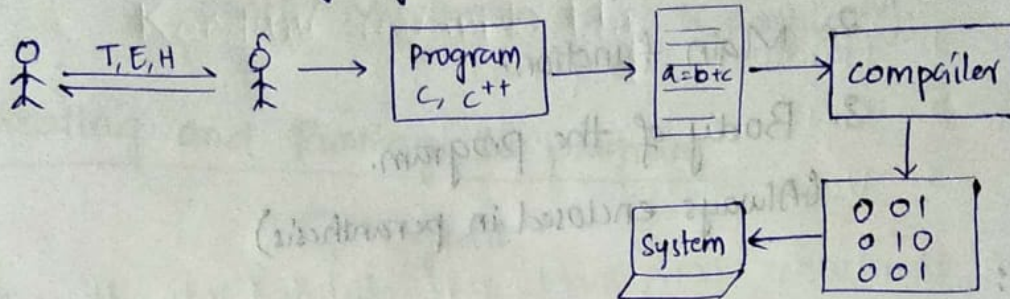


UNIT-1

Language : Language is the set of instructions.

C-language : If a human wants to communicate with system, uses C-language.



C-programming is a combination of BCPL and B.
(BCPL + B = C)

BCPL = Basic Combined Programming Language

- C-programming is introduced in 1972 by Dennis Ritchie.
- C-programming is the mother of languages, (all compilers are designed by C language only)
- C programming is platform dependent.

The single output file should not be executed in multiple system

- C-programming is a structural programming.

Structural programming is dividing one program into many sub programs.

- Because of structural compilation time is very less compared to the other languages.
- C-programming is user friendly.
- C-programming is procedural language.

1. Structure of C-programming.
2. Creating and running program.

1. Structure of C-programming: It consists of

1. Headed file
2. Main function.
3. Body of the program.

(Always enclosed in paranthesis)

Rules:

1. C-programming consists of atleast one headed file.
 $\geq 1(H)$
2. Consists of only one Main function.
 $= 1(Main)$
3. Should maintain equal no. of open and closed paranthesis
 $\{ \} (= =)$
4. Each and every executable statement is ended with semicolon.

→ Write a C-program to print your name on the desktop.

```
#include <stdio.h>
void main()
{
    printf("Sravani Reddy");
}
```

→ Write a C-program to print your mother name.

```
#include <stdio.h>
void main()
{
    printf("My mother name is Latha")
}
```


→ Write a c-program to print your sibling name

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

```
void main()
```

```
{
```

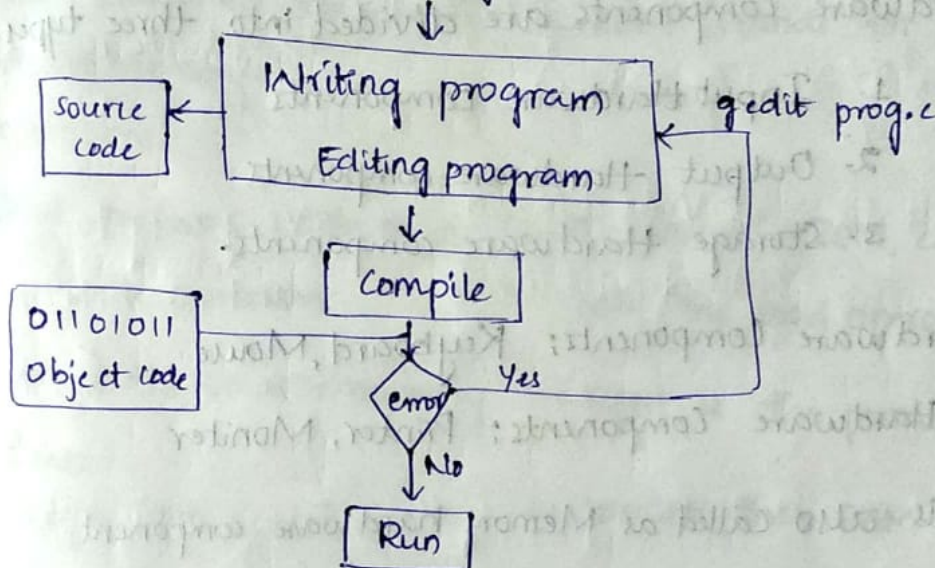
```
printf("My sister name is Shravya");
```

```
printf("My brother name is cathwik");
```

```
}
```

2. Creating and Running the program

Open gedit



For opening gedit

open → ctrl + shift + t

For writing the program

gedit prog.c

For compiling

cc prog.c

To runout program

./a.out

Computer: Computer is an electronic device which takes the input values and processes the data and finally produces the output.

Components of the computer

1. Hardware components
2. Software components
3. User

→ Again Hardware components are divided into three types.

Those are

1. Input Hardware components
2. Output Hardware components
3. Storage Hardware components.

1. Input Hardware Components: Keyboard, Mouse
2. Output Hardware Components: Printer, Monitor

→ Storage is also called as Memory hardware component

- It is categorised into
1. Primary Memory
 2. Secondary Memory

Memory is used to store any type of data.

Differences between primary and secondary Memory.

Primary

- RAM, ROM
- Accesses time.
- Small amount of data is stored.

Secondary

- Hard disk, CD, DVD
- Relatively low than primary.
- Large amount of data is stored here.

If we store data in primary memory we can access the data very fast

Secondary is relatively low compared to Primary Memory

Capacity of storing data

Small amount of data is stored in primary.

Large amount of data is stored in Secondary.

RAM

→ RAM stands for Random Access Memory.
operations:

→ RAM performs both read and write operations.

→ RAM is the volatile memory
Sizes:

→ Larger (like scale)

→ What is a compiler?

Compiler is a translator which convert high level language into low level language.

Low level Language

Introduced in
1940

→ In a low level language we always write the codes in the form of 0s and 1s

Middle level language

1950

→ In the case of middle level language we write the code in mnemonics.

High level language

1960

→ In high level language always write the code in English language.

ROM

→ ROM stands for Read Only Memory.

→ In case of ROM, can perform only the read operations.

→ ROM is the permanent memory

→ Smaller than RAM (like chip)

Low level language

→ Eg: 0110110
011111
00000

→ Any compiler is not required.

→ Low level language takes very less time to get the result.

→ Very difficult to understand.

Middle level language

→ Mnemonics

ADD A,B

SUB A,B

MUL A,B

DIV A,B

→ Compiler is required which is called Assembler.

→ Relatively higher than low level.

→ Very difficult to understand.

High level language

→ Printf ("hello");

→ Compiler is required.

→ Takes more time to get the result than Middle level language.

→ Very easy to understand.

Number systems

→ Number system used to represent the numbers with radices or base values.

→ Number system is divided into 4 types. They are Binary, decimal, Octal and hexa decimal.

Eg:

Base

(01)₂

2

Binary → 0, 1 → 2

(23)₁₀

10

Decimal → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 → 10

(12)₈

8

Octal → 0, 1, 2, 3, 4, 5, 6, 7 → 8

(1A)₁₆

16

Hexadecimal → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F → 16

Convert a decimal number into Binary number (10011)

$$(7)_{10} \rightarrow ()_2 \quad \begin{array}{r} 2 \overline{) 7} \\ \underline{4} \\ 3 \\ \underline{2} \\ 1 \\ \underline{1} \\ 0 \end{array} \Rightarrow (7)_{10} \rightarrow (111)_2$$

$$(8)_{10} \rightarrow (1000)_2 \quad \begin{array}{r} 2 \overline{) 8} \\ \underline{4} \\ 4 \\ \underline{2} \\ 2 \\ \underline{2} \\ 0 \end{array} \Rightarrow (8)_{10} \rightarrow (1000)_2$$

$$(25)_{10} \rightarrow (11001)_2 \quad \begin{array}{r} 2 \overline{) 25} \\ \underline{12} \\ 13 \\ \underline{6} \\ 7 \\ \underline{3} \\ 4 \\ \underline{1} \\ 3 \end{array} \quad \begin{array}{l} (32)_{10} \rightarrow (100000)_2 \\ (64)_{10} \rightarrow (1000000)_2 \\ (128)_{10} \rightarrow (10000000)_2 \end{array}$$

Convert decimal number into the octal

$$(25)_{10} \rightarrow (31)_8 \quad \begin{array}{r} 8 \overline{) 25} \\ \underline{16} \\ 9 \\ \underline{8} \\ 1 \end{array} \quad \begin{array}{r} 8 \overline{) 64} \\ \underline{48} \\ 16 \\ \underline{16} \\ 0 \end{array}$$

Convert decimal to the hexadecimal

$$(25)_{10} \rightarrow (19)_{16} \quad \begin{array}{r} 16 \overline{) 25} \\ \underline{16} \\ 9 \end{array} \quad \begin{array}{r} 16 \overline{) 34} \\ \underline{32} \\ 2 \end{array}$$

If the base value increases then the total digits or space is decreased.

Convert binary number into the decimal number

$$(101)_2 \rightarrow (5)_{10}$$

2^2	2^1	2^0
1	0	1

$$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \Rightarrow 4 + 0 + 1$$

$$\Rightarrow 5$$

$$(11001)_2 = (25)_{10}$$

$$\begin{array}{cccccc} 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 0 & 0 & 1 \end{array}$$

$$1 \times 2^4 + 1 \times 2^3 + 1 \times 1 \Rightarrow 25$$

$$(10101011)_2 = (171)_{10}$$

$$\begin{array}{cccccccc} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

$$128 + 32 + 8 + 2 + 1 = 171$$

$$* (102)_2 \rightarrow ()_{10} \text{ (Ans: None)}$$

Because 2 does not belong to binary number.

Note: Converting decimal to binary

	8	4	2	1
10 = 0 < 0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Converting octal to binary

	4	2	1
8 = 0 < 0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Converting hexadecimal to binary

0 → 0000	10(A) 1010
1 → 0001	11(B) 1011
2 → 0010	12(C) 1100
3 → 0011	13(D) 1101
4 → 0100	14(E) 1110
5 → 0101	15(F) 1111
6 → 0110	
7 → 0111	
8 → 1000	
9 → 1001	

Converting octal number into the decimal

$$(25)_8 \rightarrow 2 \times 8^1 + 5 \times 8^0 \Rightarrow 16 + 5 \Rightarrow (21)_{10}$$

$(78)_8 \rightarrow$ Not possible (because 8 is not present in octal)

Converting hexadecimal to decimal

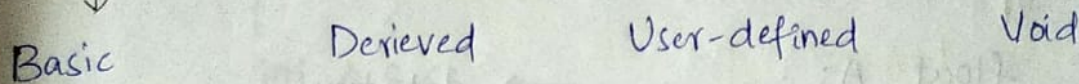
$$(12)_{16} = 1 \times 16^1 + 2 \times 16^0 \Rightarrow (18)_{10}$$

$(1FG)_{16} =$ Not possible (because G is not present in hexadecimal)

Convert given octal number into the hexadecimal.

$$(127)_8 \longrightarrow (\quad)_{16}$$

DATA TYPES



Basic data types

	size (bytes)	Key words	format specifier
Character → A, B, C, D...	1	char	%c
Integer → 0, 1, 2, 3, 4...	2	int	%d
Floating point → 2.12, 3.43	4	float	%f

With the help of data type

- We can assign with what kind of we are working
- We can also assign how much data is allocated.

Variable:

Variable is a named memory location

Syntax: datatype variable name;

char A;

int A;

float A;

Rules for defining the variables:

- Variable name start with char
- Variable should not start with integers.
- Don't start with any special characters.

Output → printf("ABC")

Input → ("format specifier", &b);

output → ("%f", b);

→ Write a C program to read and print one float value

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

```
void main()
```

```
{
```

```
float A;
```

→ data type

```
printf("Enter value of A");
```

→ User understanding

```
scanf("%f", &A);
```

→ Input

```
printf("%f", A);
```

→ Output

```
}
```

→ Write a C program to read and print one integer value

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

```
void main()
```

```
{
```

```
int A;
```

```
printf("Enter value of A");
```

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

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

```
}
```

→ Write a C program to read and print integer and float

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

```
void main()
```

```
{
```

```
int A;
```

```
float B;
```

```
printf("Enter value of A & B");
```

```
scanf("%d%f", &A, &B);
```

```
printf("A is %d B is %f", A, B);
```

```
}
```


→ Write a c-program to perform addition of two numbers.

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

```
void main( )
```

```
{
```

```
int A, B, C;
```

```
printf("enter A,B");
```

```
scanf("%d %d", &A, &B);
```

```
C = A+B;
```

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

```
}
```

→ Write a c program to perform multiplication of two numbers

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

```
void main( )
```

```
{
```

```
int A, B, C;
```

```
printf("enter A,B");
```

```
scanf("%d %d", &A, &B);
```

```
C = A*B;
```

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

```
}
```

→ Write a c program to perform subtraction of two numbers.

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

```
void main( )
```

```
{  
int A, B, C;
```

```
printf("enter A,B");
```

```
scanf("%d %d", &A, &B);
```

```
C = A-B;
```

```
printf("enter the value of C is %d", C);
```

```
}
```


Write a c program to perform addition, multiplication and subtraction of two numbers.

```
#include <stdio.h>

void main( )
{
    int A, B, C, D, E;
    printf("enter value of A and B");
    scanf("%d %d", &A & B);
    C = A + B;
    D = A - B;
    E = A * B;
    printf("C is %d and D is %d and E is %d", C, D, E);
}
```

1. Write a c program to print the hello word on desktop.
2. Write a c program to print sibling name (2 brothers, 2 sisters).
3. Write a c program to perform addition of two integers numbers.

1. W

13th August, 2018

→ Write a C-program to print your name on desktop.

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

```
void main()
```

```
{
```

```
printf("My name is Sravani Reddy")
```

```
}
```

Output:

My name is Sravani Reddy

→ Write a C-program to print mother name on desktop

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

```
void main()
```

```
{
```

```
printf("My mother name is Latha")
```

```
}
```

Output:

My mother name is Latha

→ Write a C-program to print mgit, hyd, mumbai on desktop in different lines.

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

```
void main()
```

```
{
```

```
printf("mgit\n")
```

```
printf("hyd\n")
```

```
printf("mumbai\n")
```

```
}
```

Output:

mgit

hyd

mumbai

→ Write a C-program to print your siblings names

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

```
void main()
```

```
{ printf("Akhil");
```

```
printf("satwik");
```

```
printf("shravya");
```

```
printf("pravaruthi");  
}
```

output:

Akhil

Satwik

Shravya

Pravaruthi

→ Write a C-program to perform addition of two numbers.

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

```
void main()
```

```
{
```

```
int A, B, C;
```

```
printf("enter A and B");
```

```
scanf("%d %d", &A, &B);
```

```
C = A + B;
```

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

```
}
```

A = 10 ; B = 20

Output : C = 30

→ Write a C-program to perform subtraction of two numbers.

```
#include <stdio.h>
void main()
{
    int A, B, C;
    printf("enter A and B");
    scanf("%d %d", &A, &B);
    C = A - B; sub
    printf("%d", C);
}
```

A = 20 ; B = 15

Output C = A - B = 15

→ Write a C-program to perform addition, subtraction, Multiplication for the given through integer values

```
#include <stdio.h>
void main()
{
    int A, B, C, D, E;
    printf("enter A and B");
    scanf("%d %d", &A, &B);
    C = A + B;
    D = A - B;
    E = A * B;
    printf("C = %d, D = %d and E = %d", C, D, E);
}
```

A = 5 ; B = 3

Output: C = 8

D = 2

E = 15

→ Write a c program to execute the value of c as $\frac{a+b}{2}$

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

```
void main()
```

```
{
```

```
int A, B, c;
```

```
printf("enter A and B");
```

```
scanf("%d %d", &A, &B);
```

```
c = (A+B)/2;
```

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

```
}
```

A = 50 ; B = 50
output: $c = \frac{A+B}{2} \Rightarrow 50$

→ Write a c program to execute the value $s = \frac{p \times t \times r}{100}$

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

```
void main()
```

```
{
```

```
int s, p, t, r;
```

```
printf("enter p, t and r);
```

```
scanf("%d %d %d", &p, &t, &r);
```

```
s =  $\frac{p \times t \times r}{100}$ ;
```

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

```
}
```

p = 5 ; t = 10 ; r = 20

output: $s = \frac{p \times t \times r}{100} \Rightarrow \frac{5 \times 10 \times 20}{100} = 10$

→ Find out average of three numbers.

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

```
void main()
```

```
{
```

```
int A, B, C, D;
```

```
printf("enter values of A, B, C");
```

```
scanf("%d%d%d", &A, &B, &C);
```

```
D =  $\frac{A+B+C}{3}$ ;
```

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

```
}
```

a = 3; b = 6; c = 9

Output: $D = \frac{a+b+c}{3}$

$\frac{3+6+9}{3} = 6$

Operators in C

→ Operator is a symbol which is used to manipulate (change) operate values

→ Arithmetic operators in C:

Arithmetic operators are +, -, *, ~~/~~, %

+ (plus)

- (minus)

* (multiplier)

% (modular)

~~/~~ (division)

Modular operator always supports the integer values.

Write a c program to implement arithmetic operations by using two integer values

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

```
void main( )
```

```
{
```

```
int A,B,C,D,E,F,G;
```

```
printf("enter A and B");
```

```
scanf("%d%d",&A,&B);
```

```
C = A+B;
```

```
D = A-B;
```

```
E = A*B;
```

```
F = A/B;
```

```
G = A%B;
```

```
printf("%d\n D=%d\n E=%d\n F=%d\n G=%d\n",  
A,B,E,F,G);
```

```
}
```

Relational Operators:

Relational operators are

<

>

<=

>=

== (comparison operators)

In relational operators the result is always in the form of zero and one (0 & 1)

0 → False → No

1 → True → Yes

Write a c program to implement relational operators.

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

```
void main( )
```

```
{
```

```
int A,B,C,D,E,F,G;
```

```
printf("enter A and B");
```

```
scanf("%d%d", &A,&B);
```

```
C = A < B;
```

```
D = A > B;
```

```
E = A <= B;
```

```
F = A >= B;
```

```
G = A == B;
```

```
printf("C=%d\nD=%d\nE=%d\nF=%d\nG=%d",C,D,E,F,G);
```

```
}
```

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

```
void main( )
```

```
{
```

```
int A,B,C,D,E,F,G;
```

```
printf("enter A and B");
```

```
scanf("%d%d", &A,&B);
```

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

```
void main( )
```

```
{
```

```
int A,B;
```

```
printf("enter A and B");
```

```
scanf("%d%d", &A,&B);
```

```
printf("%d", A > B);
```

```
printf("%d", A < B);
```

```
printf("%d", A <= B);
```

```
printf("%d", A >= B);
```

```
printf("%d", A == B);
```

```
}
```


Assignment operator:

Assignment operators divided into two categories.

1. Simple assignment (=)
2. Compound assignment.

Write a C program to implement simple assignment

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

```
void main ( )
```

```
{
```

```
int A = 10;
```

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

```
}
```

Compound Assignment: If we assign any arithmetic operator before simple assignment, then that is called as compound Assignment operator.

Write a C program to implement compound Assignment

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

```
void main ( )
```

```
{
```

```
int A = 10, B = 20;
```

```
printf ("%d", A += 2);
```

```
printf ("%d", B *= 4);
```

```
}
```

Logical Operators

Logical operators divided into three types.

1. Logical AND (&&)
2. Logical OR (||) → pipeline symbol.
3. Logical NOT (!=)

→ Logical operators are always associated with relational operators
→ The output of logical operations are always 0 or 1

Truth table for Logical AND

P	Q	P & Q
0	0	0
0	1	0
1	0	0
1	1	1

Truth table for logical OR

P	Q	P Q
0	0	0
0	1	1
1	0	1
1	1	1

Write a c-program to implement logical operations

```
#include <stdio.h>
void main()
{
    int A=10, B=20, C=30, d, f, g, H;
    d = A < B && A < C;
    f = A > B && A > C;
    g = A < B || B > C;
    H = B != C;
    printf("%d\n%d\n%d\n%d", d, f, g, H);
}
```

Conditional operator (or) Ternary operator

Also called as Ternary operator

Conditional operators are used to find out biggest element in given ^{two} numbers. (?)

Write a c program to implement conditional operator

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

```
void main()
```

```
{
```

```
int A=30, B=40, C;
```

```
C = (A > B) ? A : B;
```

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

```
}
```


Bitwise operators (Results displayed in decimal)

& (B-AND)

| (B-OR)

^ (B-XOR)

~ (shift + 6)

<< (left-shift)

>> (right-shift)

~ (negation)

Write a C-program to implement bit-wise and, bit-wise or and bit-wise XOR.

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

```
void main()
```

```
{
```

```
int A=2, B=5, C, D, E;
```

```
C = A & B;
```

```
D = A | B;
```

```
E = A ^ B;
```

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

```
}
```

Soln-
B-AND
A = 2 = 0010
B = 5 = 0101

0000
C = 0

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

B-OR
A = 2 = 0010
B = 5 = 0101

0111
D = 7

A	B	A B
0	0	0
0	1	1
1	0	1
1	1	1

5-XOR

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

$$A = 2 = 0010$$

$$B = 5 = 0101$$

$$\underline{\quad\quad\quad}$$

$$0111$$

$$E = 7$$

Output: C = 0
D = 7
E = 7

<< (left shift); >> (right shift); ~ (negation)

<< → Add 0

>> → Truncate (Binary value)

↓
Removing

~ → -(n+1)

Write a C-program to implement leftshift, right shift and negation operators.

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

```
void main()
```

```
{
```

```
int A=5, B=20, C=7;
```

```
A = A << 2;
```

```
B = B >> 2;
```

```
C = ~C;
```

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

```
}
```

Obj:- Leftshift is adding zeros on right hand side

Convert the given number in to binary and add no. of zeros on right hand side according to given no. of leftshift values

$$A = 5 \rightarrow 0101$$

$$A \ll 2 \Rightarrow \begin{array}{ccccccc} & & 11 & 8 & 4 & 2 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{array}$$

20

$$\text{Eg: } A = 10 \rightarrow 1010$$

$$A \lll 3 \rightarrow \begin{array}{ccccccc} & & 11 & 8 & 4 & 2 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

80

Right shift operation: It's nothing but removing the binary operations on right hand side.

$$B = 20 = 10100$$

$$B \gg 2 = 101$$

5

$$C = \sim(n+1)$$

$$C = -8$$

Output:

$$A = 20$$

$$B = 5$$

$$C = -8$$

Auto-increment or decrement operators (or) Unary operators

Increment $\begin{cases} \text{Post} \\ \text{Pre} \end{cases}$

Decrement $\begin{cases} \text{Post} \\ \text{Pre} \end{cases}$

$A++$; (post increment)

$A--$; (post decrement)

$++A$; (pre increment)

$--A$; (pre decrement)

Write a c-program to implement unary operators

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

```
void main ( )
```

```
{
```

```
int A = 10, B;
```


Write a c-program to implement auto-increment operators.

```
→ #include <stdio.h>
void main()
{
    int A=10, B;
    B=A++;
    printf("A is %d and B is %d", A, B);
}
```

output

A=11, B=10

```
→ #include <stdio.h>
void main()
{
    int A=10, B;
    B=++A;
    printf("A is %d and B is %d", A, B);
}
```

output

A=11, B=11

Special Operators

→ Write a c-program to implement sizeof operators.

```
#include <stdio.h>
void main()
{
    float b;
    char c;
    int a;
    printf("sizeof int is %d and sizeof float is %d and sizeof char is %d\n",
           sizeof(a), sizeof(b), sizeof(c));
}
```


→ Write a c-program to perform swapping of two numbers.
without using 3rd number.

```
#include <stdio.h>
void main()
{
    int A=10, B=20;
    A=A+B;
    B=A-B;
    A=A-B;
    printf("%d %d", A, B);
}
```

```
#include <stdio.h>
void main()
{
    int A, B;
    printf("enter A and B");
    scanf("%d %d", &A, &B);
    printf("before swap A is %d B is %d", A, B);
    A=A+B;
    B=A-B;
    A=A-B;
    printf("after swap A is %d and B is %d", A, B);
}
```

→ Write a c-program to swap two numbers by using third variable

```
#include <stdio.h>
void main()
{
    float f=64.2, c;
    c=0.59*(f-32);
    printf("f is %f and c is %f", f, c);
}
```

out put:

f = 64.199997 and c = 10.0

Conditional statements

Conditional statements are

1. Simple if
2. if-else
3. Nested if
4. if-else ladder

} condition (or) decision Making

1. Representation of Simple if

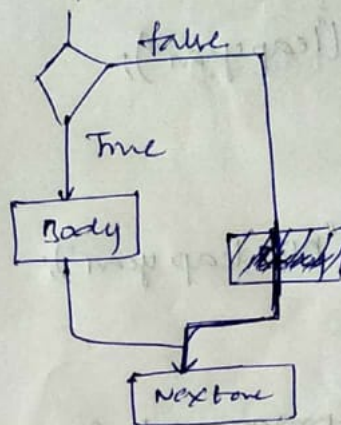
if (condition)

{

Body of _____ Statements

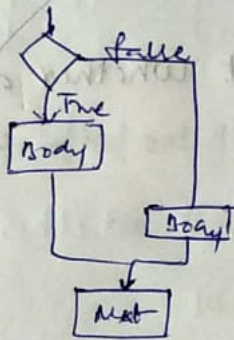
program _____

}



2. If-else

$\int$
body
 $\int$



Write a c-program to identify whether the given number is even or odd

```
#include <stdio.h>
void main()
{
    int A;
    printf("enter A");
    scanf("%d",&A);
    If(A%2 == 0)
    {
        printf("even number");
    }
    else
    {
        printf("odd number");
    }
}
```


Write a c-program to find whether given year is a leap year or not

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

```
void main()
```

```
{
```

```
int n;
```

```
printf("enter n");
```

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

```
If (n % 4 == 0)
```

```
{
```

```
printf("leap year");
```

```
}
```

```
else
```

```
{
```

```
printf("non leap year");
```

```
}
```

```
}
```

Write a c-program to find out whether a person is eligible for voting or not.

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

```
void main()
```

```
{
```

```
int A;
```

```
printf("enter A");
```

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

```
If (A > 18)
```

```
{
```

```
printf("eligible");
```

```
}
```

```
else
```

```
{
```

```
printf("not eligible");
```

```
}
```

```
}
```


Write a c-program to find out the biggest number in given two integer numbers.

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

```
void main()
```

```
{
```

```
int A, B;
```

```
printf("enter A and B");
```

```
scanf("%d %d", &A, &B);
```

```
If (A > B)
```

```
{
```

```
printf("A is bigger");
```

```
}
```

```
else
```

```
{
```

```
printf("B is biggest value");
```

```
}
```

```
}
```

Write a c-program to find out the given alphabet is vowel or not

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

```
void main()
```

```
{
```

```
char 'D';
```

```
printf("enter D");
```

```
scanf("%c", &D);
```

```
If (D == 'a' || 'e' || 'i' || 'o' || 'u' || 'A' || 'E' || 'I' || 'O' || 'U')
```

```
{
```

```
printf("vowel");
```

```
}
```

```
else
```

```
{
```

```
printf("consonant");
```

```
}
```

```
}
```

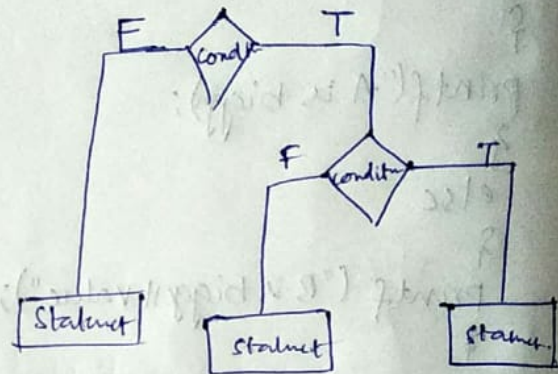

Nested-if

Nested-if is defining if statement inside the another if statement

Mismatching else is called as the Dangling else.

Syntax for the nested if

```
{
if (condition)
{
    if (condition)
    {
    }
    else
    {
    }
}
else
{
}
}
```



Write a c-program to find out whether the given integer is

- 1) Greater than 100 or not.
- 2) And the number is even or odd.

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

```
void main( )
```

```
{
```

```
int A;
```

```
printf("enter A");
```

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

```
If (A > 100)
```

```
{ printf("greater than 100");
```

```
If (A % 2 == 0)
```

```
{
```

```
printf("even");
```

```
}
```



```

else
{
printf("odd");
}
}
else
{
printf("less than 100");
}
}

```

Write a c-program to identify that whether the given number is positive or not if it is positive then that number is even or odd

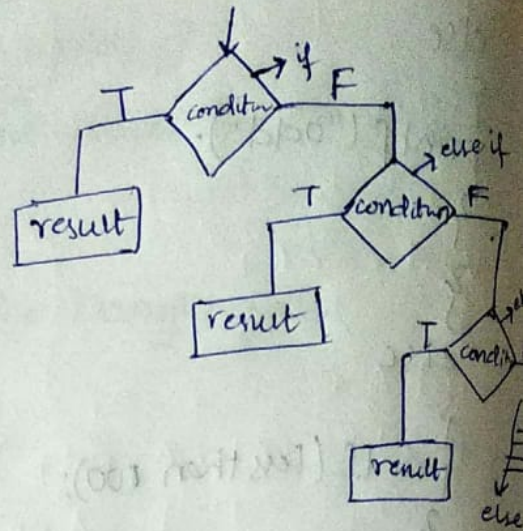
```

#include <stdio.h>
void main()
{
int A;
printf("enter A");
scanf("%d", &A);
If (A > 0)
{
If (A % 2 == 0)
{
printf("A is positive and even");
}
else
{
printf("A is positive but odd");
}
}
else
{
printf("A is negative");
}
}

```


If-else ladder:

```
if(condition)
{
    //
}
else if(condition)
{
    //
}
else if(condition)
{
    //
}
else
{
    //
}
```



Write a c-program to find out biggest number from given three integers.

```
#include <stdio.h>
void main( )
{
    int A,B,C;
    printf("enter A and B and C");
    scanf("%d%d%d", &A,&B,&C);
    If (A>B && A>C)
    {
        printf("A is biggest");
    }
    else if (B>A && B>C)
    {
        printf("B is biggest");
    }
    else
    {
        printf("C is biggest");
    }
}
```


write a program to decide the student average

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

```
void main()
```

```
{
```

```
float Avg;
```

```
printf("enter avg and B and C");
```

```
scanf("%f %f %f", &A, &B, &C);
```

```
If (avg < 40)
```

```
{
```

```
printf("fail");
```

```
}
```

```
else if (avg > 40 && avg < 55)
```

```
{
```

```
printf("C-grade");
```

```
}
```

```
else if (avg > 55 && avg < 65)
```

```
{
```

```
printf("B-grade");
```

```
}
```

```
else if (avg > 65 && avg < 75)
```

```
{
```

```
printf("A-grade");
```

```
}
```

```
else (avg > 75)
```

```
{
```

```
printf("outstanding");
```

```
}
```

```
}
```


Types of Errors

1. Compile (or) Syntax Error
2. Run-time error
3. Logical Error

1. Compilation Errors are:

- Missing semicolon.
- Write keywords in uppercase.
- Mismatching of parenthesis.
- Undeclared variables

2. Run-time errors are:

- An integer divides with zero.
- Insufficient Memory.

3. Logical Error are:

If expected output is not shown.

Operator Precedence and Associativity.

Precedence = priority

<u>Precedence</u>	<u>Associativity</u>
function braces $\{ \}$	If same equal priority operators are present, Left to Right
Array $[]$	
right left shift $\ll, \gg$	R-L
Auto increment $++$, decrement $--$	R-L
$*, /, \%$	L-R
$+, -$	L-R
$<, <=, >, >=$	L-R
$==$	L-R
$\&\&, \ \ $	L-R
$\&, \ \ $	L-R
$+=, -=$	L-R
(comma),	L-R
$=$	R-L

Algorithm

Algorithm is a step-by-step process for solving any mathematical problem

Eg: Addition of two numbers.

step-1 Read a, b, c

step-2 Insert a and b

step-3 Apply sum of a and b

+ Store sum in to c

5 print c value.

goto step-1

Input:

An algorithm consists of zero or more input variables.

output:

An algorithm consists of one or more output values.

Definiteness:

In an algorithm each and every statement defined in simple manner and should not repeat same statement twice.

Effectiveness: For any different input values, output should be correct $\rightarrow$ Effectives.

Switch Statement :

Syntax for switch statement

Switch (condition)

{

case 1: Statement; break;

case 2: Statement; break;

case 3: Statement; break;

case 4: Statement; break;

default: Statement;

}

Eg: Write a c-program of all arithmetic operators to execute switch statement

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

```
void main()
```

```
{
```

```
int a=10, b=20, c;
```

```
printf("Enter choice");
```

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

```
switch (c)
```

```
{
```

```
case 1: printf("add is %d", a+b); break;
```

```
case 2: printf("sub is %d", a-b); break;
```

```
case 3: printf("Mul is %d", a*b); break;
```

```
case 4: printf("Div is %d", a/b); break;
```

```
case 5: printf("Modular is %d", a%b); break;
```

```
default: printf("invalid option");
```

```
}
```

```
}
```



```

#include <stdio.h>
void main()
{
    int a=10, b=20, c;
    printf("1 add\n 2 sub\n 3 mul\n 4 div\n 5 mod");
    printf("enter choice");
    scanf("%d", &c);
    switch (c)
    {
        case 1: printf("add is %d", a+b); break;
        case 2: printf("sub is %d", a-b); break;
        case 3: printf("mul is %d", a*b); break;
        case 4: printf("div is %d", a/b); break;
        case 5: printf("mod is %d", a%b); break;
        default: printf("invalid option");
    }
}

```

Type casting (or) Type conversion

Type casting is convert one data type into the another data type

Type casting is divided into two types.

1. Implicit type casting
2. Explicit type casting

1. Implicit Type casting; compiler by default convert one data type into the another data type.

Implicit type casting of two types

- ① Widening
- ② Narrowing

Widening

```
{  
int a=10;  
float b=a;  
printf("%f", b);  
}
```

(No loss of data)

Narrowing

```
{  
float a=10.23;  
int b=a;  
printf("%d", b);  
}
```

(Data may loss)

2. Explicit type casting: ~~Forcefully~~ we are converting one data type into the other data type.

```
{  
int a=7, b=5;  
float c=a/b;  
printf("%f", c);  
}
```

```
{  
int a=7, b=5;  
float c=(float) a/b;  
printf("%f", c);  
}
```

Loops (or) Conditional

Loops are used to execute block of statements repeatedly with a condition.

Loops are divided into three categories

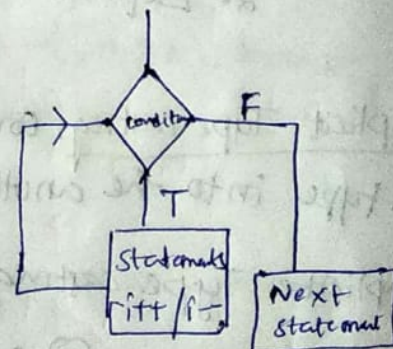
- i) While loop
- ii) do-while loop
- iii) For loop

While loop

Syntax

while (condition)

```
{  
stmts;  
Inc/dec;  
}
```



Write a c-program to print mgit ~~100~~ 100 times

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

```
void main()
```

```
{
```

```
int a=1;
```

```
while(a<=100)
```

```
{
```

```
printf("Mgit");
```

```
a++;
```

```
}
```

```
}
```

Write c-program to print from 1 to 10

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

```
void main()
```

```
{
```

```
int a=1;
```

```
while(a<=10)
```

```
{
```

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

```
a++;
```

```
}
```

```
}
```

Write a c-program to print infinite loop

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

```
void main()
```

```
{
```

```
inta=1;
```

```
while(a>0)
```

```
{
```

```
printf("mgit");
```

```
a++;
```

```
}
```

```
}
```

(control+z)

To terminate

Write a C-program to perform reverse order of given number

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

```
void main()
```

```
{
```

```
int r, s, n;
```

```
s = 0;
```

```
n = 123;
```

```
while (n != 0)
```

```
{
```

```
r = n % 10;
```

```
s = s * 10 + r;
```

```
n = n / 10;
```

```
}
```

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

```
}
```

Do-while loop

Syntax

```
do
```

```
{
```

```
_____
```

```
} while (condition);
```

(The scope of)

Pretest

Eg: While loop

→ If the condition is true then only we are executing block of statements.

→ The scope of accessing statements in while is $0, 1, 2, \dots, n$

Post test

Eg: do-while loop

→ Irrespective of condition the block of statements are executed atleast once

→ The scope of accessing statements in do-while is $1, 2, 3, \dots, n$

→ Syntax of pre-test is

while (condition)

{

=====

}

→ syntax of post test is

do

{

=====

} while (condition);

For loop

Syntax for the for-loop

for (initialization; condition; incre/decre)

Eg:- C-program for sum of first three natural numbers.

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

```
void main()
```

```
{
```

```
int A, S=0;
```

```
for (A=1; A<=3; A++)
```

```
{
```

```
S=S+A;
```

```
}
```

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

```
}
```

Write a c-program to find out factorial of given number.

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

```
void main()
```

```
{
```

```
int A, S=1;
```

```
for (A=1; A<=3; A++)
```

```
{
```

```
S=S*A;
```

```
}
```

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

```
}
```


Break and Continue

Break is the keyword which is used to exit current loop.

Continue is the keyword, used to pause a single value from a set of values.

Eg for Break

```
#include <stdio.h>
void main()
{
    int A;
    for (A=1; A<=5; A++)
    {
        if (A>3)
        {
            break;
        }
        printf("%d", A);
    }
}
```

output

1, 2, 3

Eg for continue

```
#include <stdio.h>
void main()
{
    int A;
    for (A=1; A<=5; A++)
    {
        if (A==4)
        {
            continue;
        }
        printf("%d\n", A);
    }
}
```

output

1
2
3
5

Write a C-program to print the pattern

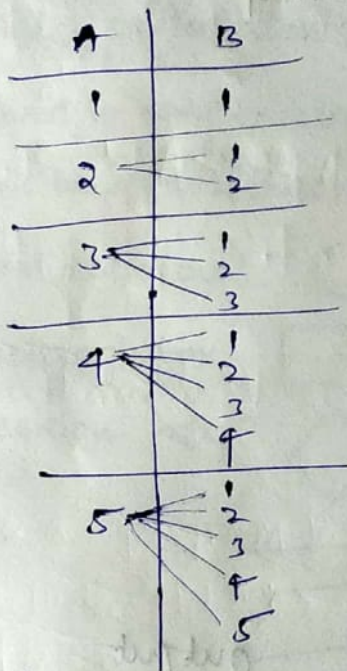
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

```
#include <stdio.h>

void main()
{
    int A, B;
    for (A=1; A<=5; A++)
    {
        for (B=1; B<=A; B++)
        {
            printf("%d", A);
        }
        printf("\n");
    }
}
```

output

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



```
* #include <stdio.h>

void main()
{
    int A, B;
    for (A=1; A<=5; A++)
    {
        for (B=1; B<=5; B++)
        {
            printf("%d", B);
        }
        printf("\n");
    }
}
```

A	B
1	1 2 3 4 5
2	1 2 3 4 5
3	1 2 3 4 5
4	1 2 3 4 5
5	1 2 3 4 5

→ #include <stdio.h>

void main()

{

int A, B;

for (A=1; A<=5; A++)

{

for (B=1; B<=A; B++)

{

printf("%d", B);

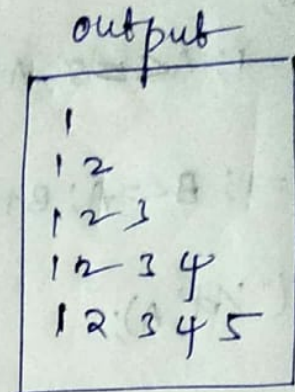
}

printf("\n");

}

}

output



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

→ Write a c-program to print pyramid

#include <stdio.h>

void main()

{

int A, S, B;

for (A=1; A<=5; A++)

{

for (S=1; S<=5-A; S++)

{

printf(" ");

}

for (B=1; B<=A; B++)

{

printf("* ");

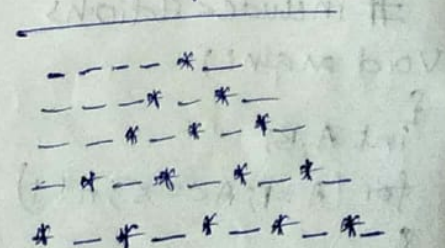
}

printf("\n");

}

}

output



```

  *
 * *
* * *
* * * *
* * * * *
```


→ Steps to solve logical and numerical problems or Program development steps

step: 1 - Understand the problem (statement)

step: 2 - Algorithm

step: 3 - Flow chart

step: 4 - Pseudo code

step: 5 - Coding

step: 6 - Debugging

step: 7 - Testing

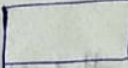
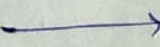
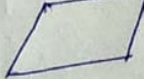
step: 8 - Documentation

Flowchart

Pictorial representation of an algorithm is called as Flowchart

Flowchart is easy to understand.

Symbols used in flowchart

1. Terminal or oval  (used for begin and stop)
2. Rectangle box  (processing the data)
3. Decision box  (True or false)
4.  (flow)
5.  (Input & output)

Draw a flowchart for addition of two numbers

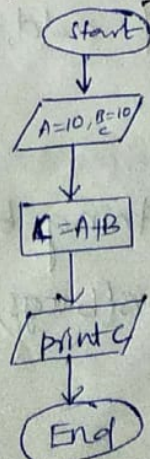
Start

Read $A=10, B=10, C$

$C = A + B$

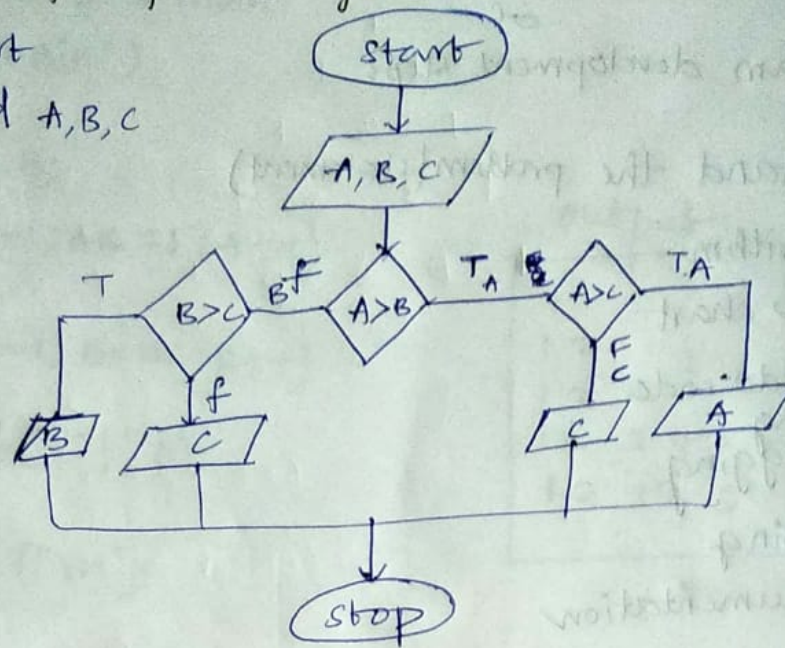
Print C

End



Draw a flow for the greatest number for given three numbers

Start
Read A, B, C



Pseudocode :- Informal representation of an algorithm

Pseudocode is in the form of neither an algorithm nor a code

Pseudocode is in the form of english statements.

Pseudocode is easy to understand, easy to perform modifications

Pseudocode is similar to the algorithm but not an algorithm

Eg: Read a=10, b=10

Add a, b

sub a, b

mul a, b

printf add, sub, mul

Debugging

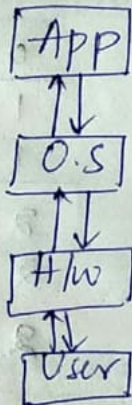
Debugging is nothing but compiling, this is easy to find the errors(bugs).

Testing: It is used to identify the efficiency of a program

Documentation: Saving

Operating system:

Operating system act as a interface between user and Application software



Formatted input and output statements:

scanf is the formatted input and printf is the formatted output

To find no. of character values to c program is

```
{  
  int i;  
  i = printf("MGIT");  
  printf("%d", i);  
}
```

To find how many inputs assigning in scanf

```
{  
  int
```


1 Write a C-program to print pascal triangle

```

      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1

```

```

#include <stdio.h>
void main()
{
    int i, s, j, c = 0;
    for (i = 0; i <= 5; i++)
    {
        for (s = 0; s <= 5 - i; s++)
        {
            printf(" ");
        }
        for (j = 0; j <= i; j++)
        {
            if (i == 0 || j == 0)
                c = 1;
            else
                c = c * (i - j + 1) / j;
            printf("%d ", c);
        }
        printf("\n");
    }
}

```

i \ j	0	1	2
0	1		
1	1	1	
2	1	2	1
3	1	3	3
4	1	6	6
5	1	10	10

UNIT-2

Array: Array is the collection of similar datatype elements. (or)

Array is the homogeneous datatypes representation

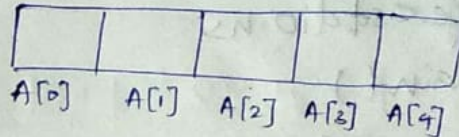
Arrays are always represented by []

Syntax:

datatype Arrayname [size of array];

Eg: `int A[5]`

Index = size - 1



Initialization and Accessing array values

```
int a[5];
printf("enter array values");
for (i=0; i<5; i++)
{
    scanf("%d", &a[i]);
}
printf("enter array values");
for (i=0; i<5; i++)
{
    printf("%d", a[i]);
}
```

→ Dynamic

(or)

`int A[5] = {10, 20, 30, 40, 50};` → static

→ Write a c-program to print the smallest element in the array values.

```
int a[5] = {50, 40, 30, 20, 10};
int min = a[0];
for (i=0; i<5; i++)
{
    if (a[i] < min)
```



```
min = a[i];
```

```
}
```

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

```
}
```

Two dimensional Array

Syntax:

```
datatype Arrayname [ ] [ ];
```

↓ ↓
column row

→ Write a c-program to perform addition of two matrices

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

```
void main()
```

```
{
```

```
int A[2][2], B[2][2], C[2][2];
```

```
int A[2][2], B[2][2], C[2][2], i, j;
```

```
printf("enter elements in A");
```

```
for (i=0; i<2; i++)
```

```
{
```

```
for (j=0; j<2; j++)
```

```
{
```

```
scanf("%d", &A[i][j]);
```

```
}
```

```
}
```

```
printf("enter elements in B");
```

```
for (i=0; i<2; i++)
```

```
{
```

```
for (j=0; j<2; j++)
```

```
{
```

```
scanf("%d", &B[i][j]);
```

```
}
```

```
}
```

```
printf("result is\n");
```



```

for(i=0; i<2; i++)
{
    for(j=0; j<2; j++)
    {
        C[i][j]=0;
        C[i][j]=A[i][j]+B[i][j];
        printf("%d\t", C[i][j]);
    }
    printf("\n");
}

```

Subtraction of two matrices

```

#include <stdio.h>
void main()
{
    int A[2][2], B[2][2], C[2][2];
    printf("enter elements of A");
    for(i=0; i<2; i++)
    {
        for(j=0; j<2; j++)
        {
            scanf("%d", &A[i][j]);
        }
    }
    printf("enter elements of B");
    for(i=0; i<2; i++)
    {
        for(j=0; j<2; j++)
        {
            scanf("%d", &B[i][j]);
        }
    }
    printf("result is \n");
    for(i=0; i<2; i++)
    {

```



```

for (j = 0; j < 2; j++)
{
    C[i][j] = 0;
    C[i][j] = A[i][j] + B[i][j];
    printf("%d\t", C[i][j]);
}
printf("\n");
}
}

```

(String) x

→ Write a c-program to print transpose of a given matrix

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

```
void main ( )
```

```
{
```

```
int A, B, i, j;
```

```
for (
```

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

```
void main ( )
```

```
{
```

```
int a[2][2], b[2][2], i, j;
```

```
printf("enter elements of A")
```

```
for (i = 0; i < 2; i++)
```

```
{
```

```
for (j = 0; j < 2; j++)
```

```
{
```

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

```
printf("enter elements")
```

```
for (i = 0; i < 2; i++)
```

```
{
```

```
for (j = 0; j < 2; j++)
```

```
{
```

```
b[j][i] = a[i][j]
```

```
printf("%d", b[i][j]);
```



```

}
printf("\n");
}

```

→ Write a c-program to find out whether the given number is prime number or not.

```

#include <stdio.h>

```

```

void main()

```

```

{

```

```

    int n, i, j = 0; c = 0

```

```

    printf("enter a positive integer");

```

```

    scanf("%d", &n);

```

```

    for (i = 2; i <= n/2; i++)
    {

```

```

        if (n % i == 0)

```

```

        {

```

```

            j = i;

```

~~c++;~~

```

            break;

```

```

        }

```

```

    }

```

```

    if (j == 0)

```

if (c == 2)

```

        printf("prime number");

```

```

    else

```

```

        printf("not a prime number");

```

```

    }

```


Strings: String is a collection of characters
(or)

string is the one dimensional character array.

Syntax: character datatype stringname [size of string]
char str [10];

To read string we are using double quotation.

String Handling Function

1. strlen(); To find the length of given string.
2. strcpy(); To copy one string into other string.
3. strcat(); one after other merging.
4. strrev(); Used to reverse the string
- 5.strupr(); To convert lowercase character into upper case
6. strlwr(); To convert uppercase character into lower case.
7. strcmp(); To compare two strings

string.h is the header file used for inbuilt strings

Formatted input and output functions in strings

gets(str name) → To read
puts(str name) → To print.

Write a C-program to perform gets and puts in strings

```
{  
char str[10]  
printf("enter string");  
gets(str);  
puts(str);  
}
```


Write a c-program to count total no. of characters in a given string

```
[char str[10]]
```

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

```
#include <string.h>
```

```
void main()
```

```
{
```

```
char A[10];
```

```
printf("enter string");
```

```
gets(A);
```

```
int l = strlen(A);
```

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

```
}
```

Write a c-program to copy from string to the other string.

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

```
#include <string.h>
```

```
void main()
```

```
{
```

```
char A[5], B[5];
```

```
printf("enter string of B");
```

```
gets(B);
```

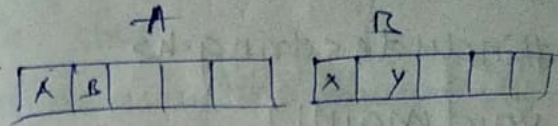
```
strcpy(A, B);
```

```
puts(A);
```

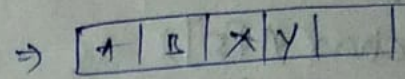
```
}
```


Write a c-program to merge the string

```
#include <stdio.h>
#include <string.h>
void main()
{
    char A[5], B[5];
    printf("enter string of A");
    printf("enter string of B");
    gets(A);
    gets(B);
    strcat(A, B);
    puts(A);
}
```



strcat(A, B)



Write a c-program to reverse the string

```
#include <stdio.h>
#include <string.h>
void main()
{
    char X[5];
    printf("enter string");
    strcpy(X, "ABCD");
    puts(X);
}
```

Write a c-program to convert the string values into uppercase

```
#include <stdio.h>
#include <string.h>
void main()
{
    char A[5];
    printf("enter string");
    strcpy(A, "abcd");
    puts(A);
}
```


Write a c-program to execute strlen

```
#include <stdio.h>
#include <string.h>
void main()
{
    char A[5];
    printf("enter string");
    strlen(A);
    puts(A);
}
```

Write a c-program to find whether string is a palindrome or not.

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

```
{
    char A[5], B[5];
    printf("enter string");
    gets(A);
    strcpy(B, A);
    if (strcmp(A, B) == 0)
        printf("palindrome");
    else
        printf("not a palindrome");
}
```

```
{
    char A[10];
    gets(A);
    int l = strlen(A);
    for (i = 0; i < l - 1; i++)
    {
        if (str[i] != str[l - 1 - i])
            i = i + 1;
    }
    if (i == 1)
        printf("palindrome");
    else
        printf("not a palindrome");
}
```

Without using
string handling
functions.

Structures;

- Structure is a collection of heterogeneous datatype.
- Key word used in structure is struct
- To initialize the structure and accessing the structure values we are using one operator that is called (.) dot.

Syntax

```
struct structure name
{
    datatype member 1;
    datatype member 2;
    datatype member 3;
};
```

Write a C program to initialize student structure.

```
#include <stdio.h>
void main()
{
    struct std
    {
        int no;
        float avg;
        char name[20];
    };
    struct std s;
    printf("enter no avg name");
    scanf("%d %f %s", &s.no, &s.avg, s.name);
    printf("%d %f %s", s.no, s.avg, s.name);
}
```

Write a structure to print employee details (id, name, salary)

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



```

int id;
float salary;
char name[20];
};
struct emp;
printf("enter id, salary, name");
scanf("%d %f %s", &s.id, &s.salary, s.name);
printf("%d %f %s", s.id, s.salary, s.name);
}

```

strcmp

```

char A[10], B[10];
gets(A)
gets(B)
c = strcmp(A, B);
if (c == 0)
{
    printf("Both are equal");
}
else if (c < 0)
{
    printf("A is greater");
}
else
{
    printf("B is greater");
}

```


Multiplication of Matrices

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

```
void main()
```

```
{
```

```
int A[2][2], B[2][2], C[2][2], i, j, k;
```

```
printf("enter elements in A");
```

```
for(i=0; i<2; i++)
```

```
{
```

```
for(j=0; j<2; j++)
```

```
{
```

```
scanf("%d", &A[i][j]);
```

```
}
```

```
}
```

```
printf("enter elements in B");
```

```
for(i=0; i<2; i++)
```

```
{
```

```
for(j=0; j<2; j++)
```

```
{
```

```
scanf("%d", &B[i][j]);
```

```
}
```

```
}
```

```
for(i=0; i<2; i++)
```

```
{
```

```
for(j=0; j<2; j++)
```

```
{
```

```
C[i][j]=0
```

```
for(k=0; k<2; k++)
```

```
{
```

```
C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

```
}
```

```
}
```

```
}
```



```

for (i=0; i<2; i++)
{
for (j=0; j<2; j++)
{
printf("%d", c[i][j]);
}
}
}

```

Differences between structure and union.

Structure

- Derived data type
- Keyword used is struct
- (.) operator is used
- Syntax Representation

```

struct std
{
datatype mem1;
datatype mem2;
datatype mem3;
};
struct std s;

```
- Size of structure depends on member functions defined inside the structure

Union

- Derived data type
- Keyword used is union
- (.) operator is used
- Syntax Representation

```

union std
{
datatype mem1;
datatype mem2;
datatype mem3;
};
union std s;

```
- In a union size is allocated for s's highest member function size

Structure

- In a structure it prints all values at a time.
- Structure is a heterogeneous datatypes

Structure with the arrays

struct std;

```
{  
    int rno;  
    char name[20];
```

```
    float avg;  
};
```

```
struct std s[5];
```

```
printf("rno avg name");
```

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

```
{  
    scanf("%d %s %f", &s.rno, s.name, &s.avg);  
}
```

```
printf("std details are");
```

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

```
{  
    printf("%d %s %f", s.rno, s.name, s.avg);  
}
```

```
}
```

→ ~~Wrong~~
(In observation)

Constants

Constants are three types

1. Integer constant.
2. Character constant.
3. Floating point constant.

→ Keyword used is const

→ Syntax is `const datatype variable name;`

Union

- Prints only one member function at a time.
- Union is also a collection of heterogeneous datatypes.

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

(Pointer)

Eg. - `const int A=10;`
`A=A+2`

Output: 10 $\times$

should not change the value of A

float b = 10

$$b = b + 40;$$

Output = 50

(Pointers) x
(Questions)

1. Write a c-program to implement bitwise operators.
2. Write a c-program to identify given numbers armstrong number or not.
3. Write a c-program to find greatest number among given three numbers.
4. Write a c-program to find out minimum number from a given array.
5. Write a cprogram to implement strlen and strcpy.

Pointers:

- Pointer is a derived datatype which is used to hold the address of another variable.
- With the help of the pointers we can access the values very fast.
- Pointers are used to allocate the memory dynamically.
- Format specifiers used in pointers are %P and %u
 ↓ ↓
 hexadecimal decimal
- Syntax for the pointer representation is
datatype * Variable name;

Eg:

```
int *p;
```

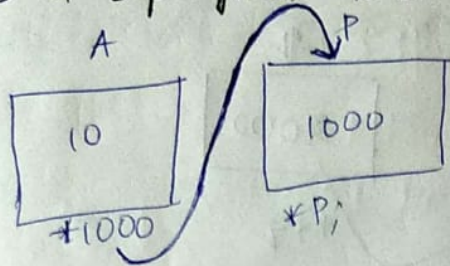
P = pointer variable

*p = pointing variable

P = &A;

*P = A

→ Write a c-program to demonstrate pointers.



```
{
int A = 10;
int *P;
P = &A;
printf("%d %d", A, *P);
printf("%u %u", P, &A);
}
```

Arithmetic operations on pointers

Add

```
{
int A = 10, B = 20;
int *P, *Q;
P = &A; Q = &B;
*P = *P + *Q;
printf("%d", *P);
}
```

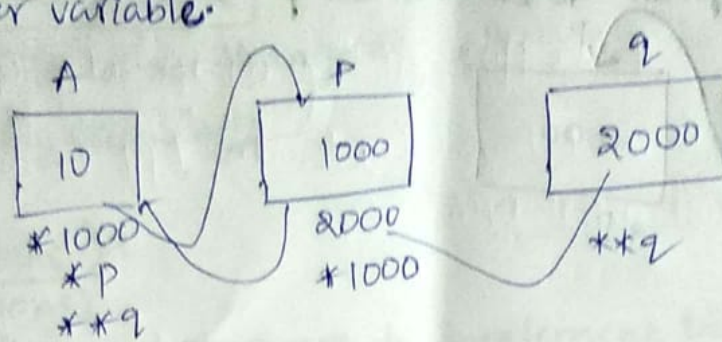
Sub

```
{
int A = 20, B = 10;
int *P, *Q;
P = &A; Q = &B;
*P = *P - *Q;
printf("%d", A);
}
```

P

Pointer to pointer variable

If a pointer variable holds the address of another pointer variable.



Write a c-program to perform addition for the pointer variables.

```
{
int a=10;
int *p;
p=&a;
P=P+1;
printf("%u", P);
}
```

Subtraction

```
{
int a=10
int *p;
p = &a
P = P-1;
printf("%u", P);
}
```

Structures with the pointers

```
struct std
{
int *p;
float avg;
};
```



```
struct std s = {10, 20.3};
```

```
struct s *p;
```

```
p = &s;
```

```
printf("%d%f", p->no, p->avg);
```

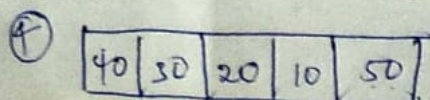
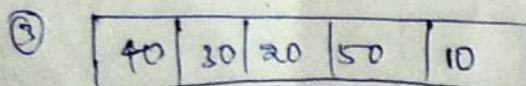
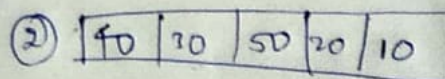
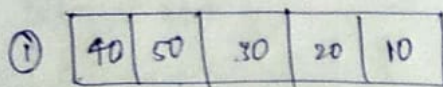
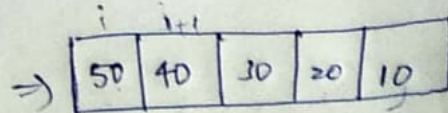

1. Preparing timetable in MS word
2. Type a leave letter in Microsoft word.
3. Draw a line graph in Microsoft excel
4. Prepare ppt about, deforestation.
 - a) Current Education system.

Day	
Monday	
Tuesday	
Wednesday	
Thursday	
Friday	
Saturday	
Sunday	

→ Write a program to show the result

$$1 - \frac{x^2}{2!} + \frac{x^2}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!}$$

→ Write a C-program to arrange the elements in ascending order in a given array.



⇒

40	30	20	10	50
----	----	----	----	----

30	40	20	10	50
----	----	----	----	----

30	20	40	10	50
----	----	----	----	----

30	20	10	40	50
----	----	----	----	----

⇒

20	10	30	40	50
----	----	----	----	----

10	20	30	40	50
----	----	----	----	----

⇒

30	20	10	40	50
----	----	----	----	----

20	30	10	40	50
----	----	----	----	----

20	10	30	40	50
----	----	----	----	----

int i, j, a[5] = {50, 40, 10, 20, 10};

for (i = 0; i < 5; i++)

{

for (j = ~~10~~; j < ~~5~~ - i; j++)

{

if [

#include <stdio.h>

void main()

{
int i, j, a[5], temp;

printf("enter a");

for (i = 0; i < 5; i++)

{

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

}

for (i = 0; i < 5; i++)

{

for (j = 0; j < ~~5~~ - i; j++)

{

if (a[j] > a[j+1])

temp = a[j+1];

a[j+1] = a[j];

a[j] = temp;

→ Write a c program to implement Bubble sort


```

}
printf
for (j=0; j<5; j++)
{
    printf("%d", a[j]);
}
}

```

Insertion sort:

16, 19, 11, 15, 10 — Apply sorting [Insertion sort]

16	19	11	15	10
----	----	----	----	----

16	11	19	15	10
----	----	----	----	----

16	11	15	19	10
----	----	----	----	----

16	11	15	10	19
----	----	----	----	----

11	16	15	10	19
----	----	----	----	----

11	15	16	10	19
----	----	----	----	----

11	15	10	16	19
----	----	----	----	----

11	16	19	15	10
----	----	----	----	----

11	16	15	19	10
----	----	----	----	----

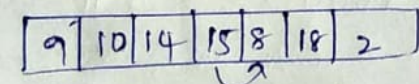
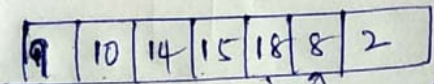
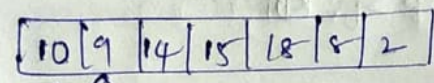
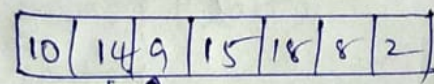
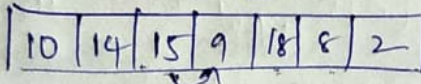
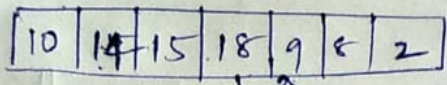
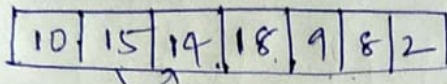
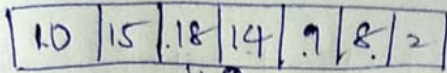
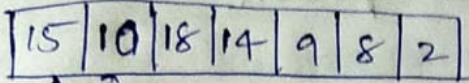
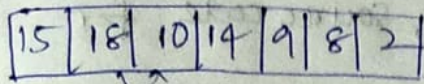
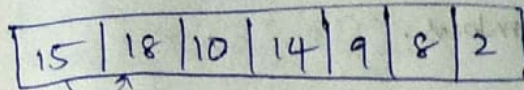
11	15	16	19	10
----	----	----	----	----

```

if (a[j-1] > a[j])
{
    swap
}

```


15, 18, 10, 14, 9, 8, 2



9, 10, 14, 8, 15, 18, 2

9, 10, 8, 14, 15, 18, 2

9, 8, 10, 14, 15, 18, 2

8, 9, 10, 14, 15, 18, 2

8, 9, 10, 14, 15, 2, 18

8, 9, 10, 14, 2, 15, 18

8, 9, 10, 2, 14, 15, 18

8, 9, 2, 10, 14, 15, 18

8, 2, 9, 10, 14, 15, 18

2, 8, 9, 10, 14, 15, 18

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

```
void main()
```

```
{
    int i, j, a[7], c;
```

```
    printf("enter array");
```

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

```
    for(i=0; i<7; i++)
```

```
{
```

```
        j=i;
```

```
        while (a[j]>0)
```

```
{
```

```
            if (a[j-1]>a[j])
```

```
{
```

```
                c=a[j];
```

```
                a[j]=a[j-1];
```

```
                a[j-1]=c;
```

```
            }
```

```
            j--;
```

```
        }
```

```
    }
```

```
    for(i=0; i<7; i++)
```

```
{
```

```
        printf("%d", a[i]);
```

```
    }
```

```
}
```


→ Write an algorithm, flowchart, source code for biggest among the three numbers.

→ Write an algorithm, flowchart, source code for roots of the given quadratic equation.

Selection sort:

20, 30, 10, 50, 40

↓
16 | 19 | 11 | 15 | 10 | 12 | 14

↓
10 | 19 | 11 | 15 | 16 | 12 | 14

10, 11, 19, 15, 16, 12, 14

10, 11, 12, 15, 16, 19, 14

10, 11, 12, 14, 16, 19, 15

10, 11, 12, 14, 15, 19, 16

10, 11, 12, 14, 15, 16, 19

#include <stdio.h>

void main()

{
int a[5] = {20, 30, 10, 50, 40};

int i, j;

for (i = 0; i < 5; i++)

{
int small = i;

for (j = i + 1; j < 5; j++)

{
if (a[j] < a[small])

small = j;

{

int t = a[i];

a[i] = a[small];

a[small] = t;

}}}

20 | 30 | 10 | 50 | 40


```
for (j=0; j<5; j++)
{
    printf("%d", a[j]);
}
```

Goto-statement:

Syntax for the goto statement

Forward goto

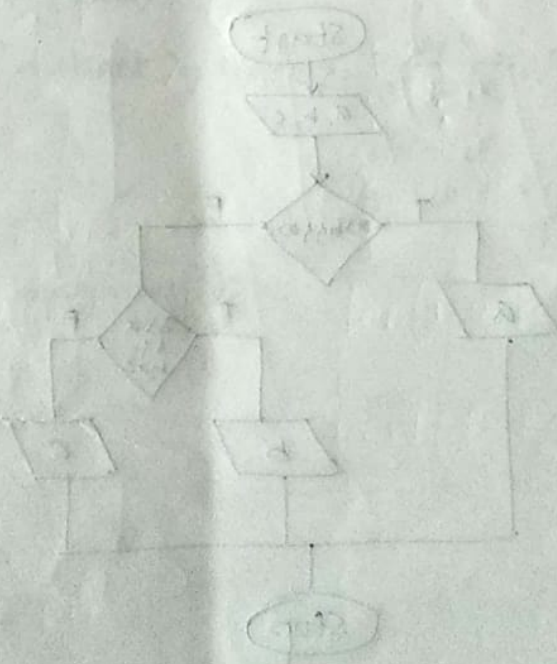
```
{
    printf("mgit");
    goto last;
    printf("cbit");
    printf("lbit");
last:
    printf("mbit");
}
```

output:

```
mgit
mbit
```

Backward goto

```
{
    printf("ngit");
last:
    printf("cbit");
    printf("lbit");
    goto last;
    printf("mbit");
}
```

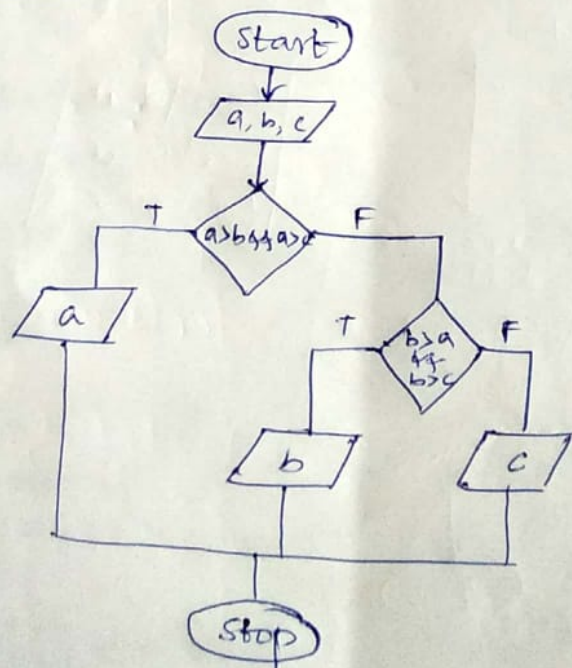



```

1. #include <stdio.h>
   void main()
   {
       int a, b, c;
       printf("enter a and b and c");
       scanf("%d %d %d", &a, &b, &c);
       if (a > b && a > c)
       {
           printf("a is greater");
       }
       else if (b > a && b > c)
       {
           printf("b is greater");
       }
       else
       {
           printf("c is greater");
       }
   }

```

Flowchart



2. #include <stdio.h>

#include <math.h>

void main()

{
float a, b, c, d, r₁, r₂;

printf("enter a and b and c");

scanf("%f%f%f", &a, &b, &c);

d = (b*b) - (4*a*c);

if (d == 0)

{

r₁ = -b/(2*a);

r₂ = -b/(2*a);

printf("roots are equal r₁ = %f r₂ = %f", r₁, r₂);

}

else if (d > 0)

{

r₁ = (-b + sqrt(d))/(2*a);

r₂ = (-b - sqrt(d))/(2*a);

printf("roots are distinct r₁ = %f r₂ = %f", r₁, r₂);

}

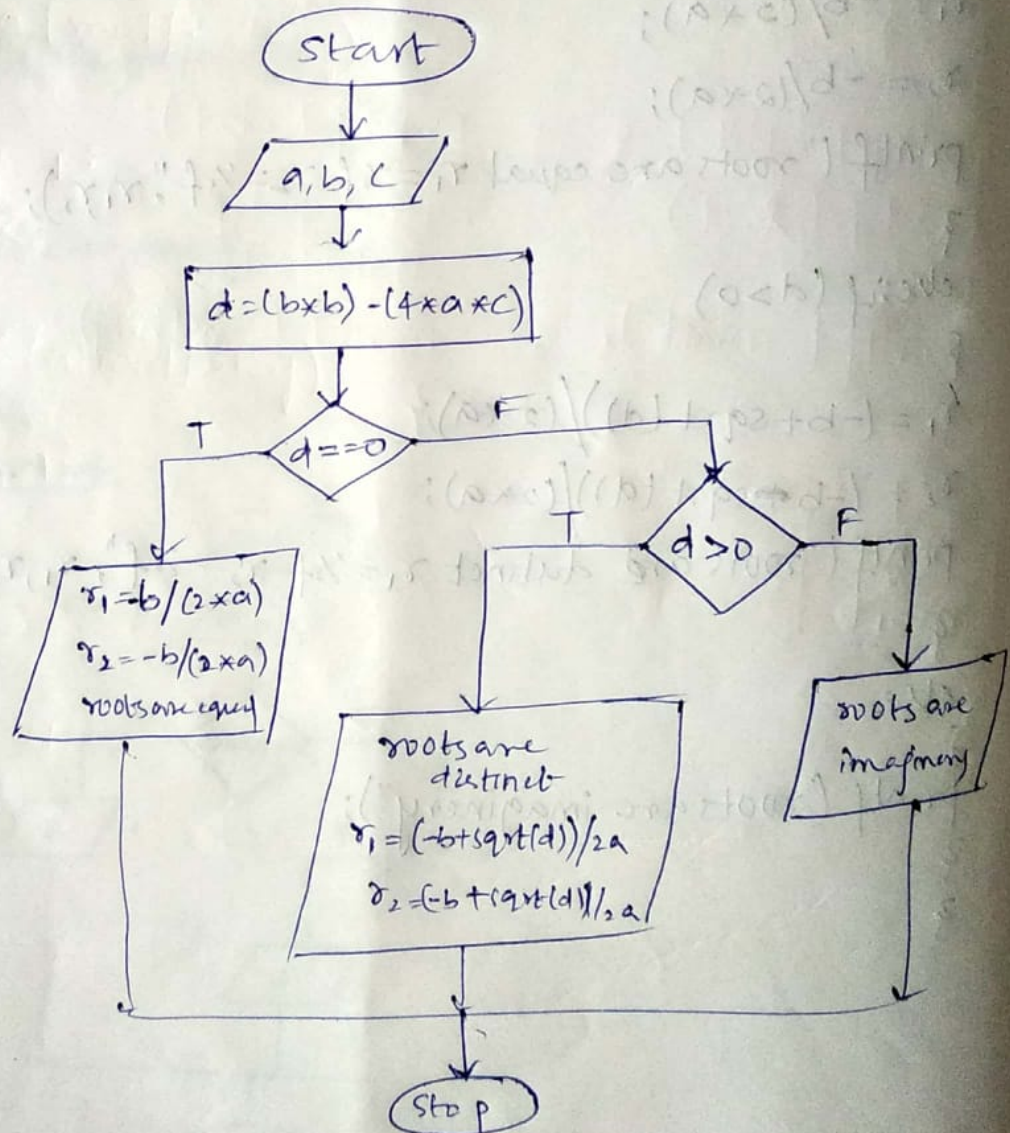
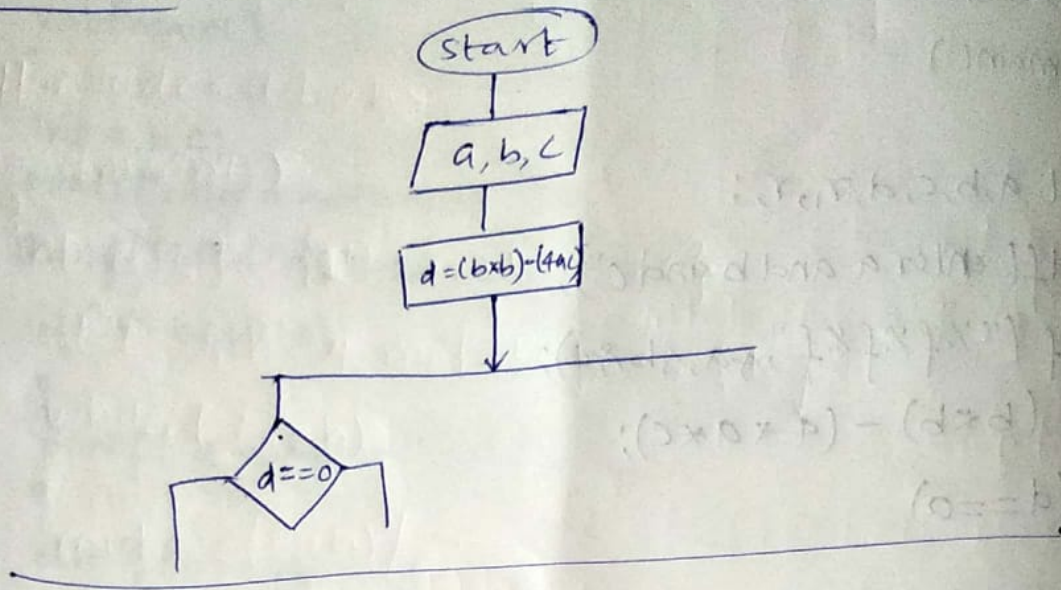
else

{
printf("roots are imaginary");

}

}

Flowchart



Searching Techniques in C-programming

Linear Search:

```
#include <stdio.h>
void main()
{
    int a[5] = {14, 25, 18, 21, 25};
    int i, flag = 0, key;
    printf("enter searching value");
    scanf("%d", &key);
    for(i=0; i<5; i++)
    {
        if(key == a[i])
        {
            flag = 1;
            break;
        }
    }
    if(flag == 1)
    {
        printf("found");
    }
    else
    {
        printf("not found");
    }
}
```

Binary Search:

10	20	30	40	50	60
l					h

$$l=0 \quad h=5 \quad m = \frac{l+h}{2} \Rightarrow \frac{0+5}{2} \Rightarrow 2.5 \Rightarrow \underline{2} \Rightarrow \text{mid} = 30$$

if (key == a[mid])

key = 50

Write a C program to perform Binary Search

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

```
void main()
```

```
{  
  int a[5] = {14, 18, 21, 25, 26};
```

```
  int l = 0, h = 4, mid; key = 25, flag = 0;
```

```
  while (l <= h)
```

```
  {
```

```
    mid = (l + h) / 2;
```

```
    if (a[mid] == key)
```

```
    {
```

```
      flag = 1;
```

```
      break;
```

```
    }
```

```
    else if (a[mid] > key)
```

```
    {
```

```
      h = m - 1;
```

```
    }
```

```
    else
```

```
    {
```

```
      l = m + 1;
```

```
    }
```


Nested structure

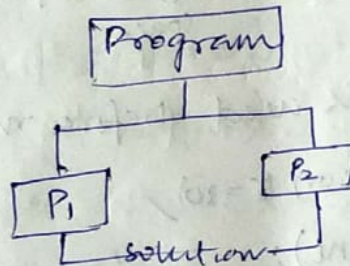
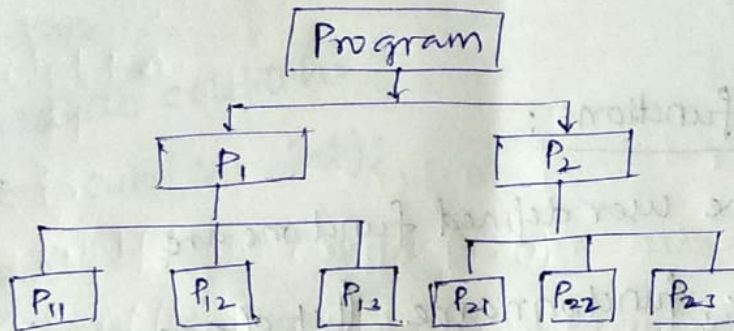
Syntax

Eg: struct std
{
int no;
float avg;
struct proj
{
int pno;
} P;
} S;
printf("enter no avg pno");
scanf("%d %f %d", &s.no, &s.avg, &s.p.pno);
printf("%d %f %d", s.no, s.avg, s.p.pno);
}

- Void pointer } → Differences.
- Null pointer
- Enum

Functions:

- Function is a block of statements are used to perform any task
- Functions are used to reduce the size of the code
- With the help of the functions we can achieve reusability concept.
- Functions are always divided and conquer process
- (or) Functions follow the modular approach method



Classifications of functions:

Library function

Used defined

In-built functions.

Eg of Library functions are

printf();
scanf();
pow();
sqrt();
gets();
puts();

→ Each program must and should maintain atleast one function

* By default we use main() 

→ Example for library function.

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

```
void main()
```

3

```
printf("abc");
```

3

- * library functions should be ended by semicolon

User defined functions:

Syntax for the user defined functions are

return type function name (list of arguments)
 ↓ ↓
 space space

спас

space

User defined function is used before main function

Ex:- $A=10$; $B=20$; Addition ($=30$),

```
int add (int, int);
```

float sub (int, int, int, int).

Write a c-program to add two numbers by using user defined functions.

#include <stdio.h>

Voldmain ()

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

```
int add (int, int);
```

```
main()
```

§

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

```
printf("enter a and b");
```

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



```

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

int add(int x, int y)
{
    int f = x + y;
    return f;
}

```

Write a c-program to subtract two numbers by using functions.

```

#include <stdio.h>
int sub(int, int);
main()
{
    int a, b, c;
    printf("enter a and b");
    scanf("%d %d", &a, &b);
    c = sub(a, b);
    printf("%d", c);
}

int sub(int x, int y)
{
    int z = x - y;
    return z;
}

```

Write a c-program for factorial of a number by using functions.

```

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

```



```

printf("enter b");
scanf("%d", &b);
a = fact(b);
printf("%d", a);
}
int = fact(int x), int
int i, f
for(i=1; i <= x; i++)
{
f = f * i
return f;
}

```

→ Types of user defined functions.

They are divided into 4 types.

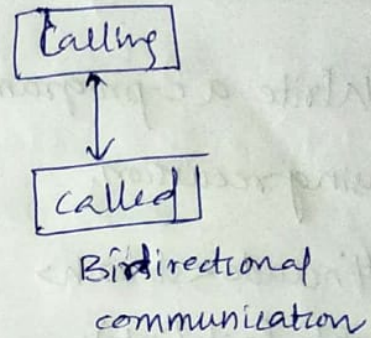
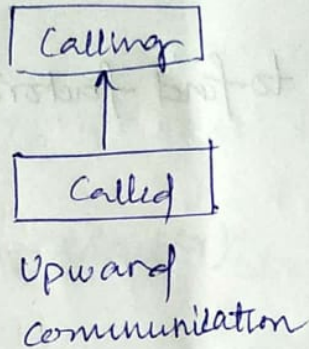
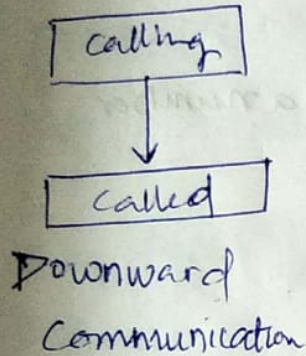
1. Without return type and without arguments.
Eg: void add(void)
2. Without return type and with arguments.
Eg: void add(int, int)
3. With return type and without arguments.
Eg: int add(void)
4. With return type and with arguments.
Eg: int add(int, int)

↓
add

Communications in functions

The communications in functions are

1. Downward communication
2. Upward communication
3. Bidirectional communication



Recursion [Recursive Function]

Recursion is the process of calling the function itself.

Eg:

```
main()
{
    printf("mgit");
    main();
}
```



→ Write a c-program to find out GCD of two numbers using Recursion

```
#include <stdio.h>
int gcd(int, int);
main()
{
    int A, B, C;
    scanf ("%d%d", &A, &B);
    C = gcd(a, b);
    printf ("%d", C);
}
```



```

int gcd(int x, int y)
{
    if (y == 0)
        return x;
    else
        return gcd(y, x % y);
}

```

→ Write a c-program to find factorial of a number using recursion.

```

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

```

→ Write a c-program to find nC_r and nPr values

```

#include <stdio.h>
int ncr(int, int);
int npr(int, int);
int fact(int);

```



```
main()
```

```
{
```

```
int n, r, c, p;
```

```
printf("enter n and r");
```

```
scanf("%d %d", &n, &r);
```

```
c = ncr(n, r);
```

```
p = npr(n, r);
```

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

```
}  
int ncr(int n, int r)
```

```
{
```

```
c = fact(n) / fact(n-r) * fact(r);
```

```
return c;
```

```
}
```

```
int npr(int n, int r)
```

```
{
```

```
p = fact(n) / fact(n-r);
```

```
return p;
```

```
}
```

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

```
{
```

```
if (x == 1)
```

```
return 1;
```

```
else
```

```
return n * fact(n-1);
```

```
}
```


Advantages of Recursion:

- Easy to understand
- With the help of Recursion we can reduce the size of the code

Limitations of Recursion:

- Compilation time is more.
- We require stack (extra memory).
- It cause more overhead.
- Very complex to write the program to write the program with the help of recursion.

Parameter passing technique (or) call by value
call by reference

(or)
Functions to pointer communication
(or)

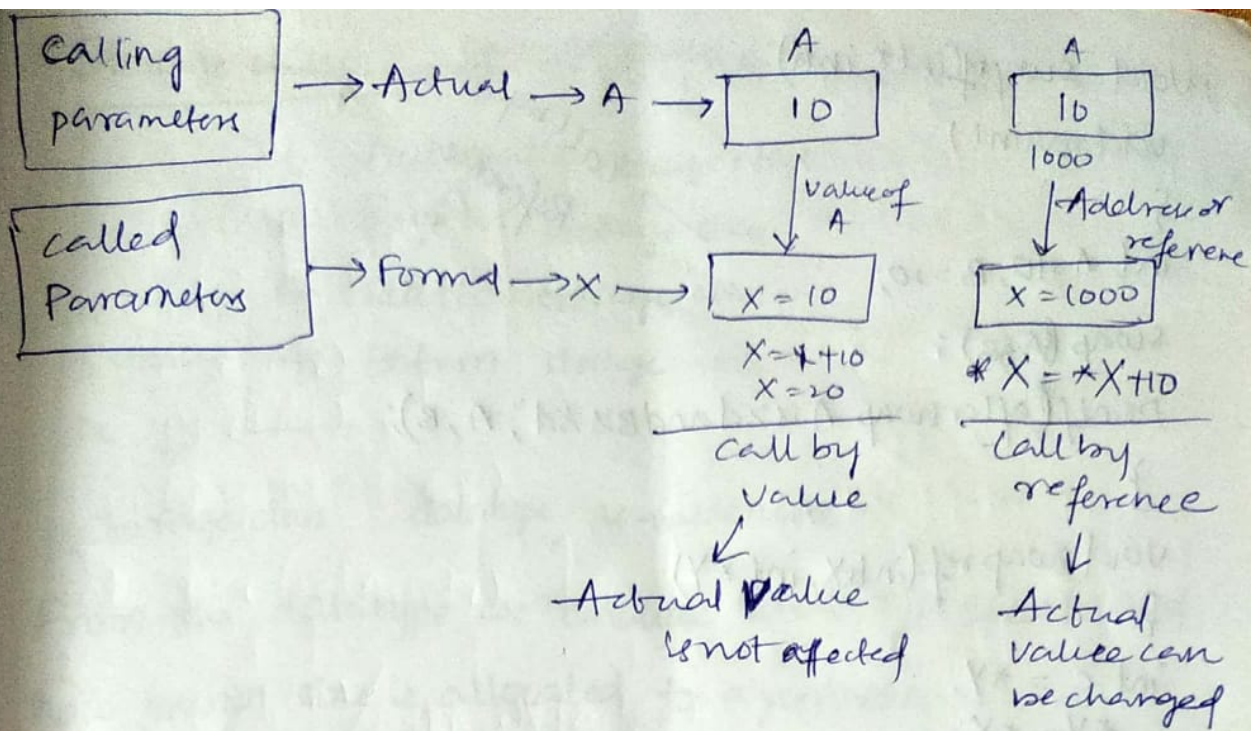
Explain about communication in userdefined functions.

Parameters are divided into the two ~~cat~~ categories

1. Actual parameter (in calling function)
2. Formal parameter (in called function)

[Calling Parameter] → Actual

[Called parameter] → Formal



Example for call by value

```

void call(int);
main()
{
    int A = 10;
    call(A);
    printf("%d", A);
}

void call(int X)
{
    X = X + 10;
    printf("%d", X);
}
  
```

Example for call by reference

```

void call_ref(int*)
void main()
{
    int A = 10;
    call_ref(&A);
    printf("%d", A);
}

void call_ref(int *X)
{
    *X = *X + 10;
}
  
```

Swapping of two numbers by using call by value or call by reference.

Call by value

```

void swap(int, int)
void main()
{
    int A = 10, B = 20;
    swap(A, B);
    printf("%d %d", A, B);
}
  
```

```

void swap(int *X, int *Y)
{
    int c = *Y;
    *Y = *X;
    *X = c;
    printf("%d %d", *X, *Y);
}
  
```



```
void swap(int*, int*)
```

```
void main()
```

```
{
```

```
int A=10, B=20;
```

```
swap(A, B);
```

```
printf("after swap A is %d and B is %d", A, B);
```

```
}
```

```
void swap_ref(int* X, int* Y)
```

```
{
```

```
int c = *Y;
```

```
*Y = *X;
```

```
*X = c;
```

```
printf("%d %d", *X, *Y);
```

```
}
```

Call by Reference

Write the differences between call by value and call by reference

call by value	call by reference
→ Function is called by directly parameter or variable <u>value</u>	→ Function is called by the <u>variable address</u> .
→ Value is <u>not changed</u> in Actual parameter	→ Value is <u>changed</u> in Actual parameter
→ In call by value in declaration we defined only normal variables	→ In call by reference in declaration we define pointer variables.
→ Call by value is a slow process compared to call by reference values.	→ Call by reference is a fast process.

Storage classes : Storage classes are four types.

1. Automatic storage class
2. Register storage class
3. Static storage class
4. ~~Extern~~ storage class

Syntax

is ↓

storage class datatype variable name

From the datatype we analyse what type of data and how much size is allocated to a variable

If you declare any variable under storage class, we can identify

1. Default value
2. Life time
3. Scope
4. Storage

Properties of storage classes :

Keywords	default	Storage	lifetime	Scope
auto	garbage	Memory (RAM)	within block	local
register	garbage	registers	block	local
static	zero	RAM	between function	program
extern	zero	RAM	program	global

(Between two continuous functions)

→ Write a C-program to implement automatic storage class and register

Sol:- #include <stdio.h>

void main()

{

auto int A=10;

{

A=A+10;

printf("%d", A);

}

Default storage class auto storage class

output
A=20

#include <stdio.h>

void main()

{

register int A;

A=A+10;

printf("%d", A);

}

output

A = garbage value

Scope → Region of the variable

Lifetime → How far variable holds the memory.

→ Write the differences between pointer, Null pointer

→ Type def → is used to change the name of datatype

Syntax of Typedef

Typedef datatype userdefine name

Eg:-
Typedef int mgit;
{
mgit A=10;
printf("%d", A);
}

→ Enum Enumeration datatype

↓
keyword



used to assign numerical values to strings
constants

Enum is defined before the main function

Eg:- #include <stdio.h>

```
enum days (sun, mon, tue);
```

```
void main()
```

```
{
```

```
int d;
```

```
enum days d;
```

```
d = mon;
```

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

```
}
```

Auto

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

```
void fun();
```

```
main()
```

```
{
```

```
fun();
```

```
fun();
```

```
fun();
```

```
}
```

```
void fun()
```

```
{
```

```
static int a=0;
```

```
a = a+1;
```

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

```
}
```

output a a a
 1 1 1

Memory used is
6 bytes

Static

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

```
void fun();
```

```
main()
```

```
{
```

```
fun();
```

```
fun();
```

```
fun();
```

```
}
```

```
void fun()
```

```
{
```

```
static int a=0;
```

```
a = a+1;
```

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

```
}
```

output a
 1 2 3

Memory used is
2 bytes.

Extern:

```
#include <stdio.h>
int A=10;
void fun();
{
fun();
}
```

prog.c

```
#include "prog.c"
extern int A;
fun()
{
A = A+10;
printf("%d", A);
}
```

prog2.c

simple example

int A=10 → acts as extern
main()

```
{
int A=20;
printf("%d", A);
}
int A;
printf("%d", A);
```

DMA

Memory is divided into two categories

1. Static Memory.
2. Dynamic Memory.

Static Memory

→ Static memory is fixed
Memory

→ Memory (or) data is constant

→ Pointers are not used

→ Examples are arrays

Dynamic Memory

→ In case of dynamic, it dynamically updates the requirements of memory based

→ No wastage of memory.

→ Pointers are used.

→ Example for Dynamic memory is free()

→ It is very easy to assign memory by the compiler in memory allocation.

→ Fast accessing

→ It is very difficult to assign memory by the compiler in memory allocation.

→ slow accessing

Dynamic Memory allocating functions

1. `Malloc()`;
 2. `Calloc()`;
- } → used for allocating the memory.

For updating the memory the function used is
3. `realloc()`;

4. `free()` → Deleting the memory

Syntax for `malloc()` is

`malloc` → `int *P = (int *) malloc(n * size of (int));`

`calloc` → `int *P = (int *) calloc(n, size of (int));`

`free (pointer);` → `free (*P);`

`Malloc()`

`Malloc()` is DMA function

Used for allocating memory

Syntax

`int *P = (int *) malloc(n * size of (int));`

memory is allocated in single block.

Default value is garbage value

`Calloc()`

`Calloc()` is DMA function

Used for allocating memory

Syntax is

`int *P = (int *) calloc(n, size of (int));`

Memory is allocated in different blocks

here default value is zero.

Syntax

Malloc program for free

$(int *p) \text{ malloc } (n * \text{sizeof}(int));$

```
{
main()
{
printf("enter size of array");
int n;
scanf("%d", &n);
int *p = (int *) malloc (n * sizeof(int));
for (i=0; i<n; i++)
{
p[i] = i+1;
}
free(p);
for (i=0; i<n; i++)
{
printf("%d", *(p+i));
}
}
```

Program for malloc:-

```
{
main()
printf("enter size of array");
int n;
scanf("%d", &n);
int *p = (int *) malloc (n * sizeof(int));
for (i=0; i<n; i++)
{
p[i] = i+1;
}
for (i=0; i<n; i++)
{
printf("%d", *(p+i));
}
}
```

Syntax for free

free(pointer)

output

Garbage value

Syntax for malloc

$int *p = (int *) \text{ malloc } (n * \text{sizeof}(int));$

n=5

output

1
2
3
4
5


```

{ program for realloc
main()
printf("enter size");
int n;
scanf("%d", &n);
int *p = (int *) calloc (n, size of (int))
for (i = 0; i < n; i++)
{
    p[i] = i + 1;
}
int *q = (int *) realloc (p, (n/2) * size of (int))
for (i = 0; i < n; i++)
{
    printf("%d", *(p+i));
}
}

```

program for calloc

```

{
main()
printf("enter size of array");
int n;
scanf("%d", &n);
int *p = (int *) calloc (n, size of (int));
for (i = 0; i < n; i++)
{
    p[i] = i + 1;
}
for (i = 0; i < n; i++)
{
    printf("%d", *(p+i));
}
}

```

Syntax for calloc

$\text{int}^*p = (\text{int}^*) \text{calloc} (n, \text{size of (int)})$

output

1
2
3
4
5


```

→ {
    int A = 10;
    char B = 20;
    float C = 20.20;
    int *P = &A;
    char *q = &B;
    float *r = &C;
    printf("%d %c %f", *P, *q, *r);
}

```

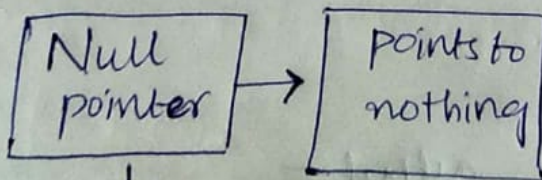
To overcome the problem of writing every datatype we use void pointers. Void pointer is called generic, universal pointer and

```

{
    int A = 10;
    float B = 20;
    void *P;
    P = &A;
    printf("%d", *(int *)P);
    P = &B;
    printf("%f", *(float *)P);
}

```

→ Program for void pointer



points to nothing

void pointer → points to everything

```

{
    int *p = null;
    printf("%d", *p);
}

```


File

File is a collection of information is stored in a computer system
(or)

File is a collection of data which is stored in second storage devices.

File handling functions (or) File input/output functions.

1. `fopen()`;

Used to open a existing file or newly created file.

2. `fclose()`;

Used to close the function.

3. `fprintf()`;

Used to add data to the existing file from the keyboard.

4. `fscanf()`;

Used to read data from the file.

5. `fgetc()`;

Used to give input to the file (or) read data from the file.
(reads character by character)

6. `fputc()`;

Used to write data in the file.

7. `fseek()`;

Used to move file pointer from one position to other.

8. `ftell()`;

Used to identify current position of file pointer.

9. `rewind()`;

Used to get back the file pointer to original position.

`fseek()`;
`ftell()`;
`frewind()`;

} file pointer positioning function.

10. `fgetwc()`;

Used to read integer in the file

11. `fputwc()`;

Used to write integer in the file.

Modes in file

1. read mode $\rightarrow r$

2. write mode $\rightarrow w$

3. append mode $\rightarrow a$

4. $\begin{cases} r+ \\ w+ \end{cases}$ read/write

$\begin{cases} r+b; \\ w+b; \end{cases}$ read and write binary files.

Different types in files.

1. Binary file } streaming

2. Text-file }

Differences between binary file and Text file.

Binary

$\rightarrow$ The extension of the binary files is `.obj`

$\rightarrow$ Very difficult to understand

$\rightarrow$ Less time for compilation.

$\rightarrow$ Less memory

Text

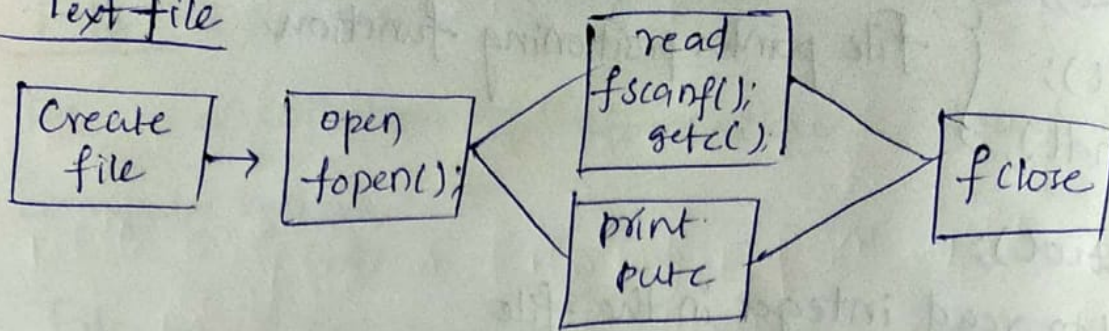
$\rightarrow$ Extension of text file is `.text`

$\rightarrow$ Easy to understand.

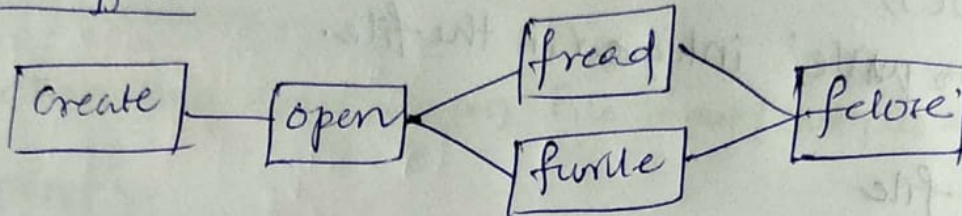
$\rightarrow$ More time for compilation.

$\rightarrow$ More memory.

Text file



Binary file



Write a c-program to demonstrate ~~fprintf~~ read and write operations.

```
{  
FILE *fp;  
fp=fopen("one.c","r");  
if (fp==NULL)  
{  
printf("does not exist");  
}  
else  
printf("file exist");  
fclose(fp);  
}
```

Write a c-program to append the data

```
{  
FILE *fp;  
fp=fopen("one.c","a");  
fprintf(fp,"ABC");  
fclose(fp);  
}
```


Write a c program to demonstrate fprintf and fscanf

```
{
int n;
FILE *fp;
fp = fopen("two.c", "w");
scanf("%d", &n);
fprintf(fp, "%d", n);
fscanf(fp, "%d", &n);
printf("%d", n);
fclose(fp);
}
```

Write a c program to copy one file information to another file.

```
main()
{
FILE *fp1, *fp2;
fp1 = fopen("prog1.c", "r");
if (fp1 == NULL)
{
printf("does not exist");
}
fp2 = fopen("prog2.c", "w");
while(1)
{
char ch = fgetc(fp1);
if (ch == EOF)
break;
fputc(ch, fp2);
}
}
```



```

fclose (fp1);
fclose (fp2);
}

```

Write a c program for merging of many files into one file

```

main()
{
FILE *fp1, *fp2, *fp3;
fp1 = fopen("prog1.c", "r");
if (fp1 == NULL)
{
printf("file does not exist");
}
fp2 = fopen("prog2.c", "r");
if (fp2 == NULL)
{
printf("file does not exist");
}
fp3 = fopen("prog3.c", "w");
while(1)
{
char ch1 = fgetc (fp1);
if (ch1 == EOF)
break;
fputc (ch1, fp3);
}
while (1)
{
char ch2 = fgetc (fp2);
if (ch2 == EOF)
break;
fputc (ch2, fp3);
}
}

```



```
fclose(fp1);  
fclose(fp2);  
fclose(fp3);  
}
```

Write a c-program to count characters, words and lines in a given file.

```
main()
```

```
{  
    int c, w, l;
```

```
    FILE *fp1; *fp2
```

```
    fp1 = fopen("prog1.c", "r")
```

```
    if(fp1 == NULL)
```

```
{  
    printf("file does not exist");
```

```
    int c=0, w=1, l=0;
```

```
    while(1)
```

```
{  
    char ch = fgetc(fp1)
```

```
    if (ch == EOF)
```

```
{  
        break;
```

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

```
    }  
    if (ch == ' ' || ch == '\n')
```

```
{  
        w = w + 1;
```

```
    }
```

```
    if (ch == '\n')
```

```
{  
        l = l + 1;
```

```
    }
```

```
}
```

```
printf("%d %d %d", c, w, l);
```

```
fclose(fp1);
```

```
}
```


Random access Files (or) File position in pointers

Random access files are three types. They are:

- | | <u>Syntax</u> | |
|----------------------------|--------------------------------|------------|
| 1. <code>fseek()</code> ; | <code>fseek (fp, 3, 0);</code> | output = 0 |
| 2. <code>ftell()</code> ; | <code>ftell (fp);</code> | " = 3 |
| 3. <code>rewind()</code> ; | <code>rewind (fp);</code> | " = 0 |

Write a c program to demonstrated random access file

```
#include <stdio.h>
void main()
{
    FILE *fp;
    fp = fopen("prog1.c", "r");
    {
        int pos;
        pos = ftell(fp);
        printf("%d \n", pos);
        fseek(fp, 25, 0);
        pos = ftell(fp);
        printf("%d \n", pos);
        rewind(fp);
        pos = ftell(fp);
        printf("%d \n", pos);
    }
}
```

Pre-processor directives

Pre processor directive is a programme that is used to process the data before passing to the compiler. Preprocessor directives are represented with #

3-preprocessor directive

- Macro substitution
- File inclusion
- Compiler controller.

Macro substitution:

(i) Are used to replace number with string constant

define stringname value

Eg:- # define Pi 3.14

Q) write a C-program to implement macro substitution

```
#include <stdio.h>
#define pi 3.14
void main()
```

```
#include Pi 3.14
{
    int r=5;
```

```
float A=pi*r*r;
```

```
printf ("%f", A);
```

```
}
```

Compiler Controller:

1. #if def

2. #else

3. #else if

```
#define no 1
```

```
{
```

```
#if def no
```

```
{
```

```
printf ("result is 1");
```

```
}
```

```
#else
```

```
{
```

```
}
```

```
#endif
```


Write a c-program to arrange strings in alphabet order,

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

```
void main()
```

```
{  
    (temp[10]) char str[10][10], temp[10];
```

```
    int i, j;
```

```
    for (i=0; i<6; i++)
```

```
    {
```

```
        gets(str[i]);
```

```
    }
```

```
    for (i=0; i<6; i++)
```

```
    {
```

```
        for (j=i+1; j<6; j++)
```

```
        {
```

```
            if (strcmp(str[i], str[j]) > 0)
```

```
            {
```

```
                strcpy(temp, str[i]);
```

```
                strcpy(str[i], str[j]);
```

```
                strcpy(str[j], temp);
```

```
            }
```

```
        }  
    }
```

```
    for (i=0; i<6; i++)
```

```
    {
```

```
        puts(str[i]);
```

```
    }
```

```
}
```

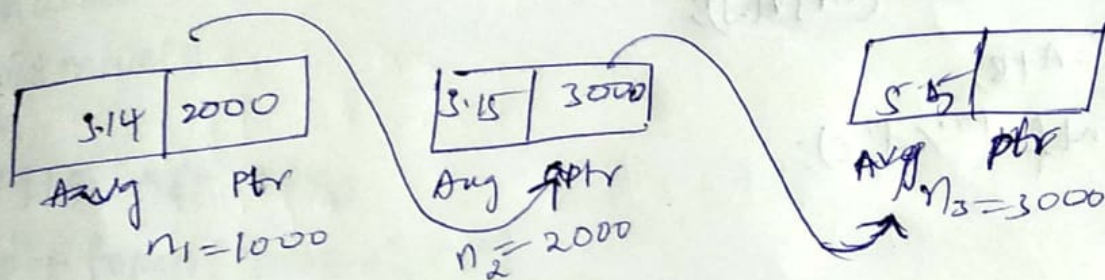

Self Referential Structure:

Syntax

struct st

```
{  
    float avg;  
    struct st *ptr;  
};
```

```
struct st n1, n2, n3;
```



n_1 . Avg = 10;

n_1 . Ptr = $\&n_2$;
 n_2 . Avg = 20;

n_2 . Ptr = $\&n_3$;

n_3 . Avg = 30;

n_3 . Ptr = NULL;

Command line Arguments:

After header file command line arguments are used.

```
int main(int argc, char *argv[])
```

↓
Argument
count
(or)

Input count

↓
Never be zero

↓
Argument
vector

↓
To count the input values.

Write a c program for addition of two numbers using command line argument

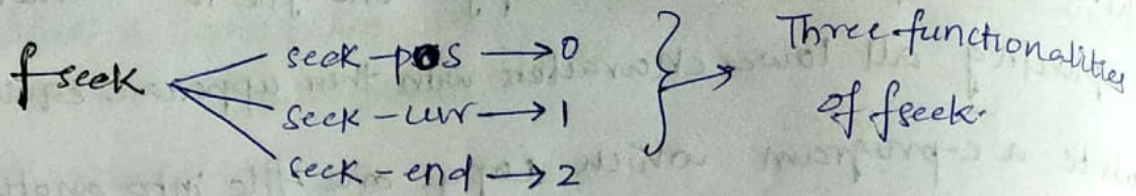
```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int c;
    int A = atoi(argv[1]);
    int B = atoi(argv[2]);
    C = A + B;
    printf("%d", C);
}
```

→ Factorial of the number by using command line Argument.

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a, fact = 1, i;
    a = atoi(argv[1]);
    for (i = 1; i <= a; i++)
    {
        fact = fact * i;
    }
    printf("%d", fact);
}
```


- Write a c-program which copies one file to another, replacing all lower characters with their uppercase equivalents.
- Write a c-program which copies one file into another file replacing all upper case letters with lower case letters.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    FILE *fp1, *fp2;
    fp = fopen("prog.c", "r");
    if (fp1 == NULL)
    {
        printf("do not exist");
    }
    else
        printf("exist");
    }
    fp = fopen("prog1.c", "w");
    while(1)
    {
        char ch = fgetc(fp1);
        to lower(ch);
        if (ch == EOF)
            break;
        fputc(ch, fp2);
        fclose(fp1);
        fclose(fp2);
    }
}
```

Write a C-program to print a ~~file~~ in reverse manner on terminal.

```

#include <stdio.h>

void main()
{
    FILE *fp;
    fp = fopen("prog1.c", "r");
    if (fp == NULL)
    {
        printf("does not exist\n");
    }
    fseek(fp, -1, 2)
    do
    {
        ch = fgetc(fp);
        putchar(ch);
    } while (fseek(fp, -2, 1));
    fclose(fp);
}

```


$$(57)_{10} \rightarrow ()_2 = (11100)_2$$

$$\# (0.625)_{10} \rightarrow ()_2 = (0.101)_2$$

$$\begin{array}{rcl} 0.625 \times 2 = 1.25 & \left| & 1 \quad 0.25 \\ 0.25 \times 2 = 0.5 & \left| & 0 \quad 0.5 \\ 0.5 \times 2 = 1.00 & \left| & 1 \quad 0.0 \end{array}$$

$$\# (12.8125)_{10} \rightarrow ()_2$$

$$12 + 0.8125$$

$$\begin{array}{r} 2 \overline{) 12} \\ \underline{2 \times 6 = 12} \quad 0 \\ \underline{2 \times 3 = 6} \quad 0 \\ 1 \quad 1 \end{array} \Rightarrow 12 = (1100)_2$$

$$0.8125 \times 2 = 1.625 \rightarrow 1 + 0.625$$

$$0.625 \times 2 = 1.25 \rightarrow 1 + 0.25$$

$$0.25 \times 2 = 0.5 \rightarrow 0 + 0.5$$

$$0.5 \times 2 = 1 \rightarrow 1 + 0$$

→ Write a C-program to print the pattern

```
AAAAA
BBBBB
CCCCC
```

→ Write a C-program to count no. of vowels and constants and wide spaces for a given string.

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

```
void main()
```

```
{
    char s[10];
    printf("Enter s");
    scanf("%d", s[i]);
```

```
for (i=0; i<3; i++)
```

```
{
    for (j=0; j<5; j++)
```

```
{
    printf("%d", s[i]);
    printf("\n");
}
```

A	B	C
0	1	2


```

② #include <stdio.h>
   #include <string.h>
   void main()

```

```

{

```

```

    int v, c, s;

```

```

    char ch[20];

```

```

    gets(ch);

```

```

    if (ch == 'a')

```

```

        ch == 'e'

```

```

        ch == 'i'

```

```

        v == 'o'

```

```

        v == 'u'

```

```

    {

```

```

        v = 0;

```

```

        v = v + 1;

```

```

    }

```

```

    if (ch == ' ');

```

```

    {

```

```

        w = 0;

```

```

        w = w + 1;

```

```

    }

```

```

    int l = strlen(ch);

```

```

    c = l - v - w;

```

```

    printf(" %d / %d / %d", v, w, c);

```

```

}

```


Performance of an Algorithm

→ Performance can be (measured) calculated in two different ways.

1. Time complexity
2. Space complexity

* 1. Time complexity: Time required to execute the program is called Time complexity.

There are two types

1. Compile time complexity.
2. Run time complexity.

1. compile time complexity, we are not considering because it depends upon compiler.
2. Run time time complexity can be calculated in two different ways.

(1) Based on number of operations, like no. of additions, no. of subtractions, no. of multiplications etc.

(2) Based on number of steps

Based on number of operations we are not considering because it is different from one machine to another machine

Eg: (Based on no. of operations) x

Alg abc (x, y, z)

{

return $(x+y+z) * (x-y/2) + (x/y+2) - 10;$

}

Based on no. of operations.

$$t_a = 4$$

$$t_s = 2$$

$$t_m = 1$$

$$t_d = \frac{2}{9}$$

Based on no. of steps = 1

Eg: Alg A(n)

```
{  
  for i=1 to n → n+1  
    Pf(i); → n  
  }  
  2n+1 → No. of steps.
```

Eg:- Alg sum(A, n)

```
{  
  s = 0 → 1  
  for i=1 to n → n+1  
    s = s + A[i]; → n  
  return s; → 1  
}  
2n+3
```

Eg:- Alg sum(A, n, m)

```
{  
  s = 0 → 1  
  for i=1 to n → n+1  
    for j=1 to m → n(m+1)  
      s = s + A[i][j] → m*n  
    }  
  return s; → 1  
}  
2mn+2n+3
```


2. Space complexity: Amount of space required to execute the program is called space complexity.

Eg:-

Alg abc(x, y, z)

{
s = 0

s = (x + y + z) / (x - y * z) + (y + z);

return s;

}

s = 1

x = 1

y = 1

z = 1

4

Eg:-

Alg sum(A, n)

{

s = 0

for i = 1 to n

{

s = s + A[i];

}

return s;

}

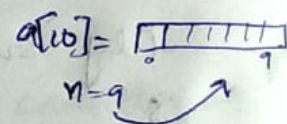
→ s = 1

i = 1

n = 1

A = n + 1

n + 4



Eg:-

Alg sum(A, n, m)

{

s = 0

for i = 1 to n

{

for j = 1 to m

{

s = s + A[i][j];

}

return s;

}

s = 1

i = 1

j = 1

n = 1

m = 1

A → (m + 1) * (n + 1)

eg:- Alg fib(n)

```

{
  if (n ≤ 1)
    return n;
  else
    f1 = 0, f2 = 1;
    for i ≤ n ton
    {
      fn = f1 + f2;
      f1 = f2;
      f2 = fn;
    }
    return fn;
}

```

Space complexity

$$n = 1$$

$$f_1 = 1$$

$$f_2 = 1$$

$$f_n = 1$$

$$i = 1$$

$$\underline{\underline{5}}$$

Time complexity

$$f_1 = 1$$

$$f_2 = 1$$

$$n$$

$$3(n-1)$$

$$1$$

$$\underline{\underline{4n}}$$

Time complexity is also represented in terms of mathematical notations. There are three notations

1. Big O notation

2. Ω notation (omega)

3. θ notation

1. Big O notation:- The function $f(n) = O(g(n))$

if and only if $f \leq c \cdot g(n)$

2. Omega notation.

Bit-field

Used to assign only one bit

If bit-field is not used 16 bits of data is used

If bit-field is used only one bit is used.

int A=0; →

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

 (eg-)

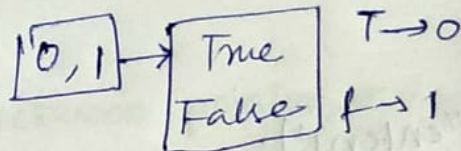
syntax for bit-field

```
int A:1;
```

```
int B:1;
```

```
A=0, B=1;
```

```
printf("%d\n%d\n", A, B);
```



Type qualifiers:

- 1. Constant ←

int
float
char
- 2. Volatile

Ex:-

```
const float A=10.53;
```

```
volatile A = A + 10;
```

```
printf("%f", A);
```

```
}
```

If we want to apply any write operation then

volatile is used

To fix any value constant is used.

Write a c program to demonstrate the function

$$f(x) = \begin{cases} 0 & y=0 \\ 1 & y=1 \\ 2 & y=2 \end{cases}$$

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

```
void main()
```

```
{  
  int x, y
```

```
  printf("enter y");
```

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

```
  if (y=0)
```

```
  {
```

```
    x=0;
```

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

```
  }
```

```
  if (y=1)
```

```
  {
```

```
    x=1;
```

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

```
  }
```

```
  if (y=2)
```

```
  {
```

```
    x=2;
```

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

```
  }
```

```
}
```

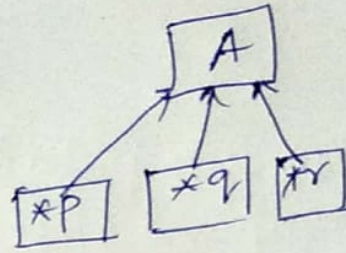

~~restrict~~ int A=10 → syntax

int *p=&A;

int *q=&A;

int *r=&A;

restrict is used not change



Write the differences between preprocessor command and preprocessor director.

What is the difference between pointer and dangling pointer.

Dangling pointer is the ~~erased~~ erased pointer.