fib(4)

7 = 2+1+2+1 = fib(6) + fib(5)

8 = 1+0+1+1+0 = fib(7) + fib(6)

variable :- to store  
for 'i' var values  
instead of var  
we can store m  
values of arry  
Any name which  
array name.  
array :- collection  
arrayname[size]  
Ex:- int a[5];  

|      |      |      |      |      |
|------|------|------|------|------|
|      |      |      |      |      |
| 2000 | 2002 | 2004 | 2006 | 2008 |

  
array is  
2 = position  
This array  
same do

it can't hold  
in 1 byte  
by stru  
let      0 1  
      10 20

To distinguish  
parts of  
index b

Each value

10 = a[0]

string

declaring  
sets  
balancing

workshop

## ARRAYS

variable :- to store only one value.

for 5 values, we need 5 variables.

Instead of using 5 variable, we can store multiple variables values in one variable using arrays.

Any name which follows  $\text{arrayname}[size]$ , it is called array name.

array :- collection of values of same data type.

$\text{arrayname}[size]$  :- cont. location of memory

e.g.  $\text{int a[5]}$ ; 5 integer values - 5 \* 2 bytes = 10 bytes.

|      |      |      |      |      |
|------|------|------|------|------|
|      |      |      |      |      |
| 2000 | 2002 | 2004 | 2006 | 2008 |

array is allocated in continuous manner locations.

This array holds 5 values of int type same data types - so it is called homogeneous collection of values.

It can't hold 10 from int & 20 from char. This is the obvious & draw back, which can be explained by structure.

Let  $\begin{matrix} 0 & 1 & 2 & 3 & 4 \end{matrix}$  → index here all values are stored under same variable 'a'.

To distinguish each value, we use indexing if index begins with '0'. i.e. 0, 1, 2, 3, 4

Each value of array is accessed by its index.

$10 = a[0]$ ,  $25 = a[3]$ .

string :- collection of characters - word

declaring variables! 3 ways (1) Initialisation :-  $\text{int a=2}$ ;  
(2) Assignment  $\text{int a=2};$

Input Output

int a

|   |   |   |   |
|---|---|---|---|
| a | b | c | d |
|   |   |   |   |

{3} Scanf. : [2] is the

: (Eg.: "123") thing is a variable of

word

Types of arrays :- based on dimensions.

int a[ ] below are plus prob of i address  
1-D      2-D      3-D  
int a[3]    int a[3][ ]    int a[3][ ][ ]

Declaration :- syntax :- datatype arrayname [size];  
Ex:- int a[10];

Address of 1-D array is pointer to smallest element.  
eg:- int arr[5]; pointer to first element.

10x2=20 bytes allocated  
maximum capacity  
we can use only 10 bytes  
minimum limited bytes  
it's a drawback.

Assigning values:  
(1) Initialisation : passing values at declaration.  
(2) Reading at run time : (3) Assignment operator.

(1) Initialisation:

int a[5]={1,2,3,4,5}; no. of elements is known  
in initialisation method, size is not mandatory as we assign values at declaration & it counts the size.

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

we can give no. of values less than the size then remaining values are stored as garbage values (0). It depends on storage classes.

storage classes :- auto, register, static, const, volatile, extern, etc.

int a[5]; value 110 print

printf("%d", a);

it prints some garbage value due to auto storage class before int. '0' after signed 'register'

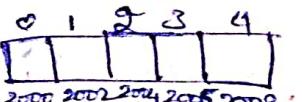
register signed because it's value is not known to user who

int a[5]={1,2,3}; [E]D=25, [O]D=0

int a[5]={0}; - 25 bytes for variables to print.

(2) Reading at run time :- using scanf or something else

int a[5]; - took



for character values  
getchar

to print  
for address :- printf("%u", &a[0]);  
→ 2000

printf("%u")  
scanf("%u")  
scanf("%u")  
let f instead use  
for(c;c<0) scanf("%u")  
if we run sun, it prints

Printing:

print  
instead  
{  
loops

for(c  
print

(3) Assignment:

int a  
a[0];  
a[1];  
:

printf("%d\n", &a[0]); // 2002 at memory 0. start  
printf("%d", a); // 2000 a - array name. <right> shallow  
add. starting address. (In main block)

scanf("%d", &a[0]);  
scanf("%d", &a[1]);  
scanf("%d", &a[2]);

let for int a[100];

instead of writing scanf for 100 times we can use loops. as only index varies each time.

for(i=0; i<100; i++)

scanf("%d", &a[i]);

if we want to store only some values for this run, then ask user to select no. of values = n.

int a[100], n;  
scanf("%d", &n);  
for(i=0; i<n; i++)  
scanf("%d", &a[i]);

Printing array "values from

printf("%d", a[0]);  
instead of writing printf for many times use  
loops:  $i < [i] \neq 0$   
for(i=0; i<n; i++)  
printf("%d", a[i]);  
%d "bx" thing

(3) Assignment operator :-

int a[5]  
a[0]=1;  
a[1]=2;

:

=

-> print 2  
<right> shallow  
(In main block)

(i,j) & [2] 0 tri  
:(0,0,0,0,0) free  
(+/-, 2) i : 0 : j ref  
(+/-, 2) i : 0 : j ref

Write a program to read and print 4 numbers.

```
#include <stdio.h>
void main()
{
 int a[4], i, n;
 scanf("%d", &n);
 for (i=0; i<n; i++)
 scanf("%d", &a[i]);
 for (i=0; i<n; i++)
 printf("%d", a[i]);
```

3rd no. of smaller ones who reads of those no. if  
the smaller from below it now start in next row  
Write a program to find max. element in an array.

```
#include <stdio.h>
void main()
{
 int a[5], i, n;
 scanf ("%d", &n);
 for (i=0; i<n; i++)
 scanf ("%d", &a[i]);
 if (a[i] > c)
 c = a[i];
```

Sorting:-

```
#include <stdio.h>
void main()
{
 int a[5], i, j, n;
 scanf ("%d", &a[0]);
 c = a[0];
 for (i=0; i<5, i++)
 {
 c = a[i];
```

```
#include <stdio.h>
Void main()
{
 int a[5], i, n;
 scanf ("%d", &a[0]);
 for (i=0; i<4; i++)
 scanf ("%d", &a[i]);
 if (a[i] > c)
 c = a[i];
 if (c > a[0])
 printf ("%d", c);
```

(2) a tri  
i = [0] 0  
j = [1] 0

for (j=0; j<4; j+)
{
 scanf ("%d", &a[j]);
 if (a[j] > c)
 c = a[j];
 printf ("%d", c);
}

```
#include <stdio.h>
void main()
{
 int a[5];
 scanf ("%d", &a[0]);
 for (i=0; i<4; i++)
 scanf ("%d", &a[i]);
 if (a[i] > a[0])
 a[i] = a[0];
}
```

```
#include <stdio.h>
void main()
{
 int a[5];
 for (i=0; i<4; i++)
 a[i] = a[0];
}
```

if  
c  
0

```

for (j=0; j<4; j++)
{
 scanf("%d", &a[j]);
 if (a[j] > c)
 c = a[j];
 printf("%d", c);
}

```

---

```

#include <stdio.h>
void main()
{
 int a[5], i, n;
 a[i]
 scanf("%d", &a[0]);
 c = a[0];
 for (i=0; j=0; i<5 && j<5; i++, j++)
 scanf("%d", &a[j]);
 if (a[j] < c)
 a[i] = a[j];
 c = a[j];
}

```

<4.0.0> abulsoit  
 ( ) niam hciu  
 3.  
 ini. Ego tm  
 :((a. "bx")) fnoz  
 (++i; n>j; o=i) rot  
 :((i>0. "bx")) fnoz  
 (+j; i>j; o=j) rot  
 (o==s. x(i>j)) fi  
 :((i>0. "bx")) fnoz  
 {
 }

<4.0.0> abulsoit  
 ( ) niam hciu  
 4. 3.  
 (2.0. ini. Ego tm  
 :((a. "bx")) fnoz  
 (++i; n>j; o=i) rot  
 :((i>0. "bx")) fnoz  
 (++i; o>j; i>j) rot  
 (o=2; o=3) f  
 (o<[j]) fi  
 (3+3=5  
 (c=5  
 (c>0+2=2  
 :((i. "bx")) fnoz  
 :((i. "bx")) fnoz  
 3 4 8 6 2  
 (++i; n>j; o=i) rot  
 (o=2; o=3

---

```

#include <stdio.h>
void main()
{
 int a[5], i, j, n;
 c = a[0];
 for (i=0; i<5; i++)
 {
 for (j=0; j<4; j++)
 {
 scanf("%d", &a[j]);
 if (a[j] < c)
 c = a[j];
 a[i] = c;
 }
 }
}

```

Write a program to find and print even numbers in an array.

```
#include<stdio.h>
Void main()
{
 int a[5], i, n;
 scanf("%d", &n);
 for (i=0; i<n; i++)
 scanf("%d", &a[i]);
 for (i=0, j=n; i<j)
 {
 if (a[i] % 2 == 0)
 printf("%d", a[i]);
 }
}
```

Write a program to count and print no. of positive, negative, even odd numbers and also display their sum.

```
#include<stdio.h>
void main()
{
 int a[5], i, n, c, s;
 scanf("%d", &n);
 for (i=0; i<n; i++)
 scanf("%d", &a[i]);
 for (i=0; i<n; i++)
 {
 if (a[i] > 0)
 c = c + 1;
 s = s + a[i];
 printf("%d", c);
 printf("%d", s);
 }
}

```

if ( $a[i] < 0$ )

```
s
c = c + 1;
s = s + a[i];
}
```

printf("%d",

printf("%d",

for (i=0; i<n; i++)

```
c = 0; s = 0;
```

if ( $a[i] >$

```
c = c + 1;
```

```
s = s + a[i];
```

printf("%d",

printf("%d",

for (i=0;

```
c = 0; s = 0;
```

if ( $a[i] >$

```
c = c + 1;
```

```
s = s + a[i];
```

printf("%d",

printf("%d",

for (i=0;

```
c = c + 1;
```

```
s = s + a[i];
```

printf("%d",

printf("%d",

#incl

Void

```
s
int
```

```
Sc
```

for

Sc

```

if (a[i] > 0)
{
 c = c + 1;
 s = s + a[i];
}
printf ("%d", c);
printf ("%d", s);
}

for (i=0; i<n; i++)
{
 c=0; s=0;
 if (a[i]*x2 == 0)
 {
 c = c + 1;
 s = s + a[i];
 }
 printf ("%d", c);
 printf ("%d", s);
}

for (i=0; i<n; i++)
{
 c=0; s=0;
 if (a[i]*x2 != 0)
 {
 c = c + 1;
 s = s + a[i];
 }
 printf ("%d", c);
 printf ("%d", s);
}

```

( $c > 0 \wedge a[i] > 0 \wedge s \neq 0$ )  $\vdash$

( $a[i] > 0 \wedge s \neq 0$ )  $\vdash$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

( $a[i] > 0 \wedge s \neq 0$ )  $\vdash$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

( $a[i] > 0 \wedge s \neq 0$ )  $\vdash$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

( $a[i] > 0 \wedge s \neq 0$ )  $\vdash$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

$[i]_0 + [i]_1 + \dots + [i]_{p-1} = s$

$i+1 \leq p$

#include <stdio.h>

Void main()

{

int a[5], i, n,

scanf ("%d", &n);

for (i=0; i<n; i++)

scanf ("%d", & a[i]);

```

for (i=0; i<n; i++)
{
 if (a[i] > 0)
 {
 c1 = c1 + 1;
 s1 = s1 + a[i];
 }
 else if (a[i] < 0)
 {
 c2 = c2 + 1;
 s2 = s2 + a[i];
 }
 else if (a[i] * 2 == 0)
 {
 c3 = c3 + 1;
 s3 = s3 + a[i];
 }
 else if (a[i] * 2 != 0)
 {
 c4 = c4 + 1;
 s4 = s4 + a[i];
 }
}

```

$c_1$ ,  $s_1$ ,  $c_2$ ,  $s_2$ ,  $c_3$ ,  $s_3$ ,  $c_4$ ,  $s_4$ ;  
 }  
 printf(" no. of positive numbers = %d in sum of positive  
 numbers = %d\n no. of negative numbers = %d in sum  
 of negative numbers = %d\n no. of even numbers  
 = %d in sum of even numbers = %d\n no. of odd  
 numbers = %d in sum of odd numbers = %d",

$c_1, s_1, c_2, s_2, c_3, s_3, c_4, s_4);$

<file:///C:/Users/Asus/Downloads/Untitled 1.c>

(Compiling)

min([2]) tri

:("m", "bx") true

(++i; (n>i, 0>i)) not

:("E") o & ("bx") false

```

#include <stdio.h>
void main()
{
 int a[5];
 for(i=0; i<5; i++)
 scanf("%d", &a[i]);
 for(i=0; i<5; i++)
 {
 for(j=i; j<5; j++)
 if(a[j]<a[i])
 {
 a[i] = a[i]+a[j];
 a[j] = a[i]-a[j];
 a[i] = a[i]-a[j];
 }
 }
 for(i=0; i<5; i++)
 printf("%d", a[i]);
}

```

Sorting

Time complexity:  $O(n^2)$

Space complexity:  $O(1)$

Properties: In-place sorting, stable sort.

Passing array as argument to a function:

```

#include <stdio.h>
void main()
{
 int a[5];
 void display(int a[], int n);
 int b[5];
 for(i=0; i<5; i++)
 scanf("%d", &b[i]);
 display(b, 5);
 for(i=0; i<5; i++)
 printf("%d", a[i]);
}

```

Time Complexity:  $O(n^2)$

Space Complexity:  $O(1)$

Properties: Non-local, Non-modifiable, Non-reentrant.

Types of arrays: based on dimension  
array array array array

One dimensional array, two dimensional, three dimensional, multi dimensional array.

(1-D array), (2-D array) . . . . . more than 1-D

1-D :- arrayname[], 2-D :- arrayname[3][3]  
3-D :- arrayname[3][3][3]

Two-dimensional array:-

Declarations:- datatype arrayname [rowsize][columnsize];

Ex:- int a[2][3];

| 0 <sup>th</sup> col | 1 <sup>st</sup> col | 2 <sup>nd</sup> col |         |
|---------------------|---------------------|---------------------|---------|
| 0 <sup>th</sup> row | a[0][0]             | a[0][1]             | a[0][2] |
| 1 <sup>st</sup> row | a[1][0]             | a[1][1]             | a[1][2] |

Assigning values:- Initialisation, Reading at run time, assignment.

Initialisation:- int a[2][3] = {1, 2, 3, 4, 5, 6};

rows=optional, columns=mandatory

to know calculate rows,

int a[ ][3] = {1, 2, 3, 4, 5, 6};

- int a[2][ ] = {

contains 2 sets  
each set holds  
3 values

{1, 2, 3},

{4, 5, 6}; } valgib hov

};

Reading at run time:-

for (i=0; i<rowsize; i++)

{

    for (j=0; j<columnsize; j++)

{

        scanf("%d", &a[i][j]);

}

}

{ a[i][j] = valgib hov

{ i + j } valgib hov

Printing :- A function which prints a 2D array.

```

for (i=0; i<rowsize; i++)
{
 for (j=0; j<colsize; j++)
 printf ("%d\t", &a[i][j]);
 printf ("\n");
}

```

Assignment :-

```

int a[10][10];
int rowsize, colsize;
scanf ("%d %d", &rowsize, &colsize);
for (i=0; i<rowsize; i++)
{
 for (j=0; j<colsize; j++)
 scanf ("%d", &a[i][j]);
}

```

Assignment:-

```

int a[10][10];
a[0][0]=10;
a[0][1]=20;
a[0][2]=30;

```

$$a[i][j] + a[i][j] = a[i][j]$$

~~a[i][j] = b[x]~~

Write a program to read and print a 2-D array.

```

#include <stdio.h>
void main();
{
 int a[10][10];
 int rowsize, colsize;
 scanf ("%d %d", &rowsize, &colsize);
 for (i=0; i<rowsize; i++)
 {
 for (j=0; j<colsize; j++)
 scanf ("%d", &a[i][j]);
 }
 for (i=0; i<rowsize; i++)
 {
 for (j=0; j<colsize; j++)
 printf ("%d\t", &a[i][j]);
 printf ("\n");
 }
}

```

Write a program to add two matrices :-

```
#include <stdio.h>
Void main()
{
 int a[3][3], b[3][3], c[3][3];
 for (i=0; i<3; i++)
 {
 for (j=0; j<3; j++)
 scanf ("%d", &a[i][j]);
 }
 for (i=0; i<3; i++)
 {
 for (j=0; j<3; j++)
 scanf ("%d", &b[i][j]);
 }
 for (i=0; i<3; i++)
 {
 for (j=0; j<3; j++)
 c[i][j] = a[i][j] + b[i][j];
 }
 printf ("%d", c[i][j]);
}
```

for loop 2 nesting has been done at memory structure

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

```
#include <stdio.h>
Void main()
{
 int a[3][3], b[3][3];
 scanf ("%d %d %d", &a[0][0], &a[0][1], &a[0][2]);
 scanf ("%d %d %d", &a[1][0], &a[1][1], &a[1][2]);
 scanf ("%d %d %d", &a[2][0], &a[2][1], &a[2][2]);
 scanf ("%d %d %d", &b[0][0], &b[0][1], &b[0][2]);
 scanf ("%d %d %d", &b[1][0], &b[1][1], &b[1][2]);
 scanf ("%d %d %d", &b[2][0], &b[2][1], &b[2][2]);
 if (a[0][0] == b[0][0])
 {
 for (i=0; i<3; i++)
 {
 for (j=0; j<3; j++)
 printf ("%d", a[i][j] + b[i][j]);
 }
 }
 else
 printf ("Matrices are not equal");
}
```

## Matrix Multiplication

```
#include <stdio.h>
Void main()
{
 int a[3][3], b[3][3];
 scanf ("%d %d %d", &a[0][0], &a[0][1], &a[0][2]);
 scanf ("%d %d %d", &a[1][0], &a[1][1], &a[1][2]);
 scanf ("%d %d %d", &a[2][0], &a[2][1], &a[2][2]);
 scanf ("%d %d %d", &b[0][0], &b[0][1], &b[0][2]);
 scanf ("%d %d %d", &b[1][0], &b[1][1], &b[1][2]);
 scanf ("%d %d %d", &b[2][0], &b[2][1], &b[2][2]);
 if (a[0][0] == b[0][0])
 {
 for (i=0; i<3; i++)
 {
 for (j=0; j<3; j++)
 printf ("%d", a[i][j] * b[j][i]);
 }
 }
 else
 printf ("Matrices are not equal");
}
```

```

#include <stdio.h>
void main()
{
 int a[3][3], b[3][3], c[3][3];
 scanf ("%d %d %d %d", &rowsize1, &colsiz1, &rowsize2,
 &colsiz2);
 if ((rowsize1 == rowsize2) && (colsiz1 == colsiz2))
 {
 for (i=0; i<rowsize1; i++)
 for (j=0; j<colsiz1; j++)
 scanf ("%d", &a[i][j]);
 for (i=0; i<rowsize2; i++)
 for (j=0; j<colsiz2; j++)
 scanf ("%d", &b[i][j]);
 for (i=0; i<rowsize1; i++)
 for (j=0; j<colsiz2; j++)
 printf ("%d + %d * %d = %d\n", a[i][j], a[i][j] * b[i][j], b[i][j], a[i][j] * b[i][j] + a[i][j]);
 else
 printf ("Addition is not possible");
 }
}

```

### Matrix Multiplication:

```

#include <stdio.h>
void main()
{
 int a[3][3], b[3][3], c[3][3], s=0, i, j, k;
 scanf ("%d %d %d %d", &rowsize1, &colsiz1, &rowsize2,
 &colsiz2);
 if (colsiz1 == rowsize2)
 {
 for (i=0; i<rowsize1; i++)
 for (j=0; j<colsiz2; j++)
 scanf ("%d", &c[i][j]);
 for (i=0; i<rowsize1; i++)
 for (j=0; j<colsiz2; j++)
 for (k=0; k<colsiz1; k++)
 s = a[i][k] * b[k][j] + s;
 c[i][j] = s;
 }
}

```

```

for (i=0; i< rowsize2; i++)
{
 for (j=0; j< colsiz2; j++)
 scanf ("%d", &b[i][j]);
 }

 for (i=0; i< rowsize1; i++)
 for (j=0; j< colsiz1; j++)
 s = a[i][j] * b[i][j];
 }

 for (j=0; j< colsiz2; j++)
 printf ("%d", s);
 }

 printf ("\n");
}

```

~~for (i=0; i< rowsize2; i++)~~  
~~for (j=0; j< colsiz2; j++)~~  
~~scanf ("%d", &b[i][j]);~~  
~~for (i=0; i< rowsize1; i++)~~  
~~for (j=0; j< colsiz1; j++)~~  
~~s = a[i][j] \* b[i][j];~~  
~~printf ("%d", s);~~  
~~printf ("\n");~~

**Matrix Multiplication:**  
 #include <stdio.h>  
 void main()
 {
 int a[3][3], b[3][3];
 scanf ("%d%d", &a[0][0], &a[0][1]);
 if (a[0][0] == r2)
 {
 for (i=0; i<r1)
 for (j=0; j<c1)
 scanf ("%d", &a[i][j]);
 for (i=0; i<r2)
 for (j=0; j<c2)
 scanf ("%d", &b[i][j]);
 for (i=0; i<r1; i++)
 for (j=0; j<c2; j++)
 c[i][j] = 0;
 for (k=0; k<r2; k++)
 c[i][j] = c[i][j] + a[i][k] \* b[k][j];
 for (i=0; i<r1; i++)
 for (j=0; j<c2; j++)
 printf ("%d", c[i][j]);
 printf ("\n");
 }
 }

~~for (i=0; i< r1; i++)~~  
~~for (j=0; j< c2; j++)~~  
~~{~~  
 ~~if (c[i][j] == 0) { i, j, a[i][j], b[j][k], a[i][k] \* b[k][j]}~~  
 ~~for (k=0; k< r2; k++)~~  
 ~~c[i][j] = c[i][j] + a[i][k] \* b[k][j];~~  
 ~~}~~  
 ~~for (j=0; j< c2; j++)~~  
 ~~printf ("%d", c[i][j]);~~  
 ~~printf ("\n");~~

$$C \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix}_{3 \times 3} = A \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \\ A_{20} & A_{22} \end{bmatrix}_{3 \times 3} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{bmatrix}_{3 \times 3}$$

$$C_{ij} = C_{ij} + A_{ik} * B_{kj}$$

## Matrix Multiplication

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

```
void main()
```

```
{
```

```
int a[3][3], b[3][3], c[3][3], i, j, k;
```

```
scanf ("%d%d%d%d", &r1, &c1, &r2, &c2);
```

```
if (c1 == r2)
```

```
{
```

```
for (i=0; i<r1; i++)
```

```
for (j=0; j<c2; j++)
```

```
scanf ("%d", &a[i][j]);
```

```
for (i=0; i<r2; i++)
```

```
for (j=0; j<c1; j++)
```

```
scanf ("%d", &b[i][j]);
```

```
{
```

```
for (i=0; i<r1; i++)
```

```
for (j=0; j<c2; j++)
```

```
{
```

```
c[i][j] = 0;
```

```
for (k=0; k<r2; k++)
```

```
c[i][j] = c[i][j] + a[i][k] * b[k][j];
```

```
→ printf ("%d", c[i][j]);
```

```
→ printf ("\n");
```

(or) → for (i=0; i<r1; i++)

```
for (j=0; j<c2; j++)
```

```
printf ("%d", c[i][j]);
```

```
printf ("\n");
```

```
{
```

```
{
```

Matrix Subtraction  
C = A - B

(+/-) (row i)(column j) rot.

Transpose of a matrix:

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

```
{
```

```
 int a[3][3], b[3][3], c[3][3];
 int i, j, k;
```

```
 scanf ("%d %d %d", &r1, &c1);
```

```
 for (i=0; i<r1; i++)
```

```
 for (j=0; j<c1; j++)
```

```
 scanf ("%d", &a[i][j]);
```

```
 for (i=0; i<c1; i++)
```

```
 for (j=0; j<r1; j++)
```

```
 scanf ("%d", &b[i][j]);
```

```
 for (i=0; i<r1; i++)
```

```
 for (j=0; j<c1; j++)
```

```
 b[i][j] = a[j][i];
```

```
 printf ("%d", b[i][j]);
```

```
 printf ("\n");
```

```
}
```

```
}
```

```
for (k=0; k<r1; k++)
```

```
 for (j=0; j<c1; j++)
```

```
 for (i=0; i<r1; i++)
```

```
 c[i][j] = a[i][k] + b[i][k];
```

```
 printf ("%d", c[i][j]);
```

```
 printf ("\n");
```

```
}
```

```
}
```

```
}
```

Transpose of a matrix:  
Input:  $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$   
Output:  $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$

(a) main function

(b) transpose function

(c) addition function

(d) multiplication function

(e) division function

(f) subtraction function

(g) multiplication function

(h) division function

(i) subtraction function

(j) multiplication function

(k) division function

(l) subtraction function

(m) multiplication function

(n) division function

(o) subtraction function

(p) multiplication function

(q) division function

(r) subtraction function

(s) multiplication function

(t) division function

(u) subtraction function

(v) multiplication function

(w) division function

(x) subtraction function

Pass 2-D array as an argument to a function :-

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

```
{ int display(int a[10][10]); int r, c, i, j; }
int main()
{
 int n, a[10][10], i, j;
 for scanf("n%d", &n);
 for (i=0; i<n; i++)
 scanf("%d", &a[i][0]);
 for (j=0; j<n; j++)
 printf("%d", a[0][j]);
}
```

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

```
{ int display(int a[10][10], int r, int c);
 int i, j, a[10][10], r, c;
 scanf("%d %d", &r, &c);
 for (i=0; i<r; i++)
 {
 for (j=0; j<c; j++)
 scanf("%d", &a[i][j]);
 }
 display(a, r, c);
}
```

```
int display(int a[10][10], int r, int c)
{
 for (i=0; i<r; i++)
 {
 for (j=0; j<c; j++)
 printf("%d", a[i][j]);
 }
}
```

if r, c are declared globally  
and scan in main,

no need to pass r, c to func.

→ Print sum of diagonal

```
#include <stdio.h>
void main()
{
 int i, j, s = 0;
 int a[10][10];
 scanf("%d", &i);
 for (i = 0; i < i; i++)
 for (j = 0; j < i; j++)
 if (i == j)
 s = s + a[i][j];
 printf("%d", s);
}
```

Program to implement addition of two matrices

```
#include <stdio.h>
void main()
{
 int add (int a[10], int b[10], int c[10]);
 int read (int a[10], int m, int n);
 scanf ("%d %d", &m, &n);
 read (a, m, n);
 for (i = 0; i < m; i++)
 for (j = 0; j < n; j++)
 a[i][j] = 0;
 read (b, n, n);
 for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
 b[i][j] = 0;
 if ((m == n) && (n == n))
 add (a, b, c);
 else
 printf ("Addition is not possible");
 int read (int a[10], int m, int n);
 for (i = 0; i < m; i++)
 for (j = 0; j < n; j++)
 scanf ("%d", &a[i][j]);
 for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
 scanf ("%d", &b[i][j]);
 int add (int a[10], int b[10], int c[10])
 {
 int i, j;
 for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
 c[i][j] = a[i][j] + b[i][j];
 printf ("\n");
 }
}
```

Print sum of diagonal elements.

#include <stdio.h>

void main()

```
{ int tri[10][10], i, j, s=0; for (i=0; i<10; i++) for (j=0; j<10; j++) tri[i][j] = (i+j)*2+1; for (i=0; i<10; i++) s += tri[i][i]; printf ("%d", s); }
```

Output: 100

Sum of diagonal elements = 100

```

#include <csdio.h>
void main()
{
 int add (int c1[10], int c2[10], int r1, int r2, int c1, int c2);
 int a[10][10], b[10][10], i, j, r1, c1, r2, c2;
 scanf ("%d%d", &r1, &r2); scanf ("%d%d", &c1, &c2);
 for (i=0; i<r1; i++)
 {
 for (j=0; j<c1; j++)
 scanf ("%d", &a[i][j]);
 }
 for (i=0; i<r2; i++)
 {
 for (j=0; j<c2; j++)
 scanf ("%d", &b[i][j]);
 }
 if (c1==r2) && (c2==c1)
 add (a, b, r1, c1);
 else
 printf ("Addition is not possible");
 int add (int a[10][10], int b[10][10], int r, int c)
 {
 int i, j;
 for (i=0; i<r; i++)
 {
 for (j=0; j<c; j++)
 {
 c[i][j] = a[i][j] + b[i][j];
 printf ("%d\t", c[i][j]);
 }
 printf ("\n");
 }
 }
}

```

elements transpose for multiplication  
transposed elements  
(matrix b)  
(matrix a)

Write a program for search

```

#include <csdio.h>
void main()
{
 int search()
 int a[10][10];
 scanf ("%d", &r1);
 for (i=0; i<r1; i++)
 scanf ("%d", &a[i][0]);
 scanf ("%d", &r2);
 for (i=0; i<r2; i++)
 scanf ("%d", &a[0][i]);
 search();
}
int search()
{
 int i, j;
 for (i=0; i<r1; i++)
 {
 if (a[i][0] == search)
 printf ("%d", i);
 }
}

```

Write a program to create a user defined function for searching an element in an array.

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

```
void main()
```

```
{ int search(int a[], int n, int k);
```

```
int a[10], n, i;
```

```
scanf ("%d", &n);
```

```
for (i=0; i<n; i++)
```

```
scanf ("%d", &a[i]);
```

```
scanf ("%d", &k);
```

```
search(a, n, k);
```

```
{
```

```
int search(int a[], int n, int k)
```

```
int i;
```

```
for (i=0
```

```
if (m==1)
```

```
printf ("Element is found");
```

```
if (m==2)
```

```
printf ("Element is not found");
```

```
{ int search(int a[], int n, int k)
```

```
{
```

```
int i;
```

```
for (i=0; i<n; i++)
```

```
{
```

```
if (a[i]==k)
```

```
return m=1;
```

```
else
```

```
c=c+1;
```

```
{
```

```
if (c==5)
```

```
return m=2;
```

```
{
```

① To display upper triangular & lower triangles of a matrix. Given no. of rows & columns no. of elements in given matrix.

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

② To add row wise, column wise, diagonal sum and display.

③ To check whether the matrix is symmetric or not.

① #include<stdio.h>

```
void main()
```

```
{
```

```
int a[10][10], i, j, r, c;
```

```
for (i=0; i<r; i++)
```

```
{
```

```
for (j=0; j<c; j++)
```

```
{ if (i==j)
```

```
printf("x%d\t", a[i][j]);
```

```
}
```

```
printf("\n");
```

```
{
```

```
for (i=0; i<m; i++)
```

```
{
```

```
for (j=0; j<c; j++)
```

```
{
```

```
if (i>j)
```

```
printf("x%d\t", a[i][j]);
```

```
}
```

```
printf("\n");
```

```
{
```

```
.
```

② #include<stdio.h>

```
void main()
```

```
{
```

```
int a[10][10], i, j, r, c;
```

```
for (i=0; i<r; i++)
```

```
{
```

```
for (j=0; j<c; j++)
scanf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (j=0; j<c; j++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (j=0; j<c; j++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (j=0; j<c; j++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```
s=0;
```

```
for (i=0; i<c; i++)
{
```

```
s=s+a[i][j];
printf("%d",
```

```
{
```

```
for (i=0; i<c; i++)
{
```

```

for(j=0; j<c1; j++)
 scanf ("%d", &a[i][j]);
}

for (i=0; i<r1; i++)
{
 s=0;
 for (j=0; j<c1; j++)
 {
 s=s+a[i][j];
 printf ("%d", s);
 }
}

for (j=0; j<c1; j++)
{
 s=0;
 for (i=0; i<r1; i++)
 {
 s=s+a[i][j];
 printf ("%d", s);
 }
}

if (r1==c1)
{
 s=0;
 for (i=0; i<r1; i++)
 {
 if (i==j)
 s=s+a[i][j];
 }
}

for (i=r1-1; i>=0; i--)
{
 for (j=0; j<c1; j++)
 printf ("%d", a[i][j]);
}

```

(System statements) 2) system swap set

swap function is written using set

(function)

(3) #include <stdio.h>

void main()

{

int a[10][10], b[10][10], r1, c1, i, j, s1; // r1, c1 = no. of rows & columns

scanf ("%d %d", &r1, &c1);

for (i=0; i<r1; i++)

{

for (j=0; j<c1; j++)

scanf ("%d", &a[i][j]);

}

for (i=0; i<c1; i++)

{

for (j=0; j<r1; j++)

~~scanf~~

b[i][j] = a[j][i];

}

for (i=0; i<c1; i++)

{

for (c1

if (r1 == c1)

for (i=0; i<r1; i++)

{

for (j=0; j<c1; j++)

{

if (a[i][j] == b[i][j])

s = s + 1;

}

}

if (s == r1 \* c1)

printf ("The given matrix is symmetric matrix");

else

printf ("The given matrix is ~~not~~ symmetric matrix");

}

## Parameter passing Techniques:- Unit-5

- pass by value  
(call by values)

Pass by value:  
formal parameters

```

void swap(int a,int b)
{
 int c;
 c=a;
 a=b;
 b=c;
 printf("a=%d,b=%d",a,b);
}

```

Output:- a=5 b=3

after a=3 b=5

→ This effects only formal parameters.

Memory locations are diff. for actual & formal.

Return statement - returns only one value.

These drawbacks overcome by pass by reference

int a; memory allocated with name 'a'.

int \*p; storing address of 'a' in 'p': p=&a.

Single pointer :- \*p.

```

int a=2;
int *p;
p=&a;

```

Pointer also have its own address.

|      |      |                                                                    |
|------|------|--------------------------------------------------------------------|
| P    | a    | Since 'a' is int type<br>pointer & should also<br>be in same type. |
| 2000 | 2    |                                                                    |
| 4000 | 2000 |                                                                    |

We can print 'a' by calling 'a' on its address.

```

printf("%d", a); // direct
printf("%d", *(&a)); // indirect
printf("%d", &a); // for address (control string).

```

&-address ; \*-value at particular address;

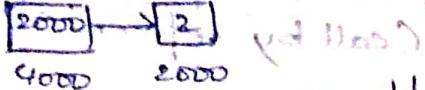
\*(&a)  
-\*(2000)  
(Value at 2000)

char ch;  
char \*p;  
p=&ch;

ch=55; p=&ch; printf("%d", \*p);

Pointer means "a link to a".

So what's stored after val 2000 is -



`printf("%u", p);` → address → unsigned int.

`printf("%u", &p);` → same value.

`printf("%d", *p);` → 2000

`*(2000)` → Value at 2000

`Value at 2000 = 2`

Pass by reference

`int *a=&a`

`void swap(int *a, int *b)`

{

`int c; if (a < b) { c=a; a=b; b=c; }`

`int a=2;`

`int *p=&a; (or) int a=2;`

`int *p; p=&a;`

`int *a=&a, a=2000`

`void swap(int *a, int *b)`

{ `a=3, b=4` } `if (a < b) { a=4, b=3; }`

`int a,b; (b=4)`

`c=*a; (a=4)`

`*a=*b; (a=4, b=3)`

`*b=c; (b=3)`

`printf("a=%d, b=%d\n", *a, *b);`

`3`

`(4000)`

`(9000)`

`→ This effects the original address, so it effects the actual parameters.`

`*(&p)`

→ 2000

`*(*(&p))`

→ 2

So a=2 → 2000

and b=4 → 4000

`void main()`

`void swap(int *a, *b);`

`int a,b; "bx" → 4000`

`scanf("%d%d", &a, &b);`

`swap(&a, &b);`

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

`3`

`(4000) *`

`(4000) *`

`(4000) to output`

Output:- after ~~a=5, b=3.~~  $a=5, b=3$ . ~~functionality~~  $a=5, b=3$ .

## Pre-processor directives :-

- Writing a code.
- pre processing  $\rightarrow$  # - hash / pound - preprocessing.
- compilation.
- linking.
- execution.

variable - varies  $\text{int } a=2;$

$\Rightarrow$  maintains preprocessor variables which are  $a=5;$   $a=6;$   $\dots$  no fixed

constant - fixed  $\#define \rightarrow$  preprocessing directives.  
in global declaration section

$\#define c=3.14\%$

Value remains fixed.

There are 3 preprocessor directives.

- Macros - Conditional compilation - file inclusion  
Macros:  $\#define$  identifier substitution text  
existing datatype  $\rightarrow$  headerfile section  
(or) variable.

Ex:-  $\#define a 20$

$\#define PI 3.14$

in preprocessing duration  
all identifiers in the main are replaced by substitution text.

we can replace existing datatypes using (typedef)

$\#define integer int$

$\#define cls clrscr();$

$\#define printf printf("simple")$

$\#define found$

$\#define cls clrscr() cls;$

$\#define AND &&$

$\#define double(a) a*2$

$\#define double(a) : (double)(a)*2$

$\#define double(a) : ((double)a)*2$

- Conditional compilation: - ~~• Standard stuff~~
- file inclusion: - `#include <stdio.h>` ? Header file to include files.  
~~• Standard stuff~~

## Conditional compilation:

```
#if def
#else
#endif
```

`#if` - to check whether preprocessing derivatives are defined or not.

```
#define E =
main()
#if E
a=2;
#else
b=3;
#endif
#endif.
```

`#ifndef` if not defined

```
#ifndef E
a=2;
#else
b=3;
#endif
#endif.
a=2;
#else
b=3;
#endif
#endif.
```

## Examples for parameter passing techniques:

```
#include<stdio.h>
void main()
{
 int add (int,int);
 int a,b,c,*p;
 scanf ("%d %d %d", &a, &b, &c);
 add (&a, &b, *p);
 printf ("%d", *p);
```

```

void add(int *a, int *b, int *p)
{
 *p = *a + *b;
}

```

To find largest of three numbers

```

#include <stdio.h>
void main()
{
 int a, b, c, d;
 int large (int, int, int, int);
 scanf ("%d %d %d", &a, &b, &c);
 large (&a, &b, &c, &d);
 printf ("%d", d);
}

int large (int *a, int *b, int *c, int *d)
{
 if (*a > *b && *a > *c)
 *d = *a;
 else if (*b > *a && *b > *c)
 *d = *b;
 else
 *d = *c;
}

```

To print sum and average of ~~of~~ numbers from ~~(int ("b" -> pro))~~ string ~~in~~ in ~~on~~.

```

#include <stdio.h>
void main()
{
 int m, n, k;
 int sum (int *, int *, int *);
 int avg (int *, int *, int *);
 scanf ("%d %d", &m, &n);
 sum (&m, &n, &k);
 avg (&m, &n, &k);
 printf ("%d %d", k, k);
}

```

```

int sum (int *m, int *n, int *k) {
 if (m < n)
 while (m <= n) {
 *k = *k + *m;
 m++;
 }
 int avg (int *m, int *n, int *l) {
 if (m < n)
 while (m <= n) {
 *l = *l + *m;
 m++;
 }
 }
}

```

(char) ~~char~~ shift for Hospital shift of  
 (char) ~~char~~ shift of  
 (char) ~~char~~ know

To print sum and average of numbers from m to n.

```

#include <stdio.h>
Void main() {
 int sum (int *, int *, int *, int *);
 int m, n, k, l;
 scanf ("%d %d", &m, &n);
 sum (&m, &n, &k, &l);
 printf ("sum = %d", k);
 printf ("avg = %d", k/l);
}

```

```

int sum (int *m, int *n, int *k, int *l) {
 if (m < n)
 while (m <= n) {
 *k = *k + *m;
 m++;
 }
 if (n < m)
 while (n <= m) {
 *l = *l + *n;
 n++;
 }
 *k = *k + *l;
 *l = *l + 1;
}

```

Area of a

```

#include <stdio.h>
#define P
#define P
#define P
#define P
Void main() {
 int s, a;
 int area (float r);
 float pi = 3.14159;
 s = area (r);
 printf ("%f", s);
}

```

3  
 int area (float r) {
 float A = pi \* r \* r;
 return A;
 }
}

To print  
 (area)

Area of a circle: (length of radius)  $\times$  (pi)  $\times$  (radius)<sup>2</sup>

#include <stdio.h>

```
#define PI 3.14
#define printf printf
#define scanf scanf

void main()
{
 int r, A;
 int area(int *, int *);
 float pi = PI;

 scanf("%d", &r);
 area(&r, &A);
 printf("%d", A);
}

int area(int *r, int *A)
{
 *A = PI * (*r) * (*r);
}
```

(values separate → solar leaflet) values separate when calculate  
(values separate because values are greater)

To print Hello world

Variables separate to socket + own socket  
Variables separate through friends separate intermodule (1)

Variables local {      Variables separate inter (2)  
Variables separate repeated (3)  
Variables local → Variables separate local (4)

Variable #include variables separate will use value  
Variables → variables → variables → variables  
Variables → variables → variables → variables  
Variables → variables → variables → variables

Variables → variables → variables → variables  
Variables → variables → variables → variables  
Variables → variables → variables → variables

Variables → variables → variables → variables  
Variables → variables → variables → variables  
Variables → variables → variables → variables

Variables → variables → variables → variables  
Variables → variables → variables → variables  
Variables → variables → variables → variables

Variables → variables → variables → variables  
Variables → variables → variables → variables  
Variables → variables → variables → variables

(?) bold syntax of variables → syntax leaflet

Storage Classes :- (Tells Initial value) (default value)

int a;

Storage class.

If not return, then it automatically has some storage  
if 'a' is not scanned and printed. Then it prints initial value - garbage value for zero.

The initial value depends on the storage class.

→ It tells initial value if the variable is not read.  
→ It also tells scope of a variable (upto which we can access the variable)

if main()

int a; scope is

'a' is only to main

(if declared globally)

(if declared locally)

→ It also tells the lifetime of a variable.

(upto which the variable is active)

→ It also tells storage area (location - memory or register)  
(where the value is stored)

depends on storage class

There are 4 types of storage classes

(1) Automatic storage class (default storage class)

(2) Static storage class. { local visibility}

(3) Register storage class.

(4) External storage class. - global visibility.

All the storage classes specify all the below :-  
- default value  
- scope  
- lifetime  
- storage area.

(1) Automatic storage class :- keyword is auto

int a; (or) auto int a;

(same)

initial value - Garbage value.

scope :- limited to specific block ({} )

for main()  
{  
int a=3;  
{  
int a=3;  
printf("%d",  
a);  
}  
for a=2;  
printf("%d",  
a);  
}  
3  
3  
printf("%d",  
a);  
3  
main()  
{  
inc();  
inc();  
inc();  
{  
inc();  
}  
auto in  
printf("%d",  
i++);  
"  
3 times"  
"3 times"  
"3 times"

(2) Static storage class

initial value :-

main()

static int  
printf("%d",

3

Scope :- only

for main()  
{

int();

inc();

inc();

3

int();

{

static

printf(

i++;

3

~~Ex:-~~ ~~int a=3;~~

~~if (a==1)~~ ~~printf("%d",a);~~ ~~else~~ ~~printf("%d",a);~~

Output :- 1 2 3

Scopes only to a specific block (blocks)

~~for (a=2; a<5; a++)~~ ~~printf("%d",a);~~ ~~for (a=2; a<5; a++)~~

Output :- 2 3 4 5

Scopes only to a specific block , const

3  
printf("%d",a); 3

3 ~~printf "%d" a;~~ ~~because words separated by commas~~ (A)

~~Ex:-~~ main()  
{  
 int a;  
 a=5;  
 a=a+5;  
 printf("%d",a);  
}  
Output :- 10

"a" is local variable

"a" is global variable

"a" is local variable

"a" is global variable

"a" is local variable

"a" is global variable

(2) Static storage class :- ~~initial value :- 0.~~ ~~key word is static~~ ~~it fixes the modified value~~

main()  
{  
 static int i;  
 printf("%d",i);

Output :- 0

fixed value is 0

Scopes only to a specific block.

~~Ex:-~~ main()  
{

int();

inc();

inc();

int();

"i" is modified and it fixed  
(static)

{  
 static int i=1;  
 printf("%d",i);  
 i++;

Output :- 1 2 3

(3) registered storage class keyword is register.  
initial value is garbage value.  
Accessing value from a register is faster than the memory.  
for iterations , accessing 'i' variable more no. of times , declare it as register.

(4) External storage class:- keyword is extern  
initial value is 0.

declaring variables as global.  
**extern int i;**  
main()  
{  
printf("%d", i);      Output :-  
}

main.c  
#include "origin.c"  
**Extern void func();**  
main(),  
{  
**int** i;  
if (i == 10)  
printf("%d", i);      we can access 'i' in another file  
func();      below
}

```
int i;
main()
{
 i=1;
 {
 fun();
 i=2;
 }
 {
 i=3; int state
 if ("bx") string
 }
}
origin.c
ext
#include "main.c"
extern int c;
void fun() separate state
{
 c=0; subset initial
 c=100; Unison
}
{
 {
 i=3; int state
 if ("bx") string
 }
}

```

Life  
Till the control remains within the block in which the variable is defined.

Scope  
local to the block in which the variable is defined.

Default Initial Value

Garbage value

Storage

Memory level

stomachic

## Life

Till the block in which the variable is defined.

## Scope

### Default initial value

Garbage value

Garbage value

Registers

Memory

Automatic

## Storage

Till the control remains within the block in which the variable is defined.

Till the control remains within the block in which the variable is defined.

Till the program's execution come to an end.

$i = d, s = b$  till

if ( $i < c$ ) then

Global  $i = d, s = b$  till

if ( $i < c$ ) then

(( $i < c$ ) then

(( $i < c$ ) then

register no to free space  
provided at lowest  
address to be taken up  
and move out to user program

level - libBm

Memory

sub1 demand +

get std no file - spooler -> ready

goal fix

variables reqd not to

disk ->

level - w1

level pmon

program -> SPP

ready -> wait

goal fix -> equal

initial condition

state to

External Memory

## Variable rules

- (1) Should not start with number, underscore.
- (2) No keywords, no special characters.
- (3) Combination of letters, digits, special characters.

## Type conversion: (Implicit & Explicit)

Converting one data type into another data type.

float f = 3.14159;  
 printf("%f", f);

internally converted.  
 output is

Implicit type conversion: internally converted, the system converts automatically.

Explicit: (type casting)

int a=2, b=3;

float

c=a/b

printf("%f", c);

int a=2, b=3;

float c;

c=(float)a/b;

printf("%f", c);

direct  
conversion  
possible  
without loss of  
precision  
and overflow  
problems.

1.000000

1.000000

1.500000

safe  
conversion  
possible  
without loss of  
precision  
and overflow  
problems.

1.500000

Square root of an integer

Decimal to binary.

Summation of sets of numbers.

Exchanging values of two numbers.

Low-level  
Assembly level

Middle-level  
C-language

High-level  
English language

1972: C-language, Dennis Ritchie

Previous B-language - only one data type

Loops: - entry loop - exit loop

Check condition  
at start

Ex: for, while

Check condition  
at last

Ex: do-while

Recursion  
iteration

## Problem Definition

Similarities, bln

## Problem Solving

Strategies to be

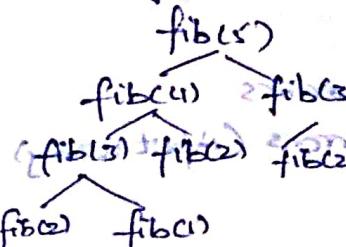
(1) Divide & Conquer  
divide into sub

Ex:- searching

(2) Dynamic Programming

for getting

- generally, for so  
we get optimal



(3) Backtracking

tracking best

finding sh

if a good so

and goes to

Ex:- rec

(4) Greedy Method

Problem Definition :- Input - may or may not be  
Output - one or more outputs

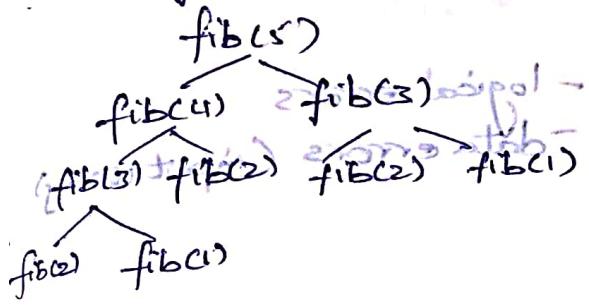
Similarities b/w problems :- common problems  
(Sorting, swapping etc.)

Problem solving Strategies :- (4) -  
1. strategies to be followed while solving any problem.

- (1) Divide & conquer strategy :-  
divide into subproblems & group all (conquer)  
Ex:- searching (Linear search) - Input -  
Binary search. (divide & conquer)  
sorted

(2) Dynamic programming: dynamically updating.  
for getting optimal solutions.

- generally, for some, we get multiple solutions  
we get optimal solution by recursion,  
memorisation



memorisation but here fib(3) is  
called two times, when fib(3)  
is called first, it stores the  
value & when it is called 2<sup>nd</sup>  
time, it doesn't calculate again.

(3) Backtracking :-

tracking best method ways in a no. of ways / methods  
finding shortest path.  
if a good solution is not found, then back tracking  
and goes to another method.

Ex:- recursion.

(4) Greedy Method :-

# Program Design and Implementation Issues

## 1. Programs and Algorithms :-

definitions of Program & Algorithm.

Properties (or) characteristics of an Algorithm :-

- finiteness - after finite no. of steps, the algorithm terminates.

- Definiteness - each and every step should be definite and unambiguous.

- Input - (0 or more inputs).

- Output - one or more outputs.

Effectiveness :- Effective logic should be there.

Properties of Flowchart :-

## Program Development steps :-

- 1) Errors detected in execution - logical errors (modif.)
- 2) Syntax errors - input (wrong)
- 3) Data errors (modif.)
- 4) Logic errors (modif.)

• prioritization :-  
bottom up form a single bottom test priority  
• testing (modif.)

priority based on the flow for i created loop n if  
bottom patterns at loop end

• iteration (modif.)

• bottom up (modif.)

Strings :- character array.

Initialisation  
for ( $i=0, i < 5, i++$ )  
`scanf("%c", &str[i]);`

`atoi()` :- converting a string into numeric

`printf("%d", atoi("123") + atoi("234"));`

`char str[] = "geek";`      `char *p = "geek"`

a & &a - same      p & &p - not same

a = "hello" - not valid      p = "India" - valid

a[0] = 'b' - valid      p[0] = 'k' - invalid

att - invalid      p++ - valid

Q1

strcpy - doesn't allocate new memory } both copies by strings data.

strdup - allocates new memory

1      (1001)0  
2      (2001)1  
1001    (2001)2

1 -> 2 -> 3  
1 + 2 + 3  
1 + 2 + 3

<4.8bits> abulwiff  
( ) Nish Bov

strncpy \* more than  
intgs \* less than

strncpy \* < strlent  
-      ; 221 = max  
; PE - SI = 20

; '0' = ds  
; max = strlent  
; DS = strlent

(strncpy("bs") fling  
(strncpy("bx") fling

(strncpy("bx") fling  
; DS = strlent

## Pointers

A variable is a named memory location.  
 A pointer is a variable that contains the memory location (address) of another variable.

Syntax:- type \* variable name; int \*ptr, "ox") float

Declaring a pointer variable:-

To declare ptr. as an integer pointer:- int \*ptr;

To " " " " character " " char \*ptr;

To " " " " float " " float \*ptr;

Accessing a variable through its pointer:-

Once we declare a pointer variable, we must point it.

Assign the address of the variable you want to point to the pointer

```
#include <stdio.h>
main()
{
 int a=10;
 int *ptr;
 ptr=&a;
 printf("%d", *ptr);
}
```

| Memory  | Value | ptr &x |
|---------|-------|--------|
| a(1001) | 5     |        |
| b(1003) | 6     |        |
| c(1005) | 1001  |        |

x | 10

```
#include <stdio.h>
void main()
{
 int num, *intptr;
 char ch, *cptr;
 float x, *floptr;
 num=123;
 x=12.34;
 ch='a';
 intptr=#
 cptr=&ch;
 floptr=&x;
 printf("%d", intptr);
 printf("%d", floptr);
 printf("%c", cptr);
}
```

## Pointer Arithmetic

Incrementing

to the next

Ex:- int

cout<<ptr

P=&P

P++

Decrementing

## Pointer Address

We can't

int P1;

P1+P2;

## Subtraction

int \*P

P=P-

x&y - y&x

P1-P2 →

\*P1 - \*P2

## Pointer Comparison

int \*P1, \*P2

P1=&a

P2=&b

P1 ==

When  
may be  
not fail

## Pointer Arithmetic

Incrementing :- increased by the size of datatype to the next location.

extra  $\text{int}^* \text{P};$   $\text{P} = \&\text{a};$   $1025$   
 statement  $\text{P} = \text{P} + 1;$   $1027$   
 address  $\text{P}++;$   $1029$

Decrementing :-  $\text{int}^* \text{P};$   $\text{P} = \&\text{a};$   $1027$   
 $\text{P}--;$   $1025$  (to the previous location)

Pointer Addition :- we can add a value to pointer variable  $\text{int}^* \text{P};$   $\text{P} = \text{P} + 3;$  (addition at  $\&\text{val}$ )

We can't add two pointer variables.  $\text{int}^* \text{P}_1, \& \text{P}_2; \text{P}_1 = 1024, \text{P}_2 = 1034$   $\text{P}_1 + \text{P}_2 = 1024 + 1034 = 2058$  - not sufficient memory

Subtraction :- subtract a value from pointer variable

$\text{int}^* \text{P};$   $\text{P} = \text{P} - 3;$   $\text{int}^* \text{P}_1, \& \text{P}_2; \text{P}_1 - \text{P}_2;$  no. of elements =  $\frac{\text{P}_1 - \text{P}_2}{\text{size}}$

char word[10] stores string in array.

strings tw[0] to tw[10] : ("9", "bx") string

$\text{P}_1 - \text{P}_2 \rightarrow$  pointer arithmetic b/w [0]

$*\text{P}_1 - *\text{P}_2 \rightarrow$  performing no value expression.

Pointer Comparisons: For same thing  $\leftarrow$  to using relational operators  $=, <, >, \neq$ .

$\text{int}^* \text{P}_1, \& \text{P}_2;$  towards sw.  $\text{P}_1 \leftarrow \text{i} + 10$   
 $\text{P}_1 = \&\text{a};$   $\text{P}_2 = \&\text{a};$   $\text{P}_1 == \text{P}_2 \rightarrow$  True.

points to

$[i]_0 \xrightarrow{10} \text{sw} \approx (i+10) \#$

When perform division  $(\frac{i}{(i+10)})$  pointers  $\frac{1024}{1025}$  but may lead to floating point also, so division is not allowed

```

main()
{
 int a=5, *ptr;
 ptr = value(a);
 printf ("%d", *ptr);
}
int * value(int x)
{
 static int p=x*2;
 return &p;
}

```

returning address of function  $\rightarrow$  possible mistake

### Pointer to Pointer

int \*ptr1;  $\rightarrow$  ptr1 can store address of a pointer.  
 int \*\*ptr2;  $\rightarrow$  ptr2 can store address of a double pointer.  
 int \*\*\*ptr3;  $\rightarrow$  ptr3 can store address of a pointer.

### Pointers to Array $\rightarrow$ value of function

int a[5] = {1,2,3,4,5};  $\rightarrow$  a is an array of 5 integers.  
 int \*p=a;  $\rightarrow$  p is a pointer to an array.  
 printf ("%d", \*p);  $\rightarrow$  prints the array, by increm. index.  
 printf ("%d", a[i]);  $\rightarrow$  will also print elements.  
 $i[a]$   $\rightarrow$  address of all elements.  
 $a+i$   $\rightarrow$  prints value of element.  
 $*(*a+i)$   $\rightarrow$  prints value of element.

$\langle 3 \rangle == \text{print}(a[0])$   $\rightarrow$  print value of a[0] only.

$a++$   $\rightarrow$  error : we cannot change base address of array.

$*(*a+i)$  is same as  $a[i]$

Two dimensional  $\rightarrow$   $(*(a+i)+j)$   $\rightarrow$  address of  $a[i][j]$ .  
 i)  $a[i][j]$  or  $*(*a+i)+j$   $\rightarrow$  value of  $a[i][j]$ .

if static is not placed  
it takes a position storage  
class - otherwise address  
releases (expires) after  
coming out of the function,  
then, it displays garbage  
value

Array of pointers  
 datatype \* (ptr1, ptr2, ...)  
 int \* p1, \* p2;  
 float b1, b2;  
 datatype \* arr;

### Pointer and char

char \* str;  
 str = "Hello";

printf ("%s", str);

Table of strings

char s1[10], s

(or) char str[]  
 if represent

Void pointer :-

datatype.

void \*p;

datatype

can store

of any

datatype.

whether, handle

a void poi

the pointer

explicit :-

int i=9;

int \*p;

float \*fp;

void \*ptr;

p=fp;

fp=&i;

ptr=p;

ptr=f;

ptr=&i;

## Array of pointers

int \*P1, \*P2, \*P3, \*P4; int a,b,c,d; int z[4];  
 datatype \*array-name [size];

## Pointers and character strings:-

char \*str; (v) str = "Hello"; printf ("%s", str);

char str; (w.r.t. memory) char \*str [10]; str = "Hello"; (x) str[5] = 'Y'

Table of strings: List of strings. → array of strings

char s1[10], s2[10], ..., s10[10];

(or) char str[257][10]; → Table of strings b/w

if represented by pointer :- char \*S[257];

Void pointer :- A generic pointer that can point to any datatype.

int \*p → can store address of int-variable only.

void \*p; no typecasting required

can store any address & no typecast required

of any variable.

a void pointer can be assigned to any type of the pointer without performing any explicit typecasting.

explicit :- float b;  
              int a=(int)b;

int i=9;

int \*p;

float \*fp;

void \*ptr;

p=fp; — incorrect

fp=&i; — incorrect

ptr=p; — correct

ptr=fp; — correct

ptr=&i — correct

(sec'd) if  
(o matter)  
int main()  
{  
    int a=7;  
    float b=7.6;  
    void \*p;  
    p=&a;  
    printf ("%d", \*(int \*)p);  
    p=&b;  
    printf ("%f", \*(float \*)p));  
(at print 3rd output was  
    : d=7  
    : f=7.6)

(at print 3rd output was  
    : d=7  
    : f=7.6)

id=7  
id=7.6

## Pointer to Functions:-

void \*pf(int) → a function  
that returns a void pointer.  
([ptr] is more generic than pointer)

main()  
{

void (\*pf)(int);

Pf = &somefunction (or) (ptr vars)

Pf = somefunction(); (all the vars)

If (5) for (\*pf)(5);

{} (appears to work, points to old & points to old)

[0][1][2] ... [n-1][2] vars

Void \*display(int argc); [void \*pf(int)] b (or)

[ptr] vars → returning pf = display.

## Calling multiple functions using pointer variable is as follows

main() create vars & pf for

{ pf = add or sub; } function pointer of array type.

void (\*ptr)[]. this array is holding addresses of functions.

void (\*fun\_ptr\_arr[])(int, int) = {add, subtract, multiply};

to get output for fun\_ptr\_arr[0], fun\_ptr\_arr[1], fun\_ptr\_arr[2]

choice :- add, subtract, multiplying int

if (ch > 2)

return 0;

(\*fun\_ptr\_arr[ch])(a, b);

return 0;

}. (r = a, t = b)

void add(int a, int b);

{ q = a + b

Print a+b; } thing

{ q = a - b

void subtract(int a, int b);

{ q = a - b

Print a-b; } thing

void multiply (int a, int b);

{ q = a \* b

Print a \* b; }

## Writing to work

void \*pf(int) → a function  
that returns a void pointer.

void somefunction(int, a);  
(represents)

(\*) (ptr vars)

: "all the" = r1

(r1, r2, r3) thing

{} (old, old)

[0][1][2] vars

## Dynamic memory

allocating

array is co size of an

Sometimes wasted.

There are malloc()

static,

Dynamic

malloc():-

stands for

allocates

it returns

Syntax:-

stdlib.h

main()

{ .(2)

int n, i;

scanf()

ptr = (c)

if (ptr)

{

printf()

for(i)

{

scanf()

sum

{

printf()

}

## Dynamic memory allocation

allocating memory at runtime  
 array is collection of fixed no. of values. Once the size of an array is declared, you can't change it.  
 Sometimes, it may be insufficient or may become wasted.

There are 4 functions of stdlib.h header file.

malloc(), calloc(), realloc(), free().

static memory all. → used in arrays.

dynamic → used in linked lists.

### malloc():

stands for memory allocation.

allocates single block of requested memory.

it returns NULL if memory is not sufficient.

Syntax:  $\text{ptr} = (\text{cast-type}^*) \text{malloc}(\text{byte-size});$

returns a void type.

to specify the type of ptr  
 cast-type is required - to convert.

stdlib.h

main() {  
 malloc from below滋生的  
 } // or when it reaches here freed

int n, i, \*ptr, s=0;

scanf("%d", &n); calloc(p = n, sizeof(int));

ptr = (int\*) malloc(n \* sizeof(int));

if (ptr == NULL)

{ to allocate memory.

printf("Unable to allocate memory.\n"); exit(0);

printf("Elements");

- for(i=0; i<n; i++)

{

scanf("%d", ptr+i);

sum = sum + \*(ptr+i);

}

printf("%d", sum);

free(ptr);

→ if here we print again, it prints because the free function releases the memory only after it comes out of main function.

if the compiler executes memory, later it releases memory automatically Ex:- functions.

For malloc & calloc it is not released automatically, so free() function is used.

## calloc() function :-

"calloc" stands for contiguous allocation. It allocates multiple block of requested memory. If memory is not sufficient, it returns NULL.

Syntax:- `ptr = (cast type *) calloc (number, byte size);`

Ex:- #include <stdlib.h> // for malloc, free  
#include <stdio.h>

main() { int n, i, \*ptr, sum=0; // n = no. of elements, i = index, sum = sum of elements }

int n; // no. of elements  
scanf("%d", &n); // n = no. of elements

ptr = (int \*) calloc(n, sizeof(int)); // n = no. of elements, sizeof(int) = size of one integer

for (i=0; i<n; ++i) // i = index  
{

scanf("%d", &ptr[i]); // n = no. of elements, &ptr[i] = address of i-th element

sum += \*(ptr+i); // sum = sum + value at address of i-th element

printf("%d", sum); // sum = sum of all elements

free(ptr); // ptr = address of first element

} // i = index of last element

if no argument is given

Free() :-  
allocates memory to pointers  
Dynamically allocated memory malloc

doesn't free automatically so use free().

realloc() :- Syntax:- `ptr = realloc(ptr, new size);`

Ex:- `ptr = (int *) malloc (n * sizeof(int));`

realloc(ptr, n+1 \* sizeof(int));

against we have to enter new data.

("string") thing

(char n[10], o=1) of

((int\*) "bx") tree

((int\*) "y + m = m")

((m\* "bx") thing

((int\*) "y + m = m")

## Command Line Argument

command line arguments  
program when it runs

C-C-A are passed  
to main function

gcc syntax: int main(int argc, char \*argv[])

Command Line Arguments: ~~Invokes a program with command line arguments which are passed to the program when it is invoked.~~

C-LA are passed to the main method.  
Before we go into code, let's see how to pass arguments to the compiler.

gcc ex.c  $\rightarrow$  /a [Input arguments]  $\rightarrow$  /a [Output arguments]

full syntax: int main(int argc, char\* argv[ ]) { }

argc means argument count of character array  
argv means argument character array

int main( int argc, char \*argv[] ) { }  $\rightarrow$  int argc  $\rightarrow$  count of character array  
char \*argv[]  $\rightarrow$  argument character array

int main( int argc, char \*argv[] ) { }  $\rightarrow$  int argc  $\rightarrow$  count of character array  
char \*argv[]  $\rightarrow$  argument character array

for( i=0; i<argc; i++ ) { }  $\rightarrow$  i  $\rightarrow$  index of character array  
printf( " %s ", argv[i] );  $\rightarrow$  argv[i]  $\rightarrow$  character array

(if no argument is supplied, argc will be 1)

Ex. C:/documents/C programs> ./a red green blue.

- argv[0] == ex.c

argv[1] == red

argv[2] == green

argv[3] == blue.

address suffices

{function}

do

→ argument character string

These are also stored as strings

→ argument character string

exit status do relation to the string red green blue

breakable

→ argument character string

{function}

→ argument character string

{function}

{function}

breakable

## Initialisation

(1) struct name  
 {  
 char name;  
 int height;  
 int weight;  
 } n1 = { "Rahul", 180, 70 };

n2 = { "Amit", 175, 65 };  
 n3 = { "Shivam", 170, 60 };

(2) struct name;

(3) struct name;

    n1.name = "Rahul";

    n1.height = 180;

    n1.weight = 70;

Accessing structure variable

operator < ->

"struct variable"

    n1.name;

If a pointer

struct \*ptr;

(or)      ptr = &n1;

ptr->name

for access

int s1, s2, s3, s4;

Struct record

Ex:- #include < stdio.h >

struct student\_record

{ char name[100];

    int id;

    float marks;

};

Struct student\_record

s1, s2, s3, s4;

Structures :- is a derived datatype. Collection of different datatypes. Heterogeneous collection. Ex:- Student data :- name, rollno., branch, section, address.

A single struct would store the data for one object.

Declaration of structures :- struct structure-name tag.

|                                       |                   |
|---------------------------------------|-------------------|
| fundamental :- int, char, float       | Datatype member1; |
| user-defined :- typedef               | Datatype member2; |
| derived :- arrays, structures, unions | Datatype member3; |
| empty :- void;                        |                   |

Ex:- struct rectangular\_prism  
 {  
 int height;  
 int width;  
 int length;  
 };

Does not reserve space (memory)

To reserve memory  
 write as below

```
struct mystruct
{
 int height;
 int width;
 int length;
};
```

→ structure variable.  
 mystruct;

To access structure members :-      obj

struct rectangular\_prism obj; [height    width    length]

We can create any no. of variables (or) objects to the same structure.

struct student\_record

```
{

 int ID;

 char name[100];

};
```

(or) :

{ s1, s2, s3, s4; }

struct student\_record s1, s2, s3, s4;

## Initialization of structure objects (structures)

(1) struct names

{ char name[100];

int height;

int weight;

} n1 = { "Tom", 180, 65 };

n2 = { "George", 170, 68 };

n3 = { "Bob", 186, 72 };

(2) struct names n1 = { "Mary", 170, 55 };

(3) struct names n1, n2;

n1.name = "Jane";

n1.height = 160;

n1.weight = 50;

n2.name = "Jane";

n2.height = 160;

n2.weight = 50;

Accessing struct members :- using structure member operator (dot ".")

"struct variable/object.member";

n1.name; id ob student

If a pointer to the struct is declared.

struct student \*ptr = &s1;

(or) \*ptr = &mystruct;

for accessing :-

ptr->id; id (\*myptr).letter;

(or) (\*ptr).id or (\*myptr).letter;

Struct rectangle - P

#include <stdio.h>

struct student

{

char name[50];

int roll; id

float marks; float

} (S)

int main()

struct student s;

printf("Enter information: ");

```

gets(s.name);
scanf("%d", &s.roll);
scanf("%f", &s.marks);
printf("%s", s.name);
printf("Roll number : %d\n", s.roll);
printf("Marks : %.1f\n", s.marks);
}

```

Nested Structure :- Structure within another another

structure is called nested structure.

By using it, complex data types are created as  
student record - rollno., address,

can be nested in two ways → (1) By separate structure.  
(2) By embedded structure.

Separate structure: -

```

struct Date
{
 int dd;
 int mm;
 int yyyy;
};

struct student_record
{
 int id;
 char name[20];
 struct Date dob;
};

```

In this way, we can use Date structure  
in many structures.

Here, we create two structures, but the dependent  
structure should be used inside the main structure  
as a member.

Embedded structure: - It enables us to declare the  
structure

```

struct student_record
{
 int id;
 struct Date
 {
 int dd;
 int mm;
 int yyyy;
 } dob;
};

```

To access date  
of student, write  
s1.dob.dd.

Initialisation :-  
members are  
initialised as

So, we include  
= Struct

User defined data  
for previously

Ex:- typed  
length of  
can now!  
Same wa  
length  
length  
in structure

isted s1 =  
254d 2

Enumeration  
using the l

- The n  
consta

of a program

choice of subunit "top"  
tribute funds

: (old man works)

struct Date dob; (Her tri  
cannot be done on top

(2) E

: 3 tribute funds → (1) Name tri  
(The maintenance cost) "Hiring"

Initialization :- student record  $st = \{ 99, "A. Anonymous", \{ 1, 4, 2003 \}, (23, 23) \}$

so:- #include <stdio.h>  
= struct student

User defined datatypes (typedef) :- to create synonyms for previously defined datatype names.

Ex:- `typedef int length;`  
length acts like a user defined datatype.  
can now be used in declarations in exactly the same way that the datatype int can be used.

length a[i];  
length numbers[10];

in structures:

```
typedef struct {
 int label;
 char letter;
 char name[20];
} first;
```

`struct st1 = { 1, 'A', "John" };` some-name; The "alias" is some-name  
`struct st2 = (struct) some_name my_struct;` → create a structure variable.

Enumeration :- is a user-defined datatype. It is defined using the keyword enum and the syntax is:-

```
enum tag-name (name-0, ..., name-n);
```

enum colours {red, yellow, green};

The names in the braces are assigned symbolic constants that integers from 0 to n.

{0} enum vars  
{red, yellow, green} int  
{red, green, blue} int  
{red, blue} int  
{red, green, blue} int

Array of Structures :- collection of multiple structure variables where each variable contains information about different entities.

for 5 students data  $\rightarrow$  5 structure variables  
for 60 students  $\rightarrow$  60 struct.  $\rightarrow$  but same structure.  
( $s_1, s_2, \dots, s_{60}$ ) variables.

instead of declaring 60 diff. variables for same structure, we can represent in array.

Once a structure is defined,

struct student class[50];

struct student  $s_1, s_2, s_3, s_4, \dots, s_{20}$ ; class[50]  $\rightarrow$  class ref.

The individual members can be accessed as:

$s_1$ .name  $\rightarrow$  still access its own

class[50].name  $\rightarrow$  still access its own

class[50].rollnumber  $\rightarrow$  still access its own

class[50].marks  $\rightarrow$  still access its own

struct student  $s[20]$ ;  $\rightarrow$  idea

```
for(i=0; i<20; i++) {
 scanf(" %d", &s[i].id);
 scanf(" %s", &s[i].name);
}
```

struct employee

sizeof(emp) = 4 + 5 + 4 = 13 bytes.

sizeof(emp[2]) = 26 bytes.

int id;

char name[5];

float salary;

};  $\rightarrow$  2 structures with both marks browser and price

struct employee emp[2];  $\rightarrow$  2 structures with both marks

(1-structure, - - - , 2-structure)  $\rightarrow$  2 structures with both marks

Array within structures:

struct student {

int id;

char name[20];

int roll\_number;

int marks[3];

char dob[10];

} a1, a2, a3;

```
struct student
{
 int roll;
 char name[20];
 int marks[3];
 char dob[10];
};

for (i=0; i<60; i++)
{
 scanf(" %d", &s[i].roll);
 scanf(" %s", &s[i].name);
 scanf(" %d", &s[i].marks[0]);
 scanf(" %d", &s[i].marks[1]);
 scanf(" %d", &s[i].marks[2]);
 scanf(" %s", &s[i].dob);
}

for (j=0; j<3; j++)
{
 printf("%d", s[0].marks[j]);
}
```

## Structures and Functions

- A structure can be passed to main function or not
- Structures can be passed to functions
- Structure definition can be passed to functions only
- It won't be passed if it's not defined
- Struct student {  
 int roll;  
 char name[20];  
};

## Passing structure

- passing each element

struct student

```
{
 int roll;
 char name[25];
 int marks[3];
};
```

It's a ~~structure~~ a ~~series~~ of memory ~~under~~ ~~array~~ within ~~structures~~ ~~array~~ ~~of structures~~ ~~structures~~ ~~array~~.

```
for (i=0; i<60; i++)
 scanf ("%d", &s[i].roll);
 scanf ("%s", s[i].name);
 for (j=0; j<3; j++)
 printf scanf ("%d", &s[i].marks[j]);
};
```

### Structures and Functions

- A structure can be passed to any function from main function or any subfunction.
- Structures can be returned from functions.
- Structure definition will be available within the function only.
- It won't be available to other functions unless passed.

```
struct student
{
 int -
 char -
};
```

} Declaring

struct student s;

defining.

's' should not be global.  
because if global, no need to pass.

Passing structure to a function -  
→ passing each item, whole structure, address.

## Structures and pointers

#include <stdio.h>  
struct person {

    char name[20];

    int age;

    float height;

};

char name[20];

int age;

float height;

};

→ Write an example program to create a structure with members of pointer type.

(+1; 00>1; 0>) ↴

((char, 0) & 2, "bx") frs2

(name, 0) & ("2x") frs2

(+1; 2>1; 0>) ↴

height

((float, 0) & 2, "bx") frs2

. {

## without base address

without base address has attribute A  
without base address has no information  
without base address has no memory +  
and with address addition and no memory attribute +  
the next

without address with address add base to }  
base address }  
+ base address

structures frs2

{

- int

- float

{

; 2 structures frs2

- primitive

pointer }

↳ data add from base '2'

↳ base can be added for access

- members at structure passed

variable members start in memory block +

```
#include <stdio.h>
void main()
{
 int
 scanf ("%d", &n);
 for
```

FUNCTIONS:- A function is a self-contained block of statements that perform a coherent task of some kind.

If a C program contains only one function, it must be `main()`.

→ Any function can be called from any other function. Even `main()` can be called from other functions.

```
#include <stdio.h>
void message();
int main()
{
 message()
 return 0;
}
void message()
{
 printf(" --- ");
}
```

→ A function can be called another function, but a function cannot be defined in another function.

Use of functions:-

→ Avoids rewriting the same code over and over.

There is no restriction on the number of return statements that may be present in a function.

```
int func()
```

```
{
```

```
int n;
```

```
scanf("%d", &n);
```

```
if (n >= 10 & & n <= 90)
```

```
return (n);
```

```
else
```

```
return (n+32);
```

```
}
```

exhibit a short life

(return value)

for

(return value) from

return

~~#include <stdio.h>~~ is a keyword in C language

~~void display(int);~~ is a user-defined identifier

```
int main()
```

is known as main function which contains a block of code

```
int i = 20;
```

```
display(i);
```

return 0; sets the exit status of program to 0

{ creates a new stack frame and uses (main) as its identifier

```
void display(int j)
```

```
{
```

```
int k = 35;
```

```
printf("%d\n", j);
```

```
printf("%d\n", k);
```

```
}
```

exhibit a short life

(display) from

(display) from

(display) from

(display) from

f(5) =

fib(4) + f(3)

0 1 1 2 ③ → (f(5) = f(3) + f(2)) fib

f(3) + f(2) + f(2) + f(1)

(fibonacci)

f(2) + f(1) + 1 + 1 + 0.

(fibonacci)

0

→ fib(4) + fib(3) (fibonacci) and uses (fibonacci)

fibonacci = 3rd fibo + 2nd fibo + 1st fibo

f(5) =

f(4) + f(3)

→ (fibonacci) to 0

f(3) + f(2)

f(2) + f(1)

f(2) + f(1)

1 + 0

0, 1, 1, 2,

## OPERATORS :-

Arithmetic Operators :- +, -, \*, /, %.

Logical Operators - &, |, !, &!, |!, !!

Relational Operators - <, >, <=, >=, ==, !=.

Increment and Decrement Operators :- i++, ++i, i--, --i.

Short hand assignment operations :- operator = Vi+ = 2.

Conditional operator (Ternary operator) :- [ condition ]? True statement : False statement ;

bitwise operators:- &, |, ^, ~n, <<, >>, <<=, >>=.

comma operator :- ,

statement1, statement2

Data types :-

integer — int /\* 4 bytes \*/

character — char /\* 1 byte \*/

float — float — %f

double — double — %lf

Variables

problem exists

Dimension

fixed — file

sets —

variables —

variable sets —

Address

existing sets

(variables)

?

statements

?

sets

?

statements

?

existing set

(variables, variables, variable)

?

statements

?

## Data types

↓  
Primitive / Built-in  
data types

→ Integer - int  
→ char  
→ fractional - float  
→ double

↓  
Derived data  
types

→ Arrays

→ Structures

→ Unions

→ Pointers

User defined  
data types

→ enum - enumerated  
→ typedef

Control statements :-

Conditional/  
decision making  
statements

→ if - simple if  
- if else  
- nested if  
- else if ladder

→ switch.

if else syntax :-

if (condition)

{

statements;

}

else

{

statements;

}

for syntax :-

for( initialization, condition, updation)

{

statements ;

.

Iterative  
statements

→ for

→ while

→ do while

for  
while  
do while

Jump  
statements

→ break

→ continue

→ to go.

break

continue

Ex:-  
scanf(" %d"  
for(i=1; i<  
{  
printf("%d  
}

do while syntax :-  
initialization;  
do

{ statements;

updation count;

} while (condition);

switch:-

switch(choice)

{

case constant1:

statements;

case constant2:

statements;

default:

statements;

}

#include<std.h>

Void main()

{

int n; j; abc;

scanf("%d", &n);

for(i=1; i<=n; i++)

{

scanf("%d", &j);

printf("%d", j);

}

scanf("%d", &abc);

printf("%d", abc);

return 0;

}

Scans

Ex:- `scanf("%d", &n);  
for(i=1; i<=n; i++)  
{  
 printf("Hello World");  
}`

do while syntax :-  
 initialization;  
 do  
 {  
 statements;  
 updation counter;  
} while (condition);

`scanf("%d", &n);  
i=1;  
(i<=n) {  
 do  
 {  
 printf("Hello World");  
 i++;  
 } while (i<=n);  
}`

switch:-

switch(choice)  
 {  
 case constant1:  
 case p: statements;  
 break;  
 case constant2:  
 case z: statements;  
 break;  
 default: statements;  
 }

#include<stdio.h>  
 void main()  
 {  
 int n, a, b, c;  
 scanf("%d", &n);  
 for(i=1; i<=n; i++) (or) for(;;) (or) while(i<=n)  
 {  
 scanf("%d %d", &a, &b);  
 printf(" 1. addition\n"  
 " 2. subtraction\n"  
 " 3. multiplication\n"  
 " 4. product & division\n"  
 " 5. Modulo division -n ");  
 scanf("%d", &c);

"Hello!" printf : Hello

equal string not  
assists both  
in switch

`(A) if ("bab") true  
(B) if ("aa") true  
(C) false.`

for i=1 to 1000  
infinite loop  
for i=1 to infinity  
infinite loop

infinite loop  
for i=1 to 1000  
infinite loop

Switch (ch)

```
{
 case 1: printf ("%d", a+b);
 break;
 case 2: printf ("%d", a-b);
 break;
 case 3: printf ("%d", a*b);
 break;
 case 4: printf ("%d", a/b);
 break;
 case 5: printf ("%d", a%b);
 break;
 default: printf ("Invalid");
}
```

3.  
Case  
#include

for infinite loops:-

```
char choice;
while(1)
{
 scanf ("%d%d", &a, &b);
 scanf ("%c", &ch);
 switch(ch)
 {
 case 'i':
 break;
 default:
 if((a>0) && (b>0))
 printf ("do you want to continue
Y-yes N-no");
 else
 printf ("do you want to continue
Y-Yes N-No");
 scanf ("%c", &choice);
 if(choice == 'N')
 break;
 }
}
```

Output :-

2 3  
1  
do you want to continue  
Y-Yes  
N-No  
Y  
2 3  
2  
do you want to continue  
Y-Yes  
N-No  
Y  
2 3  
2  
G  
do you want to continue  
Y-Yes  
N-No

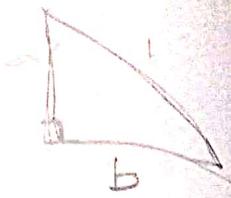
Outputs :-

2 3  
1  
5  
do you want to continue  
Y-Yes  
N-No  
Y  
2 3  
1  
5  
do you want to continue  
Y-Yes  
N-No

Swapping two numbers

```
void main()
{
 int a,b;
 int swap(int, int);
 swap (&a, &b);
}
int swap (int a, int b)
{
 int temp;
 temp = a;
 a = b;
 b = temp;
 return a+b;
}
```

Write a program using function that calculates the hypotenuse of a right-angled triangle.



```
#include <stdio.h>
#include <math.h>
void main()
{
 int hyp(int, int); int a, b;
 scanf("%d %d", &a, &b);
 printf("%d", hyp(a, b));
}
int hyp(int a, int b)
{
 return sqrt(a*a + b*b);
}
```

Write a function that accepts an integer b/w 1-12 to represent the month number and displays the corresponding (For month number=1, then display JANUARY)

```
#include <stdio.h>
void main()
{
 int month(int);
 int n;
 scanf("%d", &n);
 printf("%d", month(n));
}
int month(int n)
{
 if (n == 1)
 return JANUARY;
 else if (n == 2)
 return FEBRUARY;
```

22) Write a program to find leap year

```
#include <stdio.h>
void main()
{
 int leap(int);
 int n;
 scanf("%d", &n);
 printf("%d", leap(n));
}
int leap(int n)
{
 if (n % 400 == 0)
 return 1;
 else if (n % 100 == 0)
 return 0;
 else if (n % 4 == 0)
 return 1;
 else
 return 0;
}
```

Write a P

-sion.

```
#include <stdio.h>
void main()
{
 int revl;
```

23) Write a program to check whether the given number is leap year or not.

```
#include <stdio.h>
void main()
{
 int leap(int);
 int n;
 scanf("%d", &n);
 printf("%d", leap(n));
}

int leap(int n)
{
 if (n % 4 == 0)
 printf("The given year is a leap year");
 else if (n % 100 == 0)
 printf("The given year is not a leap year");
 else if (n % 400 == 0)
 printf("Yes");
 else
 printf("No");
}
```

Write a program to reverse a number using recursion.

```
#include <stdio.h>
void main()
{
 int rev(int);
 int n;
 rev(n);
}
```

```

int n; // takes integer input from user
scanf("%d", &n); // reads the value of n
printf("%d", rev(n));
}

int rev(int n)
{
 int r, s = 0;
 while (n > 0)
 {
 r = n % 10;
 n = n / 10;
 s = (s * 10) + r;
 }
 return s;
}

```

~~<delisted subject~~  
 (unnumbered)

~~(tri)~~ first  
 for

#include <stdio.h>

Write a program to compute  $F(x, y)$  where  
 $F(x, y) = F(x-y, y) + 1$  if  $y \leq x$  and  $F(x, y) = 0$  if  $x < y$ .

#include <stdio.h>

void main()

{

int Fun(int, int);

int

scanf("%d %d", &x, &y);

printf("%d", Fun(x, y));

}

int Fun(int x, int y);

{

if ( $x < y$ )

return 0;

if ( $y \leq x$ )

return  $F(x-y, y) + 1$ ;

~~<delisted subject~~  
 (unnumbered)

~~L(n) = L(n/2) + 1~~  
 if ( $n > 1$ )

~~return n;~~  
 else if ( $n = 1$ )

~~return 0;~~

#include <stdio.h>

Scan - arr

for (i=0; i<5;

    Scan[i];

    for (j=0; j<5;

        a[i][j] = a[i][j];

        a[i][j] = a[i][j];

~~<delisted subject~~  
 (unnumbered)

~~(tri)~~ from tri

Explain file inclusion and conditional compilation.

file inclusion :- to create a header file to include files into the program.

#include "headerfile.h".

conditional operation :- compilation :- To check whether the identifier is defined or not. by the Macros.

Ex:-

```
#include <stdio.h>
#include <math.h>
#define sqfp sqrt(p)
int main()
{
 int a, b;
 #ifdef sqfp
 {
 printf("%d", sqfp);
 }
 #else
 {
 printf("%d", sqrt(a));
 }
 #endif b
 #ifndef sqfp
 {
 printf("%d", sqrt(b));
 }
 #else
 {
 printf("%d", sqfp);
 }
 #endif
}
```

Write a program -

for number using

#include <stdio.h>

int fact (int);

int n, d;

scanf ("%d", &n);

d=fact (n);

printf ("%d", d);

}

int fact (int n);

{

if (n==1)

return 1;

else

return n\*fact (n-1);

}

Write a program using recursion

#include <stdio.h>

void main()

{

int gcd (int a, b);

scanf ("%d %d", &a, &b);

printf ("%d", gcd (a, b));

}

int gcd (int x, int y);

{

if (x > y)

x=x%y;

return gcd (x, y);

if (x == y)

return x;

else

return gcd (y, x);

}

to include

checks whether  
the Macros.  
exists  
in file

Write a program to find factorial of a given number using recursive functions.

```
#include <stdio.h>
void main()
{
 int fact (int);
 int n, d;
 scanf ("%d", &n);
 d=fact (n);
 printf ("%d", d);
}

int fact (int n)
{
 if (n==1)
 return 1;
 else
 return n*fact (n-1);
}
```

Write a program to find gcd of two numbers using recursive function.

```
#include <stdio.h>
void main()
{
 int gcd (int, int);
 int a, b;
 scanf ("%d %d", &a, &b);
 printf ("%d", gcd (a, b));
}

int gcd (int x, int y)
{
 if (x>y)
 x=x-y;
 if (x==0)
 return y;
 else
 return gcd (y, x);
}
```

find factorial of a given number using recursive functions.

( ) from biov

i=d, o=0, it is

((12,"bx")) from

((12,"bx",bx)) from

(main is userdefined

function. (d+d-1)

(d-1)

(0=d

((0,"bx")) from

{

{

exhibits should not

( ) from biov

fib(4) & fib(3) in tri

((2,"bx",bx)) from

((2,"bx",bx)) from

y=x;

{

(n tri) dif tri

10) 3473

i=d, o=0 3473

((d,0,"bx",bx)) from

((d,0,"bx",bx)) from

((d,0,"bx",bx)) from

else

(d+d-1)

(d=0 a,b

(0=d

((x,y,"bx",bx)) from

if (r==0)

return y;

else

return gcd (y,r);

}

```

fibonacci series (non-recursive)
#include <stdio.h>
Void main()
{
 int n, a=0, b=1
 Scanf ("%d", &n);
 printf ("%d %d", a, b);
 for (i=0; i<n; i++)
 {
 c=a+b;
 a=b;
 b=c;
 printf ("%d", c);
 }
}

```

at margin of stack  
pushes values

<+> sister & brother  
(inner loop)

((tri) left tri)

((tri) right tri)

((tri, tri)) frame

((b, b)) frame

((a, tri)) left tri

((b, tri)) right tri

recursive:

```

#include <cs>
void main()
{
 int n
 int fib();
 Scanf ("%d", &n);
 fib(n);
}

int fib()
{
 if (n==0)
 printf ("0");
 else if (n==1)
 printf ("1");
 else
 fib(n-1) + fib(n-2);
}

```

## recursive:-

#include <stdio.h>

void main()

{

int n;

int fib(int);

scanf ("%d", &n);

fib(n);

int fib(int n)

{

if (n==1)

printf ("%d", 0);

if (n==2)

printf ("%d", 1);

if (n>2)

printf ("%d", fib(n-1)+fib(n-2));

}

return 0;

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

};

## Algorithm:-

Algorithm means a procedure. It is a sequence of finite no. of steps to solve a particular problem.

### Properties of algorithm:-

(1) Finiteness :- An algorithm must always terminate after a finite no. of steps.

(2) Definiteness :- Each step of an algorithm is precisely defined.

(3) Input :- An algorithm may or may not require input.

(4) Output :- Each algorithm is expected to produce at least one result.

If some result is from the intermediate stage of the operation, then it is known as intermediate result, and the result obtained at the end of algorithm is known as end result.

Flowchart :- The flowchart is a diagram which visually presents the procedure to solve a particular problem.

A flowchart can be used for representing an algorithm.

### Advantages of flowchart:-

- Flowchart is an excellent way of communicating the logic of a program.
- Easy and efficient to analyse problem.
- It is easy to convert the flowchart into any programming language code.

### Advantages of algorithm:-

- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming languages so it is easy to understand for anyone even without programming knowledge.

→ Every step in sequence  
The algorithm types - (1) Sequence  
(2) Loop (Repetitive)  
(3) Decision (Branching)

(4) Parallel (a state depends on others)

To draw flowchart  
fused or  
basic symbols

rectangle  
oval

parallelogram  
diamond

rectangle  
diamond

rectangle  
diamond

rectangle  
diamond

rectangle  
diamond

rectangle  
diamond

Every step in an algorithm has its own logical sequence so it is easy to debug.

The algorithm and flowchart are classified into 3 types - (1) Sequence (2) Branching (Selection) (3) Loop (Repetition)

(1) Sequence:- In the sequence structure, statements are placed one after the other and the execution takes place starting from top to down.

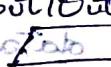
(2) Branching (Selection):- In this branch control, there is a condition based according to a condition or decision of either TRUE or FALSE is achieved.

If the condition is True, one of the branches is explored if the condition is false, the other alternative is explored. Generally the 'IF-THEN' is used to represent branch control.

(3) Loop (Repetition):- The loop or repetition allows a statement to be executed repeatedly based on a certain loop condition. Ex:- WHILE, FOR loops.

To draw a flowchart following standard symbols are

Symbol Name Symbol Function  
Terminal  Used to represent start and end of flowchart.

Input/Output  Used for any Input/Output operation. Indicates that the computer is to obtain data or output results.

Process  Indicates any type of internal operation inside the Processor/Memory.

Diamond 

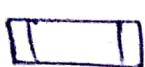
Used to represent the operation in which there are two/three alternatives.

Arrows Flowlines 

Shows direction of flow.

O

Allows the flowchart to connect from one page to other.



Predefined process steps are represented by a rectangle with a vertical line through it.

Data types in C language:- Data types are used to group.

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we use in our program. These datatypes have different storage capacities.

There are 2 different types of data types

(1) Primary data types:- These are fundamental data

types in C namely integer (int), floating point (float), character (char) and void type.

Integer type:- int, short int, long int, unsigned int,

unsigned short int, unsigned long int

Character type:- char, signed char, unsigned char.

Floating point type:- float, double, long double.

(2) Derived data types:- Derived data types are nothing but primary data types but a little twisted or grouped together like array, structure, union and pointer.

Data type determines the type of data a variable will hold. If a variable 'x' is declared as int, it means x can hold only integer values. Every variable which is used in the program must be declared as what datatype, it is.



## Operators :-

C supports a rich set of built-in operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

Operators are used in programs to manipulate data and variables. They usually form a part of statements.

C operators are classified into no. of categories:-

- (1) Arithmetic Operators.
- (2) Relational Operators.
- (3) Logical Operators.
- (4) Assignment Operators.
- (5) Increment and Decrement Operators.
- (6) Conditional Operators.
- (7) Bitwise Operators.
- (8) Special Operators.

### (1) Arithmetic Operators:-

+ → Addition or unary plus.

- → Subtraction or unary minus.

The modulo division operator % cannot be used on floating point data.

### (2) Relational Operators:-

It is an operator that compares two values.

Expressions that contain relational operators are called relational expressions.

Relational operators return true or false value depending on whether the conditional relationship between the two operands holds or not.

These are used to determine the relationships between the operands.

<, >, <=, >=

less than      greater than  
than      less than  
or equal      greater than  
or equal.

Equality operators :-    ==, !=

equal to      not equal to.

The equality operator  
on the both sides  
Value, otherwise

The not equals  
operator to op  
else it returns

### (3) Logical Operators

There are three  
logical operators  
Logical AND :-

two condition  
operations.

If expression  
operator is  
true then

Logical OR :-

two condition  
operators.

left side, right  
then the

Logical NOT :-

Single expression  
body of  
expression

| A | B |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |

Logical  
fashion

for 80%

For exa  
logical  
therefore

Sure the

tors. An operator to perform manipulate data categories:-

and  $\neq$ .

not be used on comparison operators values operators are

value, dependency between relationships between

to.

The equal-to operator ( $=$ ) returns true (1) if both operands on the both sides of the operator have the same value, otherwise it returns false (0). The not equal-to operator ( $\neq$ ) returns true (1) if the two operands do not have the same value, else it returns false (0).

### (3) Logical Operators:-

(a) AND (b) OR (c) NOT

There are three logical operators AND, OR, NOT.  
Logical AND :- It is used to simultaneously evaluate two conditions or expressions with relational operations. If expressions on both the sides of the logical operator is true then the whole expression is true. Logical OR :- It is used to simultaneously evaluate two conditions or expressions with relational operators. If one or both the expressions on the left side of the logical operator is true then the whole expression is true. Logical NOT :- The logical NOT operator takes a single expression and negates the value of the expression. It just reverses the value of the expression.

| A | B | $A \& B$ | $A \mid B$ | $\neg A$ |
|---|---|----------|------------|----------|
| 0 | 0 | 0        | 0          | 1        |
| 0 | 1 | 0        | 1          | 1        |
| 1 | 0 | 0        | 1          | 0        |
| 1 | 1 | 1        | 1          | 0        |

logical operator expressions operate in a shortcut fashion and stop the evaluation when it knows for sure what the final outcome would be. For example, in a logical expression involving logical AND, if the first operand is false then the result will be false. If the first operand is true and the second operand is not evaluated as it is for sure that the result will be false. Similarly, for

a logical expression involving logical OR if the first operand is true, then the second operand is not evaluated as it is for sure that the result will be true. If assert and (.) rotates strings then

(4) Assignment Operators: The assignment operator is responsible for assigning values to the variables.

(1) (2)

Assignment Operator

Variable operator = expression  
The 'operator' is known as Short hand assignment operator.

Simple assignment operator :-  $a = a + 1$ .

Shorthand assignment operator :-  $a += 1$

Advantages:-

- (1) what appears on the left-hand side need not be repeated and therefore it becomes easier to write.
- (2) More concise and easier to read.

(5) Increment and Decrement Operators:

Increment operator increases the value of operand by 1.

Decrement operator decreases the value of operand by 1.

Post increment or addition by -1

The operator is applied after an operand is fetched for computation.

Pre :-  $++x$ ,  $--x$ .

The operator is applied first before an operand is fetched for computation.

(6) Conditional operator:- Ternary operator

Symbol :- '?'

$exp1 ? exp2 : exp3$  gets true instead

$exp1$  is evaluated first. If it is non-zero (true) then  $exp2$  is evaluated and becomes the value of the expression. If  $exp1$  is false,  $exp3$  is evaluated and becomes the value of the expression.

(7) Bitwise operators

Used for manipulation

The bitwise operators can be integers and bits.

Bitwise AND : (1)

operand with second operand corresponding otherwise.

Bitwise OR :- (1)

operand with second operand the corresponding 0 otherwise.

Bitwise XOR :-

first operand its second corresponding

| A | B | A |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Bitwise NOT:

operation + bit of the element of

~ 101

Shift operator

To shift for left the left lost

If the first operand is not zero, result will be positive. If the operator is variables.

Assigned to  
assignment  
its assigned  
one out  
variables  
range 32.  
variables  
need not be  
er to write.

Bitwise AND  
operator by 1.  
operand by 1.

disfetched  
variables

3 operand is  
0  
1  
0  
1  
1  
1

for logical  
operator.

(true) then  
the value of the  
evaluated and  
the expression.

both ends

(a) Bitwise operators :- It takes full binary input. Used for manipulation of data at bit level. The bitwise operators except their two operands, to be integers and treat them as a sequence of bits.

Bitwise AND :- (l) It compares each bit of its first operand with the corresponding bit of its second operand. If both bits are 1, the corresponding bit in the result is 1 and 0 otherwise.

Bitwise OR :- (l) It compares each bit of its first operand with the corresponding bit of its second operand. If one or both bits are 1, the corresponding bit in the result is 1 and 0 otherwise.

Bitwise XOR :- (l) It compares each bit of its first operand with the corresponding bit of its second operand. If one of the bits is 1, the corresponding bit in the result is 1 and 0 otherwise.

| A | B | A & B | A   B | A ^ B | Same -><br>diff - 1 |
|---|---|-------|-------|-------|---------------------|
| 0 | 0 | 0     | 0     | 0     |                     |
| 0 | 1 | 0     | 1     | 1     |                     |
| 1 | 0 | 0     | 1     | 1     |                     |
| 1 | 1 | 1     | 1     | 0     |                     |

Bitwise NOT :- (Complement) It is a unary operation that performs logical negation on each bit of the operand. It actually produces 1's complement of the given binary value.

$$\sim 10101 = 01010.$$

Shift operator :- operand op num.

To shift bits either to the left or right. for left-shift:  $<<$  :- every bit in  $x$  is shifted to the left by one place. So, the MSB of  $x$  is lost and LSB of  $x$  is set to 0.

for right shift: ~~>>~~ >> - every bit in  $\text{BCD}$  is shifted to the right by one place. So, the LSB of  $\text{A}$  is lost. MSB if found is set to 0.

Comma Operator: In most target bus computers, the

for if  $\text{BCD}$  to find whose evenness  $\text{BCD} \cdot \text{BCD}$  something

if  $\text{BCD}$  is odd then branches out after branching

else if  $\text{BCD}$  is even then branches out after branching

or branch out in find evennesses

for if  $\text{BCD}$  to find whose evenness  $\text{BCD} \cdot \text{BCD}$  something

if  $\text{BCD}$  is odd then branches out after branching

else if  $\text{BCD}$  is even then branches out after branching

or branch out in find evennesses

for if  $\text{BCD}$  to find whose evenness  $\text{BCD} \cdot \text{BCD}$  something

if  $\text{BCD}$  is odd then branches out after branching

else if  $\text{BCD}$  is even then branches out after branching

or branch out in find evennesses

or branch out in find evennesses

1 =  $\text{BCD}$       0101 0110 0111 0101 0110

0      0      0      0      0

1      1      0      1      0

1      1      0      0      1

0      1      1      1      1

known as 2's RC (Binary form) - 1010 0110

the output is logical summing that contains

odd numbers also  $\text{BCD}$  branching out to find

even numbers below would result out to normal

01010 = 10101

the summing goes branching out to two

target are first left of results else find out

target is initial process  $\Rightarrow >>$  - find out if not

give to gain out, or initial process

## Control Statements

specify the flow order in which will be executed. They make decisions, to jump from one section

→ To control the execution

Control

### Conditional statements

#### Decision making

→ if - simple if

- if else - then

- nested if

- else if ladder

→ switch

→ if

(a) simple if - else

(conditions) If

else

case

statement block

group of statements

if the test-expression

will be executed

will be skipped

cases have already

(b) if - else -

else if

else statements

for two loop iterations

translates into for

if true out for two

if the condition

be executed

Control Statements: Control statements enable us to specify the flow of program control i.e., the order in which the instructions in a program must be executed. They make it possible to make decisions, to perform tasks repeatedly or to jump from one section of code to another.

→ To control the execution of the program.

### Control Statements

Conditional statements) Iterative/Repetitive)  
Decision making (Loop)

Jump statements

if - simple if      for  
- if else      while  
- nested if      do while  
- else if ladder  
switch (conditions) {  
(a) if  
    if (condition) {  
        statement block;  
    }  
    else {  
        statement block;  
    }

if (condition) {  
    statement block;  
}

Statement block may be a single statement or a group of statements.

If the test-expression is true, the statement-block will be executed, otherwise the statement-block will be skipped.

(b) if - else:-  
    if (condition) {  
        statement block 1;  
    }  
    else {  
        statement block 2;  
    }

If the condition is true, the statement block 1 will be executed. Otherwise, the statement block 2 will be executed.

(c) Nested if:- Syntax :- if (condition1) statement1  
if (condition2) statement2  
else statement3  
• If condition 1 is true, then it will execute statement1.  
• Then it will check condition2. If condition2 is true, then it will execute statement2.  
• Else it will execute statement3.  
• It is useful when we want to perform more than one operation.

↳ Placing if statement inside another if statement  
• Useful to check the condition inside a condition.  
(d) else if ladder:- Syntax :-  
Helps user to decide among multiple options.  
↳ bold transition  
↳ bold transition

(e) Switch:- Tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed & control goes out of switch.  
• If a case match is not found, the default statement is executed, and the control goes out of the switch.

↳ bold transition  
↳ bold transition

Syntax :- switch  
{  
    case value:  
        statements;  
    case value:  
        statements;  
    default:  
        statements;

↳ bold transition

(f) Iterative / Repetitive Statement  
• for :- A for loop is used to repeat a set of statements for a given number of times.

↳ bold transition  
↳ bold transition

Syntax :- for (initialisation; condition; update)  
    statements;  
• The initialisation part is executed first, then the condition is evaluated. If the condition is true, then the body part is executed. After that, the update part is executed, and then the condition is checked again.

Again the condition is checked. This process continues until the condition is false.

(g) while:-

• While statement good code to convert into while

Syntax:- switch (choices)

```

 {
 case value 1: statement 1; break;
 case value 2: statement 2; break;
 ...
 default: statements;
 }

```

Statement 1, Statement 2, ... are called cases  
 break; is used to exit from switch block.  
 default: statements; is used if none of the cases matches.

### Iterative / Repetitive / Loop:-

- (1) for :- A for loop enables a particular set of statements to be executed repeatedly until a condition is false.

Syntax:- for (initialisation, condition, updation)

The initialisation statement is executed only once. Then, the condition is evaluated; if the condition is evaluated to true, the statements inside the body of 'for' loop are executed and the updation is done. Again the condition is evaluated.

If the condition is evaluated to false, the for loop is terminated.

The process of for loop goes on until the condition is false. When the condition is false, the loop terminates.

(2) while :- Syntax: (Initialisation; Test; updation)

still maintains prior statements; if condition is true, the updation is done to condition and the condition is checked for equality. If equal, it enters the loop again.

Initialisation takes place.  
The while loop evaluates the condition inside the parenthesis ( ).

If the condition is true, the statements inside the loop are executed and updation takes place. And the condition is evaluated again. This process goes on until the condition is evaluated to false.

If the condition is false, the loop terminates.

(3) do while :- Syntax :- (Initialisation);  
do {  
    statements;  
    Updation;  
} while (condition);

The statements inside do while is executed once and updation takes place. Only then, the condition is evaluated. If the condition is true, the statements inside the loop, updation is executed and condition is evaluated.

This process goes on until the test condition becomes false and the loop terminates.

Jump statements :- Inside

(1) Break :- Used in loops and switch case.

Used to bring the control out of the loop.

The break statement breaks the loop one by one i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops.

Syntax:- break;

(2) Continue :- Instead of forcing termination like break, it forces the next iteration of the loop to take place by skipping any code in between.

Syntax:- continue;

(3) goto :- It transfers the statement to a label defined. It can transfer same blocks and you want to.

Defining a label  
Transferring control

### FUNCTIONS :-

A function is a block of code which performs some task at least one time.

There are two types of functions:-

(1) Pre-defined some functions are built-in hence they are pre-defined.

Ex:- scanf(),  
      strcpy(),  
      strlen(),  
      etc.

to use these, include header files.

(2) User Defined

defined by program.

Parts of function

(1) Function

(2) Function

(3) Function

(4) Function

It tells what type of return type.

Syntax:-

inside takes again, is

(3) goto: It transfers the program's control from one statement to another statement where label is defined.

It can transfer the program's control within the same block and there must be a label, where you want to transfer the control.

Defining a label :- labelname: int a = 10;  
Transferring control :- goto labelname;

### FUNCTIONS :-

A function is a group of statements that together perform a task. Every C program has at least one function, which is main().

There are two types of functions :-

(1) Pre defined functions :- The system provides some functions and are stored in the library. Hence they are also called Library functions.  
Ex:- scanf(), printf(), ~~< stdio.h >~~, pow(), sqrt(), ~~< math.h >~~, etc.

To use these functions, you need to ~~call the~~ include the appropriate header files at

(2) User Defined functions :- These functions are defined by the user at the time of writing the program.

Parts of function :-

(1) Function declaration (2) Function Definition.

(3) Function Call.

(Function declaration) (Function prototype)

It tells the compiler about the function's name, return type and parameters.

Syntax :- returntype functionname( arguments, arguments );  
(datatype names)

Arguments names are not important in function declaration, only their type is required.

So:- `int max(int, int);`

(2) Function declaration-definition-referent are for it consists of a function header and a function body.

Syntax:- `returntype functionname (argument1, argument2);`  
function body;

Return type is a group of data types. A function returns the value, the return type is the datatype of the value the function returns. If the function is not returning the value, the return type is void.  
function body: It contains a block of statements that define what the function does. Argument names and their datatype is also required.

(3) Function call:- To use a function you will have to call that function to perform the predefined task. When a program calls a function, the control is transferred to the called function. The function performs the task and when its return statement is executed or when its function ending closing brace is reached, it returns the control back to the main program.

Syntax:- `functionname (argument1, argument2);`

variables.

(String, Number) variables equate to string, number, float etc.

- function categories:
- (1) with return type with arguments.
  - (2) with return type without arguments.
  - (3) without return type with arguments.
  - (4) without return type without arguments.

### Recursive functions:-

A function calls itself.

In this case, it is important to define an exit condition from the function otherwise it will go into an infinite loop.

Non-recursive:- A function calls another function.

### ARRAYS:-

An array is a collection of values of same types, accessed using a common name.

One-dimensional is like a list and two-dimensional is like a table (matrices). In basic work An array is a variable that can store multiple values. It can't hold values of different datatypes. It can hold only same type values so it is called homogeneous collection of values.

The memory for the values are allocated in a continuous manner.

Elements of an array are accessed by its index value starting from 0.

If size of an array is n then the index of the last element is n-1.

Declaration of Arrays:- type name[size]; (1-D)  
size :- max. no. of elements that can be stored in the array. It must have a value.

2-dimensional :- type name [rowsize][column size];

Storing values in arrays :-

- Initialise the elements
- Input values for the elements
- Assign values to the elements

(ii) Initialisation :- We need to provide a value for every element in the array. At the time of declaration.

type arrayname[size] = { list of values};  
for one-dimensional:  
type name[size] = { list of values};  
(or) type name [] = { list of values};  
Size is not mandatory, it counts the no. of values.

for two-dimensional:-

type name [rowsize][colszie] = { list of values }  
(or) type name [rowsize] [colszie] = { list of values }  
Here's column size is not mandatory, but the row size is compulsory.

If the no. of values provided is less than the no. of elements in the array, the un-assigned elements are filled with garbage values or zero according to the storage class.

(2) Input values :- Reading at run time.

Using loops, the input value for each element is read.

(3) Assigning values :- Assign values to individual elements of the array by using assignment operator.

(Ex:-)  $a[3]=10$ ; sqrt is sent to a function  
but  $a[3][5]=11$ ; it means from row 3 to col 5  
 $a[3][5]$  is sent from the form of  $a[3]$  to the function - 6

## Types of array

One-dimensional

array [ ]  
or more than one

Two-dimensional  
the rows and columns

## Passing

To pass an array to the array function.

Use subscript to indicate address.

Use name

## 1-D

```
int fun (int
int a[5];
fun(a);
{
}
```

## 2-D

```
int fun (
{
}
}
```

## Parameter

Pass by

## Ex) Pass by

Variables to be changed

Value

The if

change

current

left

Types of arrays will based on dimension and that is  
One-dimensional, Two-dimensional, Three-dimensional  
more than one-dimensional :- Multidimensional array  
Two-dimensional array will be used for representing  
the elements of the array in the form of rows  
and columns and used to represent Matrix.

Passing array as an argument to a function

To pass an array to function only the name of  
the array is passed as an argument in the  
function.

Use subscript [ ] in the function declaration to  
indicate an array.

Use name & [ ] in the function definition.

1-D

```
int fun(int a[]); // Function Declaration
int a[5];
fun(a);
}

int fun(int a[])
{
 ...
}
```

2-D

```
int fun(int a[][10]);
int a[10][10];
fun(int a[10][10]);
}

int fun(int a[10][10]);
}

// 2D matrix passing
```

Parameter passing Techniques

Pass by value or Pass by reference

i) Pass by value:- In this, the original value cannot  
be changed. In call by value when you passed  
value to the function it is locally stored by  
the function parameter in the memory. When you  
changed argument, it is changed for the  
current function, not for the main function.

## (e) Pass by reference :- or call by reference

In this, the original value is changed because we pass address of the value to the function, so formal and actual arguments share the same address. Hence, any value changed inside the function will be reflected inside as well as outside the function.

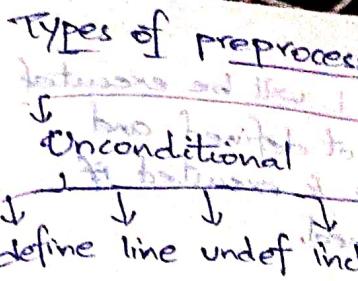
```
#include <stdio.h>
Void main()
{
 int a=50, b=70;
 swap(&a, &b);
 printf("%d", a);
 printf("%d", b);
}
```

```
(C) #include <int.h>
int swap(int *a, int *b)
{
 int c;
 *a = *b;
 *b = c;
}
```

## Pre-processor directives:-

Preprocessor directives are lines included in a program that begin with the character `#`, which makes them different from a typical source code text.

These are only one line long because as soon as a newline character is found, the preprocessor directive is considered to end. No semicolon (`;`) can be placed at the end of a preprocessor directive because it is treated as a comment. If a `#define` directive needs a lot of lines, then it must be enclosed in braces { }.



`#define` is also  
simply a macro

Syntax: `#define`  
Every occurs  
It is simply

Substitution

## File inclusion

### File inclusion

-tive tells  
source code

## Conditional

The code  
true.

It is used  
macro def

`#ifdef`

`#define`

main

{

`#ifdef`

{

state

}

`#else`

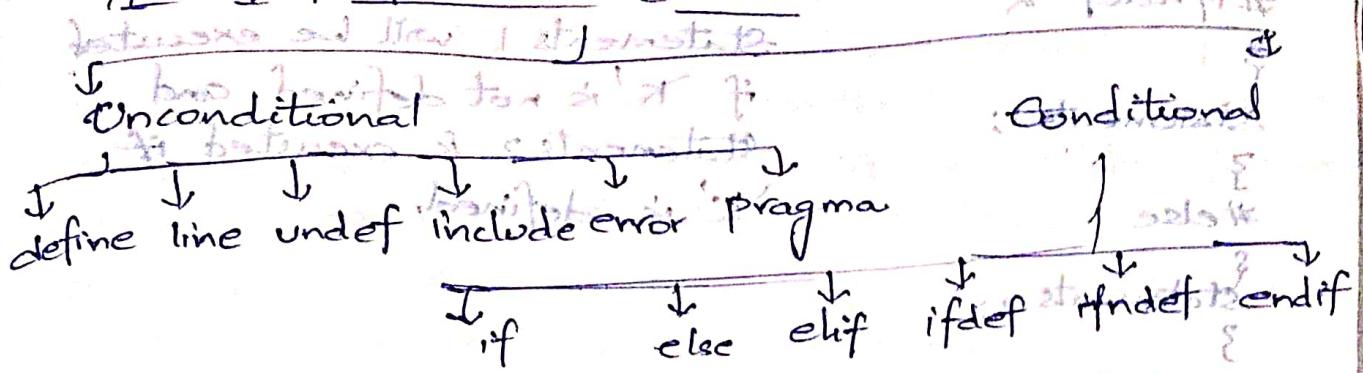
{

state

}

`#endif`

## Types of preprocessor directives:-



#define is also known as Macro definition (or) simply a macro.

Syntax :- `#define identifier substitution_text`  
 Every occurrence of the identifier is replaced by the substitution text in the source code.

File inclusion :- Ex:- `#define printf pf`

File inclusion :- This type of preprocessor directive tells the compiler to include a file in the source code. Ex:- `#include <stdio.h>`.

## Conditional compilation :-

The code is compiled if certain condition holds true.

It is used to check whether the existence of macro definitions.

```

#ifndef K
#define K
main()
{
 #ifdef K
 {
 statements 1;
 }
 #else
 {
 statement 2;
 }
#endif

```

The statements will be executed if and only if the K is defined.

If not defined, the statements will be executed.

```

#ifndef K
{
 statements1;
}
#else
{
 statements2;
}
#endif

```

statements 1 will be executed if 'K' is not defined and statements 2 is executed if 'K' is defined.

#endif  
continues until the next code or #endif

Test multi-line statements  
with multiple statements in {}  
or code blocks or multi-line statements in if  
else etc. cases where each line has its own structure  
for if, for loops, functions etc.

for (i=1; j<5; i++) next line is considered as  
with a new definition of variables after first visit  
if (a>10) then next line is  
a(1)

Scap,

last definitions carries if beginning of block or  
a(2) previous

isn't static set members of base in j  
a(3) next

Static int a[20];

i=0

a[0]=i++;

a[0]=i=0      a[0]=0  
i=i+1=1      i=1

not static set, last value

Print a[0]=0

j=i+1; b[i++]

j=0; if b[0]

i=1; while

if b[0]#

a[0]=default value of static = 0

a[0]=0

i=1.