ALGORITHMS

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

start/begin
Step1
Step2
:
Stop/end.

Program can be solved in three ways.

→ Algorithm
→ Flowchart
→

(1) A simple Algorithm:-

Start
Step1 : Print "Welcome"
Stop.

(2) Write an algorithm to add two numbers.

Start
Step1 : Read two input values i.e., a and b
Step2 : Add the two values c = a+b.
Step3 : Print the result i.e., c
Stop.

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

Start
Step1 :- Read two input values i.e., l and b.
Step2 : Calculate area of a rectangle i.e., A = l*b.
Step3 : Calculate perimeter of a rectangle i.e., P = 2(l+b)
Step4 : Print the result i.e., A and P.
Stop.

(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);
printf("%d",A);
Printf("%d",P);
}
```

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

Start

Step1: Read the values a, b, c.

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

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

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

Stop.

```
#include <stdio.h>
#include <math.h>
Void main( )
{
int a,b,c; x,y;
x = (-b+SQRT(b*b-4*a*c))/2*a;
y = (-b-SQRT(b*b-4*a*c))/2*a;
Printf(enter a,b,c);
scanf("%d%d",&a,&b,&c);
printf("%d",x);
printf("%d",y);
};
printf("root1=%d",x);
printf("root2=%d",y);
}
```

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

Start

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

Step2: Take c=a.

Step3: Take a=b.

Step4: Take b=c.

Step5: Print the result 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.

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

Step 2: Take a = a+b

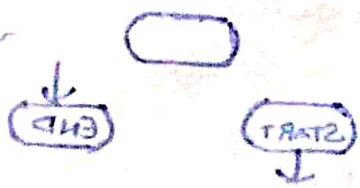
Step 3: b = a-b. (or) a = a*b.
b = a/b

Step 4: a = a-b a = a/b

Step 5: Print the result i.e., a and b

Stop.

```
#include<stdio.h>
void main()
{
int a,b;
printf("enter a and b");
a= a+b;
b= a-b;
a= a-b;
printf("%d %d",a,b);
}
```



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

The steps in an algorithm can be divided into 3 categories

(1) Sequence (2) Selection (based on condition) (3) Iterations (repetition)

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

(2) 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
incrementation
condition.

(3) 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

(1) Algorithm (2) Flowchart

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

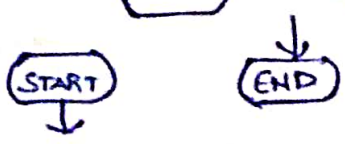
By representing various steps in the form of a diagram called as Flowchart.

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

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

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

(1) – start/stop — oval



(2) – process : — rectangle.

It is used for
Process symbol is used for data handling and assigning values to variables

Eg:-

```
z=x+y
x=2
```

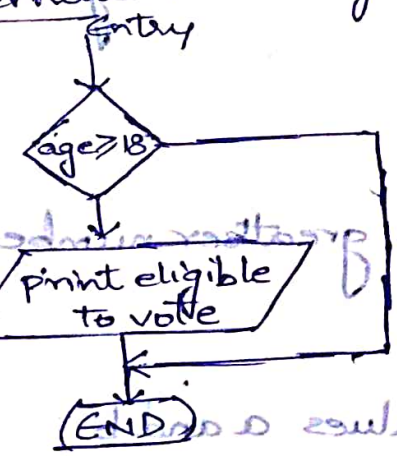
(3) Input & output symbol — parallelogram.

printf, scanf, Read input values.

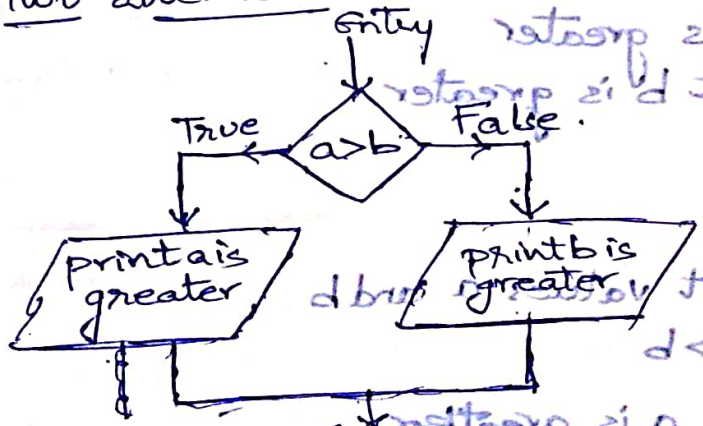
(4) Decision / Test symbol — Rhombus

- single alternate Decision
- Two alternate Decision

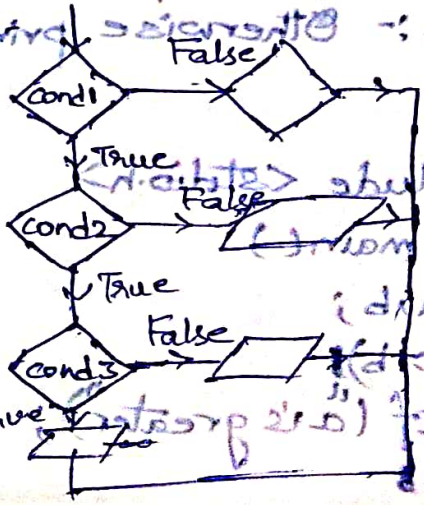
Single alternate :- only TRUE case exists
if false directly end.



Two alternate :- Two alternates exist



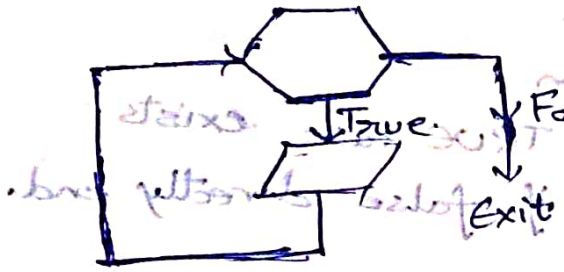
Multiple alternative decisions



(5) connector symbol ○ : circle Eg:- (A).
connects two parts of a single flow chart from one to another page.

(6) Loop symbols:- Hexagon

for repeting 'n' no. of times



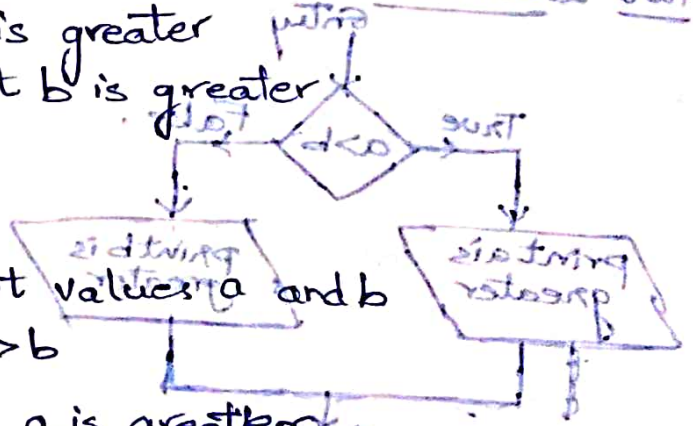
if at any one condition, it is not false, then it goes to infinity.

Write an algorithm to find greatter number between two numbers.

Start
Step1: Read two input values a and b
step2: If a>b
then print a is greater
otherwise print b is greater.
Stop.

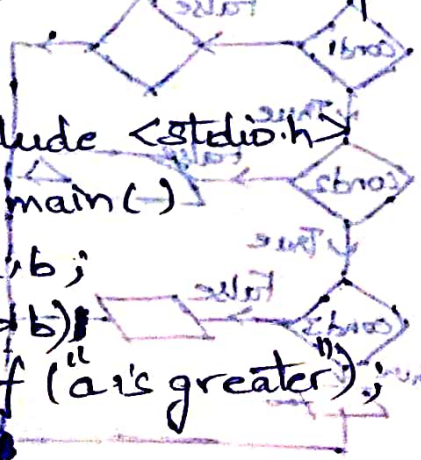
(or)

Start
Step1:- Read two input values a and b
step2:- Check if a>b
step3:- Then print a is greater.
step4:- Otherwise print b is greater
Stop



Program

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



printf(b is greater);
}.

Start
step1:- Read two input values a and b
step2:- if (a>b) then go to step3
otherwise go to step4.
step3:- Print a is greater and go to stop
step4:- Print b is greater and go to stop
Stop.

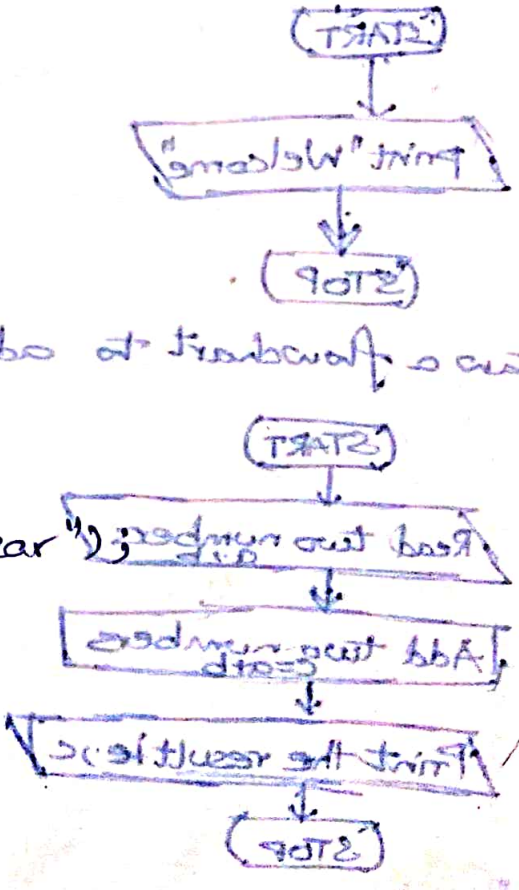
==

Write an algorithm to check the given year is leap year or not.

% gives remainder
== it is a type of >= or <= or != to assign values

Start
Step1: Read year.
Step2: If (year % 4 == 0) then go to step3
otherwise go to step4.
Step3:- Print it is a leap year and go to stop
Step4:- Print.
Stop.

```
#include<stdio.h>
Void main()
{
int a;
if (a%4==0)
scanf
printf ("Enter the year");
scanf ("%d",&a);
if (a%4==0)
printf ("it is a leap year");
else
printf ("it is a non-leap year");
}
```

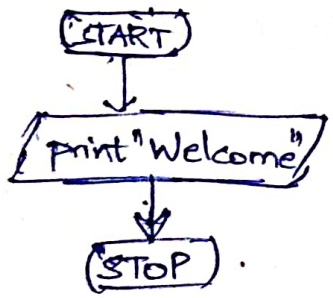


Write an algorithm to check the given number is even number or odd number.

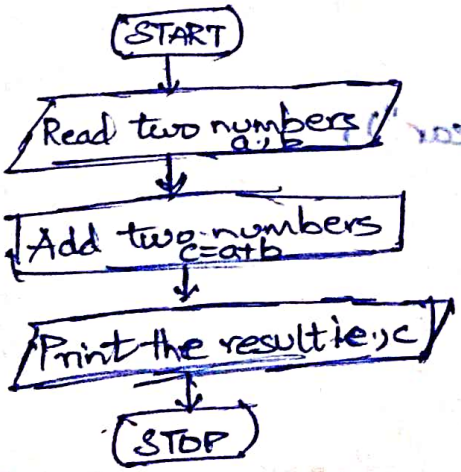
Start
Step 1:- Read the number
Step 2:- If (number % 2 == 0), then go to Step 3 otherwise go to step 4.
Step 3:- Print it is an even number and go to stop
Step 4:- Print it is an odd number and go to stop
Stop.

```
#include<stdio.h>
void main()
{
int a;
printf("Enter the number");
scanf("%d",&a);
if(a%2==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 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 it is a positive number and go to stop
Step 4:- Print it is a negative number and go to stop
Stop.

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

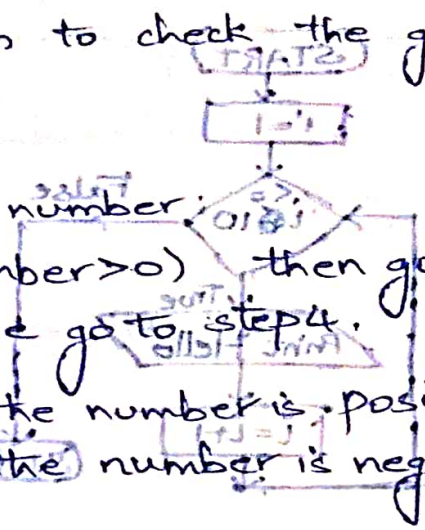
→Write an algorithm to print Hello World 5 times.

Start
Step 1:- Initialize i=1
Step 2:- Repeat
2.1 Print Hello World
2.2 i=i+1 until i<=5
Stop.

```
#include<stdio.h>
void main()
{
int i=1;
for(i=1;i<=5;i++)
{
printf("Hello World");
}
}
```

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

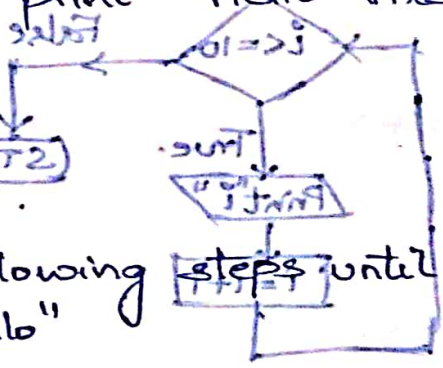
Start
Step1 :- Read the number
Step2 :- If (number>0) then go to step3
otherwise go to step4.
Step3 :- Print the number is positive and go to stop.
Step4 :- Print the number is negative and go to stop.
Stop.



```
#include<stdio.h>
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");
}
```

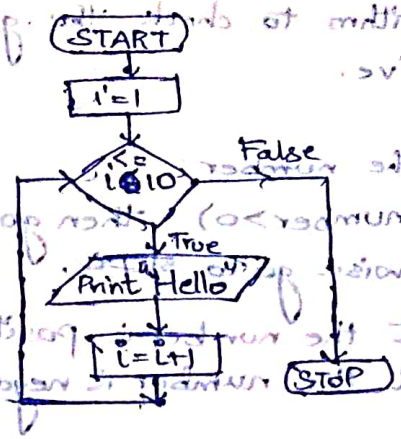
→Write an algorithm to print 'Hello' message for 10 times'

Start
~~Step1:- Print Hello~~
Step1:- Initialise i=1.
Step2: 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");
}
}
```

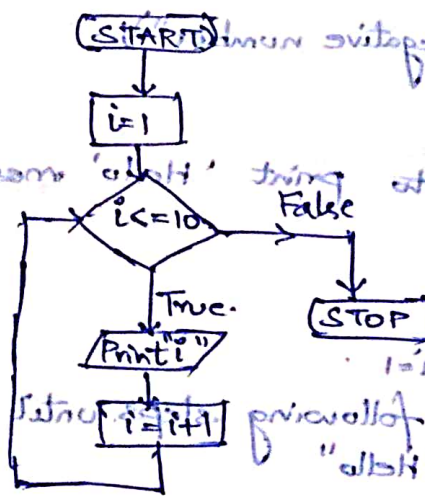
Flowchart:-



→ Write an algorithm to print 1 to 10 numbers

Start
Step1:- Initialise i=1
Step2:- 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");
}
}
```

→ Write an algorithm to

Start
Step1:- Read a number
Step2:- Initialise i
Step3:- Repeat the
3.1
3.2 i=i+1
Step4:- Print s
Stop

Flowchart:-

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

→ Write an algorithm to print sum of '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.

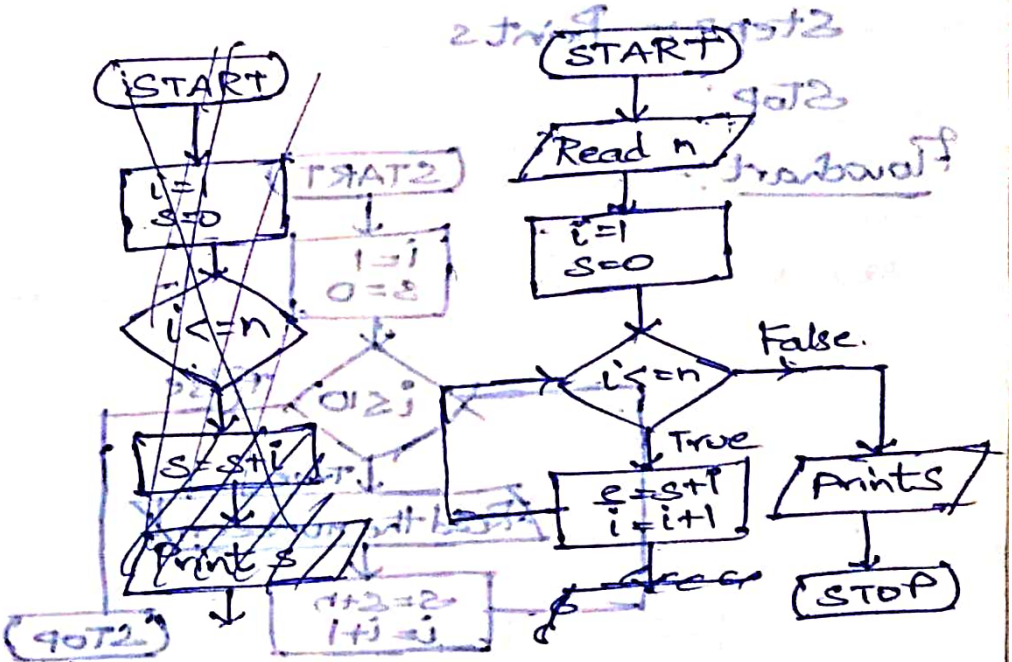
3.1 s=s+i

3.2 i=i+1

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 'n' numbers is s=%d", s);
}.
```

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

Start

Step1 :- Initialise $i=1$ and $s=0$.

Step2 :- Repeat the following steps until $i=11$

2.1 Read a number n

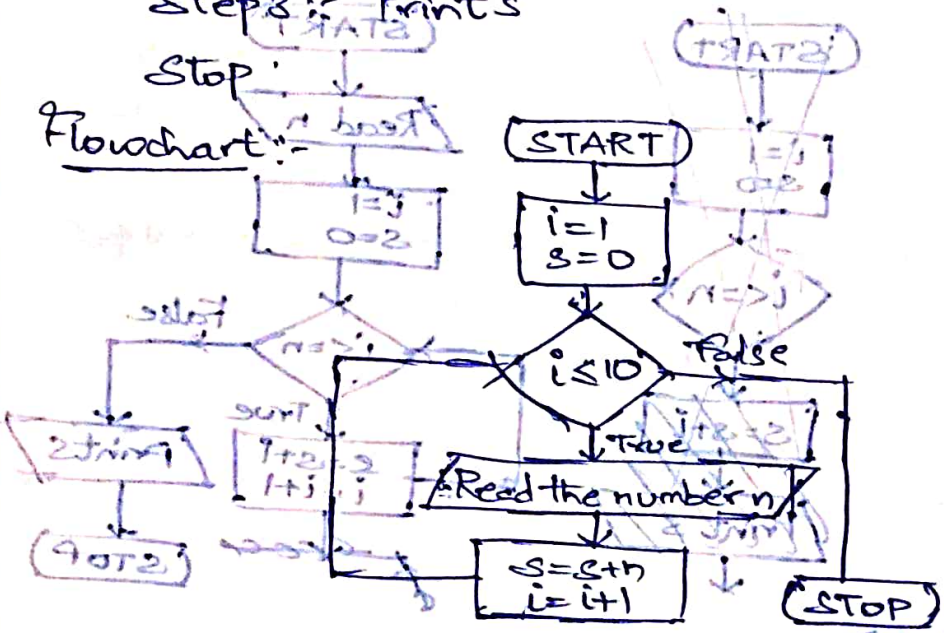
2.2 $s=s+n$

2.3 $i=i+1$

Step3 :- 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 %s=%d", s);
}.
```

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

Start

Step1:- ~~Read the temperatures F and C~~

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 Centrigrade value.

step4:- Read the temperature in degree Centigrade

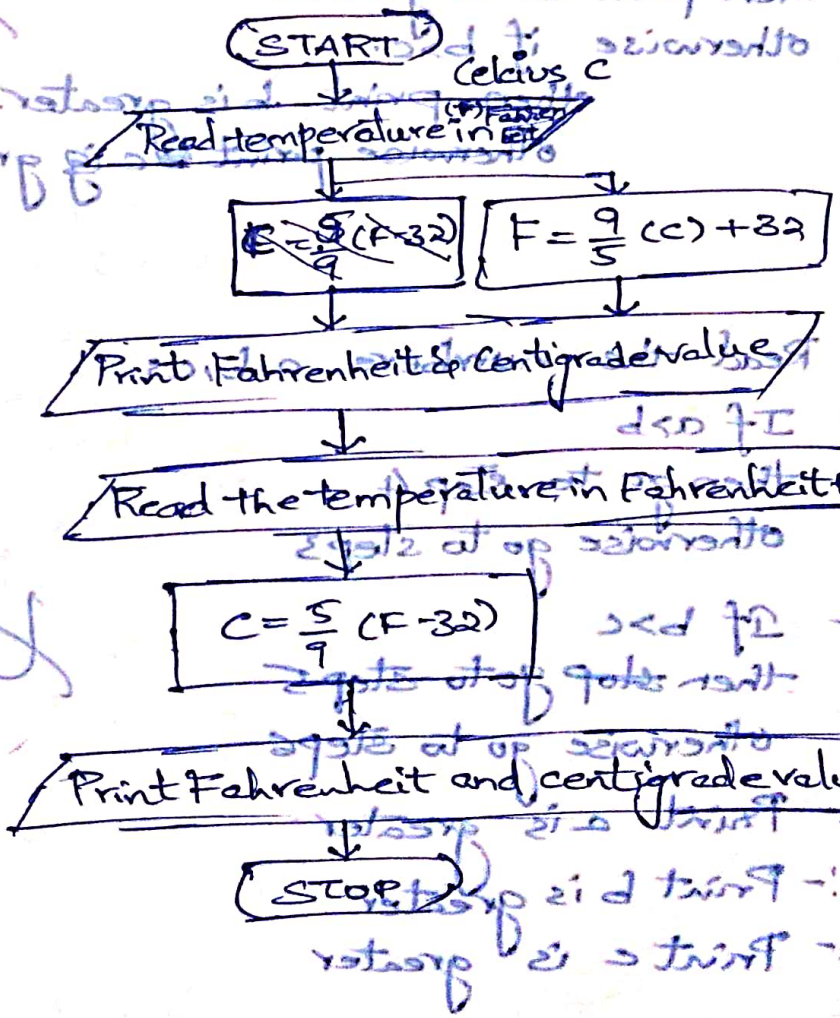
step5:- Convert the Centigrade to Fahrenheit

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

Step6:- 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);
c=(f-32)/1.8;
printf("c=%d",c);
printf("Enter the temperature in Celcius");
scanf("%d",&c);
f= 1.8*c+32;
printf("f=%d",f);
}
```

=

→ Write an algorithm to compare any three numbers

Start

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

Step 2 :- If a>b

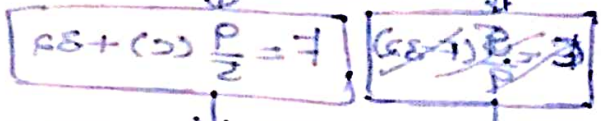
then print a is greater

otherwise if b>c

then print b is greater.

otherwise print c is greater.

Stop.



Start

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

Step 2 :- If a>b

then go to step 4

otherwise go to step 3

Step 3 :- If b>c

then go to step 5

otherwise go to step 6

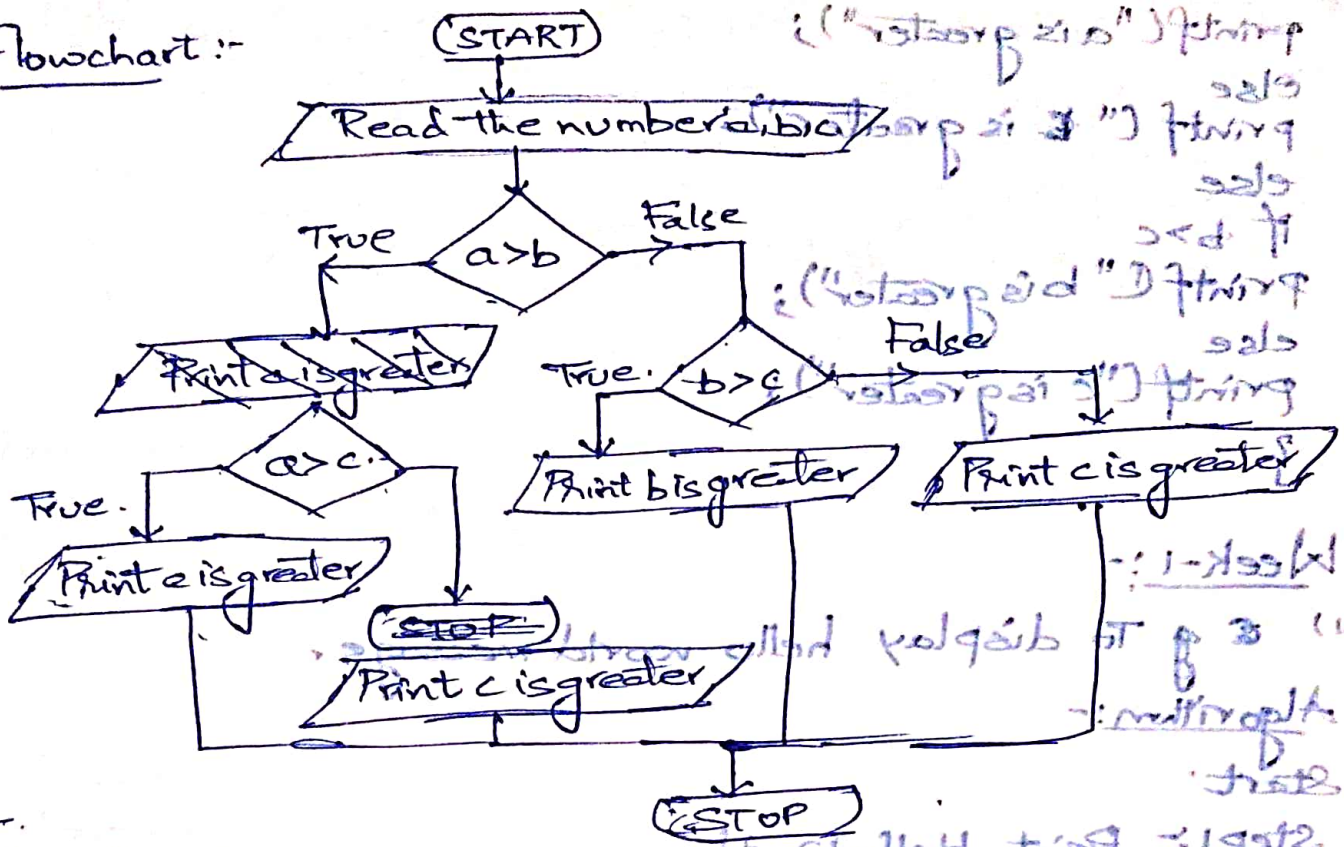
Step 4 :- Print a is greater

Step 5 :- Print b is greater

Step 6 :- Print c is greater

Stop.

Flowchart :-



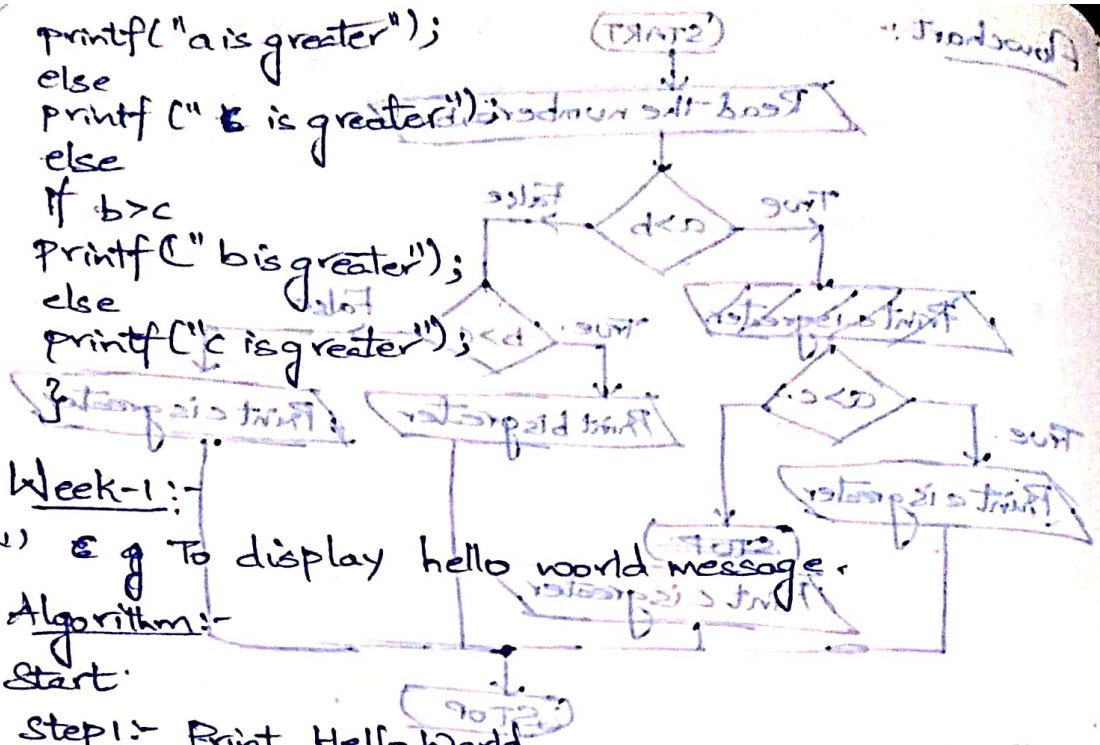
Algorithm :-

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 7
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");
}



Week-1:-

1) e.g To display hello world message.

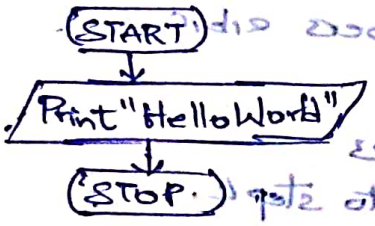
Algorithm:-

Start.

Step1:- Print Hello World.

Stop.

Flowchart:-



Program:-

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

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

Algorithm:-

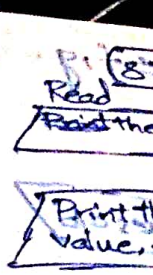
Start

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

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

Stop.

Flowchart:-



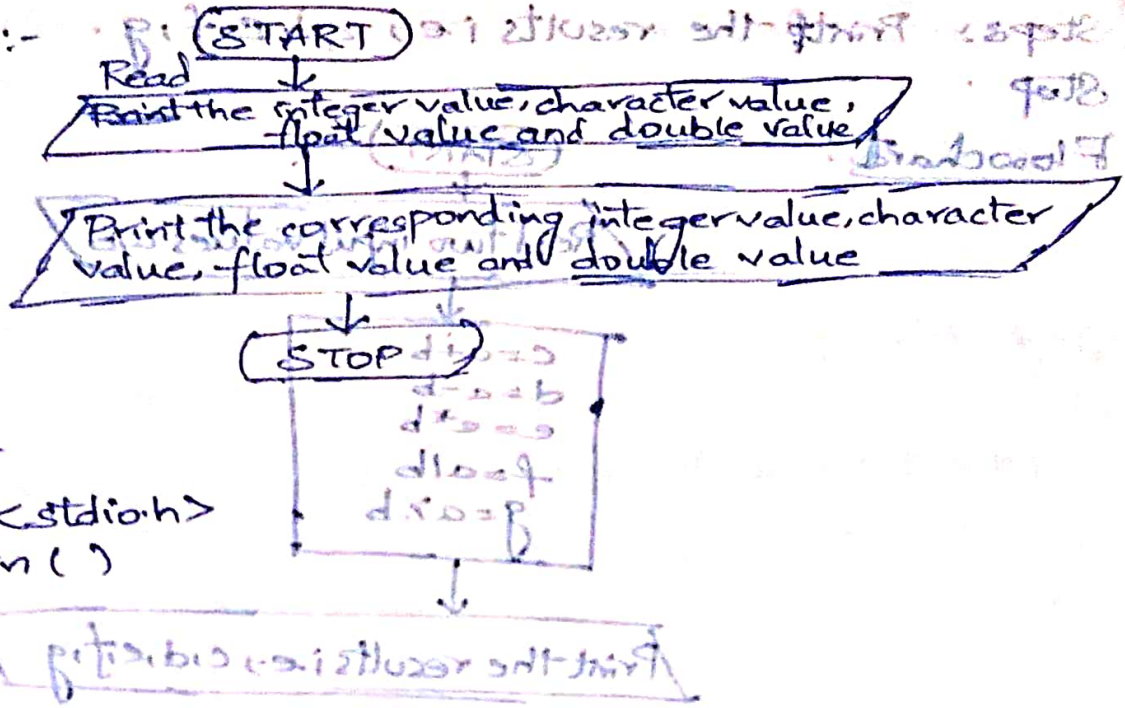
Program:-

```
#include<stdio.h
void main()
{
int a;
char b;
float c;
double d;
printf("Ente
scanf("%d"
printf("a
printf("E
scanf("%
printf("
printf("
Scanf("
printf(
printf(
scanf(
printf
}
```

(3) To per
on tw

Algorithm
Start
Step1:-
Step2:-

Flowchart:-



Program:-

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

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

Algorithm:-

Start

Step1 :- Read two input values a and b.

Step2 :- Add two values i.e., c=a+b.

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

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

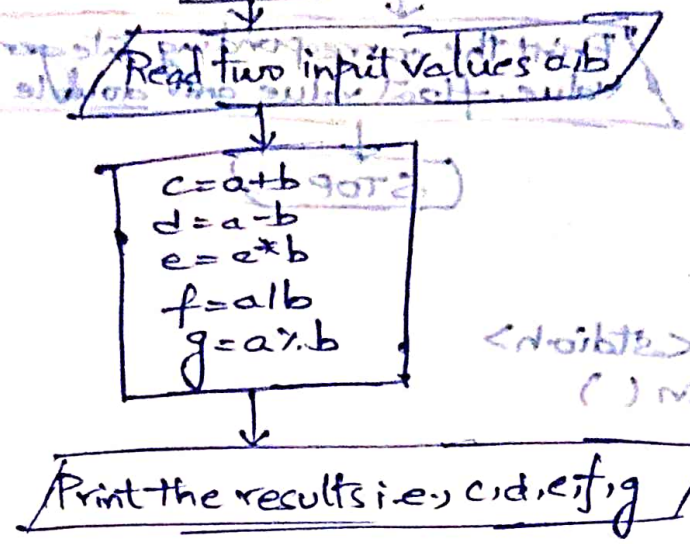
Divide two values i.e., f=a/b.

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

Steps:- Print the results i.e., c,d,e,f,g.

Stop.

Flowchart:-



Program:-

```
#include<stdio.h>
void main()
{
int
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",
c,d,e,f,g);
}
```

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

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.

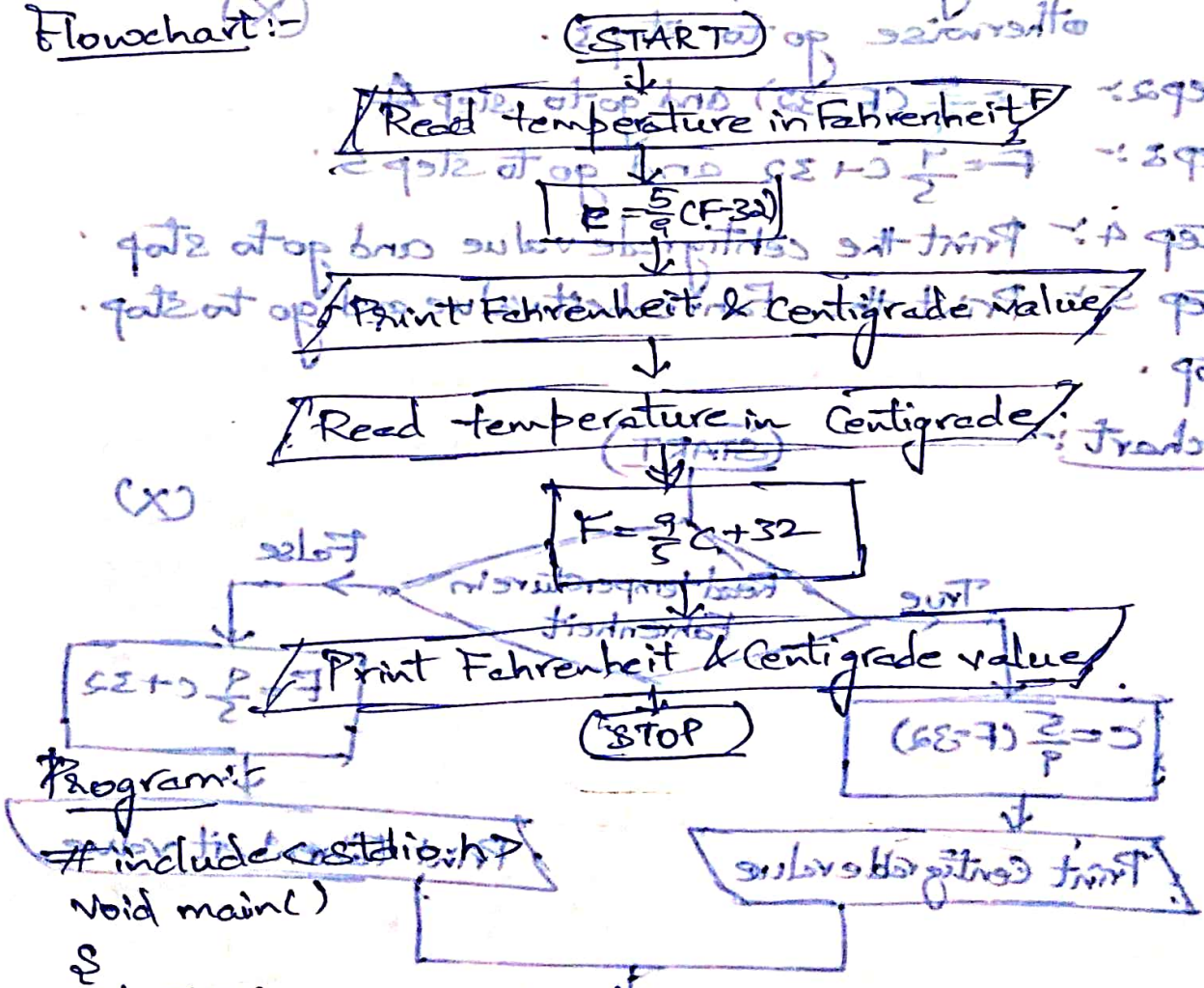
Step5:- Convert the Centigrade to Fahrenheit value.

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

Step6:- 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=5/9*(F-32);
printf("C=%d",C);
printf("Enter the temperature in Centigrade");
```

```
scanf ("%d", &C);
F = 9/5 * C + 32;
printf (" F = %d", F);
}
```

Algorithm:

Start

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

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

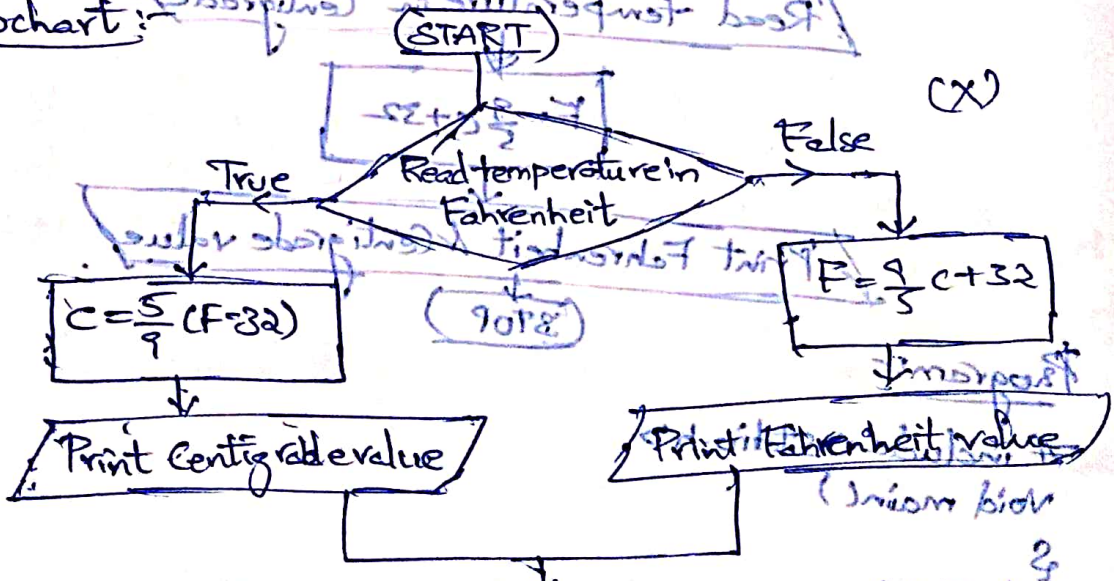
Step 3:- $F = \frac{9}{5}C + 32$ and go to step 5.

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

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

Stop.

Flowchart:-



```
Program:-
#include <stdio.h>
void main()
{
int c,F;
if
printf("Ente
scanf ("%d"
printf(" C
else
scanf ("%d
printf("
}.
```

Start
Step 1:-
°C to F & F

```
#include
void main
int c,f;
char x;
printf("R
scanf ("
#includ
void m
int a,b
printf
scanf
printf
scanf
scanf
if (x
{
printf
b= (
printf
}
else
```

Program:-

```
#include<stdio.h>
void main()
{
int C,F;
C= 5/9*(F-32);
F= 9/5*C+32;
printf("Enter temperature in Fahrenheit");
scanf("%d",&F);
printf("C=%d",C);
else
scanf("%d",&C);
printf("F=%d",F);
}
```

Start

Step1:-

°C to F & F to °C acc. to users choice:-

```
#include
void main()
int c,f;
char x;
printf("Read temperature and character x=c or f");
scanf("%d%d",&
```

```
#include
void main()
{
int a,b,x;
printf("Enter temperature");
scanf("%d",&a);
printf("1. Celcius to fahrenheit
2. Fahrenheit to Celsius");
printf("Enter users choice");
scanf("%d",&x);
if(x==1)
{
printf("Entered temperature is in degree celcius");
b=(9*c/5)+32;
printf("Temperature in fahrenheit is %d",b);
}
else if(x==2)
```

Perform arithmetic operations
users choice:

```
{
printf(" Entered temperature is in Fahrenheit ");
  a= (5*(b-32))/9;
printf (" Temperature in celsius is :%d ",a);
}
else.
printf ("reenter choice");
}.
```

Write an algorithm. to find sum of the following series $1+x+x^2+x^3+\dots+x^{10}$.

Start.

Step1 :- Read x.

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

Step3 :- Repeat the following steps until $i=11$.

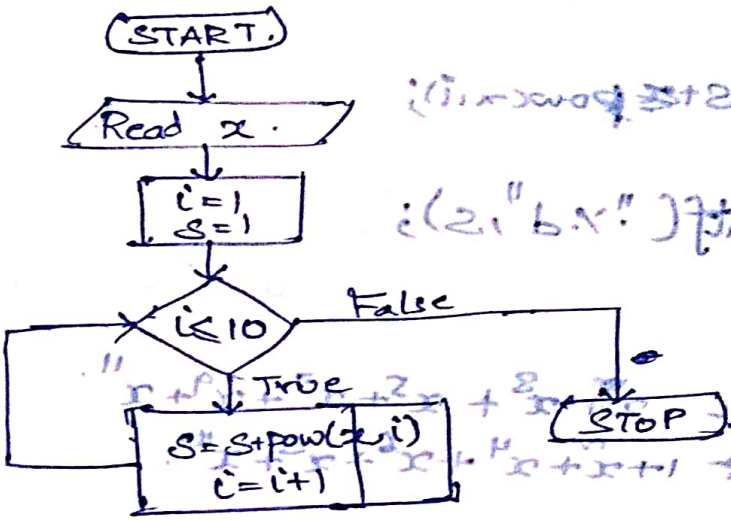
$S = S + pow(x,i)$

$i = i+1$

Step4 :- Print the result S.

Stop.

Flowchart :-



Start

Step1 :- Read x.

Step2 :- Initialise $i=0, S=0$

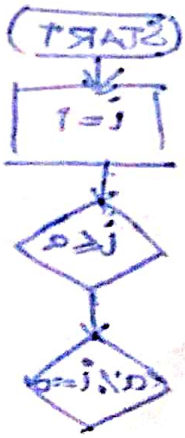
Step3 :- Repeat the following steps until $i=11$

$S = S + pow(x,i)$

$i = i+1$

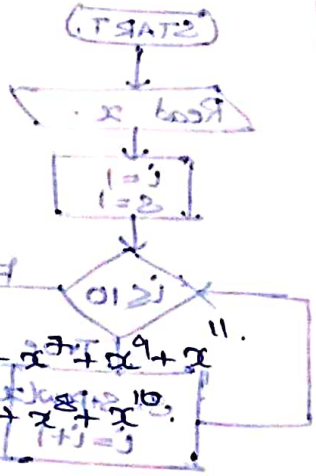
Step4 :- Print S

Stop.



Program:-

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



Series: $x + x^3 + x^5 + x^7 + x^9 + x^{11}$.

Series: $1 + x + x^4 + x^6 + x^8 + x^{10}$.

Write an algorithm to find factors of a given number

Start

Step1:- Read the number a.

Step2:- Initialise i=1

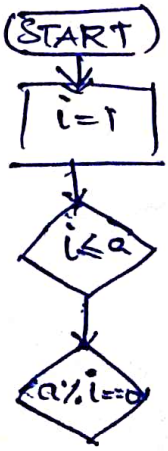
Step3:- Repeat the following steps until i=a+1 or i<a.

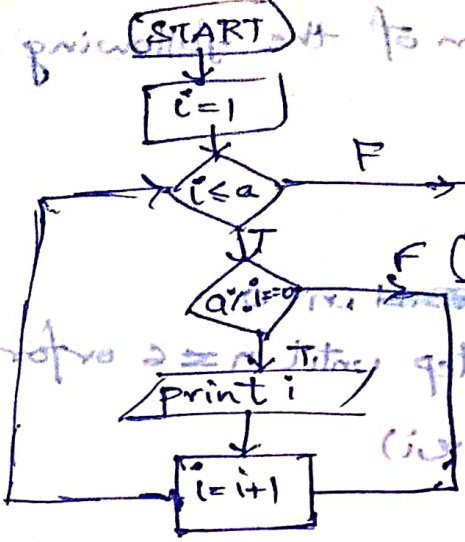
3.1:- if (a%i==0) then print i,

3.2:- i=i+1

Stop.

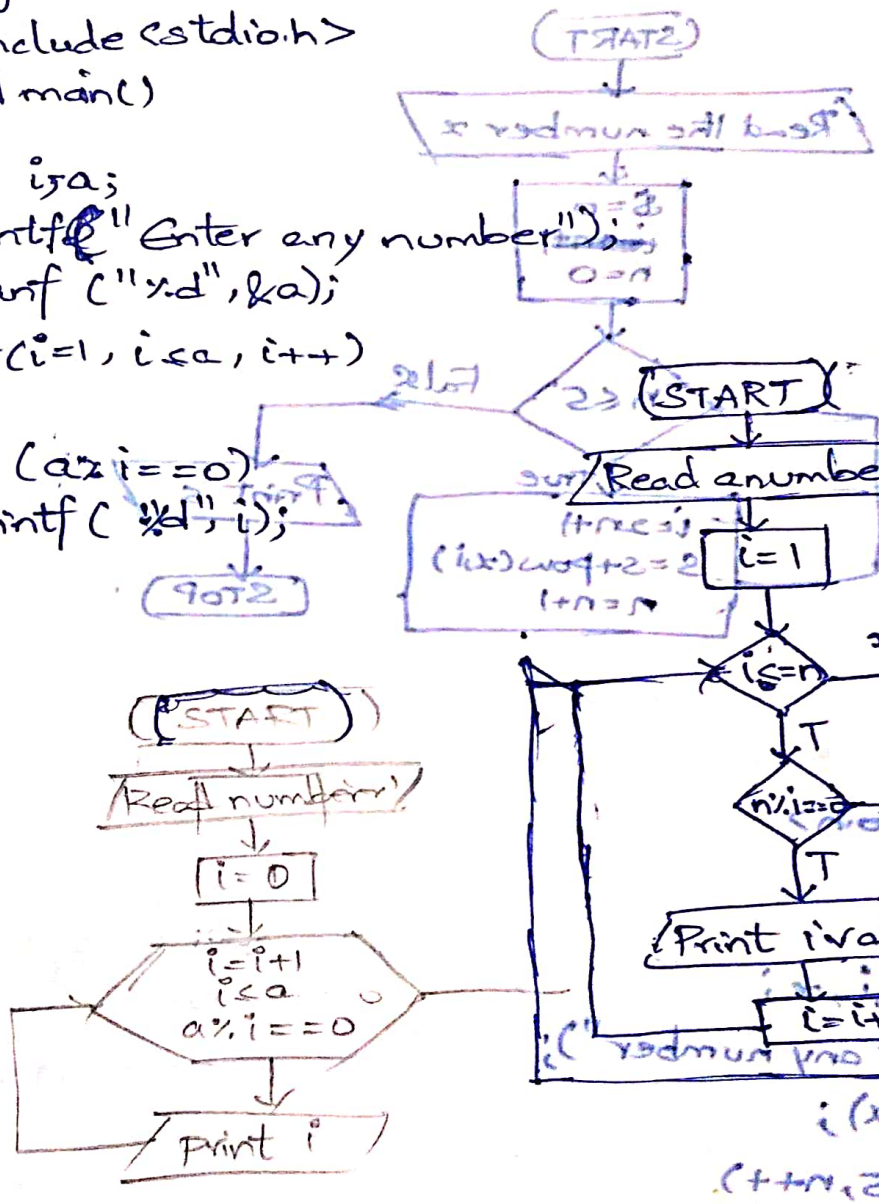
Flow Chart:-





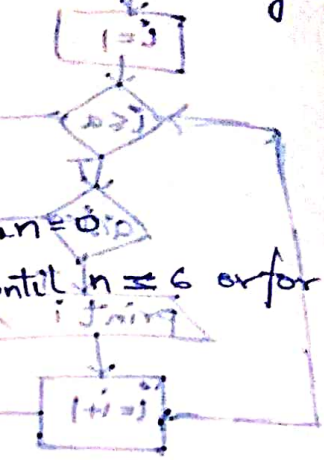
Program:-

```
#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);
}
}
```

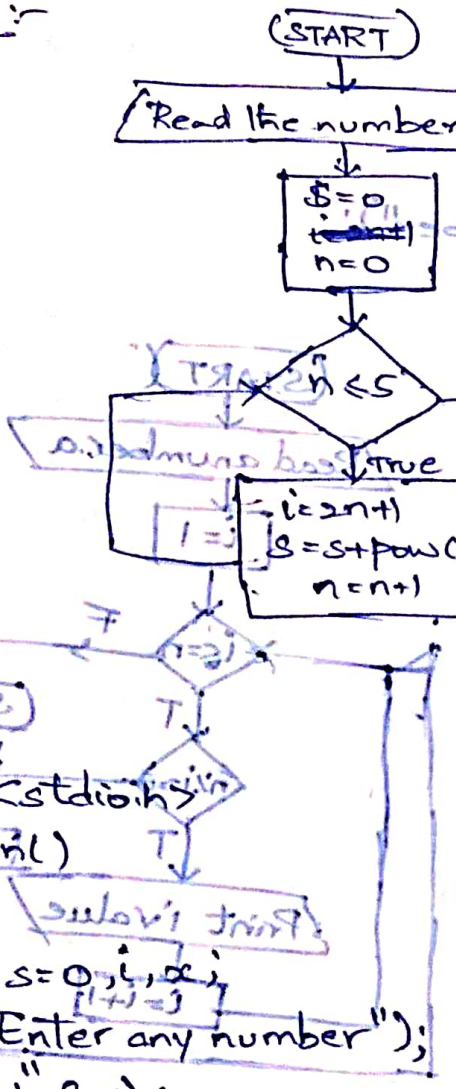


Write an algorithm to find sum of the following series
$x+x^3+x^5+x^7+x^9+x^{11}$.

Start
Step1:- Read the number x.
Step2:- Initialise s=0, n=0
Step3:- Repeat the following step until n <= 5 or for n<5
i = 2n+1
s = s + pow(x,i)
n = n+1
Step4:- Print s
Stop



Flowchart:-



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+1;
s= s+pow(x,i);
}
printf("%d", s);
}
```

Write an algorithm to find sum of following series $1+x^2+x^4+x^6+x^8+x^{10}$.

Start

Step 1:- Read the number x

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

Step 3:- Repeat the following step 2 until n=6 or for n=5.

i=2n

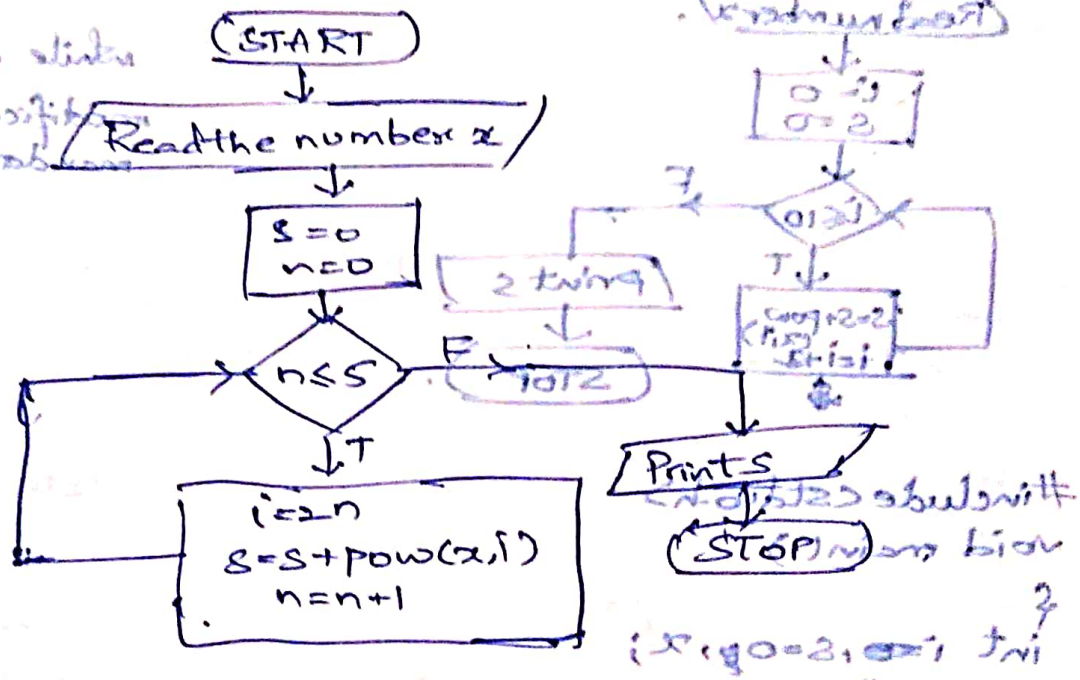
s=s+pow(x,i)

n=n+1

Step 4:- Print s

Stop

Flowchart:-



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

$1+x+x^2+x^4+x^6+x^8+x^{10}$

Start

Step 1:- Read the number x.

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

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

s=s+pow(x,i)

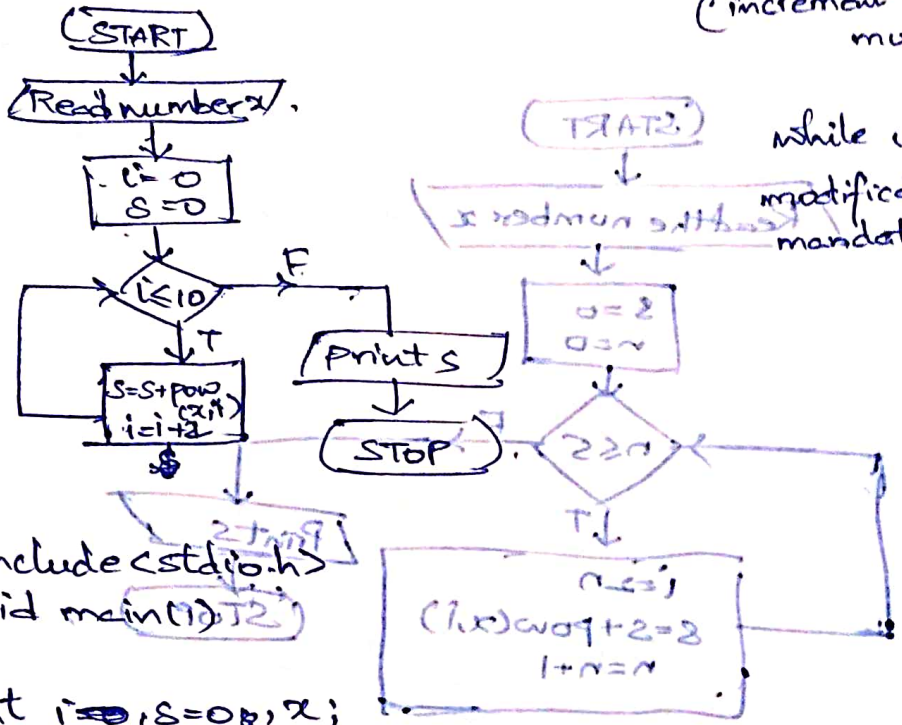
i=i+2.

Step 4:- Print s

Stop.

modification counter (increment, decrement, multiply etc.)

while using loops, modification counter is mandatory




```
#include<stdio.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);
}
```

$x + x^3 + x^5 + x^7 + x^9 + x^{11}$

Start

Step1:- Read the number x.

Step2:- Initialise s=0, i=1.

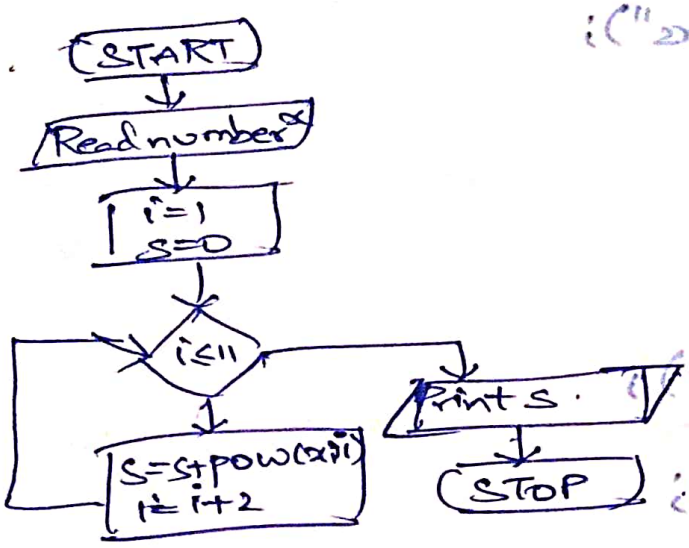
Step3:- Repeat the following steps for i<=11.

s = s + pow(x,i)

i = i + 2

Step4:- 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);
}
```

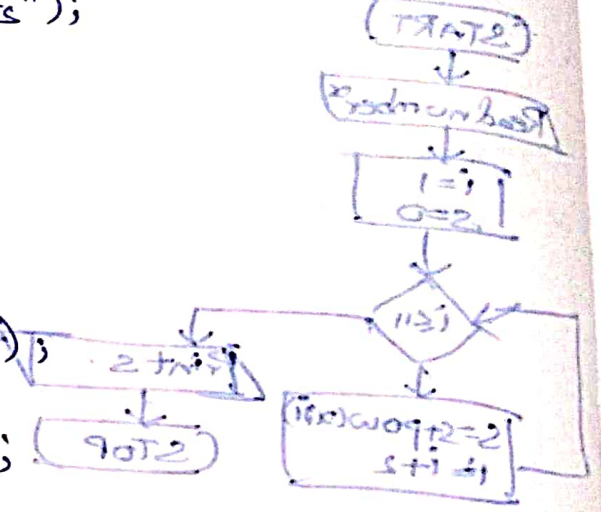
Perform arithmetic operations. "+, -, *, /, %" on user choice.

Start

Step 1:- Read two numbers

Step 2:-

```
#include<stdio.h>
Void main()
{
int a,b,c,x;
Printf(" Enter two numbers");
Scanf ("%d%d", &a, &b);
Printf (" 1. Addition \n
          2. Subtraction \n
          3. Multiply \n
          4. Divide \n
          5. Give remainder");
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)
{
```

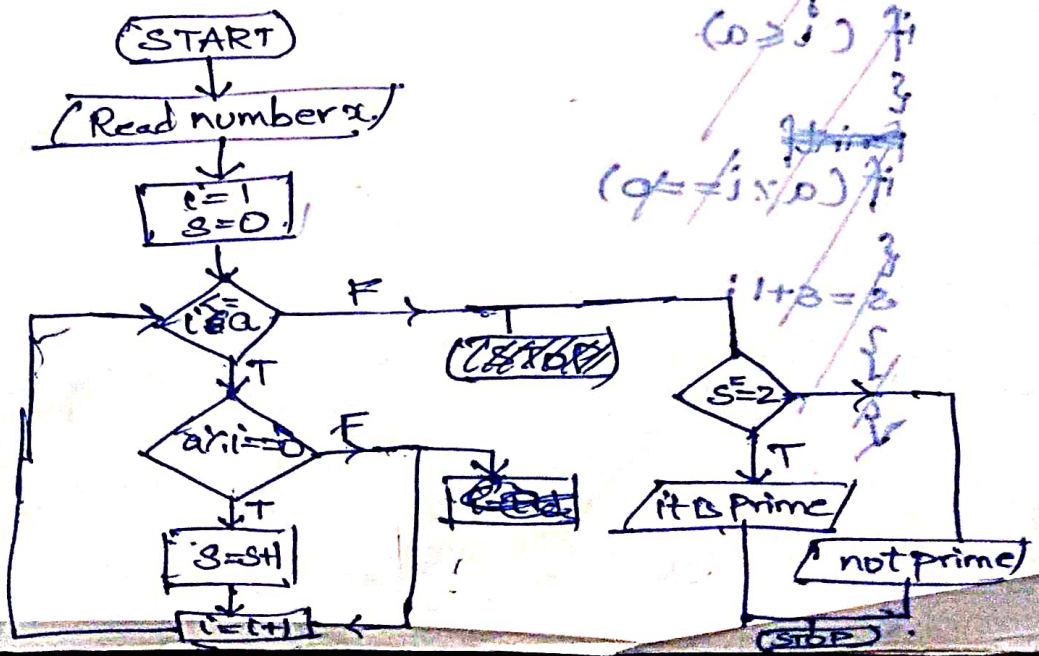


```
printf(" Dividing");
c=a/b;
printf("%d",c);
else if (x==5)
{
printf("Remainder");
c=a%b;
printf("%d",c);
}
else
printf("re. enter choice");
}.
```

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

start
step1 :- Read the number 'a'
step2 :- Initialise i=1, s=0.
step3 :- Repeat the following steps until i<=a+1
3.1 :- if a%i==0
then s=s+1.
3.2 :- i=i+1
step4 :- if s=2.
then printf("it is a prime number")
otherwise printf("it is not a prime number")
stop.

Flowchart :-

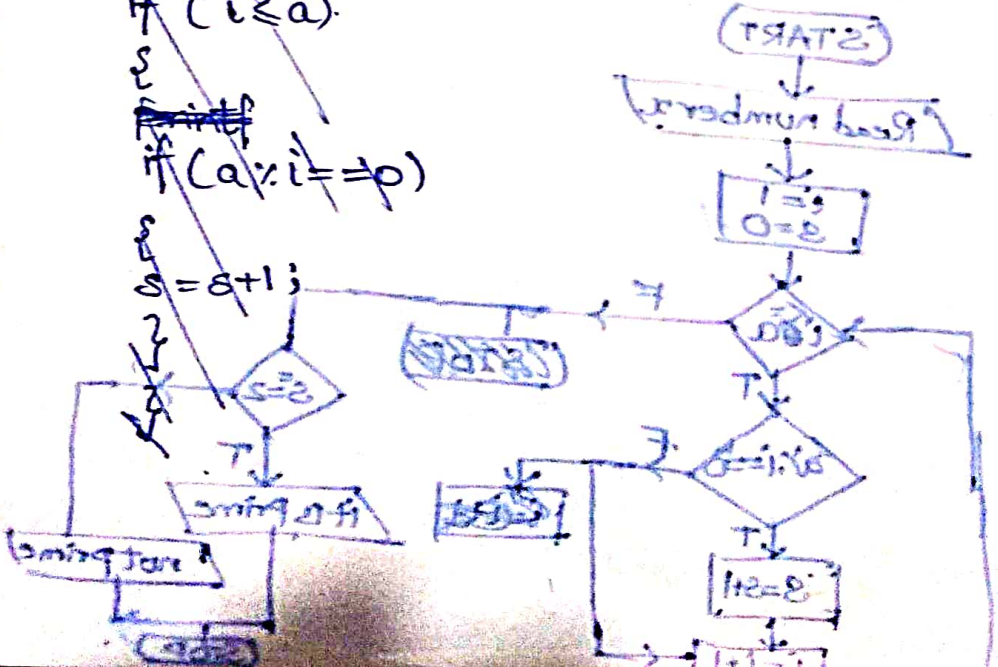


Program :-

```
#include <stdio.h>
void main()
{
 int i,s=0,a;
 printf ("Enter any number");
 scanf ("%d",&a);
 for (i=1; i<a; i++)
 {
 if (a%i==0)
 s=s+i;
 }
 if (s=2)
 printf ("it is prime");
 else
 printf ("it is not prime");
 }.
```

→

```
# include <stdio.h>
void main()
{
int i=0,s=0,a;
printf ("Enter any number");
scanf ("%d",&a);
if (i<a)
{
 if (a%i==0)
{
s=s+1;
}
if (s=2)
```



1 to 100.

Write an algorithm to print even numbers from 1 to n if given 1 to n then read 'n'

Start

Step1:- Initialise i=1

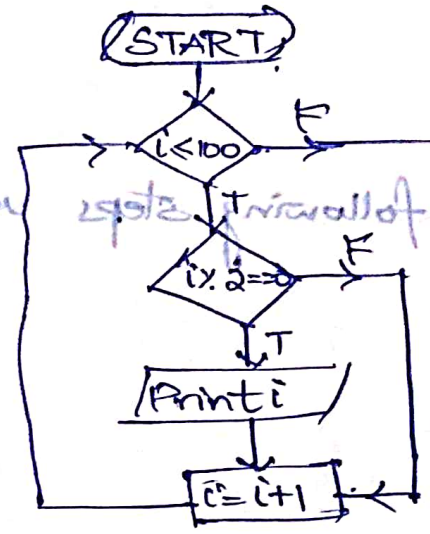
Step2:- Repeat the following steps until i=101.

2.1:- if (i%2==0)
then printi

2.2:- i=i+1.

Stop.

Flowchart:-



Program:-

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

Start

① Initialise. i=2.

② Repeat the following until i=102.

2.1 print i

2.2 i=i+2

Stop

for odd numbers.

Start

Step 1:- Initialise i=1

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

2.1 :- if (i%2!=0)

then print i

2.2 :- i = i+1

Stop

(or).

Start

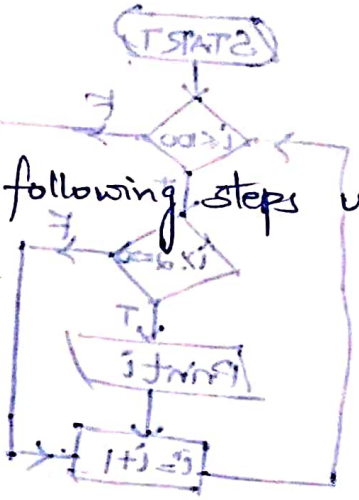
Step 1:- Initialise i=1

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

2.1 :- print i

2.2 :- i=i+2

Stop



Program:

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

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

Start

step1:- Read a number 'n'

step2:- Initialise i=1.

step3:- Repeat the following steps until i=21

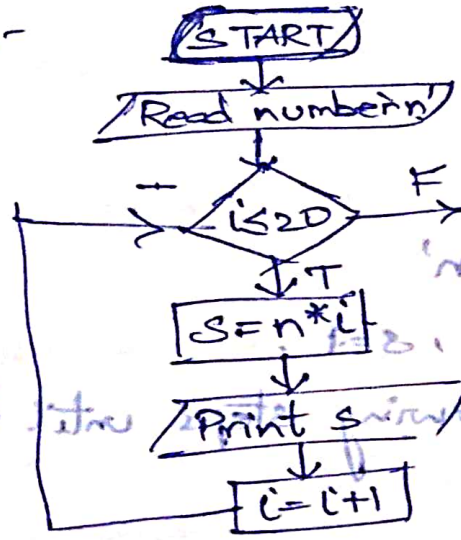
3.1:- $s = n*i$

Print s

3.2:- i+1

Stop.

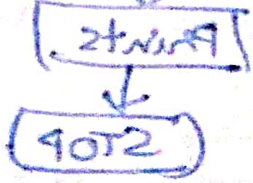
Flow chart:-

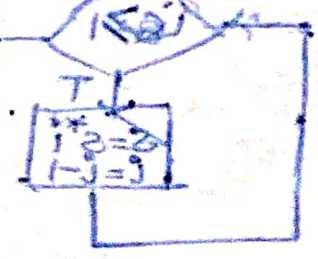


Program:-

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

Step 3 :- Repeat the following steps until $i = 0$

3.1 :- $s = s * i$

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 $i = 1$.

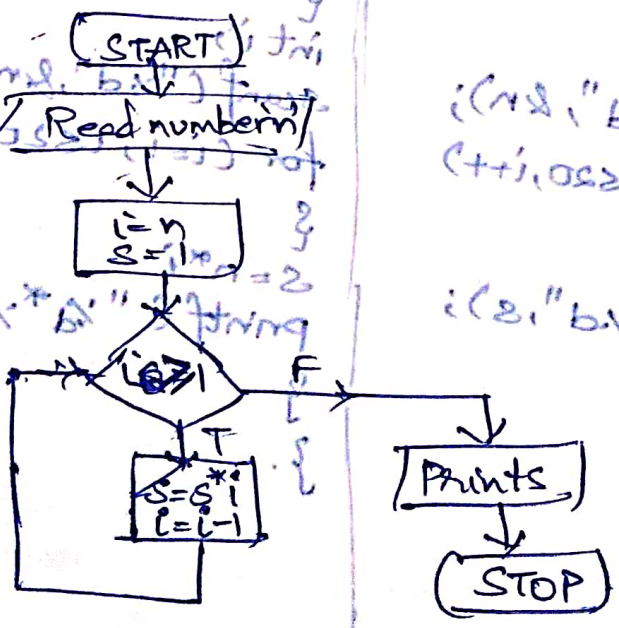
3.1 :- $s = s * i$.

3.2 :- $i = i - 1$.

Step 4 :- Prints

Stop.

Flowchart :-



Program :-

```
#include<
void main(
{
int i,s.
scanf("%
for(i=n,
{
s=s*i
}
printf(
}.
```

code for

Start

Step 1 :- R

Step 2 :-

Step 3 :-

Step 4 :-

Stop

Flowcha

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

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

Start

step1:- Read the number x

step2:- Initialise i=1, s=0, k=1

step3:- Repeat the following steps until i=6

~~s=s+pow(x,i)~~

k=k*i

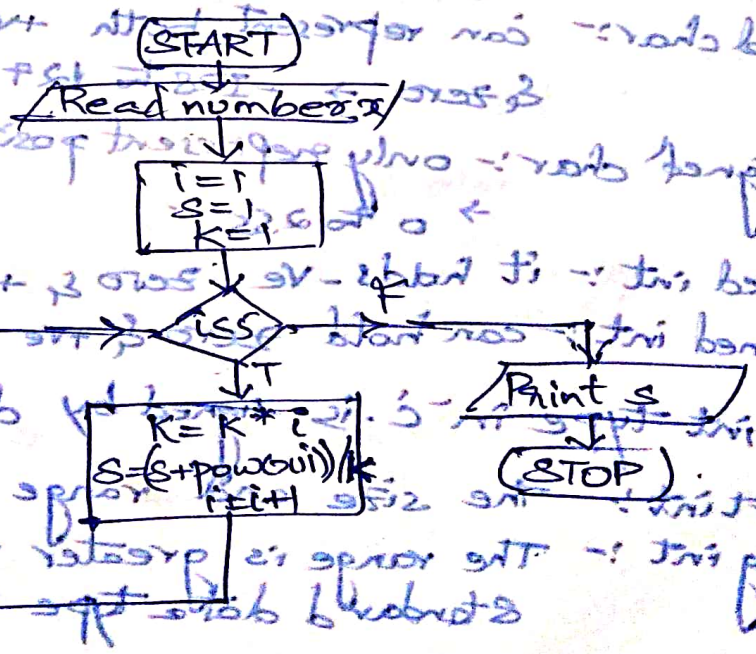
s=(s+pow(x,i))/k

i=i+1

step4:- Print the result s

Stop

Flowchart:-



Program :-

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

Data types in C :-

(1) int :- for declaring an integer variables zero, positive and negative values in 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 → -128 to 127.

(6) unsigned char :- only represent positives values & zero → 0 to 255.

(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 and range is short.

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

(11) long dou

Datatypes to use These on characters

Keyword
char
unsigned char
signed char
int
unsigned int
signed int
short int
long int
unsigned lon
unsigned
short
float
doubl
Long
doubl

(11) long double :- it is more precise than double precision.

Datatypes are used to define a variable before to use in a program.
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) %hhu (for num output)
signed char	1byte	-128 to 127.	%c (or) %hhi (for num output)
int.	2bytes	-32,768 to 32,767	%d
unsigned int.	2bytes	0 to 65,535.	%u
signed int.	2bytes	-32,767 to +32767	%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 4,294,967,295.	%lu
unsigned short		0 to 65,535	%hu
float	4bytes	1.2E-38 to 3.4E+38 (6 decimal places)	%f
double	8bytes	2.3E-308 to 1.7E+308 (15 decimal places)	%lf
long double	10bytes	3.4E-4932 to 1.1E+4932 (19 decimal places)	%LF

Operators

→ Performs operations on variables/operands

(1) Arithmetic operators + , - , * , / , %

(2) Relational operators (comparison operators)

(3) Logical operators :- logical 'and' &&
(we can join conditions) logical 'or' ||
logical 'not' !

True condition → 1, False condition → 0.

a	b	a&&b	a||b	!a
0	0	0	0	1
1	0	0	1	0
0	1	0	1	1
1	1	1	1	0

Assignment operator :- Shorthand assignment operator

operator =

Same variable should be on 2 sides

Ex :- i = i * 2
i * = 2

Increment & Decrement Operators :-

Increment < pre ++i
post i++.

Pre :- Ex :- int i = 3, j;
j = ++i;
printf("i = %d, j = %d", i, j);

j = ++i → ① i = i + 1
② j = i

Post :- j = i++; → ① j = i
② i = i + 1.

i = 3
4
j = 3

Decrement → pre
→ post.

pre:- j = --i
① i = i - 1 ② j = i

post:- j = i--
① j = i ② i = i - 1

Control Statements :- to control execution of program

Control Statements
- Conditional (Selection) statements (decision making)
  - if – Simple if
    - if else
    - nested if
    - else if ladder
  - switch.
- Iterative/loop/ Repetitive
  - for
  - while
  - Dowhile
- Jump statements
  - break
  - Continue
  - goto

Conditional / Selection statements:-

if statement :-

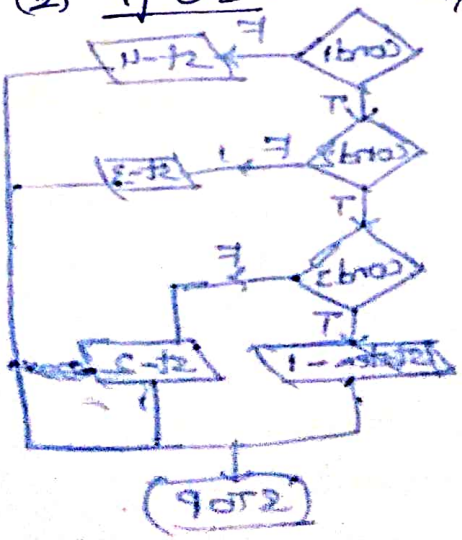
(1) **Simple if** :- Syntax :-

```
if (condition)
{
Statements 1;
Statement 2;
}
```

Block (no. of statements placed within braces)

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

(2) **if else** :- Syntax :-

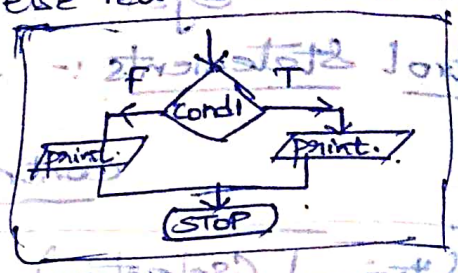


```
if (condition)
{
statement 1;
statement 2;
}
else
{
statement 3;
statement 4;
}
```

Ex:- a=2, b=3

```
if(a<b)
{
printf("a is smaller");
}
else
{
printf("b is smaller");
}.
```

if a<b
True returns '1'
else returns '0'



(3) Nested if:- one 'if' within another 'if'

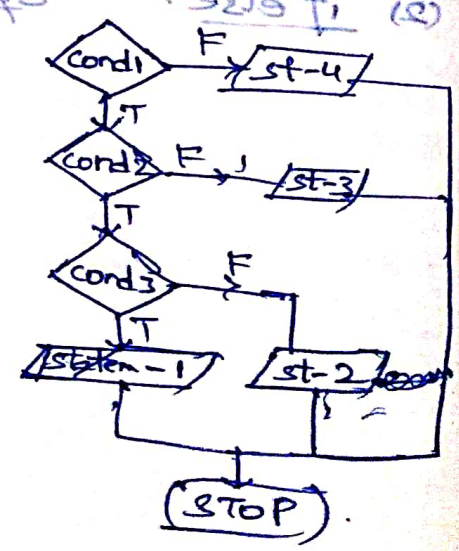
Syntax:-

```
if(condition1)
{
if(condition2)
{
if(condition3)
{
statement1;
}
else
{
statement2;
}
}
else
{
statement3;
}
}
else
{
statement4;
}
```

multiple conditions

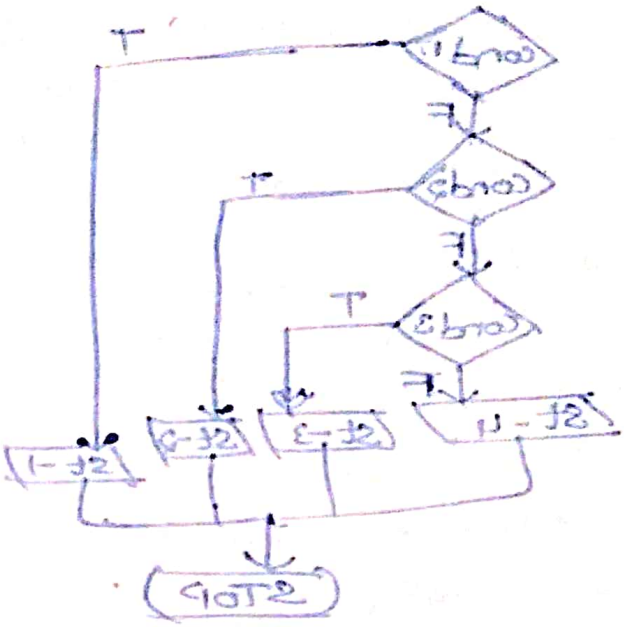
```
if(condition1)
{
  if(condition2)
  {
    if(condition3)
    {
    statement1;
    }
    else
    {
    statement2;
    }
  }
  else
  {
  statement3;
  }
}
else
{
statement4;
}.
```

Flowchart :-



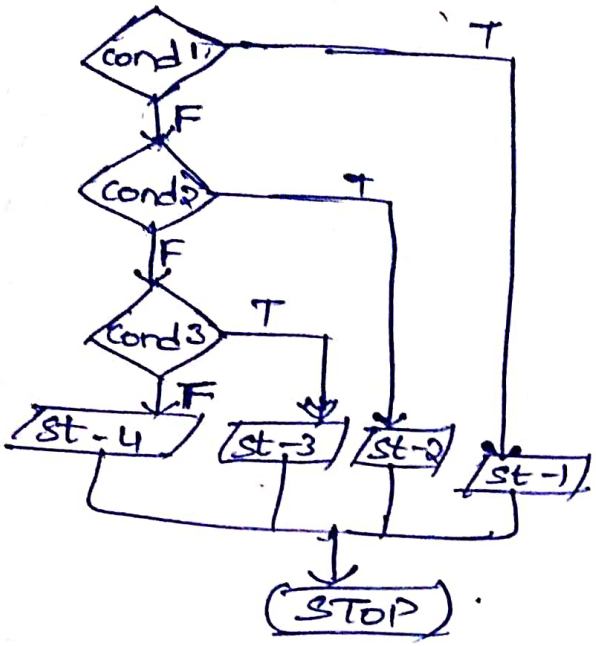
Ex:- Comparing 3 numbers

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



(4) else if ladder :-

Flowchart :-




```
if (condition1)
{
statement1;
}
else if (condition2)
{
statement2;
}
else if (condition3)
{
statement3;
}
else
{
statement4;
}
```

Write a program to give grades to a student based on following data.

>90	A
80-90	B.
70-80	C.
40-70	D
<40	F.

```
#include<stdio.h>
void main()
{
scanf("%d",&a);
if(a>90)
{
printf("%d",A);
}
else if(a>80 && a<90)
{
printf("%d",B);
}
else if(a>70 && a<80)
{
printf("%d",C);
}
else if(a>40 && a<70)
{
printf("%d",D);
}
else
{
printf("%d",E);
}
```

```
#include<stdio.h>
void main()
{
scanf("%d%d%d%d%d",
&a,&b,&c,&d,&e);
n=(a+b+c+d+e)/5;
if(n>90)
{
printf("%d",A);
}
else if(a>80&& a<90)
{
printf("%d",B);
}
else if(a>70&& a<80)
{
printf("%d",C);
}
else if(a>40&& a<70)
{
printf("%d",D);
}
else
{
printf("%d",E);
}
```

Switch :- menu driven programs

Syntax:-

```
switch (choice)
{
case value1 : statement 1;
              statement 2;
              break;
case value2 : statement 3;
              statement 4;
              break;
default : printf ("invalid choice");
}
```

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

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

for integer → no need. if double codes.

String → group of char's

Ex:-

```
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=a-b;
         printf("%d",c);
         break;
default: printf("Invalid choice");
}
```

Loops :- while
do while
for.

while :- syntax :-

```
Initialisation
while(condition)
{
  statement;
  Updation counter;
}
```

Ex :-

```
i=0;
while(i<=10)
{
printf("Hello");
i=i+1;
}
```

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

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

Program :-

```
#include <stdio.h>
int main()
{
int x,n,k=1,i,i;
scanf("%d%d",&x,&n);
for(i=0, i<=n i=i+2)
{
s=(s+pow(x,i))/k
```

Start

Step 1:- Read x, n.

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

Step 3:- Repeat the following until i<=n+1

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

$k = k * l * t$

$l = l + 2$

$t = t + 2$

$i = i + 2$.

Step 4:- Print s

Stop.

$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$

Start

Step 1:- Read x, n.

Step 2:- Initialise k=1, i=0, s=0, l=2, t=3.

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

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

$k = k * l * t$

$l = l + 2$

$t = t + 2$

$i = i + 2$.

Step 4:- Print s

Stop.

$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$

Program

```
#include <stdio.h>
int main()
{
float k=1, i, s=0, l=1, t=2;
for( i=0, i<=n, i=i+2);
{
s = (s + pow(x,i))/k;
k = (k * l * t);
l = l+2;
t = t+2;
}
printf("%d", s);
}
```

```
int main()
{
float x=1, i, s=0; [illegible]
for(i=[illegible]; [illegible]; [illegible])
{
 s=(s+pow(x,i))/k;
 k=-(k)*(-t);
 i=i+2;
 t=t+2;
}
printf("%d",s);
}
```

① $0+(0+1^0)/1 = 1$

$1+(1^2)/(-2) = 1-\frac{1}{2}$

$\frac{1}{2}+(1^4)/(+24) = \frac{1}{2}+\frac{1}{24} = 0.54167$

$(1^6)/(-24\times5\times6) = \frac{-1}{720} + 0.54167$

$= 1.389\times10^{-3} + 0.54167$

$= 0.001389 + 0.54167$

$= 0.543$

$x=1,\ n=6$

$1-\frac{1}{2}+\frac{1}{24}-\frac{1}{720} = \frac{360+30-1}{720} = \frac{389}{720} = 0.540$

② $1-\frac{1}{6}+\frac{1}{120}-\frac{1}{5040} = \frac{5040+42-840-1}{5040} = \frac{4241}{5040} = 0.8414$

→ $0+(1^1)/1 = 1$

→ $1-(1^3)/6 = 1-\frac{1}{6} = 0.8334$

→ $1.1667+(1^5/6\times4\times5) = 1.1667+\frac{1}{120} = 1.1751$

→ $1.1751+(1^7/6\times4\times5\times6\times7) = 0.0001984+1.1751$

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

$= 0.8417$

$\Rightarrow 0.8417 - \left(\frac{1}{5040}\right) = 0.8417 - 0.0001984$

$= 0.8415$

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

```
#include<stdio.h>
void main()
{
int a,c,d;
scanf("%d",&a);
if(a<1000 && a>99)
{
c=a%100;
d=c/10;
printf("%d\n",d%10);
printf("%d",c%10);
}
else
{
printf("invalid input");
}
}
```

```
#include<
void main
{
int a,c,d;
scanf("
while(a>99)
{
c=a*10
while(a>
{
c=a/1
printf(
}
}
else
```

```
{
int a,c,d;
scanf("%d",&a);
if(a>99)
{
c=a%100
d=c/10
printf("%d\n",d%10);
}
else
{
printf("
}
```

```
printf("%d", c%10);
}
else
printf("invalid input");
}
```

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

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

rite 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);
}
```

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

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

}.

→ ~~to reverse letters of a word.~~

for memory operations - 0x100

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 \n
         2. Subtraction \n
         3. Multiplication \n
         4. Division \n
         5. Remainder \n");
scanf("%d", &n);
switch (choice)
        n
{
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);
        break;
case 4: printf(" the ...ratio is %d",a/b);
        break;
case 5: printf(" the remainder is %d", a%b);
        break;
default : printf( " invalid choice");
}.
```

fflush(stdin); ⇒ for two inputs

choice = getchar(); (int)

predefined string handling function

⇒ C program to check even or odd using switch.

```
#include<stdio.h>
void main()
{
int n,k;
scanf("%d",&n);
k = n%2;
switch(k)
{
case 0 : printf("it is even");
         break;
case 1 : printf("it is odd");
         break;
default : printf("Invalid choice");
         break;
}
}
```

(or)

```
{
case 0 : printf("it is even");
         break;
default : printf("it is odd");
         break;
}
```

Nested switch:-

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

This part can also be taken (Nested) switch

A-Z = 65 to 90 , a-z = 97 to 122

=

→ 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)
{
```

≥90 A
80-90 B
70-80 C
40-70 D
<40 F

```
case 10 : printf("A");
          break;
case 9 : printf("A");
          break;
case 8 : printf("B");
          break;
case 7 : printf("C");
          break;
case 6 : printf("D");
          break;
case 5 : printf("D");
          break;
case 4 : printf("D");
          break;

case
default : printf("F");
          break;
```

(or)

```
case 10 : printf
case 9 : printf("A");
          break;
case 8 : printf("B");
          break;
```

Iterative Statements : (Loops / Repetitive) - initialisation
- while - for - dowhile - condition.
- Modification / Updationcounter.

while :-

```
initialisation ——— variable is called loop variable.
while (condition)
{
statements;
Updationcounter;
}
```

Print Hello for 100 times

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

```
i=1;
while(i
{
printf("
i=i+1;
}
```

→ Write a

```
#includ
Void m
{
int i;
i=1;
while
{
if (
{
print
}
i=i+
}
}
```

Flowcha

To prin

```
#inclu
void m
{
int i=
while
{
prin
i=i+1
}
```

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

```
i=0;
while (i<100)
{
printf ("Hello");
i=i+1;
}.
```

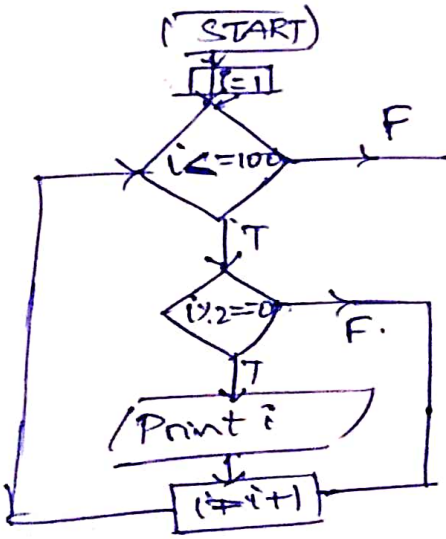
→ Write a program to print even number using while. (1-100)

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

(or)

```
{
int i;
i= 1;
while ( i<=100)
{
printf ("%d", 2*i);
i= i+1;
}
}.
```

Flowchart :-



To print 1-10 numbers (while).

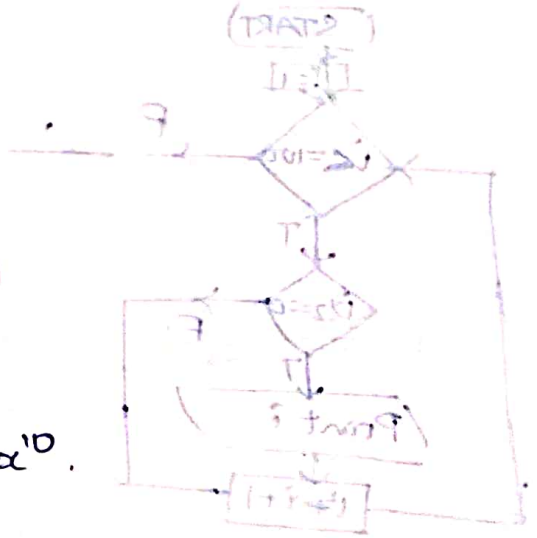
```
#include <stdio.h>
void main()
{
int i;
while (i<=10)
{
printf("%i");
i=i+1;
}
```

To print series, $1+x+x^2+\cdots+x^{10}$. (while)

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

$x+x^3+x^5+\cdots+x^n$:

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



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

```
#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 factors of a given number (while).

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

→ 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;
}
}
if(s==2)
{
printf("prime");
}
else
{
printf("not prime");
}
}
```

→ To print 'n' multiplication table (while)

```
#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);
}
```

→ $1+x^1+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots\cdots+\frac{x^n}{n!}$ (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
updation
}while(co
```

For loop

```
for(initia
{
statemen
}
```

flowcha

→ writ

#incl

void

{

int

int

sca

for

{

pri

}

}

→ w

-i

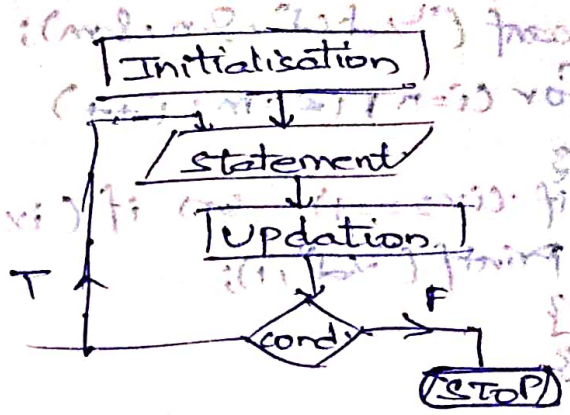
#

v

{

Do while

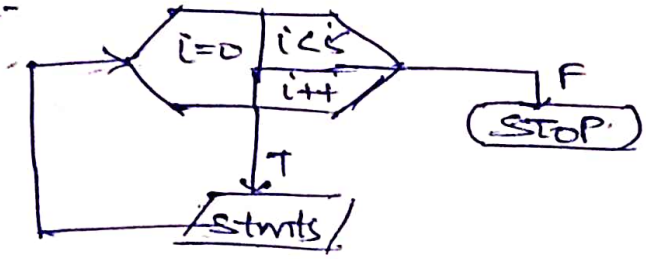
Intialisation
do
{
statement
updation
}while (condition);



For loop

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

flowchart:-



→ Write a program to print nth table.

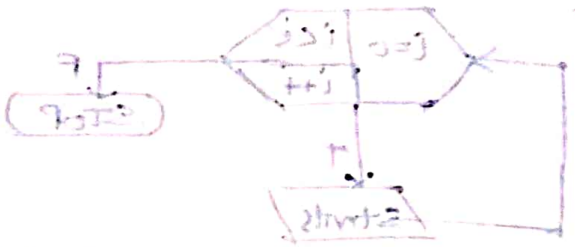
```
#include<stdio.h>
void main()
{
int n;
int i;
scanf ("%d",&n);
for (i=1; i<=10; i++)
{
printf ("%d * %d = %d", n, i, n*i);
}
}
```

→ Write a print odd no's bin specified range using for loop

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

```
scanf ("%d%d", &n, &m);
for (i=n ; i<=m; i++)
{
if (i%2==1) (or) if (i%2 !=0)
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,n;
scanf ("%d", &n);
for(i=1;i<=n;i++)
{
if (n%i==0)
printf ("%d", i);
c++;
}
}
if (c==2)
```

```
printf ("pri
else
printf ("
}.
```

→
```
#include
void mai
{
int i,j
for (i=5
{
for (k=
printf(
for (j=
printf
}
printf
}
}.
```

→
```
#inclu
void
{
int a
scanf
scanf
while
{
r=a
a=a
s=s
}
prin
if (
prin
else
pri
}.
```

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

→
```
#include<stdio.h>
void main()
{
int i,j,k;
for (i=5 ; i>=1 ; i--)
{
for (K=5; K>=i ; K--)
printf(" ");
for (j=1; j<=i ; j++)
{
printf( "%d",j);
}
printf("\n");
}
}.
```

e) Nested for

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

→
```
#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");
}.
```

→ Write a syntax for 'while':-

```
initialisation;
while (condition)
{
statements;
updation;
}
```

→ 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);
}
```

```
int i=1;
for(; i<=100;) (or) for(;;)
{
:
i=i+1;
}
```

infinite loop

Jump State
Condition
Switch
Iteration
→ break :-
for in
break
→ contin
for
for
if(i
conti
prin
}
break
→ goto
if (n
go to
else
goto
Ever
odd

Jump Statements :- - break , - continue , - goto.

Condition :- based on T/F.

Switch :- only one case executed.

Iteration :- executed many times.

→ break :- to stop the program in middle.

Ex :- in 'switch' and also for

break & continue only in loops.

→ continue :- to continue with next iteration.

i = 1, 2, 3, 4, 5.

if 3rd iteration should be executed.

'continue' continues the next iteration

i++

i == 3
continue

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

Ex :-
```
for (i=1 ; i<=5; i++)
{
if (i==3)
break;
printf ("%d",i);
}
```

Break

Output :- 1 2

```
for (i=1 ; i<=5; i++)
{
if (i==3)
continue;
printf ("%d", i);
}
```

Continue

Output :- 1 2 4 5

break :- to stop the process

→ goto label :-

```
if (n%2==0) → label
goto Even;
else
goto odd;
Even: printf("even"); both will be
odd : printf("odd"); executed.
```

go to label; control is transferred then its goes to the definition of label.

Syntax:- goto label;
↳ user defined name.

label definition ↳ label: statements

Nested for:-

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

for each value of 'i', inner loop is repeated 3 times

So total iterations = 3x3 = 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.
```

Program :-
```
1
1 2
1 2 3
```

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

outer loop :- rows
inner loop :- columns

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

Output:-

```
1
2 2
3 3 3
```

=> this is for outer loop.

Program:-

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
1
2 2
3 3 3
4 4
5 5
```

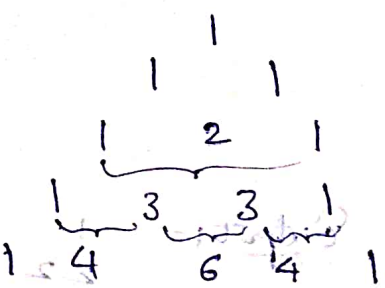
```
#include<stdio.h>
void main()
{
int i,j;
for(i=5;i>=1;i--)
{
for(j=1;j<=i;j++)
printf("%d\t",j);
printf("\n");
}
}
```

Output:-

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
#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 :-




```
4 3 2 1      1 2 3 4 5      1 1 1 1 1
3 2 1        1 2 3 4        2 2 2 2
2 1          1 2 3          3 3 3
1            1 2            4 4
             1              5
```

Operators :- (same as if else) Arithmetic, Relational, logical (to join conditions), Bitwise, Conditional (ternary), Increment & Decrement (pre, post), comma, Assignment (short hand Assignment (operator=))

↓ Seperate the expressions

Expressions :- combination of operators and operands

2+3*5 ~~the~~ follow priority (preference)

Multiplication - high priority. * / % level

+ - level

2+15
=17;

Operator precedence table :-

2*3/4 % - based on associativity
L-R or R-L

Operator
()
[]
.
->
++ --
++ --
+ -
! ~
(type)
*
&
sizeof
* / %
+ -
<< >>
< <=
> >=
== !=
&
^
|
&&
||
?:
=
+= -=
+= /=
%= &=
^= |=
<<= >>=
,

Operator	Description	Associativity
()	Parentheses or function call	left to right
[]	Brackets or array subscript	
.	Dot or Member selection operator	
->	Arrow operator	
++ --	Positive increment/decrement	
++ --	Prefix increment/decrement	right to left
+ -	Unary plus and minus	
! ~	not operator & bitwise complement	
(type)	type cast	
*	Indirection or dereference operator	
&	Address of operator	
sizeof	Determine size in bytes	
* / %	Multiplication, Division & modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift & right shift	left to right
< <=	relational lessthan/lessthanequal to	left to right
> >=	relational greaterthan/greaterthan equal to	
== !=	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	
+= -=	Addition/Subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus and bitwise assignment	
^= |=	Bitwise exclusive/inclusive OR assigment	
<<= >>=		
,	Comma operator	Left to right

4321
321
21
1

8Program:-

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

12345
1234
123
12
1

Program:-

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

```
1
11
111
1111
11111
```

Program :-

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

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

FUNCTION :-

Structure of C program :- Documentation section
Preprocess directives.
Header file section
Global declaration
void main() function
{
local declaration
Executable section
}.

Successful compilation
→ objective code
→ no errors
→ .i & .o format

functions / Pre-defined ⇒ Built-in ⇒ only limited
\ User-defined.

name() → function

After main function.
userdefined function1()
{
=
=
}
userdefined function2()
{
=
=
}
:
:

Key words :- int, float, if, else, ... (32 also called reserved words)

Functions are two types :-
(1) Predefined / Built-in functions → Ex :- printf, scanf, SQRT, POW, ABS.
(2) Userdefined functions

Header files
stdio.h
conio.h
math.h
string.h
} 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

function declaration:-

Syntax:-

Returntype functionname(argument1, argument2);

functionname → user defined name (cannot be key words)

argument → data type & name (space; name optional)

return type:- int, float, ... return the same.

if no return → void.

Ex:- int fact(int n);
(or)
int fact(int);

Ex:- int add(int a, int b);
int add(int a, char b);
(or)
int add(int, int);

mandatory → type & name

function definition:- Returntype functionname(arg1, arg2)

is called function header.

{
=====
=====
return value;
}

is called function body.

keyword ← return

returns only one value
cannot be used for swapping.

if returntype is void.
return value; (*) not needed.
then print value

Ex:- int fact(int n)
{
int i, fact1=1;
for(i=1; i<=n; i++)
fact1=fact1*i;
return fact1;
}

for calling, fact(5)

variable names/values.

function call:- functionname(argument1, argument2);

Ex:- fact(5);
(or)
fact(n);
↓
actual argument, actual parameter

formal(?)

returntype	arguement	→ 2parameters, 4ways
with	with	
without	without	
with	without	
without	with	

Based on returntype & arguements, user defined functions are four types.

User defined function categories:-

- without returntype without arguments.
- without returntype with arguements.
- with returntype without arguements
- with returntype with arguements.

1) without returntype (void) without arguements.

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

```
void main()
{
void add(void);
add();   no arguements.
}
void add()
{
int a,b,c;   not arguements so declare & read here.
scanf("%d%d",&a,&b);
c=a+b;
printf("%d",c);
}
```

2) without returntype with arguements:-

```
void main()
{
void add(int a, int b);
int a,b;
scanf("%d%d",&a,&b);
add(a,b); ← before here, read the values first.

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

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

(3) with return

(or)

```
void mai
{
void ad
int a,b;
scanf("
add(a
}
void a
{
int c;
c=a+
printf
}.
```

(3) with retu

```
void mai
{
int add
int a,b
add();
}
int add
{
int a,b
scanf(
c=a+b
return
}
```

(4) with re

```
void m
{
int ad
int a
scanf
D=add(
```

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

3) ~~with return type without arguements.~~

(or)

```
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);
}.
```

4) with returntype without arguements

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

(5)

```
recursive function
()int
{
}
```

(4) with return type with arguements

```
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 arguement :-

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

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

=

Recursion :- A function calls itself → recursion

recursive function

```
fun1()
{
fun1();
}
```

fun1 calls fun1 only (itself)

non-recursive function

```
main()
{
fun1();
}
fun1()
{
=
}
```

main function calls fun1.

```
fact(5)   fact=1;
          for(i=1; i<=5; i++)
          fact=fact*i;
```

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

$5 \times 4!$ here no need to use any loop by calling itself, it is repeating.

fact(n) =
n * fact(n-1)
5 * fact(4)
5 * 4 * fact3
3 * fact(2)
2 * fact(1)

```
int fact(int n)
{
```

fact(1)
should return to fact(2) 2*fact(1)
fact(2) = 2*1 = 2 should return to fact(3)
:
fact(5) should return to main

Queue
DS :- Data Structure

Places all the calls in →

2 × fact(1)
3 × fact(2)
4 × fact(3)
5 × fact(4)
fact(5)

Solver first.

LIFO last in first out
FILO (or) first in last out.

internal function calls should be solved first.

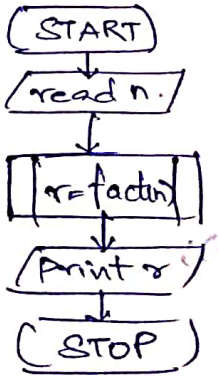
```
main()
{
int r;
int fact(int);
r = fact(5);
printf("%d", r);
}
int fact(int n)
{
if (n==1)
return 1;
else
return (n * fact(n-1));
}
```

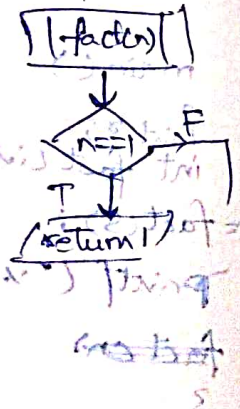
Print gcd using recursive :-

```
#include <stdio.h>
void main()
{
int gcd (int ,int);
int a,b;
Printf ("%d", gcd
scanf ("%d%d", &a, &b);
printf ("%d", gcd (a,b));
}
int gcd (int a, int b)
{
if (a<b)
{
b=a+b;
a=b-a;
b=b-a;
}
if (a%b==0)
return b;
else
return (gcd (b, a%b));
}
```

b	a
3	29 | 9
	27 b=b
r= a%b	2 | 3 | 1
	2 b
a%b	1 | 2 | 2
	2 a
b	0 | 1 |

Flowchart for function :-





fibonacci series

0, 1

```
#include<stdio
void main()
{
int i=0,j=1,k,
scanf("%d",&
for(k=0;k<
{
k=i+j;
printf("%d",
i=j;
j=k;
}
}
```

with non-rec

```
#include<st
void main()
{
int n;
scanf("%d"
printf("%
}
int fibonac
{
int i=0,j=
for(k=0;
{
k=i+j;
Printf("%
i=j;
j=k;
}
}
```

fibonacci Series :-

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

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

upto 'n' terms

'5' terms

0, 1, 1, 2, 3,

i=0, j=0

i=0

with non-recursive function :-

```
#include<stdio.h>
void main()
{ int fibonacci(int);
int n;
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("%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);
```

4+3
3+2+2+1
2+1+2+2+1
1+0+1+1+0

fib(4) +fib(3)

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

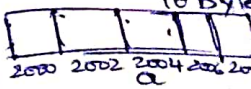
ARRAY

variable :- to store
for 5 values
instead of
we can store
using arrays
Any name which
array name.

array :- collection

arrayname[size]

Ex:- int a[5];



array is

This array

same da

it can't hold

So, it

by stru

Let

0	1
10	20

To distinguis
index

Each valu

10 = a[

string:-

declaring

ARRAYS

variable :- to store only one value
for '5' varia values, we need '5' variables

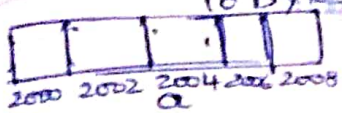
instead of using '5' variable we can store multiple variables values in one variable using arrays.

thy name which follows subscript [ ], it is called array name.

array :- collection of values of same data type.

arrayname[size]

Ex: int a[5]; for one value → 2 bytes
→ 10 bytes.

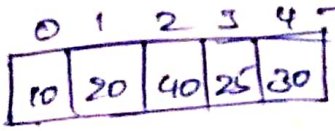


array is allocated in continuous manner locations.

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

it can't hold 2 from int & 2 from char.
so, it is a drawback, which can be explained by structure.

Let



Here all values are stored under same variable 'a'

To distinguish each value, we use 'index'
'index' begins with '0'
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 :- int a=2;
(2) Assignment int a; a=2;
(3) Scanf.

Types of arrays :- based on dimensions.

int a[] → 1-D , int a[][] → 2-D , int a[][][] → 3-D

Declaration :- syntax :- datatype arrayname [size];

Ex :- int a[10];

10×2 = 20 bytes allocated
maximum capacity
we can use only
not minimum - limited bytes
it is 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};

- in Initialisation method, size is not mandatory
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, the remaining values are stored as garbage values. (or) '0'.
it depends on storage classes :- ~~atto~~ auto

int a;
printf("%d", a);

it prints some garbage value due to auto storage class before int.

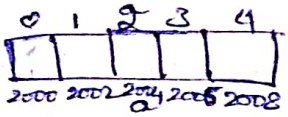
auto int a;

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

int a[5] = {0};

(2) Reading at run time :- using scanf (or) something else for character values getchar

int a[5];



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

printf("%u", &a[i]); 2002

printf("%u", a); a - array name.

2000 add. starting address.

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~~ scan 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

printf("%d", a[0]);

instead of writing printf for many times use loops.

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

(5) Assignment operator :-

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

Write a program to read and print

```
#include<stdio.h>
void main()
{
int a[100],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]);
}
```

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]);
c=a[i];
```

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

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];
```

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

```
#include<s
void main
{
int a[5]
scanf("
c=a[0]
for(i=0
{
scanf
if(a[
a[i]=
}
```

```
#inclu
void m
{
int
for(
{
fo
{
sc
if
c
}
}
```

```
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;
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];
}
```

```
#include <stdio.h>
void main()
{
int a[5], i, j, c;
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, i<n; i++)
{
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++)
{c=0; s=0;
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]<0)
{
c=c+1;
s=s+a[i];
}
printf("%
printf("%
}
for(i=0; i
{
c=0; s=0;
if(a[i]>
{
c=c+1;
s=s+a[
}
printf("
printf(
}
for(i=0
{
c=0; s=
if(a[i
{
c=c+1
s=s+
}
printf
print
}
#inc
Void
{
int
sca
for
Scan
```

```
{
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]%2==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]%2!=0)
{
c=c+1;
s=s+a[i];
}
printf("%d",c);
printf("%d",s);
}
```

```
#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];
}
}
printf(" no.of positive numbers=%d in sum of positive numbers =%d in no.of negative numbers=%d in sum of negative numbers =%d in no.of even numbers =%d in sum of even numbers = %d in no.of odd numbers = %d in sum of odd numbers= %d in",
c1,s1,c2,s2,c3,s3,c4,s4);
}
```

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

Passing array as argument to a function:

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

fun.

Types of arrays:- based on dimension:
One dimensional array, two dimensional array, three dimensional array, multi dimensional array.

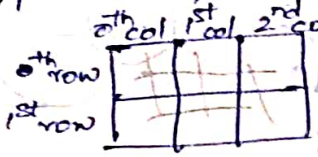
(1-D array), (2-D array) . . . more than 1-D

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

Two-dimensional array:-

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

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



0th row, 1st col
=> a[0][1]
1st row 2nd col.
=> a[1][2]

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

Initialisation:- int a[2][3] = { 1,2,3,4,5,6 };
0,0 0,1 0,2 1,0 1,1 1,2

rows-optional, columns-mandatory
to calculate rows.

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

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

contains 2 sets
each set holds
3 values

Reading at run time :-

```
for(i=0; i< rowsize; i++)
{
for (j=0; j< columnsize; j++)
{
scanf ("%d", a[i][j]);
}
}
```

Printing :-

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

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

```
#include<stdio.h>
void main()
{
int a[3][3], b[3][3]
scanf("%d%d%d

if((rowsize1 == row
{
for(i=0; i<rowsi
{
for(j=0; j<cols
scanf("%d", &a[i
}
for(i=0; i<rows
{
for(j=0; j<colsi
scanf("%d", &b
}
for(i=0; i<row
{
for(j=0; j<col
{
printf("%d\t
}
printf("\n");
}
}
```

Matrix Multipli

```
#include<stdio
void main()
{
int a[][];
scanf("%d

if(colsize1=
{
for(i=0; i<r
{
for(j=0; j<
scanf("%d
}
```

Addition

```
#include<stdio.h>
void main()
{
int a[3][3], b[3][3], c[3][3];
scanf("%d%d%d%d", &rowsize1, &colsize1, &rowsize2,
                  &colsize2);
if((rowsize1==rowsize2) && (colsize1==colsize2))
{
for(i=0; i<rowsize1; i++)
{
for(j=0; j<colsize1; j++)
scanf("%d", &a[i][j]);
}
for(i=0; i<rowsize2; i++)
{
for(j=0; j<colsize2; j++);
scanf("%d", &b[i][j]);
}
for(i=0; i<rowsize1; i++)
{
for(j=0; j<colsize1; j++)
{
printf("%d\t", a[i][j]+b[i][j]);
}
printf("\n");
}
}
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, &colsize1, &rowsize2,
                  &colsize2);
if(colsize1==rowsize2)
{
for(i=0; i<rowsize1; i++)
{
for(j=0; j<colsize1; j++)
scanf("%d", &a[i][j]);
}
```

```
for (i=0; i< rowsize2; i++)
{
for (j=0; j<colsize2; j++)
scanf ("%d", &b[i][j]);
}
for (i=0; i<rowsize1; i++)
{
for (j=0; j<colsize2; j++)
{s=0;
for (k=0; k <rowsize2; k++)
{
s= s+ a[i][k] * b[k][j];
}
printf ("%d\t", s);
}
printf ("\n");
}
}
```

$$C\begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix}_{3\times3} = A\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \\ A_{20} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \end{bmatrix}_{2\times3}$$

$r_1 \times c_2$ — $r_2\ c_2$

$C_{ij} = C_{ij} + A_{ik} * B_{kj}$

```
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];
}
```

Matrix Multiplication :-

```
#include<stdio.h>
void main()
{
int a[3][3] ,b[
scanf ("%d%d%
if (c1==r2)
{
for (i=0; i<r1
{
for (j=0; j<c1
scanf ("%d",
}
for (i=0; i<r2
{
for (j=0; j<c
scanf ("%d",
}
for (i=0; i<r
{
for (j=0; j<c
{
c[i][j] = 0
for (k=0;
{
c[i][j]=c
}
}
}
for (i=0
{
for (j=0
printf
printf
}
}
```

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<c1; j++)
scanf("%d", &a[i][j]);
}
for(i=0; i<r2; i++)
{
for(j=0; j<c2; 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\t", c[i][j]);
}
printf("\n");
}
for(i=0; i<r1; i++)
{
for(j=0; j<c2; j++)
printf("%d\t", c[i][j]);
printf("\n");
}
}
```

i-row, k-column; k-row, j-column

c[i][j] = a[i][k] * b[k][j]

(or)

c[i][j] += a[i][k] * b[k][j];

Transpose of a matrix :-

```
#include<stdio.h>
void main()
{
int a[3][3], b[3][3]
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++)
{
b[i][j]=a[j][i];
printf("%d\t", b[i][j]);
}
printf("\n");
}
}
```

Pass 2-D array as an argument to a function:-

```
#include<stdio.h>
void main()
{
int display(int [10][10]);
int i,n, d[10][10],j;
for scanf("%d",&n);
for(i=0; i<n; i++)
scanf("%d",&a[i][j]
{
for(j=0; j
```

```
#include<stdio.h>
void main()
{
int display(int [][10],int ,int );
int i,j, a[10][10], c1;
scanf("%d%d",&r1,&c1);
for(i=0; i<r1; i++)
{
for(j=0; j<c1; j++)
scanf("%d", &a[i][j]);
}
display(a, r1, c1);
}
int display(int a[][10], int r1, int c1){
{
for(i=0; i<r1; i++)
{
for(j=0; j<c1; j++)
printf("%d", &a[i][j]);
}
}
```

if r,c are declared globally and scan in main, no need to pass r,c to func.

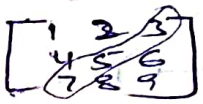
```
#include<stdio.h>
void main()
{
int add (int a[][10], int b[][10], int, int);
int read (int a[][10], int, int);
scanf ("%d%d", &r1, &c1);
for (i=0; i<r1; i++)
for (j=0; j<c1; j++)
scanf ("%d%d", &r2, &c2);
if ((r1==r2) && (c1==c2))
add (a, b, r1, c1);
else
printf ("Addition is not possible");
}
int read (int a[][10], int r1, int c1)
{
for (i=0; i<r1
}
int add (int a[][10], int b[][10], int r1, int c1)
{
int i, j;
for (i=0; i<r1; i++)
{
for (j=0; j<c1; j++)
printf ("%d", a[i][j] + b[i][j]);
printf ("\n");
}
```

→ Print sum of diagon

```
#include<stdio.h>
void main()
{
int a[][10], r1,
scanf ("%d%d",
for (i=0; i<r1;
{
for (j=0; j<c1; j+
scanf ("%d", a
} → if (r1==c1
{
for (i=0; i<r1
{
for (j=0; j<c1
{
if (i==j) (i==
s=s+a[i][j]
}
}
printf ("%d",
}
```

→ Print sum of diagonal elements.

```
#include<stdio.h>
void main()
{
int a[][10],r1,c1,i,j,s=0;
scanf("%d%d",&r1,&c1);
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
scanf("%d",a[i][j]);
}
if(r1==c1)
{
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
{
if(i==j)
s=s+a[i][j];
}
}
printf("%d",s);
}
```



1,5,9 → i=j
(0,0) (1,1) (2,2) => i=j

3,5,7 →
(0,2) (1,1) (2,0) => i+j=n-1

```
#include<stdio.h>
void main()
{
int add(int [][10], int [][10], int, int);
int a[][10], b[][10], i, j, r1, c1, r2, c2;
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<r2; i++)
{
for(j=0; j<c2; j++)
scanf("%d", &b[i][j]);
}
if((r1==r2) && (c1==c2))
add(a,b,r1,c1)
else
printf("Addition is not possible");
}
int add(int a[][10], int b[][10], int r1, int c1){
{
int i,j;
for(i=0; i<r1; i++)
{
for(j=0; j<c1; j++)
{
c[i][j]=a[i][j]+b[i][j];
printf("%d\t", c[i][j]);
}
printf("\n");
}
}
```

Write a prog
for search

```
#include<st
void main
{
int searc
int a[10
scanf("
for(i=0
scanf
scanf(
search
}
int se
{ int
for(i
if(m=
print
if(r
prin
}
int
{
int
for
{
if
re
el
c
}
i
r
}
```

Write a program to create a user defined function for searching an element in an array.

```
#include<stdio.h>
void main()
{
int search(int [][], int, int);
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 triangular of a 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,r1,c1;
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
{
if(i<=j)
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(i=0;i<r1;i++)
{
for(j=0;j<c1;j++)
{
if(i>=j)
printf("%d\t",a[i][j]);
}
printf("\n");
}
}
```

(2)
```
#include<stdio.h>
void main()
{
int a[10][10],i,j,r1,c1;
for(i=0;i<r1;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);
}
}
for(i=0; i<r1; i++)
{s=0;
for(j=0; j<c1; j++)
{
if(i==j)
s=s+a[i][j];
}
}
for(i=r1-1; i>=0; i--)
{
for(i
```

```
if(r1==c1)
{
s=0
```

(3)
```
#include<stdio.h>
void main()
{
int a[10][10], b[10][10], r1, c1, i, j, s;
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++)
b[i][j] = a[j][i];
}
for(i=0; i<c1; i++)
{
for(
if(r1==c1)
for(i=0; i<r1; j++)
{
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-III

- pass by value (call by value)
- Pass by reference (address) (call by reference)

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);
}
```

```
void main()
{
void swap(int, int);
int a,b;
scanf("%d%d",&a,&b);
swap(a,b);
printf("after a=%d, b=%d",a,b);
}
```

Actual arguments

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 are overcome by pass by reference.

-

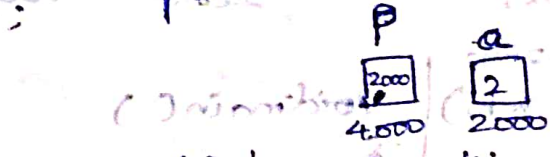
int a; memory allocated with name 'a'.

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.



Since 'a' is int type pointer should also be in same type.

we can print 'a' by calling 'a' or its address.

```
printf("%d", a);      2          (direct)
printf("%d", *(&a));  2
printf("%u", &a);     2000       (indirect).
```

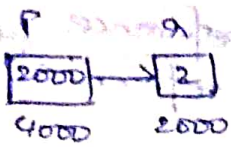
(for address, control string %u)

&-address, *-value at particular address.

*(&a)
= *(2000)
(Value at 2000)
= 2.

```
char ch;
char *p;
p=&ch;
```

Pointer means a link to a.




```
printf ("%u", p);   2000   -> address - unsigned - %u.
printf ("%u", &p);  4000
printf ("%d", *p);  2
```

*(2000)
Value at 2000
= 2

*(&p)
=> 2000

((&p))
= 2

Pass by reference :-

```
                    int *a=&a
void swap(int *a, int *b)
{
int c;
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)
{
int c;
c=*a;
*a=*b;
*b=c;
printf("a=%d, b=%d", *a, *b);
}
```

```
void main()
{
void swap(int, int);
int a, b;
scanf("%d%d", &a, &b);
swap(&a, &b);
Printf("after a=%d, b=%d", a, b);
}
```

=> this effects the original address, so it effects the actual parameters.

Output :- ~~after a=5, b=3.~~ a=5, b=3.
after a=5, b=3.

Pre-processor directives :-

- Writing a code.
- preprocessing ⟶ # - hash / pound. - preprocessing.
- compilation.
- linking.
- Execution.

variable - varies
```
int a=2;
a=5;
a=6;
```

constant - fixed #define ⇒ 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 (or) variable.
in header file section.

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 data types. using (typedef)

```
#define integer int
#define cls clrscr(); cls.
#define found printf("simple")
#define cls clrscr()  cls;
#define AND &&
#define double(a) a*2
```
: double(2); 2x2=4.

- Conditional compilation:-

- file inclusion:- #include <stdio.h> } Header file to include files.
  #include "stdio.h"

— Conditional compilation :-

```
#if def
#else
#endif
```

#if – to check whether preprocessing derivatives are defined or not.

```
#define E =
main()
#if E
aE2;
bE3;
#else
a=2;
b=3;
#end if.
```

#ifndef – if not defined

```
#ifndef E
a=2;
b=3;
#else
aE2;
bE2;
#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=&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,(*p);
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 numbers from m to n.

```
#include<stdio.h>
void main()
{
int m,n,k,l;
int sum(int*, int*, int*);
int avg(int*, int*, int*);
scanf("%d%d", &m,&n);
sum(&m, &n, &k);
avg(&m, &n, &l);
printf("%d%d", *k, *l);
}
```

```
int sum (int *m, int *n, int *k)
{
if (m<n)
while (m<=n)
{
*k = *k + *m;
}
}
int avg (int *m, int *n, int *()
{
if (m<n)
while (m<=n)
{
*k = *
```

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 *()
{
if (m<n)
While (m<=n)
{
*k= *k + *m;
*l = *l+1;
}
if (n<m)
while (n<=m)
{
*k=*k + *m;
*l = *l+1;
}
```

Area of a

```
#include<s
#define P
#define P
#define
Void main
{
int a,A
int area
Scanf("
Sf("%d
area(
Pf("
}
int ar
```

To print

Area of a circle:-

```
#include<stdio.h>
#define P 3.14
#define pf printf
#define sf scanf
void main()
{
int r,A;
int area(int *, int *);
scanf("%d",&r);
sf("%d",&r);
area(int &r, int&A);
pf("%d",A);

}
int area(int *r, int *A)
{
*A= P*r*r;
}
```

~~To print Hello world~~

Storage Classes :- (Tells Initial value) (default value)

int a;

storage class. (automatic storage class)

if not return, then it automatically has some storage class.

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.

→ It also tells the lifetime of a variable. (upto which the variable is alive)

→ 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 ({...})

```
Ex:- main()
     {
     int a=3;
     {
     a=1;
     printf("
     }
     a=0;
     printf
     }
     }
     printf("
     }
```

```
Ex:- main()
     {
     incc();
     incc();
     incc();
     }
     incc()
     {
     auto in
     printf("
     i++;
     }
```

(2) Static storag

initial value :-

```
main()
{
static int
printf("%d"
}
```

Scope :- only

```
Ex:- main(
     {
     inc();
     incC);
     incC);
     }
     inc()
     {
     static
     printf(
     i++;
     }
```

Ex:-
```
{
int a=3;
{
a=1;
printf("%d",a);
{
a=2;
printf("%d",a);
}
}
printf("%d",a);
}
```

scope only to a specific block (braces)

Ex:-
```
main()
{
incc();
incc();
incc();
}
incc()
{
auto int i=1;
printf("%d",i);
i++;
}
```

Here 'i' variable is local.

Output:- 1 1 1

(2) Static storage class:- Keyword:- static

initial value :- 0.

static:- it fixes the modified value.

```
main()
{
static int i;
printf("%d",i);
}
```

Output:- 0

Scope:- only to a specific block.

Ex:-
```
main()
{
incc();
incc();
incc();
}
incc()
{
static int i=1;
printf("%d",i);
i++;
}
```

'i' is modified and it fixed (static)

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:-
'0'

```
int i;
main()
{
i=1;
fun1()
{
i=2;
}
```

(5)

main.c

```
#include "origin.c"
extern void fun();
main()
{
int c=10;
printf("%d",c);
fun();
}
```

origin.c

```
#include "main.c"
extern int c;
void fun()
{
c=100;
}
```

we can access 'c' in another file

Table

Properties of Algorithm, flowchart.

Storage	Default Initial Value	Scope	Life
Memory	Garbage value	local to the block in which the variable is defined.	Till the control remains within the block in which the variable is defined
stomatic			

	Storage	Default Initial Value	Scope	Life
Automatic	Memory	Garbage value	local to the block in which the variable is defined.	Till the control remains within the block in which the variable is defined.
Register	Registers	Garbage value	local to the block in which the variable is defined.	Till the control remains within the block in which the variable is defined.
Static	Memory	Zero	local to the block in which the variable is defined.	Value of the variable continues to exist between different function calls.
External	Memory	Zero	Global	Till the program's execution doesn't come to an end.

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 = 3.141414;
printf ("%d", f);
```

internally converted.

output = 3

implicit type conversion:- internally converted, the system converts automatically.

explicit:- (type casting)

```
int a=2, b=3;
float
c = a/b
printf ("%f", c);
```

1.000000

```
int a=2, b=3;
float c;
c = (float) (a)/b;
printf ("%f", c);
```

1.500000.

---

square root of an integer
Decimal to binary.
summation of set of numbers
exchanging values of two numbers.

low-level	middle-level	high-level
Assembly level	C-language	English language

1972, C-language, Dennis Ritche
Previous B-language - only one data type

loops:- entry loop, exit loop

check condition at start	check condition at last
Ex:- for, while	Ex:- dowhile

Problem Definition

Similarities bln

Problem solving s
strategies to be
(1) Divide & con
divide into su
Ex:- searchi

(2) Dynamic progr
for getting
- generally, for so
we get optima

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

(3) Backstracking
tracking best
finding sh
if a good so
and goes to
Ex:- recu

(4) Greedy Me

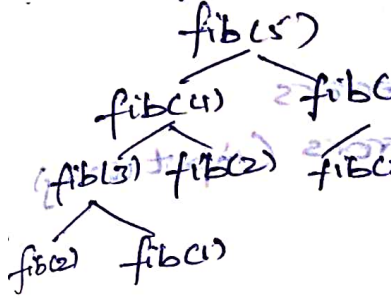
Problem Definition :- Input - may or maynot be
Output - one or more outputs

Similarities bln problems :- common problems (sorting, swapping etc.)

Problem solving strategies :- (4)
strategies to be followed while solving any problem.

(1) Divide & conquer strategy :-
divide into subproblems & group all (conquer)
Ex :- searching - Linear search
- 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 :- Here fib(3) is called two times, when fib(3) is called first, it stores the value & when it is called 2nd 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:-

Errors detected in execution - logical errors
- data errors (input (wrong))

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 a[6] = "geek"
```

a & &a - same

a = "hello" - not valid

a[0] = 'b' - valid

a++ - invalid

```
char *p = "geek"
```

p & &p - not same.

p = "India" - valid

p[0] = 'k' - invalid.

p++ - valid.

strcpy - doesn't allocate new memory } both copies strings data.

strdup - allocates new memory

Pointers

A variable is a named memory lo

A pointer is a variable that contains the memory location (address) of another variable.

Syntax:- type * variable name;

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 to something. Assign the address of the variable you want to point to the pointer

ptr = #

x 10

ptr &x

```
x=10
ptr=&x printf("%d", *ptr);
main()
{
int a=5, b=6;
int *c;
c=&a;
}
```

Memory	Value
a(1001)	5
b(1003)	6
c(1005)	1001

Ex
```
#include<stdio.h>
void main()
{
int num, *intptr;
char ch, *cptr;
float x, *floptr;
num=123;
x=12.34;
ch='a';
intptr=&num;
cptr=&ch;
floptr=&x;
printf("%d", intptr);
printf("%d", floptr);
printf("%d", cptr);
}
```

Pointer Arit

Incrementing

to the next

Ex:- int

P=&

P=P

P++

decrementing

Pointer Add

variable

we cant

int *P1, *

P1+P2;

Subtraction

int *P

P=P-

P1 - P2 →

*P1 - *P2

Pointer com

int *

P1 = &a

P2 = &

P1 ==

When

may le

not al

Pointer Arithmetic

Incrementing :- increased by the size of datatype to the next location.

Ex :- `int *p;`
`p=&a;` 1025
`p=p+1;` 1027 (+2)
`p++;` 1029 (+2)

decrementing :- `int *p;` to the previous location
`p=&a;` 1027
`p--;` 1025.

Pointer Addition :- We can add a value to pointer variable

```
int *p;
p=p+3
```

We can't add two pointer variables.

`int *p1, *p2;` 1024, 1034
`p1+p2;` - 1024 + 1034 = 2058 - not sufficient memory

Subtraction :- subtract a value from pointer variable

```
int *p;
p=p-3;
```

```
int *p1, *p2;
p1-p2;
no. of elements = (p1-p2)/2;   // in array.
```

$p_1 - p_2$ → pointer arithmetic

$*p_1 - *p_2$ → performing on value i.e expression.

Pointer comparisons :- using relational operators ==, <, &.

```
int *p1, *p2;
p1=&a;
p2=&a;
```

p1 == p2 → True.

When perform division on pointers → $\frac{1024}{1025}$ it 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;
}
```

if static is not placed it takes as auto storage class - then address releases (expires) after coming out of the function, then it displays garbage value.

Pointer to Pointer

```
int *ptr1;
int **ptr2;
int ***ptr3;
```

ptr2 can store address of a pointer.
ptr3 can store address of a double Pointer.

Pointer to Array

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

printf ("%d",*p); → prints the array, by increm. index

printf ("%d", a[i]); → will also print elements.

i[a]); → address of all elements.

a+i); → prints value of elements

*(a+i));

*a); → print value of a[0] only.

a++; → error : we cannot change base address of array.

*(a+i) is same as a[i]

Two dimensional: (*(a+i)+j) - address of a[i][j].

((a+i)+j) - value of a[i][j]

Array of pointers

int *p1, *p2,
datatype *arr

Pointer and char

char *str;
str = "Hello";
printf ("%s", str);

Table of strings

char s1[0], s
(or) char str[
if represent

Void pointer :-

datatype.
void *p;
can store
of any
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 a[4];
datatype *array_name[size];
```

Pointer and character strings:-

```
char *str;          (✓)
str = "Hello";
printf("%s",str);
```

```
char str;
str = "Hello" ; (X).
```

```
char *str[10];
```

Table of strings:- List of strings. - array of strings.

```
char s1[10], s2[10], ..... , s5[10];
```

(or) `char str[25][10];` → Table of strings

if represented by pointer :- `char *str[25];`

Void pointer :- A generic pointer that can point to any datatype.

```
void *p;
```

can store address of any variable.

int *p → can store address of int variable only.

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 = f   — correct
ptr = &i  — correct
```

```
int main()
{
int a=7;
float b=7.6;
Void *p;
p=&a;
printf("%d",(*(int*)p));
p=&b;
printf("%f",(*(float*)p));
```

Pointer to Functions:-

```
Void (*pf)(int);
```

`void *pf(int)` ⇒ a function that returns a void pointer.

```
main()
{
void (*pf)(int);
Pf = &somefunction (or)
Pf = somefunction;
Pf(5); (or) (*pf)(5);
}
```

```
void somefunction(int a);
{
}
```

```
Void *display(int arg);
```

```
void *pf(int)
pf = display.
```

Calling multiple functions using pointer variable:-

```
main()
{
void (*ptr);
```

function pointer of array type. This array is holding addresses of functions.

```
void (*fun_ptr_arr[])(int,int) = {add, subtract, multiply};
(or) {&add, &subtract, &multiply.
unsigned int ch, a=15, b=10;
choice :- 0 - add, 1 - sub, 2 - multiply
if (ch>2)
return 0;
(*fun_ptr_arr[ch])(a,b);
return 0;
}.
void add(int a, int b);
{
Print a+b;
}
void subtract(int a, int b)
{
print a-b;
}
Void multiply(int a, int b)
{
print a*b;
}
```

Dynamic mem

allocating
Array is co
size of an
Sometimes
wasted.
There are
malloc()
Static
Dynami

malloc():-

stands for
allocates
it returns

Syntax:-

```
stdlib.h
main()
{
int n,i,
scanf(
ptr = (
if (ptr
{
printf
exit
}
Printf
for(i
{
Scan
Sum
}
prin
free
}.
```

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 :- ptr = (cast_type*) malloc (byte_size);

malloc returns a void type.

to specify the type of ptr cast_type is required - to convert.

stdlib.h

```
main()
{
int n,i, *ptr, s=0.
scanf("%d",&n);
ptr = (int*) malloc (n* sizeof(int));
 if (ptr==NULL)
 {
 printf("Unable");
 exit 0;
 }
 printf ("Elements");
 for(i=0; i<n; i++)
  {
  scanf ("%d", ptr+i);
  sum= sum+ *(ptr+i);
  }
 printf("%d", sum);
 free(ptr);
}
```

to allocate memory.

if the compiler executes memory, later it releases memory automatically
Eg:- functions.
For malloc & calloc it is not released automatically, so free() function is used.

→ if here we print again, it prints. because the free function releases the memory only after it comes out of main functi

calloc() function :-

"Calloc" stands for contiguous allocation.
allocates multiple block of requested memory.
It returns NULL if memory is not sufficient.

Syntax :- ptr = (cast-type *) calloc (number, byte-size);

Ex :-
```
#include<stdio.h>
#include<stdlib.h>
main()
{
int n,i,*ptr,sum=0;
scanf("%d",&n);
ptr=(int*)calloc(n,sizeof(int));
for(i=0;i<n;++i)
{
scanf("%d",ptr+i);
sum+=*(ptr+i);
}
printf("%d",sum);
free(ptr);
}
```

Free() :-

Dynamically allocated memory malloc & calloc. doesn't freed automatically so use free().

realloc() :- Syntax :- ptr = realloc(ptr, new_size);

Ex :- ptr = (int*) malloc (n1*sizeof(int));
ptr = realloc(ptr, n2*sizeof(int));
again we have to enter new data.

Command Line Argu

command line argume
program when it
C.L.A are passed
gcc ex
Syntax :-
int ma
int main(int
{int i;
printf("The
for(i=0;i<a
printf("%s
if(no argume
Ex :- C:/docu
argv[0]=
argv[1]=
argv[2]
argv[3

Command Line Arguments:-

Command line argument is a parameter supplied to the program when it is invoked.

C.L.A are passed to the main method.

gcc ex.c i/a inputs arguments

Syntax:- `int main(int argc, char *argv[]);`

argc → count of arguments

argv → pointer character array

```
int main(int argc, char *argv[])
{
int i;
printf("The arguments supplied are:\n");
for(i=0; i<argc; i++)
 printf("%s\t", argv[i]);
```

if no argument is supplied, argc will be 1.

Ex:- C:/documents/Cprograms> ./a red green blue.

- argv[0] == ex.c
- argv[1] == red
- argv[2] == green
- argv[3] == blue.

./a 10 20 30 → These are also stored as strings

Structures:- is a derived datatype. Collection of different datatypes. Heterogeneous collection.

Ex:- Student data :- name, roll no., branch, section, address.

A single struct would store the data for one object.

Declaration of structures:-

```
struct structure_name tag.
{
Datatype member1;
Datatype member2;
:
Datatype member3;
};
```

fundamental :- int, char, float
user-defined :- typedef
derived :- arrays, struct
empty :- void

Ex:-
```
struct rectangular_prism
{
int height;
int width;
int length;
};
```

```
struct student_record
{
int ID;
char name[100];
};
```

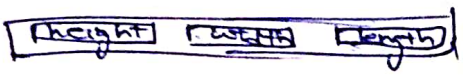
Does not reserve space (memory)

To reserve memory

```
struct tag
{
:
} mystruct;
```
→ structure variable.

To access structure members:- obj

struct rectangular_prism obj;



We can create any no. of variables (or) objects to the same structure.

```
struct student_record
{
int ID;
char name[100];
};
struct student_record s1, s2, s3, s4;
```

(or)

```
:
:
} s1, s2, s3, s4;
```

Initialisation

(1) struct name
{
char name[
int height;
int weight
} n1 = {"
n2 = {
n3 = {

(2) struct name

(3) struct nam
n1.name="
n1.height
n1.weigh

Accessing stru
operator (
"struct_vari
n1.name
If a pointer
struct student
(or)
st->it so
for acces

(or)

Struct rect

Ex:-
#in
str
{
ch
int
flo
};(
in
{
P

Initialisation 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}, n2 = {"Sue", 176, 56};

3) struct names n1, n2;

n1.name = "Jane";	n2.name = "Jane";
n1.height = 160;	n2.height = 160;
n1.weight = 50;	n2.weight = 50;

Accessing struct members:- using structure member operator (dot ".").

"struct_variable/object.member;

n1.name;

If a pointer to the struct is declared.

struct student_record *ptr = &s1;

(or)

struct some_name *ptr = &mystruct;

for accessing :- ->

ptr->id; (or) myptr->letter;

(or) (*ptr).id (or) (*myptr).letter;

~~struct rectangular *p~~

Ex:-
```
#include<stdio.h>
struct student
{
char name[50];
int roll;
float marks;
} s;
int main()
{
printf("Enter information:\n");
```

(or) struct student s;

```
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.

Student record - rollno., address, ...
↓
streetno., city, state & pincode.

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;
} s1;
```

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;
} s1;
```

struct Date dob; cannot be done.

To access date
s1.dob.dd.

Initialisation :

Ex:- #include
struct s

User defined d
for previously
Ex:- typed
length o
can now
same wa
length
length
in structure

Enumeratio
using the

- The n
conste

To access date of birth: outerstructure.nestedstructmember
st.dob.dd.

Initialisation :- student record st = {99, "A.Nonymous", {1,4,2003}};

Eg:- #include <stdio.h>
= struct student

User defined datatypes (typedef) :- to create synonyms for previously defined datatype names.

Eg:- 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,b;
length numbers[10];

in structures:-

```
typedef struct
{
int label;
char letter;
char name[20];
} some_name;
```

The "alias" is some_name

struct some_name mystruct; → 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~~ a symbolic constants ~~that take~~ integers from 0 to n.

Array of Structures :-

collection of multiple structure variables where each variable contains information about different entities.

for 5 students data → 5 structure variables
for 60 → 60 stru. → but same structure.
$(s_1, s_2, \ldots, s_{60})$ variables.

instead of declaring 60 diff. variables for same structure, so we can represent an array.

Once a structure is defined,

```
struct student class[50];
struct student s1,s2,s3,s4; => s[20];
```

The individual members can be accessed as:

```
class[i].name.
class[5].roll_number;
```

```
struct student s[20];

for(i=0;i<20;i++)
{
scanf("%d",&s[i].d);
scanf("%s",&s[i].name);
}
```

```
struct employee
{
int id;
char name[5];
float salary;
};
struct employee emp[2]
```

sizeof(emp) = 4+5+4 = 13 bytes.
sizeof(emp[2]) = 26 bytes.

Array within structures :-

```
struct student
{
char name[30];
int roll_number;
int marks[5];
char dob[10];
} a1, a2, a3;
```

a1.marks[2], a1.dob[3], ...

```
struct student
{
int roll;
char name[5];
int marks[3];
} s[60];

for(i=0;i<60;i++)
{
scanf("%d",&s
scanf("%s",s
for(j=0;j<3;j+
printf
scanf("%d",&s
}.
```

Structures and F

→ A structure can
main function (or)
→ Structures can be
→ Structure definition
-tion only
→ It won't be av
passed.
→

```
struct student
{
int_
char_
}
```

Passing structure
→ Passing each

```
struct student
{
int roll;
char name[25];
int marks[3];
}s[60];
for(i=0;i<60;i++)
{
scanf("%d", &s[i].roll);
scanf("%s", s[i].name);
for(j=0;j<3;j++)
scanf("%d", &s[i].marks[j]);
}.
```

array of structures and array within structures

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.
{
```

→ Write an example program to create a structure with members of pointer type.

==.

```
#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 functions. Even main() can be called from other functions:

```
#include<stdio.h>
void message();
int main()
{
message()
return 0;
}
void message()
{
printf("---");
main();
}
```

→ A function can be called another functions, 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 fun()
{
int n;
scanf("%d",&n);
if(n>=10&&n<=90)
return (n);
else
return (n+32);
}
```

```
#include<stdio.h>
void display(int);
int main()
{
int i=20;
display(i);
return 0;
}
void display(int j)
{
int k=35;
printf("%d\n",j);
printf("%d\n",k);
}
```

$f(5) =$

$fib(4) + f(3)$

$f(3) + f(2) + f(2) + f(1)$

$f(2) + f(1) + 1 + 1 + 0.$

$1 + 0 + 1 + 1 + 0$

$= 3.$

$f(5) =$

$f(4) + f(3)$

$f(3) + f(2) \quad f(2) + f(1)$

$f(2) + f(1) \quad 1 \quad 0, 1$

$1 + 0$

0, 1, 1, 2,

f(0) f(1) 0 1 1 2 3 f(2)+f(1) f(3)+f(2)

OPERATORS :-

Arithmetic Operators :- +, -, *, /, %.

Logical Operators - &&, ||, !

Relational Operators - <, >, <=, >=, ==, !=

Increment and Decrement Operators :- i++, ++i, i--, --i.

Short hand assignment operations :- operator= i*=2.

Conditional operator (Ternary operator) :-

(condition) ? True statement : False statement ;

bitwise operators :- &, |, ~, <<, >>,

comma operator :- ,

Data types :-

integer	—	int	%d
character	—	char	%c
float	—	float	%f
double	—	double	%lf

Datatypes

- 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.
- Iterative statements
  - for
  - while
  - do while
- Jump statements
  - break
  - continue
  - to go.

if else syntax:-

```
if (condition)
{
statements;
}
else
{
statements;
}.
```

for syntax:-

```
for(initialization, condition, updation)
{
statements;
}.
```

Ex:-

```
scanf("%d"
for(i=1; i<=
{
printf("He
}
```

do while syntax:-

```
initialization;
do
{
statements;
updation coun
} while (condi
```

switch:-

```
switch(choice
{
case constant1 : State
                 bre
case constant2 : Sta
                 brea
default : Sta
}
```

```
#include<st
void main(
{
int n,i,j,a,b,c
scanf("%d
for(i=1; i<
{
scanf("%c
printf("
```

scanf(

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;
do
{
printf("Hello World");
&i++;
} while(i<=n);
```

switch:-
```
switch(choice)
{
case constant1: statements;
        break;
case constant2: statements;
        break;
default: statements;
}
```

```
#include<stdio.h>
void main()
{
int n,i,a,b,ch;
scanf("%d",&n);
for(i=1; i<=n; i++)
{
scanf("%d%d",&a,&b);
printf(" 1. addition\n
        2. subtraction\n
        3. multiplication\n
        4. division\n.
        5. Modulo division\n");
scanf("%d",&ch);
```

infinite loop (or) for(;;) (or) while(1)

for 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");
}
}
}
```

for infinite loops:-

```
char choice;
while(1)
{
scanf("%d%d", &a, &b);
scanf("%c", &ch);
switch(ch)
{
case '1': ____
          break;
default: ____ ;
}
printf("do you want to continue\n
        Y-yes\n
        N-no");
scanf("%c", &choice);
if(choice == 'N')
break;
}.
```

Output:-

```
2  3
1
5
do you want to cotinue
Y-Yes
N-No
Y
2  3
2
-1
do you want to continue
Y- Yes
N-No
Y
2  3
2
6
do you want to continue
Y-Yes
N-No.
N
```

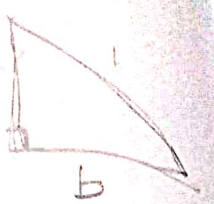
Output:-

```
2  3
1
5
do you want to continue
Y-Yes
N-No
Y
```

Swapping two num

```
void main()
{
int
int swap(int, int)
scanf("%d%d",
Printf("%d%d",
swap(a,b)
}
int swap(int
{
a=a+b;
b=a-b;
a=a-b;
```

Write a program using function that calculates the hypothenuse 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(pow(a,2)+pow(b,2));
}
```

Write a function that accepts an integer b/n 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;
```

Writ a Program
number if leap yea

```
#include<stdio.h>
void main()
{
int leap(int);
int
scanf("%d",&n)
if (leap(int)==0)
printf(" The giv
else if (leap(n)
printf("The gi
}
int leap(int n)
{
if (n%4==0)
{
if (n%100==
{
if (n%400==
return 0;
else
return 1;
}
else
return
}
else
return
}
```

Write a p
-sion.

```
#include<s
void main(
{
int rev(
```

Write a program to check whether the given number if leap year or not.

```
#include<stdio.h>
void main()
{
int leap(int);
int
scanf("%d",&n);
printf
if (leap(int)==0)
printf (" The given year is a leap year");
else if (leap(n)==1)
printf ("The given year is not a leap year");
}.
int leap (int n)
{.
if (n%4==0)
{
if (n%100==0)
{
if (n%400==0)
printf ("%d is a
return 0;
else
return 1;
}
else
return 0;
}
else
return 1;
}.
```

Write a program to reverse a number using recursion.

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

```
int n;
scanf("%d",&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;
}
```

~~#include<stdio.h>~~

Write a program to compute F(x,y) where F(x,y) = F(x-y,y)+1 if y≤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<=x)
return F(x-y,y)+1;
```

Explain file inclusion and conditional compilation

file inclusion:- to create a header file to include files into the program.

#include "headerfile.h".

Conditional compilation:- To check whether the identifier is defined or not by the Macros.

Ex:-

```
#include <stdio.h>
#include <math.h>
#define sq(p) sqrt(p)
int main( )
{
int a, b;
#ifdef sq(p)
{
printf ("%d", sq(a));
}
#else
{
printf ("%d", sqrt(a));
}
#endif
#ifndef sq(p)
{
printf ("%d", sqrt(b));
}
#else
{
printf ("%d", sq(b));
}
#endif
}
```

Write a program to find factorial of a number using recursion

```
#include <stdio.h>
void main( )
{int fact (int);
int n, d;
scanf ("%d", &n
d=fact (n);
printf ("%d"
}
int fact (int n
{
if (n==1)
return 1;
else
return n*fa
}
```

Write a program to find GCD of two numbers using recursion

```
#include <s
void main(
{
int gcd (
int a,b;
scanf ("
printf (
}
int gcd
{
if (x>
{
y=x%y;
if (r=
return
else
return
}
```

to include

eck whether
the Macros.

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);
}
```

'main' is user defined function.

Write a program to find gcd of a given 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)
{r=x%y;
return gcd(y,r);
if(r==0)
return y;
else
return gcd(y,r);
}
else
{
x=x+y;
x=x-y;
x=x-y;
r=x%y;
if(r==0)
return y;
else
return gcd(y,r);
}
```

fibonacci series :-

```
#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);
}
}
```

(non-recursive)

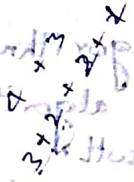
```
#include<stdio.h>
void main()
{
int fib(int);
int n;
scanf("%d", &n)
fib(int n);
}
int fib(int n)
{
int a=0,b=1;
printf("%d%d", a,b);
for(i=0; i<n; i++)
{
c=a+b;
a=b;
b=c;
printf("%d",c);
}
}
```

recursive :-

```
#include<s
void main(
{
int n
int fib(i
scanf("
fib(n);
}
int fib(
{
if(n==
printf
if(n==
printf
if(n>
printf
}
```

recursive:-

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



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 atleast 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
Sequence
The algorithm
types – (1)
(3) Loop (Re
(1) Sequence
are place
takes pla
(2) Branchin
is a condi
decision of
If the cond
-ed, if the
-tive is
to repre
(3) Loop
a state
on a cer

To draw
used:-
Symbol N
Oval
Parallelo
Rectangl
Diamond
Arrows

→ 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 up to down.

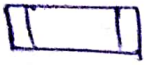
(2) Branching (Selection) :- In a branch control, there is a condition and according to a condition, a decision of either TRUE or FALSE is achieved. If the condition is True, one of the branches is explored; if the condition is false, the other alternative is explored. Generally the 'IF-THEN' is used to represent branch control.

(3) Loop (Repetition) :- The loop or repetition allows a statement(s) to be executed repeatedly based on certain loop condition. Ex:- WHILE, FOR loops

To draw a flowchart following standard symbols are used :-

Symbol Name	Symbol	Function
Oval	Terminal	Used to represent start and end of flowchart
Parallelogram	Input/Output	Used for any Input/Output operation. Indicates that the computer is to obtain data or output results
Rectangle	Process	Indicates any type of internal operation inside the Processor/Memory.
Diamond	Decision	Used to represent the operation in which there are two/three alternatives.
Arrows	Flowlines	Shows direction of flow.


connector : Allows the flowchart to connect from one page to other.


Predefined process.

Data types in C Language :-

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 programs. 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 datatypes but a little twisted or grouped ~~too~~ 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 data-type 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~~

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 :-** +, −, *, %, / and %.

+ → Addition or unary plus.
− → Subtraction or unary minus.

The modulo division operator % cannot be used on floating point data.

(2) **Relational Operators :-** also known as comparison operator. 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, less than or equal, greater than or equal.

Equality operators :- ==, !=
equal to, not equal to.

The equal to oper
on the both si
value, otherwise
The not equal to
operators to. op
else it retur

(3) Logical Ope

There are thr

Logical AND :-
two conditio
operations.
If expressio
operator is
true.

Logical OR :-
two conditio
operators.
left sides, rig
then the

Logical NO
Single exp
expression
expression.

A	B
0	0
0	1
1	0
1	1

Logical
fashion
for sur
For exa
logical
the se
sure the

The equal-to operator (==) returns true (1) if operands on the both sides of the operator have the same value, otherwise it returns false (0).
The not equal-to operator (!=) returns true (1) if the operands do not have the same value, else it returns false (0).

(3) Logical Operators:- (&&) (||) (!)

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, right 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||B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Logical 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 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.

(4) Assignment Operators :- The assignment operator is responsible for assigning values to the variables. ('=')

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 :- x++ (or) x--

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;

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 its value becomes the value of the expression.

(7) Bitwise operators

Used for manipu

The bitwise operati

be integers and

bits.

Bitwise AND :- (&) It

operand with

second operand

corresponding

otherwise.

Bitwise OR :- (|)

operand with

second operand

the correspond

0 otherwise.

Bitwise XOR :-

first operan

its second

corresponding

A	B	A
0	0	
0	1	
1	0	
1	1	

Bitwise NOT :

operation

bit of the

-lement of

~ 101

Shift oper

To shift

for left-s

the left

lost

(4) Bitwise operators:-

Used for manipulation of data at bit level.

The bitwise operators expect their operands to be integers and treat them as a sequence of bits.

Bitwise AND :- (&) 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 :- (|) 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 :- (^) 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.

same → 0
diff → 1

A	B	A & B	A | B	A ^ B
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.

~ 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 x is shifted to the right by one place. So, the LSB of x is lost, MSB of x is set to 0.

Comma Operator :-

Control Statements :-

Specify the flow
order in which
be executed. They
make decisions, to
jump from one se

→ To control the e

Contr

Conditional Statements)
Decision making

- if - simple if
- if else
- nested if
- else if ladde
- switch

(1) if

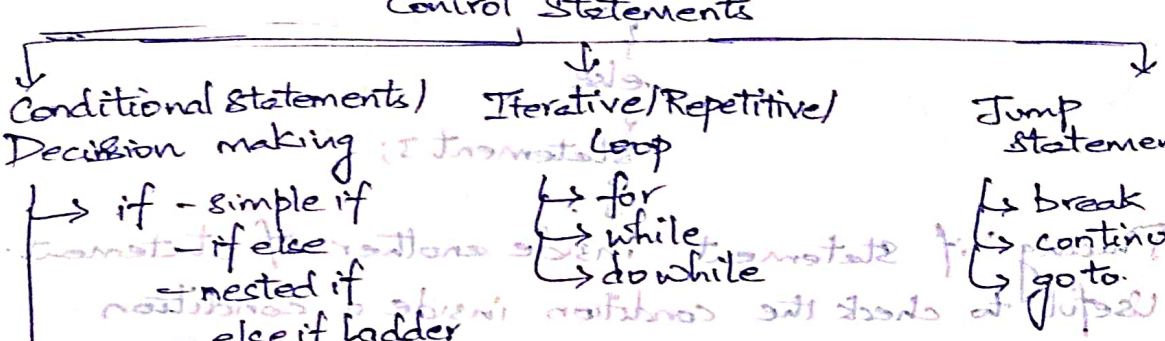
(a) Simple if :-

Statement block
group of statem
if the test-expr
will be execut
will be skipp

(b) if - else :-

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 making make decisions, to perform tasks repeatedly or to jump from one section of code to another.

→ To control the execution of the program.



(a) Simple if :- Syntax:-

```
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:- Syntax:-

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

Placing if statement inside another if statement.
Useful to check the condition inside a condition.

(d) else if ladder :- Syntax :-

```
if (condition 1)
{
statements 1;
}
else if (condition 2)
{
statements 2;
}
else
{
statements 3;
}
```

Helps user to decide among multiple options.

(2) 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 statement.

(Menu driven program)

Syntax :- switch (che
{
case value
case value
default
}

Iterative / Repet

(1) for :- A for
statements
conditions to
condition is

Syntax :- for
{
st

The initialis
Then, the co
-tion is eval
the body
updation is
Again the
If the co
loop is ter
The process
condition
the loop

(2) while :-

Syntax:- switch (choice)
{
case value 1 : statement 1;
break;
case value 2: statement 2;
break;
default : statements;
}.

Iterative / Repetitive / Loop :-

(1) for :- A for loop enables a particular set of statements ~~conditions~~ to be executed repeatedly until a condition is false.

Syntax:- for (initialisation , condition, updation)
{
statements;
}.

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;
while (condition)
{
statements;
Updation;
}

Initialisation takes place.
The while loop evaluates the condition inside the paranthesis ( ).
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 ~~exec~~ 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 :-

(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 ie., in the case of nested loops, it breaks the inner loop first and then preceeds to outer loops.
Syntax :- break;

(2) Continue :- Instead of forcing termination like break, it forces the next iteration of the loop to take place & skipping any code in between.
Syntax :- continue;

(3) goto :- ~~bata~~
It transfers th
statement to a
is defined.
It can transfer
same block and
you want to
Defining a labe
Transferring cont

FUNCTIONS :-
A function is
-ther perform
atleast one
There are two
(1) Predefined
some function
hence they
Ex :- scanf (
pow (,
etc.,
to use the
include th
(2) User Defi
defined by
program.
Parts of fu
(1) Function
(3) Function
(1) Function
It tells
return typ
Syntax :- re

(3) goto :-

It transfers the program's control from one statement to another statement where label is defined.

It can transfer the programs control within the same block and there must be a label, where you want to transfer the control.

Defining a label :- labelname:

Transferring control :- goto labelname;

FUNCTIONS :-

A function is a group of statements that together perform a task. Every C program has atleast one function, which is main().

There are two types of functions in C.

(1) Pre defined functions :- The system provided 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 include the appropriate header files.

(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.

(1) Function declaration :- (Function prototype :-

It tells the compiler about the function's name, return type and parameters.

Syntax :- returntype functionname(argument1, argument2);
(data type names)

Arguments names are not important in function declaration, only their data type is required.
Eg:- int max (int, int);

(2). Function declaration definition:-
It consists of a function header and function body.

Syntax:- returntype functionname (argument1, argument2),
{
function body;
}

Returntype:- If the function returns the value, the returntype is the data type 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 data type is also required.

(3) Function call:- To use a function, you will have to call that function to perform the defined 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.

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 :- arrayname[size]

An array is a collection of values of same type, accessed using a common name.

One-dimensional is like a list and two-dimensional is like a table (matrices)

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][columnsize];

Storing values in arrays :-

- Initialise the elements
- Input values for the elements
- Assign values to the elements

(1) **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][colsize] = { list of values of row & column };

(or)

type name[rowsize][] = { list of values of row and column };

Here 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 value 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 of individual elements of the array by using assignment operator.

Ex :- a[3] = 10;

(or) a[3][5] = 11;

Types of array

One-dimensional

name[]

more than on

Two dimensional

the elements

and columns

Passing

To pass an

the array

function.

Use subs

indicate

Use name

1-D

```
}
int fun (int
int a[5];
fun(a);
..
..
}
int fun (
{
..
..
}
```

Parameter

a) Pass b

Ex) Pass

be cha

value

the

chang

curren

Types of arrays :- based on dimension
One-dimensional , Two-dimensional , Three dimensional
name[] name[][] name[][][]
more than one-dimensional :- Multidimensional array

Two dimensional array is used for the 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 ag argument in the function.
Use subscripts [] in the function declaration to indicate an array.
Use name & [] in the function definition

1-D

```
int fun (int []);
int a[5];
fun (a);
}
int fun (int a[])
{
}
```

2-D

```
{
int fun (int [0][]);
int a[10][10];
fun (int a[10][]);
}
int fun (int a[10][]).
{
}
```

Parameter passing Techniques:
(1) Pass by value (2) Pass by reference

(1) 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 only, not for the main function

(2) 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 is reflected inside as well as outside the function.

Ex:-
```
#include<stdio.h>
void main()
{
int a=50, b=70;
int swap(int *, int *);
swap(&a, &b);
printf("%d", a);
printf("%d", b);
}
int swap(int *a, int *b)
{
int c;
c=*a;
*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 soones a newline character is found, the preprocessor directive is considered to end. No semicolon (;) can be placed at the end of a preprocessive directive.

Types of preproces

↓ Unconditional

↓ ↓ ↓ ↓

define line undef inc

#define is also

Simply a macr

Syntax :- #

Every occur

It is simply

Substitution

File inclusion

File inclusion

-tive tells

source code

Conditional

The code

true.

It is use

macro de

#ifdef

#defi

main

{

#ifde

{

state

}

#els

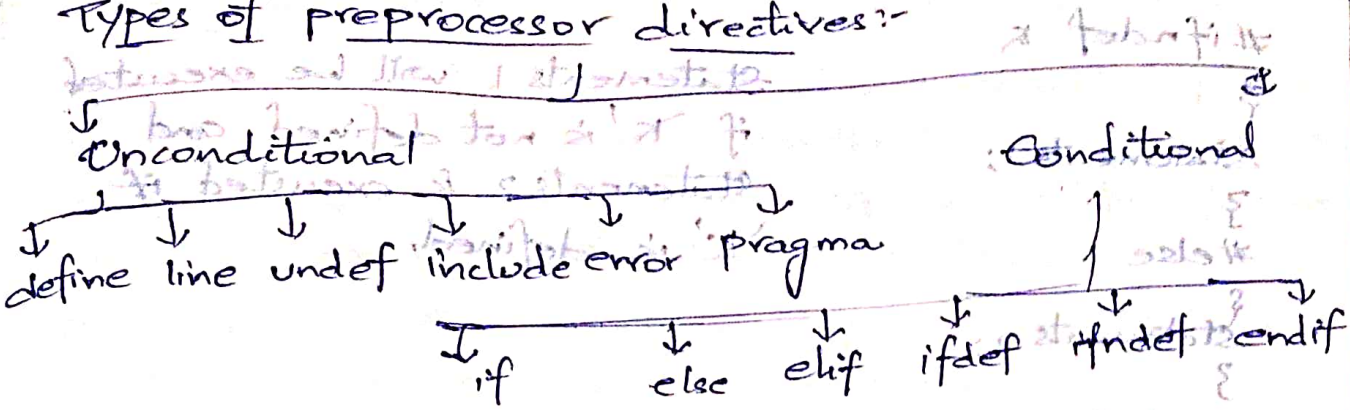
{

state

}

#

Types of preprocessor directives:-



#define is also known as Macro definition (or) simply a macro.

Syntax :- #define identifier substitution text

It is simply an identifier which is replaced by the substitution text in the source code. Every occurance of the

Ex :- #define printf pf

File inclusion :-

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~~ for the existence of macro definitions.

```
#ifdef
#define K
main()
{
#ifdef K
{
statements1;
}
#else
{
statement2;
}
#endif
```

The statements 1 will be executed if and only if the K is defined.

If not defined, the statements 2 will be executed.

```
#ifndef K
{
statements 1;
}
#else
{
statements 2;
}
#end if
```

Statements 1 will be executed if 'K' is not defined and statements 2 is executed if 'K' is defined.

```
for(i=0;i<5;i++)
{
for(j=i;j<5;j++)
{
if(a[j]<a[i])
{swap;}
}
}
```

```
Static int a[20];
i=0
a[i]= i++.
```

a[0]=i=0
i=i+1=1
} a[0]=0
i=1

```
j= i++
j=i
i=i+1
```

```
print a[0]=0
```

a[1] = default value of static = 0

a[1] = 0

i = 1.