LANGUAGE

C

The C Language

(by Dannis Richi)

Significance of C →

- C was developed in 1972 by 'Dannis Richi' at AT and T bell laboratory in USA.
- The main motive of Dannis Richi to create C was to recreate Unix operating system.
- C is a general purpose programming language.
- It is because it have some features of low level and some features of high level.
- C is procedural oriented (POP) or execute code step by step
- Softwares are developed in C language - eg - Unix OS, Database Software, Device Drivers, compilers.
- Different versons of C -
  - K & R verson of C
  - ANSI - C
  - C-89 (1989)
  - C-90
  - C-99
  - C-11
  - C-17 (2017, current)

Translators —

- compiler
- Interpretor

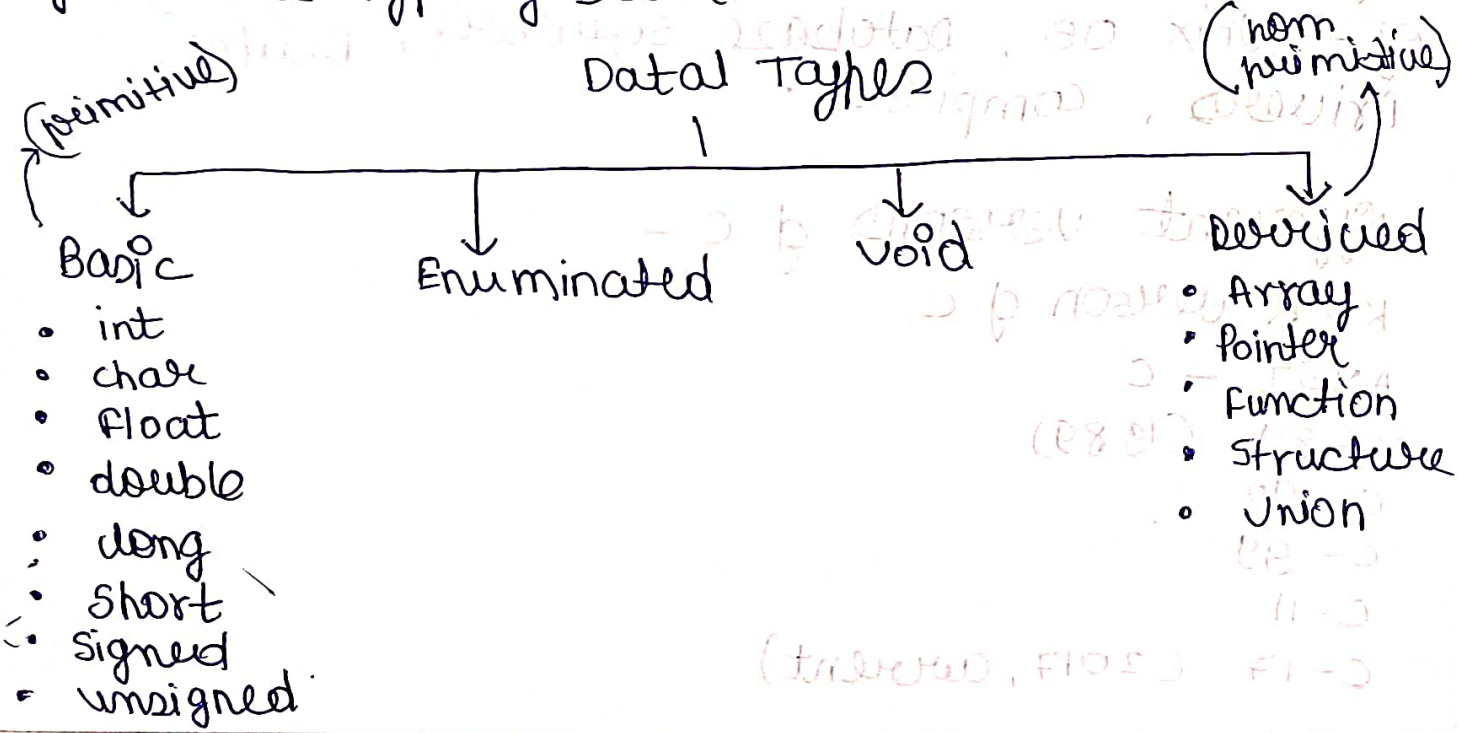
① compiler – compiler is translator which can translate High level language to machine level or low level (0101). It can translate complete code at a time

② Interpreter also do same thing ie HLL to LLL. It is slow since it translate line by line code.

- C uses compiler as translator

Data Types →

There are mainly 4 type of Data Types – Data types are those by which we can find the type of Data.



- Basic / Primary / ~~user~~ predefined (Pre defined)
- Derived / secondary / user defined
- int —
- primitive (Fixed size)
- integer is a numeric data type
- Int can't store decimal point value
- Format specifier of int → %d
- If we have to print one int we have to use 1 %d if 2 integers then 2 %d should be used

eg →
```
int a = 5;
printf ("value of a = %d", a);
return 0;
```

```
int a = 5, b = 2;
printf ("value of a = %d", "value of b = %d",
        a, b);
return 0;
```

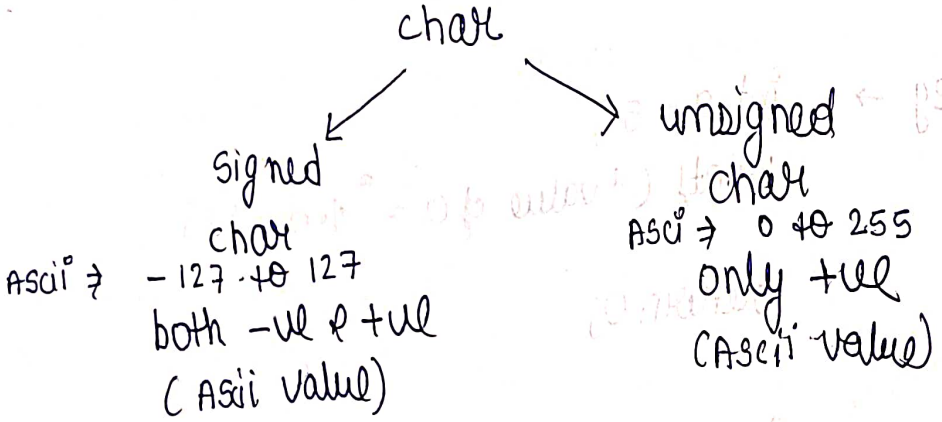
- short Int takes two byte memory where as long int takes 4 bytes memory.
- Range of int →

  −32768 to 32767 (2 bytes)

  −2147483648 to 2147483647 (4 bytes)

- float —
- Numeric data type
- In float we always store decimal values
- format specifier → '%f'
- It takes 4 byte memory
- Range → $1.2e^{-38}$ to $3.4e^{38}$

- char — (character)
- A to Z and @, #, $, & etc.



(Ascii - American std. code for information interchange)

eg →

char	Ascii
A	65
B	66
a	97
b	98
⋮	⋮

- format specifier → '%c'

- char is always defined in single quotes

eg - char a = 'A'
↙ variable ↘ char

- char a = 'A'
  Printf ("%d", a) ⇒ 65 (Ascii value)
  Printf ("%c", a) ⇒ A

Double —

- used to store decimal point values
- takes 8 byte memory to store variable
- stores data type upto 8 decimal places
- format specifier → %lf;
- Range → $-1.7e^{308}$ to $1.7e^{308}$

Long Double —

Takes 10 byte memory.

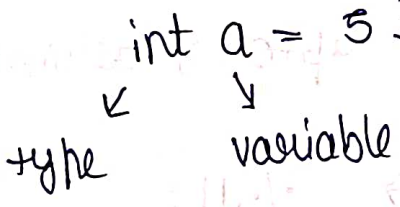
format specifier → %Lf;

Decimal point value upto 15 places

C Tokens →

- Keywords –
- Reserved word (can't used as variable)
- They have predefine meaning
- There are 32 keyword in C
- eg → Printf, for, if, while etc

- variables –
- They are container who can hold the values.



- Local variable are those who are defined in any function
- variables are mutable means means value can be changed.

like a = a+6;

(next page ⇒)

- Constant —
- Also a container who can hold data, once a variable define as a constant variable, whose value can't be changed.
- There are two ways by which we can create constants —

(i) By using cont keyword—

(Const int a = 5 ;) ; → local variable

(ii) By using # define (macro style) Pi 3.14 ;

→ global variable (visibility throughout the programe)

- Identifier —

variable name or function name is known as identifier.

- String literals —

collection of character is called string

In C to store string we use the character array. There is no str data type in c/cpp.

(PTO)

- operators

To be continued....

Program Structure →

#include <stdio.h> } → header files
#include <conio.h> } → collection of library

(used only in turbo C

- stdio → standered input output headerfile
  eg → it stored printf, scanf etc.
- conio → consol input output headerfile
  eg - getch, clrscr stored which are required in turbo C
- # → pre processor directive (pound)
  it loads / saves the header file in to memory while execution of programe
- .h → extension of header file

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

- main function is entry point of programme ie. compiler always call the main function. It is also the exit point of programme.
- void is return type. Function ka return type होता है and variable ka Data type Hota h.

(i)
```
void main()
{
}
```

(ii)
```
int main()
{
return 0;
}
```

(iii)
```
char main()
{
return A;
}
```

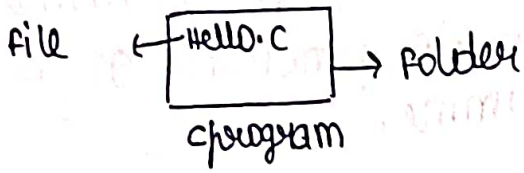
- Line terminator → ; (semi colon)
  This is put after every line to end that lin
- { } → defination
- Hello world →

```
# include < stdio.h>
void main()
{
  printf ("Hello word!");
}
```

↓ string is written in double quotes

To make programe in Notepad / txt editor

→ open c file in terminal



- open terminal
- now → cd program — to open folder
- now → gcc hello.c — to compile
- now → ./a.out (or a.out) — to Run code

- escape sequence character -

/n → new line character
/t → tab space character (8 space)

eg → #include <stdio.h>
void main ()
{
printf ("hello\n");
printf ("bye\n");
}

& output -

← hello
bye

- WAP to add 2 no's -

void main ()
{ int a, b, c;

(next page ⇒)

Rules for variable declaration —

1) comma used as a seperator

eg- int a, b, c ;
3 variables

int abc ;
1 variable

2) space is not allowed in declaration -

eg- int a b c ;
error

int roll number ;
error

3) only '_' underscore is allowed in declaration

eg - int roll_number ; (no other symbols)

4) alphanumeric variable declaration is allowed

eg → a_1, b_1, c_1 ;

" but the first letter should be alphabet or underscore "

eg → int 1a ;
error

- Programe to add 2 no's — (static)

```
# include <stdio.h>
int main ()
{ int a = 5, b = 10, c;
  c = a + b;
  printf ("sum = %d", c);
  return 0;
}
```

output → sum = 15

WAP to add 2 nos (Dynamic ⇒ input)

```
# Include <stdio.h>
int main()
   { int a, b, c;
     printf ("Enter value of a, b \n");
input  Scanf ("%d %d", &a, &b);
```

↓ 2 values a & b int

↓ address operator a के add. पर

```
     c = a + b;
     printf ("sum = %d \n", c);
     return 0;
   }
```

WAP to find Area of Rectangle –

```
# include <stdio.h>
int main()
  { int A, l, b;
    printf ("Enter value of l, b \n");
    Scanf ("%d %d", &l, &b);
    A = l * b;
    printf ("Area = %d \n", A);
    return 0;
  }
```

WAP to find area of circle

```
# include <stdio.h>
int main()
{ float
  int r, A;
  Printf ("Enter value of r\n");
  Scanf ("%.f", &r);
  A = r*r* 3.14;
  Printf ("Area is = %.f", A);
  return 0;
}
```

make habit of use decleration

```
const float pi = 3.14;
```

↳ we declare a float constant pi

WAP TO Find simple Intrest —

```
# include <stdio.h>
int main()
{ float P, R, T, SI;
  Printf ("Enter Principle, Rate & time \n");
  scanf ("%.f %.f %.f", &P, &R, &T);
  SI = P*R*T / 100;
  printf ("simple intrest is = %.f \n", SI);
  return 0;
}
```

[also we can use decleration —
const int a = 100; . $SI = \frac{P*R*T}{a}$;]

WAP To convert temp from

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

```
# include < stdio.h>
int main()
{
  Float F, C;
  Printf(" Enter temp. in °F ");
  scanf("%f", &F);
  C = (F-32)*5/9;
  Printf("Temp in °C = %f", C);
  return 0;
}
```

WAP which takes input of sub. marks and give percent —

```
# include <stdio.h>
void main()
{ float h, m, e, s, c, t, p;
  printf(" Enter marks of h, m, e, s, c");
  scanf("%f %f %f %f %f", &h, &m, &e, &s, &c);
  t = (h+m+e+s+c);
  p = t x 100 / 500 ;
  printf(" percentage = %f", p);
}
```

WAP to find

Sol.

```
#include<stdio.h>
int main()
{ float A, s;
  Printf ("Enter side");
  scanf ("%f", &s);
  A = s* s;
  Printf ("Area = %f", A);
  return 0;
}
```

Swaping Programme →

- Using third variable —

```
# include <stdio.h>
void main()
{ int a, b, c;
  Printf ("Enter value of a, b");
  scanf ("%d %d", &a, &b);
  Printf ("value of a= %d and b=%d before swap\n",
                                    a, b);

    c = a;
    a = b;
    b = c;

  Printf ("value of a = %d & b = %d after swap\n",
                                    a, b);
}
```

• without using 3rd variable -

```
#include <stdio.h>
int main ()
{  int a, b;
   Printf (" Enter value of a, b");
   Scanf ("%d %d", &a, &b);
   Printf (" value of a = %d and b = %d
            before swap", a, b);
```

① ② ③

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

take a = 10, b = 20
a = a+b = 30
b = a-b = 30-20 = 10

```
   printf (" value of a = %d and b = %d
            before swap", a, b);
   return 0;
}
```

WAP to calculate gross salary

```
#include <stdio.h>
{ float bs, ta, da, gs;
  printf (" Enter basic salary bs ");
  scanf ("%f", &bs);
  ta = (bs x 10)/100;
  da = (bs x 20)/100;
  gs = bs + ta + da;
  printf (" gross salary = %f", gs);
}
```

WAP to convert km to m, cm, feet, inch →

Sol.
```
# include <stdio.h>
int main()
{ float km, m, cm, feet, inch;
  printf ("Enter dis. in km \n");
  scanf ("%f", &km);
  m = km * 1000;
  cm = km * 100000;
  feet =
  inch = cm * 2.5;
  feet = inch * 12;
  printf ("m = %f \n", m);
  printf ("cm = %f \n", cm);
  printf ("inch = %f \n", inch);
  printf ("feet = %f \n", feet);
  return 0;
}
```

#. WAP to print ASCII value of character-

Sol.
```
# include <stdio.h>
void main()
{
  char ch = 'A';
  printf ("value of ch = %d \n", ch);
  printf ("value of ch = %c \n", ch);
}
```

Asci value

value of ch ie A

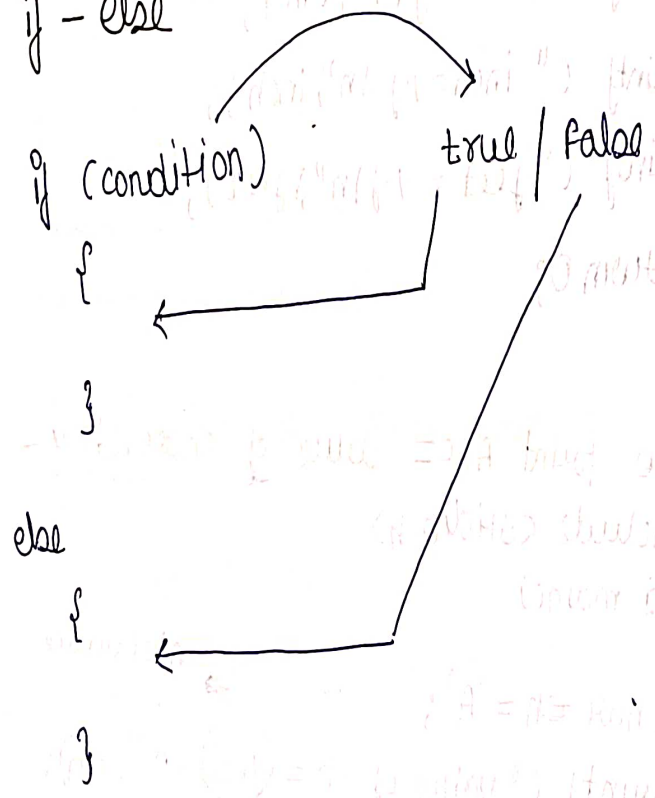
WAP to print size of a datatype -

Sol

```
#include <stdio.h>
void main()
{
    int a;
    a = sizeof(int);
    printf("size of int = %d bytes", a);
    b = sizeof(char);
    printf("size of char = %d bytes", b);
}
```

Conditional Statements →

- if - else



- In If block we can check the condition if condition is satisfied the if block is executed otherwise else block will be executed.
- only if block can be used alone but else block is always accesed with respect to if without if we can't access else.

WAP to check result pass or fair

```
#include<stdio.h>
int main()
{   int p;
    Printf("Enter percentage");
    scanf("%d",&p);

    if (p<30)
      { printf("fail");
      }

    else
      {
       printf("Pass");
      }

return 0;
}
```

WAP whether no. is +ve

```
# include <stdio.h>
int main()
{ int a;
  printf ("Enter no");
  scanf ("%d", &a);

  if (a>=0);
  {printf (" no. is +ve");
  }
  else
  { printf ("no. is -ve");
  }
return 0;
}
```

Operators ⟶

operators are those who can perfor operations on operants.

There are different types of operators in C language—

1) Arithematic operators —

+ , − , * , & , / , %

modulus
remainder

/ → stores quotient

%. → stores remainder

eg → int a=5, b=2;

c = a / b; ⇒ ②

c = a %. b; ⇒ ①

```
   2 → /
2 ) 5
   -4
   ---
    1 → %.
```

2) Assignment operators —

=, +=, -=, *=, /=, %.=

eg → a = 5;

a+ = 10; ⇒ a = a+10

printf ("a/od", a)

output ⇒ 15

a*= 10; ⇒ a = a*10

3) Relational operators —

<, >, <=, >=, ==, !=

↳ not equal to

eg → int a = 5, b=2, c;

c = a>b;

printf ("%d", c);

output → 1 ⇒ (true)

#

eg - int a=5, c;

c = a==5;

Printf ("%d", c);

output → 1 ∵ a==5 is true

①

int a=4, c;

c = a==3;

printf ("%d", c);

output → 0 (false) ∵ a ≠ 3

②

eg → int a=5, c;

c = a != 4;

output → true ∵ a is not equal to 4 (a=5)

#

4) ~~condition~~

WAP to check no. is even or odd-

sol. # include <stdio.h>

void main ()

{ int no.;

printf ("Enter no.");

scanf ("%d", &no.);

if (no.%2==0)

{ printf ("even");

}

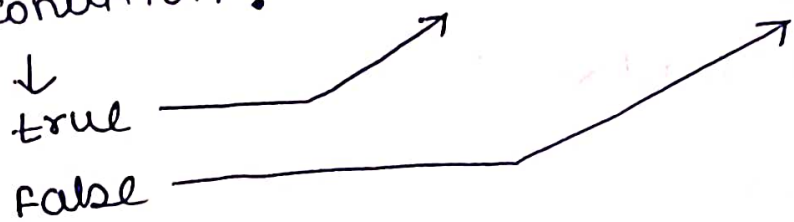
else { printf ("odd");

}

4)

Syntax →

condition ? Statement 1 : Statement 2 ;



eg- Programme to check even or odd-

```
# include <stdio.h>
void main()
{  int no;
   printf (" Enter no");
   scanf ("%d", &no);
   no%2==0 ? printf ("Even") : printf ("odd");
}
```

WAP to find / check max of two numbers -

```
# include <stdio.h>
void main ()
{ int a, b;
   printf (" Enter a & b");
   scanf ("%d%d", &a, &b);
   if ( a>b)
   { printf (" a is max");
   }
   else{
    printf (" b is max");
}
```

else if –

used for nested if-else statements

- WAP to compare 2 no's –

①

```
#include <stdio.h>
int main()
{ int a, b;
  printf ("Enter a and b");
  scanf ("Enter a & b");
  if (a>b)
   { printf ("a is greater");
   }
  else if (b>a) {
    printf ("b is greater");
  else {
    printf ("a = b");
   }
return 0;
}
```

② Error/mistake

ALTER (using conditional operator)

```
a>b ? printf ("a is max") : a==b ? printf ("a & b are equal") : printf ("b is max");
```

cond. 1 — cond. 2

```
# include <stdio.h>
void main ()
{ int a, b;
  printf ("Enter value of a, b");
  scanf ("%d %d", &a, &b);
  a>b? printf ("a is max") : a==b? printf ("b is max");
       : printf ("a is equal to b");
}
```

An example of ternary operators

```
# include <stdio.h>
void main ()
{ int a, b, c;
  Printf ("Enter value of a & b");
  scanf ("%d %d", &a, &b);
  c = a>b? 1 : 0;
        ↓ true   ↘ false
  Printf ("value of c = %d", c);
}
```

5) Logical operators —

- Logical AND (&&) 3/12

All conditions must satisfied than result will true -

eg – if (per >= 0 && per < 33)
{ fail }
elseif (per >= 33 && per < 45)
{ 3rd division }
else if (per >= 45 && per < 60)
{ 2end devision }
elseif (per >= 60 && per <= 100)
{ 1st division }
else
{ invalid percentage }

- Logical OR → (||) या

If any one condition or atleast one condition is true the program will run.

eg →
```
if (ch == 'A' || ch == 'a' || ch == 'E' ||
    ch == 'e' || ch == 'I' || ch == 'i' ||
    ch == 'O' || ch == 'o' || ch == 'U' ||
    ch == 'u')
{ printf ("char is vavel");
}
else
{ printf ("char is consonant");
}
```

but if we and enter no. / int it will give output as garbage value so →

```
#include <stdio.h>
void main()
{ char ch;
  printf (" Enter ");
  scanf ("%c", &ch);

  if ((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))

     {   if (ch=='a' || ch=='A' || ---
         {  }      ----  )
         else { }
         //(full vowel code)

     }

else if (ch>='0' && ch<='9')
   {  printf (" Digit");

   }

else
   {
     printf (" symbol");
   }

}
```

testing

WAP to calculate electricity bill

r → rate, amt → amount, unite → reading

```
#include <stdio.h>
int main()
{  int unit;amt, r;
   Printf ("Enter Unit" \n);
   scanf ("%d", &unit);

   if (unit>0 && unit<200)
      { r=5;
      }
   else if (unit>=200 && unit<500)
     { r=7;
     }
   else
     { r=10;
     }
   amt = r* unit;
   Printf ("bill amount = %d\n", amt);
   return 0;
}
```

WAP to tell the Post of a person

```
include <stdio.h>
main ()
int salary ;
Printf ("Enter salary \n") ;
scanf ("%d," & salary ) ;
if ( salary >0 && salary <= 15000)
  { printf (" peon") ;
  }
else if ( salary> 15000 && salary <= 25000)
  { printf (" clerk") ;
  }
else if ( salary > 25000 && salary <= 50000)
  { printf ("officer") ;
  }
else if ( salary > 50000 && salary <= 80000)
  { printf ("manager") ;
  }
else
  { printf ("not an employee") ;
  }
return 0;
```

Data
0 - 15000
peon
15 - 25
clerk
25 - 50
officer
50 - 80
manager
else
not employe

WAP to check uppercase or lowercase char

```
#include <stdio.h>
void main ()
{ char ch;
  printf ("enter character \n");
  scanf ("%c", &ch);
  if (ch>='A' && ch<='Z') ~~|| (ch>='a'~~
    { printf ("upper case");
    }
else if (ch>='a' && ch<='z')
    { printf ("lower case");
    }
else if (ch>='0' && ch<='9')
    { printf ("Digit");
    }
else
    { printf ("invalid input");
    }
}
```

WAP to convert uppercase to lowercase & vise versa

```
#include <stdio.h>
int main()
{  char ch;
   Printf ("Enter character /n");
   Scanf ("%c, &ch);
   if (ch>='A' && ch<='Z')
      {   ch = ch+32
          printf ("%c", ch);
      }
   else if (ch>='a' && ch<='z')
      {   ch = ch-32
          printf ("%c", ch);
      }
   else {
      print ("syntax error")
      }

return 0;
}
```

Logic →
A → 65
a → 97

diff of 32

A → 65+32 = 97
⇒ a

a → 97-32 = 65
⇒ A

- Logical NOT (! or !=)

```
#include <stdio.h>
void main()
{ int u = 123, p = 456; u1, p1;
  printf ("Enter the value of username
           and password");
  scanf ("%d %d", &u1, &p1);

  If ((u!=u1) && (p!=p1))
    { printf ("login Fail");
    }
  else
    { printf ("login");
    }
}
```

6) Increment & Decrement operators

- used to inc. or dec. the value of variable
- preincrement or predecrement
- postincrement or postdecrement

++ → increement by 1

-- → decrement by 1

postincrement

```
# include <stdio.h>
void main
{ int a=5;
  printf ("%d", a++);
  printf ("a=%d", a);
}
```

a is written 1st so first a is executed & 5 will printed then increment will done

output → 5
6

preIncrement

```
void main
{ int a=5;
  printf ("%d", ++a);
}
```

++ or increment will be done firstly then print statement will executed

output → 6

example →

```
# include <stdio.h>
 void main ()
 {  int a=5;
    printf ("value of a= %d \n", ++a);
    printf ("value of a= %d \n", --a);
    printf ("%d \n", --a);
  }
```

output →

```
6
5
4
```

7) Bitwise operators –

- Bitwise AND → (&)
- Bitwise AND checks all the bits if when all the bits are 1 then it will written 1 and if any bit is 0 then it will return 0.

```
1 & 1 = 1
0 & 1 = 0
1 & 0 = 0
0 & 0 = 0
```

a && b

eg → a = 5 0101

b = 4 0100

AND 0100 = (4)

∴ a && b = 4

```
# include <stdio.h>
void main ()
{  int a=5, b=4;
   printf ("%d", a&&b);
}
```

output → 4

Bitwise OR → (|)

Bitwise OR check all the bits when all the bits are are 1 then it will writtern 1 if any one bit is 1 it will writtern the 1 else 0.

1 | 1 = 1

0 | 1 = 1

1 | 0 = 1

0 | 0 = 0

• Bitwise XOR (^) →

If both bits are same then will give 0 and if both bits are diff then will return 1.

①

$$1 \wedge 1 = 0$$
$$0 \wedge 0 = 0$$
$$1 \wedge 0 = 1$$
$$0 \wedge 1 = 1$$

②

• Bitwise NOT (~) →

It inverts the bits. (gives 2's comp

$$0 \sim 1$$
$$1 \sim 0$$

LSB → least significant bit
MSB → most significant bit

1011 0 → LSB
↙
MSB

**

MSB	sign
1	-ve
0	+ve

computer can ...
it always store the 2's complement form (binary no sys) with sign bit

To find 2's complement first we have to find 1's complement form then add 1 to LSB to make its 2's complement.

To converting in 1's complement just invert the bits.

eg → 1 0 1 0

1's comp = 0 1 0 1

2's comp = 0 1 0 1
+ 1 (add 1 to LSB)

0 1 1 0

(−6)

∵ 2's & 1's comp. represent -ve no.

• Bitwise leftshift & Rightshift →

① Leftshift (<<)

① eg → a = 4 → 0100

leftshift 2 bits

0100 00
← 2 bits towards left

② = 16 (decimal)

m-2

2 (②) × (④)

how many bits to be shift

number

Syntax →

```
{ int a = 4;
  printf("%d", a<<2);
}
```

left shift 2

output → 16

② Right shift (>>)

eg → b = 4

0100 → 3 bits

Right shift 3 bits

0000

= 0

m-2

$\frac{1}{2^a} \times b$

a = how many bits to be shift

b = number

syntax → { int a = 4;
printf ("%d", a >> 2);
}

output → $\frac{1}{2^2} \times 4 = 1$

WAP which takes amount as input and calculates the notes / currency →

```
#include <stdio.h>
int main()
{ int amt, n2000, n500, n200, n100,
        n50, n20, n5, n2, n1;
  printf ("Enter amount = /n");
  scanf ("%d", &amt);
  if (amt >= 2000)
      { n2000 = amt/2000; // n2000 is no. of 2000 notes
        printf ("2000 Rs note = %d", n2000);
        amt = amt - n2000.*2000;
      }                          // updated amount
If (amt >= 500)   // we use if again because we want to check all conditions
      { n500 = amt/500;    // updated amount
        printf ("500 Rs note = %d", n500);
        amt = amt - n500 * 500;
      }
```

```
If (amt >= 200)
  { n200 = amt/200;
    printf ("200 Rs notes = %d", n200);
    amt = amt - n200 * 200;
  }

If (amt >= 100)
  { n100 = amt/100;
    printf ("100 Rs notes = %d", n100);
    amt = amt - n100 * 100;
  }

If (amt >= 50)
  { n50 = amt/50;
    printf ("50 Rs notes = %d", n50);
    amt = amt - n50 * 50;
  }

If (amt >= 20)
  { n20 = amt/20;
    printf ("20 Rs notes = %d", n20);
    amt = amt - n20 * 20;
  }
```

```
If (amt >= 10)
    { n10 = amt / 10;
      printf ("10 Rs notes = %d", n10);
      amt = amt - n10 * 10;
    }

If (amt >= 5)
    { n5 = amt / 5;
      printf ("5 Rs coins = %d", n5);
      amt = amt - n5 * 5;
    }

If (amt >= 2)
    { n2 = amt / 2;
      printf ("2 Rs coins = %d", n1);
      amt = amt - n2 * 2;
    }

If (amt >= 1)
    { n1 = amt;
      printf ("1 Rs coins = %d", n1);
    }
return 0;
}
```

Switch Case →

Switch is conditional statement. If we want to execute one condition from multiple then switch is better and faster way than if else.

Syntax →

```
Switch (choice)
{   case 1:
    ---
    ---
    break;
    case 2:
    ---
    ---
    break;
    case 4:
    ---
    ---
    break
    :
    :
    default:
    ----
}
```

" choice is always integer or character.

{⇒}

Example

```
#include <stdio.h>
int main()
{   int a, b, c, ch;          // ch = choice
    printf ("press 1 for add \n");
    printf ("press 2 for sub \n");
    printf ("press 3 for mul \n");
    printf ("press 4 for div. \n");
    scanf ("%d", &ch);
    switch (ch)
    { case 1:
        printf ("Enter value of a, b");
        scanf ("%d %d", &a, &b);
        c = a+b;
        printf ("sum = %d", c);
      break;
      case 2:
        printf ("Enter value of a, b");
        scanf ("%d %d", &a, &b);
        c = a-b;
        printf ("sub = %d", c);
      break;
      case 3:
        printf ("Enter value of a, b");
        scanf ("%d %d", &a, &b);
        c = a * b;
        printf ("mul = %d", c);
      break;
```

case 4:

```
printf("Enter value of a, b");
scanf("%d%d", &a, &b);
c = a / b;
printf("div = %d", c);
```

① break;

default:
printf("invalid input");

return 0;

}

② # WAP to check even or odd using switch case

```
#include <stdio.h>
int main()
{ int ch, no;
  switch(ch)
    { case 0;
        printf("even no")
      break;
      default;
        printf("odd no:");
    }
return 0;
}
```

ch = no:%2;

wrong code ⇒

correct

```
#include
int main()
{ int ch, no;
   printf("Enter no. \n");
   scanf("%d", &no);
   ch = no.%.2;
   switch(ch)
     { case 0:
          printf("even no.");
       break;
       default:
          printf("odd no.");
       break;
     }
  return 0;
}
```

WAP to find max of the numbers -

```
#include<stdio.h>
void main()
{ int a, b, ch;
  printf("Enter numbers");
  scanf("%d%d", &a&b);
  ch = a>b;
 switch(ch)
  { case 0: // : condition false ⇒ 0
      printf("b is max");
    break;
    default:
      printf("a is max");
} }
```

```
#include <stdio.h>
void main ()
{ int ch; no.;
  printf ("Enter no");
  scanf ("%d", &no);
  ch = no. > 0;
  switch (ch)
  { case 0:
      printf ("no. is -ve");
    break;
    default;
      printf ("no. is +ve");
  }
}
```

Assignment → case 1 area of ○
case 2 area of □
case 3 area of □

```
#include <stdio.h>
int main ()
{ int ch; a, b, c;
  printf ("Enter 1 to find ar(○)");
  printf ("enter 2 to find area of sq.");
  printf ("Enter 3 to find area of □");
  scanf ("%d", &ch);
```

```
switch (ch)
{   case 1 :
        printf ("enter radius");
        scanf ("%d", &a);
        c = π * a * a;
        printf ("area = %d", c);
    break;

    case 2 :
        printf ("Enter sides");
        scanf ("%d %d", &a &b);
        c = a * b;
        printf ("area = %d", c);
    break;
    case 3 :
        printf ("Enter sides");
        scanf ("%d %d", &a);
        bc= a* b;
        printf ("area = %d", c);
                break;
    default :
  } printf ("invalid input");
 return 0;
 }
```

Dot not use global variable more than one time.

Switch statements.

```
#include <stdio.h>
void main()
{ char ch;
  printf ("Enter the character\n");
  scanf ("%c",&ch);
  switch (ch)
  {
     case 'A':
     case 'a':
     case 'E':
     case 'e':
     case 'I':
     case 'i':
     case 'O':
     case 'o':
     case 'U':
     case 'u':
          printf ("Vowel");
     break;

     default:
          printf ("consonant");

}
```

• Nested switch

WAP to find max of two no's with equals to condition -

```
#include <stdio.h>
void main ()
{ int a, b, ch;
  printf ("Enter value of a, b");
  scanf ("%d %d", &a, &b);
  ch = a<b;
  switch (ch)
  { case 0 :
      ch = a>b;
      { switch(ch)
        { case 0:
            printf ("a & b are equal");
            break;
          case 1 :
            printf ("a is greater than b");
            break;
        }
      break;

    case 1 :
      printf ("b is greater than a");
      break;
    default :
      printf ("syntax error");
}
```

Loops

Loops are iterative statements. They are used to execute the statements again and again.

There are two types of loops →

- Entry control loop eg- for, while
- Exit control loop eg - do while

FOR LOOP —

for loop have three parts —

- initialisation
- condition
- increment or decrement

Syntax →

```
for (init; condition; inc/dec)
   {


   }
```

First init & condition will executed
After that code will executed
then inc / dec will be executed

WAP to print hello world 10 times

```
#include <stdio.h>
void main()
{ int i;
  for (i=1; i<=10; i++)
      { printf ("Hello world \n");
      }
}
```

WAP to print series of Hello world n times

```
#include <stdio.h>
int main()
{ int i, no;
  printf ("Enter the no. of times");
  scanf ("%d", &no);
  for (i=1; i<=no; i++)
     { printf ("%d", i);
       printf (" Hello world!");
     }
  return 0;
}
```

WAP to print square series —

```
#include <stdio.h>
void main()
  { int i, n;
    printf ("Enter no");
    scanf ("%d", &no.);
```

```
    for (i=1; i<=no.; i++)
        { printf ("%d,", i*i);
        }
}
```

① output → 1, 4, 9, 16, 25, -----

** If we dont want that last comma

```
for (i=1; i<=no.; i++)
  {
② if (i<no.)
    { printf ("%d,", i*i);
    }
  else
    {
    printf ("%d", i*i);
    }
  }
```

WAP to Print the table of a no. till 10 times

```
#include <stdio.h>
int main ()
{  int i, t, no.;
   printf ("Enter the no");
   scanf ("%d", &no.);
```

⇨

```
for (i<= 1 ; i<=10 ; i++)
    {   t = no * i ;
    }
printf ("%d x %d = %d \n", no, i, t) ;

  }

return 0;

}
```

WAP to print series of even no.

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

WAP to print 10, 9, 8 ------ 1.

```
for (i = 10;  i>=1 ;  i--)
    {   printf ("%d", i);
    }
```

WAP to print sum of series —

```
#include <stdio.h>
int main()
{   int i, no., sum=0;
    printf("Enter the no.");
    scanf("%d", &no);
    for(i=1; i<=no.; i++)
    {
        sum = sum + i;
    }
    printf("sum = %d", sum);
    return 0;
}
```

write WAP to Print factorial of no.

```
#include <stdio.h>
int main()
{   int i, no., fact=1;
    printf("Enter no.");
    scanf("%d", &nd);
    for(i=1; i<=no.; i++)
    {   fact = fact * i;
    }
    printf("fact = %d", fact);
    return 0;
}
```

WAP to print Prime numbers (**)

```
#include <stdio.h>
int main()
{ int i, n;
  printf("Enter the n");
  scanf("%d", &n);
  for (i=2; i<=n; i++)
  {   if (n%i==0)
        { break;
        }
  }
  If (n==i)
  { printf("Prime no.");
  }
  else
  { printf("not prime no.");
  }
return 0;
}
```

0, 1, 1, 2, 3, 5, 8 - - - -

WAP to print Fibonacci series (**)

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

```
int f1 = -1, f2 = 1, f, i, no.;
for (i = 1; i <= no.; i++)
    {   f = f1 + f2;
        printf ("%d", f);
        f1 = f2;        // swaping
        f2 = f;
    }
return 0;
}
```

① ②

✱✱ Logic → eg → no. = 5 (5 times)

f_1	f_2	$f (f_1+f_2)$	i
-1	1	0	1
1	0	1	2
0	1	1	3
1	1	2	4
1	2	3	5

Fibonacci till 5 terms

#WAP to

```
#include <stdio.h>
int main()
{ int i, n, a;
  printf("Enter no.");
  scanf("%d", n);
  for (i=1; i<=n; i++)
     {   a = 1/i;
         If (no. < i) {
              printf("%d+", a); }
         else {
              printf("%d", a); }
     }
  return 0;
}
```

OR

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

(PTO)

- Double for Loop —

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

output → 1,2,3,4,5,

WHILE LOOP →

syntax →

```
int i=1;
while ( condition)
  {
    — —
    — —
    — —
    increment/ decrement;
  }
```

example →

```
void main()
  { int i=1;
    while (i<=10)
      { printf ("%d", i);
        i++;
      }
  }
```

output →

1
2
3
4
⋮
10

```
void main ()
{ int i = 1;
  while (i<=10)
  { if (i==5){
        continue;
    }
    else {
      printf ("%d", i);
    }
    i++;
  }
}
```

output → 1 2 3 4 6 7 8 9 10

∴ continue statement skips the condition

WAP to print sum of digits eg 123 (1+2+3) = 6

Logic →

```
10)123(12
  -120
  ---
    3
```

⇒

```
10)12(1
  -10
  ---
   2
```

⇒

```
10)1(0
  -0
  ---
   1
```

Remainder gives last digit

12 → Quotient gives Rest no.

no. becomes zero

```
# include <stdio.h>
int main()
{
    int no, sum=0; r;
    while(n
    Printf("Enter no.\n");
    Scanf("%d", &no.);
    while(no.>0)
    {
        r = no.%.10;      // remainder
        sum = sum + r;
        no. = no/10;      // Quotient
    }
    Printf("sum = %d", sum);
    return 0;
}
```

WAP to reverse a number -

```
# include <stdio.h>
int main()
{   int no, r, sum=0;
    Printf("Enter no.\n");
    Scanf("%d", &no.);
    while(no.>0)
    {   r = no.%.10;
        sum = sum*10 + r;
        no = no./10;
    }
} return 0;   Printf("reverse no = %d", sum);
```

Palindrome number →

A no. which is same after reversing digits ⇒ eg → 111, 33, 121

```
# include < stdio.h>
void main()
{  int no, r, no1, sum = 0;
   Printf ("Enter no. \n");
   scanf ("%d", &no.);
   no. = no.1;
   while (no > 0)
      {  r = no. % 10;
         sum = sum * 10 + r;
         no = no / 10;
      }
   If (sum == no.1) {
         printf ("palindrome no.");
      }
   else {
         printf ("not palindrome no");
      }
}
```

Armstrong Number →

A no. whose digit are cubed and added then we get same no. is Armstrong no.

eg → is 3, 370, 371, $\{(3)^3 + (7)^3 + (0)^3 = 370\}$

```
# include <stdio.h>
int main ()
{ int no, r, sum=0, no1;
  printf (" Enter no.");
  scanf (" %d", &no);
  no = no1;
  while (no>0)
    { r = no % 10;
      sum = sum + (r*r*r);
      no = no/10;
    }
  If (sum == no1){
      printf ("Armstrng no.");
    }
  else {
    printf (" not Armstrong no.");
  }
return 0;
}
```

WAP to print the first and last digit sum eg → 1234 (1+4) = 5

```
# include <stdio.h>
void main ()
{ int r1, r2, sum, n0;
```

```
printf ("Enter no ");
scanf ("%d", &no);
r1 = no%10;
while (no>0) {
        r2 = no%10;
        no = no/10;
    }
sum = r1+r2;
printf ("sum = %d", sum);
return 0;
}
```

WAP to find no. of digits in given no.

```
#include <stdio.h>
void main ()
int no, count = 1;
printf ("Enter no");
scanf ("%d", &no);
while (no>0){
    count++;
    no = no/10
  }
printf ("no. of digit = %d", count);
}
```

Do While Loop →

Syntax - do {

}

① while (condition);

Example - #include <stdio.h>

② void main()

{ int i=1;

do {

printf ("%d", i);

i++;

}

while (i<=10);

}

WAP to code ATM -

```
#include <stdio.h>
void main()
{ int amt, bal=5000, ch;
  char choice;
  do {
     printf ("press 1 for deposit amount \n");
     printf ("press 2 for deposit/withdraw amount \n");
     printf ("press 3 to check balance \n");
     scanf ("%d, &ch);
```

```
switch (ch)
{  case 1 :
   printf ("Enter amount to be deposit \n");
   scanf ("%d", &amt);
   if (amt > 0)
      {  bal = bal+ amt;
         printf ("amount deposite = %d \n", amt);
         printf ("Total balance = %d \n", bal);
      }
   else {
         printf ("Invalid amount");
        }
   break;

case 2 :
   printf ("Enter amout to be withdraul \n");
   scanf ("%d", &amt);
    if (amt > 0 && bal >= amt)
      {  bal = bal - amt;
         printf ("amount withdraw = %d", amt);
         printf ("Total balance = %d", bal);
      }
   else {
         printf ("Invalid amount");
         break;
```

(PTO)

case 3 :

printf ("Total balance = %d \n", bal);

break;

}

printf ("Press y to continue \n");

getchar ();

scanf ("%c", &choice);

}

while (choice == 'y' || choice == 'Y');

Nested FOR LOOPS →

Pattern →

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

Column ~~Rows~~ depend on Rows

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

j is local so again become 1

i = 1 j wala e
i = 2 j wala 2

i → Row , j → Column

column values are changing so j will printed

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

```
#include <stdio.h>
int main()
{  for (int i=1 ; i<=5 ; i++) {
        for (int j=1 ; j<=i ; j++) {
            printf ("%d", j);
        }
        printf ("\n");
    }
return 0;
}
```

#

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

depends on Row

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

#

```
*
0 0
* * *
0 0 0 0
* * * * *
```

even row → 0
odd row → *
depends on row

①

```
#include <stdio.h>
void main()
{ for (int i=1; i<=5; i++) {
    { for (int j=1; i<=j; j++)
        If (i%2==0)
            { printf ("*");
            }
        else {
            printf ("0");
        }
    }
    printf ("\n");
}
}
```

②

#

```
*
* 0
* 0 *
* 0 * 0
* 0 * 0 *
```

Same but depend on column

even column → 0

odd column → *

#

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

i, j both should run 5 times

```
# include < stdio.h >
int main ()
{  for (int i = 1 ; i <= 5 ; i++)
        {  for ( int j = 1 ; j <= 5 ; j++)
            {  printf ("* ");
            }
            printf ("\n");
return 0;  }
    }
```

#

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

1st & Last Row & column

```
# include <stdio.h>
void main ()
{  for (int i = 1 ; i <= 5 , i++) {
            for (j = 1 ; j <= 5 , j++)
                {  If ( i==1 || i== 5 || j== 1 || j==5
                    { printf ("*") }
                else { printf (" ");}
                        space
                }
            printf ("\n");
}
```

Floric Triangle →

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

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

①

②

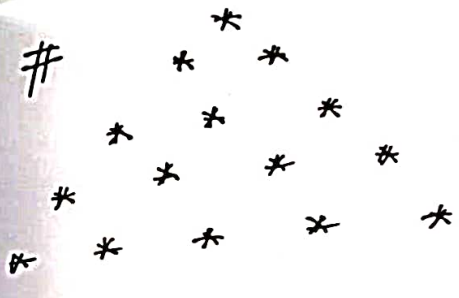

```
A
A B
A B C
A B C D
A B C D E
```

Column dependent

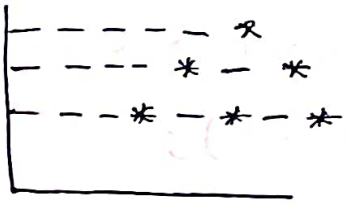
```
#include <stdio.h>
using namespace std;
void main ()
{ char i = 'A', j;
  for ( i = 'A' ; i <= 'E' ; i++) {
        for (j = 'A' ; j <= i ; j++)
        { printf ("%c", j);
        }
        printf ("\n");
    }
}
```

i = 65 or i = 'A'

#



Logic

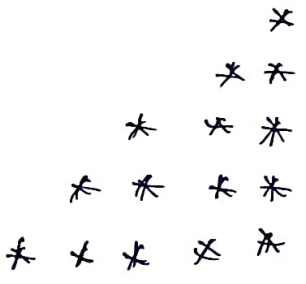


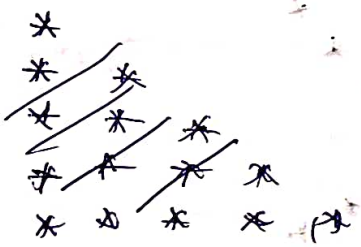
— space
* star

```
for ( i = 1 ; i <= 5 ; i++)   // row
  {
    for ( k = 5 ; k >= i ; k--)   // space
      {
        printf ("  ");
      }
    for ( j = 1 ; j <= i ; j++ )   // column
      {
        printf ("* ");
      }
    printf ("\n");
  }
```

space

If space not given then pattern be like






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

①

```
for (i=1 ; i<=5 ; i++)
{  for (k=1 ; k<=i ; k++)
   {  printf (" ");
   }
```

②

```
for (j=5 ; j>=i ; j--).
{  printf ("* ");
}
printf ("\n");
}
```

space

If space not used

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

#

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

```
#
void main()
{ int i, j, k;
  for (i=1; i<=5; i++)
    { for (k=1; k<=i; k++)
      { printf(" ");
      }
    for (j=5; j>=i; j++)
      { printf("* ");
      }
    printf("\n");
  }
for (i=1; i<=5; i++)
  { for (k=5; k>=i; k--)
    { printf(" ");
    }
  for (i=1; j<=i; j++)
    { printf("* ");
    }
  printf("\n");
  }
}
```

WAP to print series of prime number -

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

FUNCTIONS →

Function is self contained block of statements. Dividing a programme in smaller parts is called acc. to their functionality is called function.

By function we can achive modularity. function is a module.

there are two types of functions—

- Pre defined

  eg → Printf();
  scanf();
  getch();
  gets();
  pow();
  sqrt();
  floor();
  ceil();

- User defined

  are those which are created by user. there are 4 types of user defined functions

  1) Default / NO Argument
  2) Paramatrice func.
  3) ↙ call by value ↘ call by reference
  3) Return type func.
  4) Reccursive func.

there are three imp. Parts of functions →

- function declaration → void show();
- function calling →
```
{
  show(); // inside main
}
```
- function defination →
```
void show()
{ printf();
```

One function can be call from another function or any other function but we can not define a function inside another function.

Example – #include <stdio.h>

```
void show();

void main()
{
  printf("I am in main function\n");
  show();
  printf("I am back in main\n");
}
  void show()
  {
    printf("I am in show function\n");
  }
```

function return always there from where it is called.

Example → programe with two functions which calc. area of circle and area of rectangle –

```
#include <stdio.h>
void area_c();
void area_r();

int main()
{
  area_c();
  area_r();
  return 0;
}
```

```
void area_c ()
  { float pi=3.14, a, r;
    printf ("Enter radius");
    scanf ("%f", &r);
    a = pi *r * r;
    printf ("area of circle = %f", a);
  }
void area_r ()
  { float a, l, b;
    printf ("Enter value of l & b");
    scanf ("%f %f", &l, &b);
    a = l * b;
    printf ("Area of rectangle = %f", a);
  }
```

WAP for even odd as function

```
#include <stdio.h>
void oddeve ();
void main()
{ oddeve();
}
void oddeve() { int no;
{ printf ("Enter no");
  scanf ("%d", &no);
  If (no%2 == 0) {
        printf ("even");
  }
  else {
        printf ("odd");
  }
}
```

make a function for simple intrest →

```
# include <stdio.h>
void sim();
int main()
{
    sim();
}
void sim()
{ int-si float si, pr, p, t;
  printf("Enter rate, principle, time");
  scanf("%f %f %f", &pr, &p, &t);
  si = (pr * p * t) / 100;
  printf("simple intrest = %f", si);
}
```

- Paramatrice function : call by value -

In call by value we always pass the copy of actual argument which is known as formal argument/parameter.

example → max of 2 no.

```
# include <stdio.h>
void max (int a, int b);
void main()
{ int a, b;          // actual argument
  printf("Enter value of a & b");
  scanf("%d %d", &a, &b);
  max(a,b);
}
```

```
void max(int a, int b)
{   if (a>b)
    { printf ("a is max");
    }
    else
    { printf (" b is max");
    }
}
```

// formal argument or copy variable

WAP by call by value to calculate mean/avg.

```
#include <stdio.h>
void avg (int a, int b);
void main ()
{ int a, b ;
  printf (" enter value of a & b ");
  scanf (" %d %d ", &a, &b);
  avg (a, b);
}
void avg (int x, int y)
{ float mean;
  mean = (x + y) /2;
  printf ( " average = %f ", mean );
}
```

// no problem if we change name of variable

WAP to swap

Sol.

```
#include <stdio.h>
void swap (int a, int b);
void main ()
{  int a, b;
   printf ("%d %d", &a &b);
   printf ("value of a = %d before swap
                    in main \n", a);
   printf ("value of b = %d before swap
                    in main \n", b);

   swap (a, b);

   printf ("value of a = %d after swap
                    in main \n", a);
   printf ("value of b = %d after
                    swap in main \n", b);
}
void swap (int a, int b)
{  int c;
   c = a;
   a = b;
   b = c;
   printf ("value of a = %d after swap
                    in swap func. \n", a);
   printf ("value of b = %d after swap
                    in swap funct. \n", b);
}
```

output → a = 2
b = 5

a = 5
b = 2 } swap func.

a = 2
b = 5 } main function (∵ copy)

POINTERS

Pointer to variable →

format specifier for address printing ⇒ %p

If we try to store the address of any variable in normal variable than that normal var is not capable to store the address. If you want to store the address of variable than you have to create a pointer type variable. that means Pointer →

"Pointer is a variable who can store the address of another of another var."

To create any variable as a pointer variable we have to use astrict sub symbol (*) before its name.

Pointer var. stores a same address of variable which is of same type or Data type

(*) → value at that address

Example →

```
#include <stdio.h>
void math ()
{
   int a = 5;
   int *b;
   b = &a;
   printf ("Enter value of a = %d \n", a);
   printf ("address of a = %p \n", &a);
   printf ("address of a by b = %p \n", b);
   printf ("value of a by pointer variable
                  b = %d \n", *b);
}
```

output →

```
5
23125  (added)
23125
5
```

%d & *b → will give value

%p & *b → will give address

Doubt →

%p *b → ?

%d b → ?

- Paramatrice function : call by reference →

```
void show (int *ptr);
void main ()
{
   int a = 5;
   show (&a);   // send address of var
}
void show (int *ptr)   → will catch value of a
                         ∵ we use *ptr
   {
   printf ("value of a = %d", *ptr);
   }
```

swaping programe using Pointers —

" In call by reference if we change the value of former argument then value of actual argument will also change "

```
# include <stdio.h>
void swap (int *a, int *b);
void main ()
{
   int a, b;
```

```
printf ("Enter value of a,b");
scanf ("%d%d", &a&b);
printf ("value of a = %d before swap in
                main\n", a);
printf ("value of b = %d before swap
                in main\n", b);
swap (&a, &b);
printf ("value of a = %d after swap in
                main\n", a);
printf ("value of b = %d after swap in
                main\n", b);
}
void swap ( int *a, int *b)
{   int c;
    c = *a;
    *a = *b;
    *b = c;
printf ("value of a = %d after swap in
                swap function\n", *a);
printf ("value of b = %d after swap
                in swap function\n", *b);
}
```

b = 5

a = 5, b = 1 } swap func.

a = 5, b = 1 } main func ∵ we send address

WAP to calculate full salary using call by reference →

```
# include<stdio.h>
void sal (int* bs);
void main ()
{ int bs;
  printf ("Enter basic salary ");
  scanf ("%d", &bs);
  sal (&bs);
}
void sal (int *bs)
{
  int ta, da, gs;
  ta = (*bs x 10) / 100;
  da = (*bs x 20) / 100;
  gs = ta + da + *bs;
  printf ("total salary = %d", gs);
}
```

WAP to check .ASP

```
# include < stdio.h>
Void asp( int *a,  int *b, int *c);

Void main ()
{   int a, b, c;
    printf ( " Enter value of three angles of Δ");
    scanf ( "%d %d %d", &a, &b, &c );

    asp (&a, &b, &c);

}

void asp (int *a; int *b, int *c)

{   int sum = 0;
    sum =  *a + *b + *c;
    If ( sum == 0)
        {
          printf ("True triangle, asp verified");
        }
    else
      {
        printf ( " False triangle");
      }
}
```

Return of

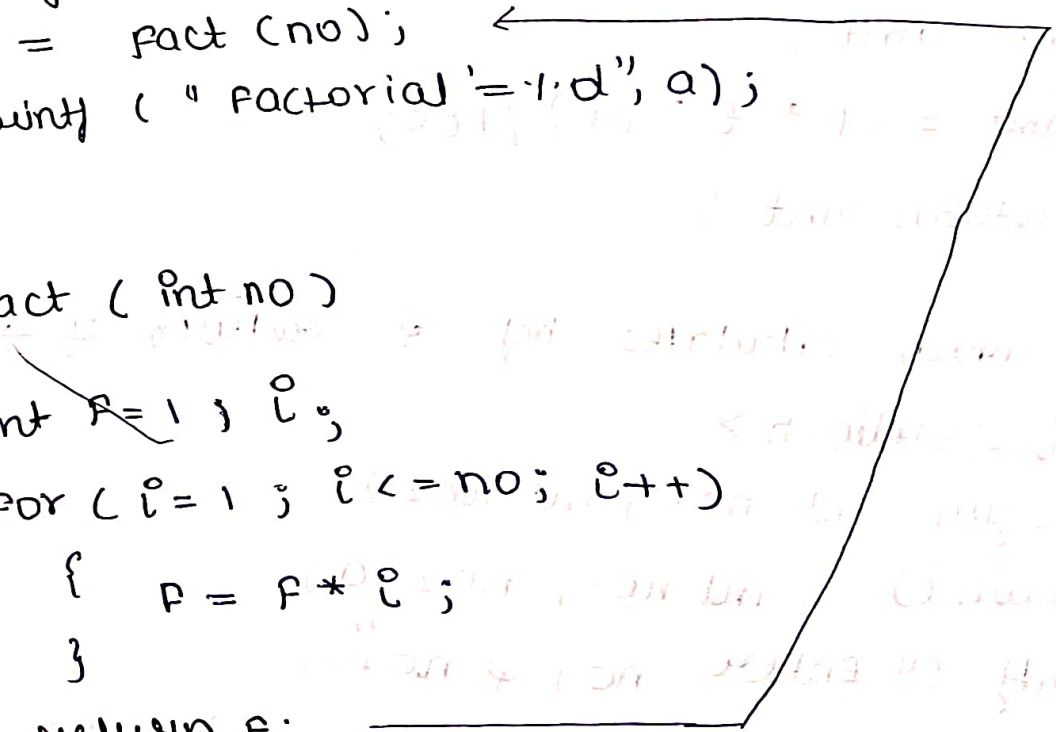
A function returns some value from where it is called.

Example → Prog. which return int Factorial –

```
# include <stdio.h>
int fact (int no);

void main ()
{ int no, a;
  printf ("Enter the no");
  scanf ("%d", &no);
  a = fact (no);
  printf ("Factorial = %d", a);
}

int fact (int no)
{
  int f=1, i;
  for (i=1; i<=no; i++)
    { f = f * i;
    }
  return f;
}
```



WAP which returns simple intrest →

```
#include <stdio.h>
int sip( int p, int t, int r);

void main ()
 { int a; p, t, r;
   printf ("Enter p, t, r");
   scanf ("%d, %d %d", &p, &t &r);
   a = sip ( p, t, r);
   printf (" amount = %d", a);
 }

int sip (int p, int t, int r)
 { int amt;
   amt = (p * t * r) / 100;
   return amt;
 }
```

WAP which calculates hcf & return it →

```
#include <stdio.h>
int gcd_fun (int no1, int no2);
void main ()  int no1, no2; a;
{ printf (" Enter no1 & no2");
  scanf ("%d %d", &no1, &no2);
  a = gcd_fun (no1, no2);
  printf (" hcf of no1 & no2 is = %d", a);
}
```

```
int gcd_fun (int no1 , int no2)
{   int i, hcf ;
    for ( i=1 ; i<=no1 && i<=no2 , i++ )
        {    If (no1 %. i ==0 && no2 %. i ==0)
                { hcf = i ;
                }
        }
    return hcf ;
}
```

$$LCM = \frac{a*b}{hcf}$$

Reccursion type functions →

when function called itself again and again then such function is called reccursive function.

Example

```
#include <stdio.h>
void show (int no);

void main()
{   int no;
    printf ("Enter no");
    scanf ("%d", &no);
```

```
    show (no);
}

void show (int no)
{
    if (no>0)
    { printf ("%d", no);
      show (no-1);
    }
}
```

LCM logic code (Assignment)

```
#include <stdio.h>
int main()
{ int n1, n2, i, lmax, j;
  printf (" Enter n1 & n2 ");
  scanf ("%d %d", &n1, &n2);
  If (n1>n2)
  { max = n1;
  }                          // since LCM is greater
  else (n2>n1)                  than both numbers
  { max = n2;
  }
  while (max > 0)            // ∞ loop condition not
  { i = max %. n1;              matter
    j = max %. n2;
```

```
    If (i==0 && j==0)
      {  break;
      }
else { max++; }
 }
 printf ("lcm = %d", max);
 return 0;
}
```

hcf code (assignment)

```
#include <stdio.h>
int main()
{ int n1, n2, i, min, j;
   printf ("Enter n1 & n2");
   scanf ("%d %d", &n1, &n2);
   If (n1>n2)
   {  min = n2;
   }
   else
   {
      min = n1;
   }
   whi
```

(P T O)

```
while (min>0)
{ i= n1 % min;
  j= n2 % min;
  if (i==0 && j==0)
    { break;
    }
  else {
    min--;
    }
}
printf ("hcf = %d", min);
return 0;
}
```

decimal to binary (assignment)

```
#include <stdio.h>
void main()
{ int no, r, a, sum=0, bin=0;
  printf ("enter the decimal no \n");
  scanf ("%d", &no);
  while (no>0)
  {
    r = no % 2;
    sum =(sum * 10) + r;
    no = no/2;
  }
```

```
// reversing
while (sum>0)
{   a= sum%10;
    bin = (bin*10) +a;
    sum = sum/10;
}
printf ("binary form is = %d", bin);
}
```

Header file thru function

we have to make function in other file for example →

File 1 show.c

```
void show (int no)
{   if (no>0)
    {  printf ("%d", no);
       show (no-1);
    }
}
```

File - 2 a.c

```
# include <stdio.h>
# include "show.c"
void main ()
{ int no;
Printf ("Enter no");
scanf ("%d", &no);
show (no);
}
```

WAP to print sum of series using recursion

```
#include <stdio.h>
void show (int no);
void main()
{  int no;
   printf ("Enter no");
   scanf ("%d", &no);
   show (no);
}

void show (int no)
{
   static int sum=0;        // making sum as global var
   if (no>0)                // while recursion it will go out of func (scope) so its value will again initialize to zero.
   {
     sum = sum + no;
     show (no-1);
   }
   printf ("%d", sum);
}
```

WAP to print factorial using recursion.

```
#include <stdio.h>
void show (int no);
void main()
{  int no;
   printf ("Enter no");
   scanf ("%d", &no);
   show (no);
}
```

```
void show (int no)
{ static int fact = 1;
  if (no>0)
  { fact = fact * no;
    show (no-1);
  }
}
```

Fibonacci using Recursions →

Logic →

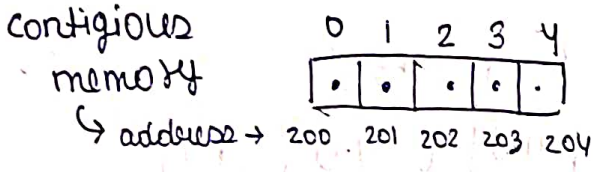
```
void feb (int no)
{
  static int f1 = -1, f2 = 1, f;
  { f = f1 + f2;
    printf ("%d", f);
    f1 = f2;
    f2 = f;
    feb (no-1);
  }
```

ARRAY

Array is the collection of similar data types → int, char, float etc.

If we want to store the same type of data then array is recommended rather than individual variable.

Array is based on index system. that is elements of array stored according to index. These index starts from (0) zero and ends at (n-1) index.



"why array index starts from 0?"

Array is a contigious memory allocation

memory taken by array

= no of elements in array × size of data type

(m1)
```
printf (
printf ("%d", a[1]);
        :
```

(m2)
```
for (i=0; i<=4; i++)
{ printf ("%d", a[i]);
}
```

Types of array -

- 1-D array
- 2-D array
- multi dimension array

Be. each type of array have static and dynamic array

If elements of array are written in code (or at code level) then such type of array case is called static array.

else if user inputs the values during run time then such type of array is called dynamic array.

initialization →

```
int a[] = { 1, 2, 3, 5 };
          0  1  2  3

float a[] = { 2.5, 3.2, 6.1 };

char a[] = { 'A', 'B', 'C' };
```

Program to [illegible]

```
#include <stdio.h>

void main()
{   int a[] = {2,5,7,9,3};
    int i;
    for (i=0; i<=4; i++) {
      printf ("%d \n", a[i]);
      }
}
```

" Array index starts from 0 because in memory array store elements like this -

$a = *(a+i)$

if i=1 the it will skips first block :: a+i

so array indexing starts from 0 "

WAP to sum all elements of array

```
#include <stdio.h>
void main()
{ int a[] = {1,2,3,4,5};
  int i, sum = 0;
  for (i=0; i<=4; i++)
    { sum = sum + a[i];
    }
  printf ("%d", sum);
}
```

WAP to [illegible] given by user and sum them (Dynamic)

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

   for (i=0; i<=4; i++)
   {  sum = sum + a[i];
   }
  printf("sum = %d", sum);
}
```

WAP to print max element of array →

```
#include <stdio.h>
void main()
{  int a[] = {2, 3, 9, 5, 7};
   int i, max;
   max = a[0];
   for (i=1; i<=4; i++)
  {   if (a[i]>max)
       {  max = a[i];
       }
  }
   printf("max element = %d", max);
}
```

why i=1 // 0 index is stored in max & we will compare it to 2nd element ie 1 index

WAP which prints min element of array

```
# include <stdio.h>
void main()
{
    int a[] = {2,3,9,5,7};
    int i, min;
    min = a[0];
    for (i=1; i<=4; i++)
    {
        if (a[i] > max)
        {
            max = a[i];
        }
    }
    printf ("min = %d", min);
}
```

WAP to add elements of two different array

```
# include <stdio.h>
void main()
{
    int a[] = {1,2,3,4,5};
    int b[] = {5,4,3,2,1};
    int c[];
    for (int i=0; i<=4; i++)
    {
        c[i] = a[i] + b[i];
    }
    printf ("Sum of array elements");
```

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

output → {6, 6, 6, 6, 6}

WAP to print which is max of two arrays

```
# include <stdio.h>
void main ()
{  int a[ ] = { 1,2,3,4,5};
   int b[ ] = { 5, 4, 3,2,1};
    Pnt i , max , max1 ;
    max = a[0];
    max1 = b[0];
    for (# int i = 1; i<= 4; i++)
    {   if ( a[i] > max)
          { max = a[i];
          }
        if ( b[i] > max1)
          {
            max1 = b[i];
          }
    }
  if (max > max1)
    { printf ("max element = %d", max);
    }
  else { printf ("max element = %d", max1);
       }
}
```

-: ALTER :-

```
# include <stdio.h>
Void main()
{
   int  a[] = {1, 2, 3, 4, 5};
   int  b[] = {5, 4, 3, 2, 1};
   int  i, max;
   max = a[0];
   for (i=0; i<=4; i++)
   {
      if (a[i] > max)
        { max = a[i];
        }
      if (b[i] > max)
       {  max = b[i];
       }
   }
   printf (" max = %d ", max);
}
```

Linear Search

Logic

5	7	9	13	12
0	1	2	3	4

```
if no = 9
no == a[0]
no == a[1]
;
;
```

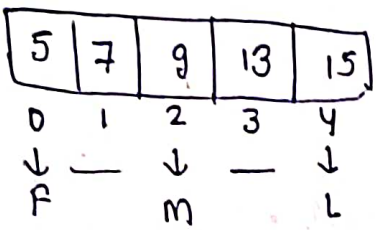
Slow search technique

```
#include <stdio.h>
void main()
{ int a[] = {5,2,7,8,9};
  int i, no;
  printf("Enter no for search");
  scanf("%d", &no);
  for (i=0; i<=4; i++)
     { if (a[i] == no)
        { printf("no. found");
          break;
        }
     }
  If (i> 4)
    {
     printf("no. not found");
    }
}
```

Binary search

"Binary search always work on sorted array."

Logic →



$f = 0$, $l = 4$ } $m = \frac{f+l}{2} = 2$

Ascending order { $m > no.$ ⇒ ~~Right~~ left side of array
$m < no$ ⇒ Right side of array

```
#include <stdio.h>
void main()
{
    int a[] = {2, 5, 7, 9, 13};
    int no, f, l, m;
    f = 0;
    l = 4;
    m = (f+l)/2;
    printf("Enter the no");
    scanf("%d", &no);
while (f<=l)
  { if (a[m]<no)
      { f = m + 1;
      }
   else if (a[m] == no)
      {
        printf("no found");
        break;
      }
   else { l = m-1;
        }
   m = (f+l)/2;
}
```

```
if (f > l)
{ printf ("no not found");
}
```

2-D ARRAY

$$\begin{bmatrix} a[0,0] & a[0,1] & a[0,2] & \cdots & a[0,n] \\ a[1,0] & a[1,1] & a[1,2] & \cdots & a[1,n] \\ a[2,0] & a[2,1] & a[2,2] & \cdots & a[2,n] \\ \vdots & \vdots & \vdots & & \vdots \\ a[n,0] & a[n,1] & a[n,2] & \cdots & a[n,n] \end{bmatrix}_{n \times n}$$

eg → $\begin{bmatrix} - & - \\ - & - \end{bmatrix}$ 2 row & 2 column

```
# include <stdio.h>
void main()
{   int a[2][2], i, j;        (// Dynamic)
    printf ("Enter elements in matrix \n");
    for (i=0; i<=1; i++)
    {  for (j=0; j<=1; j++)
       {  scanf ("%d", &a[i][j]);
       }
    }
```

```
printf ("Elements of matrix");
for (i=0; i<=1; i++)
{
    for (j=0; j<=1; j++)
    {
        printf ("%d\t", a[i][j]);
    }
    printf ("\n");
}
}
```

WAP to print sum of all elements of matrix

```
#include <stdio.h>
void main()
{
    int a[2][2], i, j, sum = 0;
    printf ("Enter elements of matrix");
    for (i=0; i<=1; i++)
    {
        for (j=0; j<=1; j++)
        {
            scanf ("%d", &a[i][j]);
        }
    }
printf ("elements of matrix \n");
```

Ans

```
for (i=0 ; i<=1 ; i++)
    { for (j=0 ; j<=1 ; j++)
        {   sum = sum + a[i][j];
        }
    }
printf (" sum of all elements of matrix = %d",
                                    sum);
}
```

WAP to find transpose of matrix

eg → $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^T$

a i j a j i

```
# include <stdio.h>
void main ()
{ int a[2][2], b[2][2], i, j;
  printf (" Enter the elements of matrix A \n");
  for (i=0; i<=1; i++)
     {
       for ( j=0 ; j<=1 ; j++)
         {
           scanf ("%d",&a[i][j]);
         }
           printf ("\n");
     }
```

```
printf ("normal matrix \n");
for ( i=0 ; i<=1 ; i++)
  { for ( j=0 ; j<= 1 ; j++)
      { printf ("%d \t", a[i][j]);
      }
  } printf ("\n");

printf ("transpose matrix \n");
for ( i=0 ; i<=1 ; i++)
  { for ( j=0 ; j<=1 ; j++)
      { printf  b[i][j] = a[j][i];
        printf ("%d \t", b[i][j]);
      }
    printf ("\n");
  }

}
```

WAP to add two matrix →

logic → input a[i][j] & b[i][j]

c[i][j] = a[i][j] + b[i][j]

```
#include <stdio.h>
void main()
{ int a[2][2], b[2][2], c[2][2]; i, j;
  printf("Enter elements of matrix A");
  for(i=0; i<=1; i++)
    { for(j=0; j<=1; j++)
      { scanf("%d\t", &a[i][j]);
      }
      printf("\n");
    }
  printf("Enter in matrix B");
  for(i=0; i<=1; i++)
    { for(j=0; j<=1; j++)
      { scanf("%d\t", &b[i][j]);
      }
      printf("\n");
    }

  for(i=0; i<=1; i++)
    { for(j=0; j<=1; j++){
        c[i][j] = a[i][j] + b[i][j];
        printf("%d", c[i][j]);
    }
}
```

- initialisation of
  - int a [2][2] = {{2,3}, {3,4}}
  - int a[3][2] = {{1,2,3}, {2,5,6}}

WAP to find matrix multiplication

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 1 \end{bmatrix} = \begin{bmatrix} 2\times5 + 3\times7 & 2\times6 + 3\times1 \\ 4\times5 + 5\times7 & 4\times6 + 5\times1 \end{bmatrix}$$

```
# include <stdio.h>
void main()
{  int sum=0, i, j, k;
   int a[2][2], b[2][2], c[2][2];
   int sum = 0;

   printf("Enter elements of matrix a \n");
   For (i=0; i<=1; i++)
     {   FOR (j=0; j<=1; j++)
         {  scanf("%d", &a[i][j]);
         }
     }
```

```
printf ("Enter ...");
for (i=0; i<=1; i++)
    { for (j=0; j<=1; j++)
        { scanf ("%d", &b[i][j]);
        }
    }

printf ("multiplied matrix \n");
for (i=0; i<=1; i++)
 { for (j=0; j<=1; j++)
     { for (k=0; k<=1; k++)
         { sum = sum + (a[i][k] * b[k][j]);
         }
       c[i][j] = sum;
       sum = 0;
     } printf ("%d/t", c[i][j]);
   } printf ("/n");
}
```

sparse matrix →

A matrix in which max or more than 50% elements are zero

no of elements = $i * j$

$\therefore$ 50 percent elements means = $\frac{i * j}{2}$

Programme to check sparse matrix

```
#include <stdio.h>
Void main()
{
  int a[2][2], i, j, count = 0;
  printf("Enter elements of matrix\n");
  for (i=0; i<=1; i++){
      for (j=0; j<=1; j++){
          scanf("%d", &a[i][j]);
      }
  }

for (i=0; i<=1; i++){
      for (j=0; j<=1; j++)
  {   if (a[i][j] == 0)
          {
              count ++;
          }
      }
  }
```

```
if (count > (i*j)/2)
    {
    printf ("sparse matrix\n");
    }

else
 { printf ("not a sparse matrix\n");
 }

}
```

Pointer to Array

```
#include <stdio.h>
void main()
{ int a[5] = {1,2,3,4,5};
  int *ptr, i;
   ptr = &a[0]
   for (i=1; i<=5; i++)
      { printf ("%d", *ptr);
        ptr++;
      }
}
```

address को आगे बढ़ाएगा।

WAP to find max element using pointer

```
# include <stdio.h>
void main ()
{
    int a[] = {1, 2, 3, 4, 5};
    int i, * max;
    max = & a[0];
    for (i=0; i<=4; i++)
        { if (*max < a[i]) {
            *max = a[i];
        }
    }
    printf ("%d", *max);
}
```

Array of Pointer

To store string type array we have to always use pointer type array.

Example

```
# include)< stdio.h>
void main()
{
    char * name[] = {"rahul", "Dev", "Ram"};
    int i = 0;
```

```
for (i=0 ; i<=2 ; i++)
    {
      printf ("%s \n", name[i]);
    }
}
```

array is Pointer type

Passing a Array in function

```
#include <stdio.h>
void show ( int a[5] );

void main()
{ int a[] = {1, 2, 3, 4, 5};
  show (a);
}
void show (int b[5])
   { int i;
     for (i=0; i<=4; i++)
     {
       printf ("%d", b[i]);
     }
   }
```

STRING

"we not use '&' to scan string"

collection of characters is known as string. Last character of string is called null character.

format specifier %s

null character → '\0'

There are diff. str handling functions To use these func we include <string.h> file

- **strlen()** →

  This function is used to find the length of given string

- example →

```
# include <stdio.h>
# include <string.h>
void main()
{
    int d;
    char name[] = "Devansh";
    d = strlen(name);
    printf("length = %d\n", d);
}
```

without using strlen ⇒

```
char name[] = "Devansh";
int i=0, count=0; i = 0;
```

(**) while (name[i] != '\0')
→ last char or null char
{
i++
}

printf ("%d", i);

strupr()
convert the given string in uppercase
strlwr()
strlwr() convert in lowercase

"without using these functions" –

To convert in upercase

```
# include <stdio.h>
# include <string.h>
void main()
{ int i = 0;
  char name[] = "mohit";
  while (name[i] != '\0')
  { if (name[i] >= 'a' && name[i] <= 'z')
```

```
        {
            name[i] = name[i] - 32;
        }
    i++;
}
printf("upper case = %s\n", name);
}
```

To convert in lowercase

```
#include <stdio.h>
#include <string.h>
void main()
{   int i = 0;
    char name = "MOHIT";
    while(name[i] != '\0')
    {  if(name[i] >= 'A' && name[i] <= 'Z')
        {   name[i] = name[i] + 32;
        }
    i++;
    }
printf("lower case = %s\n", name);
}
```

wh... ... array →

```
#include <stdio.h>
void main()
{   int a[] = {5, 10, 15, 20, 25};
    int i, size = 5, temp;
    for (i=0; i<size/2; i++)
    {
        temp = a[i];
        a[i] = a[size-i-1];
        a[size-i-1] = temp;
    }
    printf("Reverse of array elements\n");
    for (i=0; i<=4; i++)
    {
        printf("%d", a[i]);
    }
}
```

- strrev()

This function is used to reverse the string

without using strrev() we will reverse an string →

(PTO)

Program to reverse an string

```
# include <stdio.h>
# include <string.h>
void main()
{
    char name = "mohit";
    char temp;
     int i=0, l;
    l = strlen(name);
    while ( i< l/2)
    {
       temp = name[i];
       name[i] = name[l-i-1];
       name[l-i-1] = temp;
       i++;
    }
 puts (name);
}
```

- Puts () →

This function is used to point string.

- strcmp () →

used to compare two string

If both string are identical then it returns zero otherwise returns the difference of asci value.

example →

```
char name[] = "mohit";
char heap[] = "abcd";
d = strcmp (name, heap);
printf ("value of d = %d", d);
```

- gets ()

This function ~~[illegible]~~ the reads the string.

- strcpy ()

This function used to copy one string to another.

```
char name[] = "abcd";
char lame[] = "efgh";
char heap[10];

strcpy (~~name~~ heap, name);
strcpy (name, lame);

puts (heap);
puts (name);
puts (lname;
```

- strcat()

  concunitate or adds or joins the two string

```
char name[] = "mohit";
char lame[] = "thakkar";

strcat(name, lame);   name में lame को add करेगा
puts(name);

output → mohit thakkar
```

- gets()

  input a string / scan a string

```
char name[20];
gets(name);
```

Assignment → make all above as user defined functions strcmp, strcpy, strcat

WAP to cut a string from a given position entered by user

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

```
char name[20];
int pos, i, s;
printf("Enter a string");
gets(name);
s = strlen(name);   // size
printf("Enter the position");
scanf("%d", &pos);
if(pos<s)
  {  while(name[pos] != '\0')
       {
         printf("%c", name[pos]);
         pos++;
       }
  }
else
  {
    printf("invalid");
  }
}
```

-: STRUCTURE :-

- Structure is a collection of dissimilar or hetrogeneous data types.
- Structure can be defined by → 'struct' keyword.
- Structure members can be access with the help of structure variable with dot operator.
- Structure is a user defined data type
- Structure ends with a semicolon.

syntax for structure variables

```
struct student
{
    char name[20];
    int age;
    int RollNo;
} s;
```

structure members

↓ structure variable

```
void main()
{
    struct student s;
}
```

structure variable

Example →

```
struct student
{   char name[20];
    int age;
    int roll;

} s;

void name()
{   printf("Enter name, age & roll no");
    scanf("%s %d %d", &s.name, &s.age,
                      &s.roll no);

    printf("name = %s\t aage = %d \t
                roll no = %d \t",
                    s.name, s.age, s.roll no);
```

WAP to make structure of book

```
#include <stdio.h>
struct BOOK
{ char name[20];
  int price;
  int page;
  char authorname[20];

};
```

```
void main()
{   struct BOOK s;
    printf("Enter name, price, page and
            authors name of the book \n");
    scanf("%s %d %d %s", &s.name,
          &s.price, &s.page, s.author
          name);
    printf("name = %s\t, price = %d
           page = %d, author = %s)
          , s.name, s.price,
           s.page, s.author name);
}
```

Array of structure

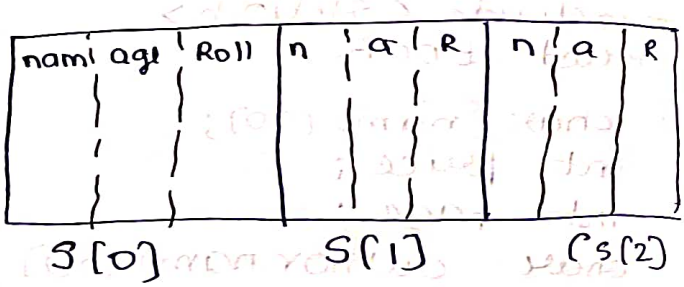
It is used to store multiple records

example →

```
struct student
{ char name[20];
  int age;
  int rollno;
} [s[3]];
```

structure array variable

⇒



memory allocation

Accessing → s[0].name
s[1].name
s[2].rollno
etc.

Programme —

```
# include <stdio.h>
struct student
{   char name[20];
    int age;
    int roll no;
} s[3];

void main()
{  int i;
   for (i=0; i<=2; i++)
   { printf ("Enter name, age & roll \n");
     scanf ("%s %d %d", s[i].name, &
                        s[i].age, & s[i].roll);
   }

printf ("name \t age \t rollno \t\n");
```

```
for (i=0; i<=2; i++)
{
    printf ("%s\t%d\t%d\n",
        s[i].name, s[i].age, s[i].rollno);
}
}
```

in above if we want to make condition that we have to print whose age is greater than 25

```
if (s[i].aage > n)
```

Inside loop for printing structure

inputed age by user

Assignment → make student record of 10 students and then search diff. things.

name, Age, salary, Post employee struct & find those whose age exp > 5 year & salary < 25000 and increment their salary by 20 %.

```
#include <stdio.h>
struct st employee
{  struct name;
   int age;
   int salary;
   int experience;
} s[5];

void main()
{  int i;
   for (i=0; i<=4; i++)
      { printf("Enter name, age, sal & exp \n");
        scanf("%f.2f %d %d %d", s[i].name, &s[i].age,
                                &s[i].sal, &s[i].exp);
      }
   printf("name \t age \t \ salary \t exp \n");
   for (i=0; i<=4; i++)
      {  if (s[i].salary < 25000 && s[i].exp > 5)
         {
            s[i].salary = s[i].salary + (s[i].sal
                                  x 20/100);
            printf("%f.2f \t %d \t %d \t
                        %d \n");
            s[i].name, s[i].age,
            s[i].sal, s[i].exp);
         }
      }
}
```

Pointer to Structure

' Accessed by Arrow (→) operator:

```
# include <stdio.h>
Struct student
{   char name[20];
    int age;
    int rollno;
} S1;
void main()
{   struct student *ptr;
    ptr = &S1;
    printf("Enter the student name, age & rollno
    scanf("%s %d %d", ptr->name, &ptr->age,
                                 &ptr->roll no);
    printf("name\t age \t rollno\n");
    printf("%s\t%d\t%d\n", ptr->name,
                ptr->age, ptr->rollno);
}
```

u

→ operator means value at that address therefore we are not using *ptr→"

WAP for [illegible] of array

```
#include <stdio.h>
struct employee
{   char name[20];
    int age;
    int sal;
}   s[3];

void main()
{   int i;
    struct employee *ptr;
    ptr = &s[0];

    for (i=0; i<=2; i++)
    {   printf("Enter name age & salary\n");
        scanf("%s %d %d", ptr->name, &ptr->age,
                          &ptr->sal);
        ptr++; // increasing address s[0] -> s[1] ---
    }

    ptr = &s[0];

    for (i=0; i<=2; i++)
    {   printf("%s %d %d", ptr->name, ptr->age,
                           ptr->sal);
        ptr++;
    }
}
```

Nested Structure

Whenever we create a structure inside any structure then its called nested struct

Example →

```
struct student
{   char name[20];
    int age;
    int rollno;
    struct std_fees
           {
              int fees;
           } F;
} S;

void main ()
{  printf ( " Enter name, age, rollno with fees \n");
   scanf ( "%s %d %d %d", s.name, &s.age,
                          &s.rollno, &s.F.fees );

   printf ("%s %d %d %d", s.name, s.age,
                          s.rollno, s.f.fees );
}
```

WAP ... fees is less than total fees.

```
#include <stdio.h>
struct student
{ char name[20];
  int age;
  int rollno;
   struct std_fee
      { int fees;
        int total_fees;
      } f[3];
} s[3];

void main()
{ int i;
  for (i=0; i<=2; i++)
  { printf ("Enter student name, age, rollno & fees\n");
    scanf ("%s %d %d %d %d", s[i].name, &s.(i).age,
           &s[i].rollno, &s[i].f[i].fees);
  }                , &s[i].f[i].total_fee

  printf ("name \t age \t roll no \t fees \n");
  for (i=0; i<=2; i++)
  {
    if (s[i].f[i].fees< s[i].f[i].total_fees)
      {
        printf ("%s %d %d %t", s[i].name, s[i].age,
                 s[i].rollno, s[i].f[i].fees);
  }
}
```

Project →

Student Magnagement system

welcome to stu. mag. sys.

1. insert data
2. view all
3. search by fees
4. search by result
5. Exit

Structure to function

```
# include <stdio.h>

struct student
{
   char name[20];
   int age;
};

void show (struct student (s));
```

```
void main ()
{   struct student s;
    printf ("Enter name & age \n");
    scanf ("%s %d", s.name, &s.age);
    show (s);
}

void show (struct student s)
{   printf ("name \t age \n");
    printf ("%s %d", s.name, s.age);
}
```

UNION

- union is also a collection of disimilar data types.
- structure works on idivijual memory concept that is structure allocate memory to all data types differently but union works on share no memory concept. that is union allocates the highest required memory only and this memory will be share one by one in all data members.

- Defining a union we use the union keyword. for example →

∵ it is shared memory so first name will take memory then to scan & print name & then age will take memory - - - -

```
#include <stdio.h>
union student
{
    char name[20];
    int age;
} s;
void main()
{
    printf("Enter name\n");
    scanf("%s", s.name);
    printf("name = %s", s.name);
    printf("Enter age");
    scanf("%d", &s.age);
    printf("new age = %d", s.age);
}
```

FILE HANDELING I/O

- File handeling is a mechanism which is used to store the data of c programe in file or we can read data from the file.
- To perform this, we require some pointer (File pointer ⇒ FILE *ptr) to store the address of file.
- To open a file there is a function called Fopen(); which is used to open file in different modes.
- Fopen() function have two parts
  - ↳ File name
  - ↳ opening mode

  Syntax : fp (pointer) = fopen ("A.txt", "r");
- There are different modes in which the files are opened.

- r mode - In r mode [illegible] mode we can read data from file.
  "If file is already exist"
  If file not exist then file pointer have null value so no file will be opened.

- w mode - this mode is used to the write the data in the file If file is not already exist it will create a new file and write data in it.
  " w to mode always over -write ~~that~~ data in the fil

- a mode — Append mode used to append or add data to the previous data

- rb / wb / ab modes - All above modes works with text file but these (rb / wb / ab) works with binary files.
  extention for binary → a.dat

- r+ | w+ | a+ — These perform both read and write operation

- File handeling supports diff. functions →

- eg fgetc( ); — This fuction is used to read single char for from the file.

{Example → char ch;
fp = fopen("A.txt","r");}

full code →

```
#include <stdio.h>
void main()
{
    char ch;
    FILE * fp;
    Fp = Fopen("A.txt", "r");
    if (fp == NULL)
    { printf("File not found");
    }
    else {
        ch = fgetc(fp);
        printf("%c", ch);
    }
    fclose(fp);
```

A.txt
Hello
^

output → H

Always close the file

• Reading full file using fgetc() →

**

```
#include <stdio.h>
void main()
{
    char ch;
    FILE *fp;
    fp = fopen("A.txt", "r");
    if (fp == NULL)
    {
      printf("File not found");
    }
    else
    {
       while ((ch = fgetc(fp)) != EOF)
       {
          printf("%c", ch);
       }
    }
}
```

End of file

In files pointer will move forward automatically so no increment or decrement in the loop.

• fputc(); — It is used to write the single character of the file (input)

eg →

```
#include <stdio.h>
void main ()
{
  FILE * fp;
  char ch;
  printf ("Enter char to write");
  scanf ("%c", &ch);
  fp = fopen ("B.txt", "w");
  fputc (ch, fp);
  printf ("data written");
  fclose (fp);
}
```

so a file B.txt will created (if not exist) and data i.e. single char will be loged in the file.

fputc in append mode →

```
fp = fopen ("B.txt", "a");
fputc (ch, fp);
```

so data will be loggend without overwriting in the file.

multi characters to be logged in fputc()

```
# include < stdio.h>
void main ()
{
    ~~FILE~~ FILE *fp;
    char ch[20];
    int i=0;
    printf ("Enter name \n");
    scanf ("%s", ch);

    fp = fopen ("A.txt", "a");

    while ( ch[i] != '\0')
    {  fputc ( ch[i] ; fp);
       i++ ;
    }

fputc ('\n', fp); // for cline change
printf ("data loged");
fclose (fp);
}
```

* WAP to copy data from file to another.

```
# include <stdio.h>
void main()
{  FILE *fs, *ft;
   char ch;
   fs = fopen("A.txt","r");
   ft = fopen("B.txt","a");
   if (fs == NULL)
   { printf("file not exist");
   }
   else
   {
      while ((ch= fgetc(fs)) != EOF)
          {
            fputc(ch, ft);
          }
      printf("file copied\n");
   }
   fclose(fs);
   fclose(ft);
}
```

- fprintf () → This function used to write formatted output in the file

example →

```
#include <stdio.h>
void main()
{
    File FILE *fp;
    char ch[200];
    fp = fopen("B.txt", "w");
    fprintf (fp, "welcome to string\n");
    fclose (fp);
}
```

- fscanf () → This function used to read formatted input from given stream in C

```
# include <stdio.h>
void main()
{
  FILE *fp;
  char ch[200];
  fp= fopen ("B.txt", "a");
  printf ("Enter data\n");
```

```
scanf ("%c", &ch);
fprintf (fp, "%c", ch);
fputc ('\n', fp);
fclose (fp);
fp = fopen ("B.txt", "r");
while (fscanf (fp, "%c", ch) != EOF)
    { printf ("%c", ch);
    }
}
```

- fputs() → This function used to write formatted string in the file..

```
#include <stdio.h>
void main()
{
    FILE *fp;
    fp = fopen ("A.txt", "w");
    fputs ("welcome to the city", fp);
    fclose (fp);
}
```

- fgets() → this function used to read formatted string

```
#include<stdio.h>
void main()
{
  FILE *fp;
  char data[200];
  fp = fopen("A.txt","r");
  printf("%s", fgets(data,200,fp));
  fclose(fp);
}
```

- fseek() → this function is used to set file pointer to specified offset. It is used to write data in a file at desired location.

```
#include<stdio.h>
void main()
{
  FILE *fp;
  fp = fopen("A.txt","w");
```

```
fputs ("mohit sraten tata", fp);
          0 1 2 3 4
fseek (fp, 4, SEEK_SET);
          ↳ set pointer to 4th index

fputs ("file start here", fp);

fseek (fp, 0, SEEK_END);    // set to end

int l = ftell (fp);
printf ("length = %d", l);
fclose (fp);

}
```

- ftell () → This function used to tell the current position of the file pointer.

- Rewind () → This function used to set file pointer at default position or again to start ie zero

  syntax → rewind (fp);

```
#include <stdio.h>
void main()
{
    FILE * fp;
    char ch;
    fp = fopen ("A.txt", "r");
    while ( (ch = fgetc(fp)) != EOF)
    {
        printf ("%c", ch);
    }
    rewind (fp);
    while ((ch = fgetc(fp)) != EOF)
    {
        printf ("%c", ch);
    }
    fclose (fp);
}
```

- Remove() → This function used to delete the file

```
# include <stdio.h>
void main()
{ if ( remove ("a.txt") == 0) // if delete then return 0 :-
    {
      printf ("file deleted");
    }
  else {
      printf ("file not deleted");
  }
}
```

- fread() and fwrite() function → To read and write the structure data type in binary form

```
# include <stdio.h>
# include <string.h>
struct student
{
   char name[20];
   int age;
} s1;
```

```
void main()
{
    FILE *fp;
    fp = fopen("A.txt", "w");
    printf("Enter student details");
    scanf("%s%d", s1.name, &s1.age);
    fwrite(&s1, sizeof(struct student), 1, fp);
                                  ↓
                          no. of record
                          in structure

    FILE *read;
    struct student inp;
    read = fopen("A.txt", "r");
    if (read == NULL)
    {
        printf("File not found");
    }
    else {
        while (fread(&inp, sizeof(struct
                                  student), 1, fr
                                              ↓
                                           point
        {
            printf("name=%s
                and age=%d", inp.name
                                  inp.age);
    }
}
```

MACRO

macro is used to define constant (global) variable and some functions.

There are two types of macro →

- macro as argument
- macro as function

* macro argument →

```
# include <stdio.h>
# define PI 3.14
```

→ global variable ∴ macro creates global var

PI → macro template

3.14 → macro expansion

```
void main ()
{
    float a, r;
    Printf ("Enter value of r");
    Scanf ("%f", &r);
    a = PI * r * r;
    Printf ("area of O = %f", a);
}
```

★ macro as function →

```
# include <stdio.h>
# define area (s) (s×s)
```

func. name ↙ | var given to function ↓ | value given by function ↘

```
void main()
{ int a, s;
  Printf ("Enter the side");
  Scanf ("%d", &s);
  a = area (s);
  Printf ("area of square = %d", a);
}
```

'macro creates one line function'

WAP to create macro function for area of rectangle

```
# include <stdio.h>
# define area (l, b) (l * b)
void main()
```

```
{ int a, l, b;
  Printf ("Enter value of l and b");
  Scanf ("%d%d", &l, &b);
   a = area (l, b);
  Printf ("Area of rectangle = %d", a);
}
```

Conditional - Compilation

It is used to compile particular parts of the code.

```
# include <stdio.h>
# define a
void main ()
{
   # if def a
       printf ("hello");
   # else
       printf ("bye");
   # endif
}
```

→ only this part will compile so time of run will be very low.

output → hello

∵ we define a

MATH FUNCTION LIBRARY

```
# include <stdio.h>
# include <maths.h>
void main ()
{
    int a., p; b;
    Printf ("Enter the number");
    scanf ("%d", &a);
    Printf (" Enter Power");
    scanf ("%d", &p);
    b = pow (a, p);    // Power function
    Printf (" value of p = %d \n", p);
}
```

Run by these commands →

- compile comand → gcc filename.c -o a -lm
- Run command → a

  (not a.out)

different functions with syntax →

① p = sqrt (a); → calculates square root $\sqrt{a}$

② p = pow (a, b) → calculates a^b or power func.

③ p = abs (a) → Absolute which converts -ve value to +ve value |a|

④ p = ceil (a) → Round of the value nearest greater value

eg → 3.4 ≈ 4

⑤ p = floor (a) → Round of value nearest lower value

eg → 3.4 ≈ 3

⑥ p = log10 (a) → log of a to the base 10

$\log_{10} a$

⑦ p = fmod (a, b) → stores the remainder

a % b

ENUM

- The Enum in C is also known as Enumated type.
- Enum is a data type.
- Its a user defined type that consists of int values and it provids meaningful name to these values.

```
# include <stdio.h>
enum week { sun = 1, mon, tues,
            wed, thur, fri, sat };
```

automatically give value (1, 2, 3, 4, 5, 6, 7)

```
void main ()
{
    enum week w;        // enum variable
    w = sat;
    printf ("%d", w);
}
```

output → 7

eg-2 → `enum name { ai = 6, bt = 7, cd = 10, dn };`

DYNAMIC MEMORY ALLOCATION

Dynamic memory Allocations is used to allocate the memory at run time. It uses some functions →

- malloc ()
- calloc ()
- realloc ()

and fourth function to free the memory →

- free ()

In dynamic memory allocation to allocate the memory we don't require any variable. we require a pointer.

- Malloc ()

This function allocates the single block of memory. Malloc function always return void pointer value we have to typecast it in desire data type of pointer variable

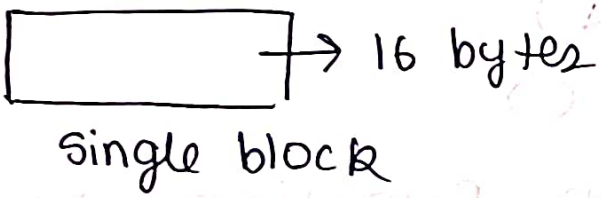
Syntax →

```
int *ptr ;
ptr = (int *) malloc (4 * sizeof (int)) ;
```

(4 byte)

(int *) → type casting

4 → 4 int store



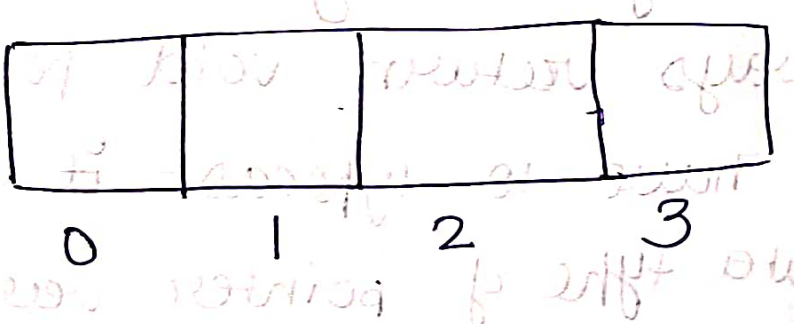
- calloc ( )

This is same as malloc but it gives array rather than single block.

syntax →

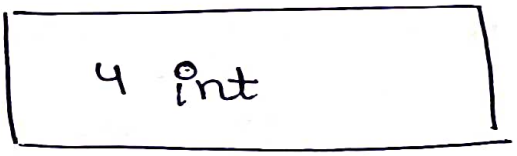
```
int *ptr ;
ptr = (int *) calloc (4 , sizeof (int)) ;
```

, → Diff in syntax



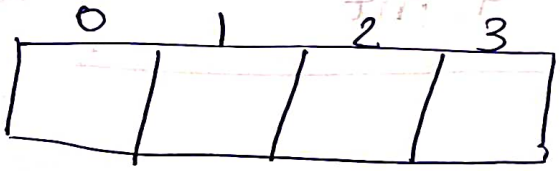
Example –

```
#include <stdio.h>
void main ()
{
    int n, *ptr, sum=0, i;
    printf ("Enter size of array");
    scanf ("%d", &n);
    ptr = (int*) malloc (n * sizeof(int));
    printf ("array elements \n");
    for (i=1 ; i<=n; i++)
        {
            scanf("%d", ptr);
            sum = sum + (*ptr);
            ptr++;
        }
    printf ("Sum of all elements = %d \n",
                                    sum);
    free (ptr);        // free the allocated
                          memory
}
```



16 byte

```
# include <stdio.h>
void main()
{
    int n, *ptr, sum=0, i;
    printf("Enter size");
    scanf("%d", &n);
  ptr = (int*) calloc (n, sizeof(int));
  printf("array elements");
  for (i=0; i<=n; i++)
      {
          scanf("%d", ptr);
          sum = sum + (*ptr);
          ptr++;
      }
  printf("Sum of all elements=%d", sum);
    free(ptr);
}
```



4 elements array

STORAGE CLASS IN C

They are used to determine the life time visiblity memory location and initial value of var.

There are 4 types of storage classes -

- Automatic
- static (eg - Reccursions)
- Register
- extern (may be dec. any where in prog)

storage class	storage Place	default value	scope	Lifetime
Auto	RAM	garbage or 0	Local	within function
Extern	RAM	0	global	till end of main prog.
Static	RAM	0	local	till the end of main prog & retain its value in multi fcn call
Register	CPU Registors	garbage	local	within function