freelance_Project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48		React+Springboot+MySql
49		React+Springboot+MySql
50		React+Springboot+MySql

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/cceesept2023

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project

```c
#include<stdio.h>
//--> Single line comment
// -->means Comment --> Documentation ,Explaination of program
/* --> Multline Comment
      Hello Om25 batch
      We have started with C programming
*/
// main is a function
// function --> ( )
int main()
{
   printf("Hello world\n");
   return 0;
}
```

```c
#include<stdio.h>
//stdio --> standard input output
// Header file Inclusion
// It contains standard function declarations
// This is a single line comment
/*
   This is multiline comment
*/
// main is a entry point function
// There is only one main( ) function
// main is a user-defined function
// we will learn function in detailed topic
// Execution starts from main ()
// program must have main ( ) function atleast

int main()
{
      //printf( ) is used to print data/string on terminal

      printf("Welcome OM25 batch");
      // printf is a library function
      return 0;
      // return is a jump statement
   // return 0 indicates successful execution of program
   // 0 indicates success
   // non-zero indicates failure  }
```

```c
#include<stdio.h>
// void --> nothing
// user defined function
// non-standard declaration
void main(  )
{
   // function not returning anything
   // 0 value is returned automatically
   printf("Hello world");
}
```

```c
#include<stdio.h>
// user-defined function
// Entry-point function
// main --> identifier
int main()
{
   printf("Hello world");
   //"Hello world" --> string
   // (  --> punctuator  // )
   // printf --> identifier
   // ;
   return 0;
   // return --> keyword
   // 0  --> constant
}
```

Tokens

- Smallest Individual unit of the program is called as token
- C program is made up of functions.
- Function is made up of statements.
- Statement contain multiple tokens.

1. Keywords 2. Data Types 3. Identifiers 4. Variables 5.Constants 6. Operators

Keywords

- Keywords are predefined words used in program, which have special meanings to the compiler.
- They are reserved words, so cannot be used as identifier.
- K & R C has 27 keywords. C89 added 5 keywords. C99 added 5 new keywords.

Identifiers

- Identifiers give names to variables, functions, defined types and pre-processor macros.

- Rules of Identifiers:

• Should start with alphabet or with _ (underscore)
• The first character of an identifier cannot be digit it should be letter (either uppercase or lowercase)
• Can include alphabets, _ (underscore),digits
• Case sensitive • Examples: • Var_1 //Valid
• 1_var // Not Valid
• _var //valid
• Var-1 // invalid
• Basic Salary //invalid

Data Types Range

• char • signed char (-128 to 127)
• unsigned char (0 to 255)
• int / long (32-bit)
• signed int (-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647)
• unsigned int (0 to 65,535 or 0 to 4,294,967,295)
• short int • signed short (-32,768 to 32,767)
• unsigned short(0 to 65,535)
• long long / long (64-bit) • signed long (-9223372036854775808 to 9223372036854775807)
• unsinged long(0 to 18446744073709551615)
• float: ± 3.4E +/- 38
• double: ± 1.7E +/- 308

```c
#include<stdio.h>
int main()
{
   // printf prints data/string on terminal
   // string --> "Hello world"
   // data --> integer,char,double
   // i want to print 10 on terminal
   // 10 --> data --> int
   /*
         format specifier
         10 --> integer format
         %d --> decimal integer
   */


    printf("%d ",10);
   // What is i want to print 'A' on terminal
   // 'A' --> character
   // character --> %c
```

```
      printf("%c ",'A');

   // What if i want to print 10.33 on terminal
   // 10.33 --> %lf
   // double --> 10.33 --> %lf
      printf("%lf ",10.33);

   // i have printed 10,'A',10.33 on terminal
   // i used format specifier
   // 10 --> int --> %d
   // 'A' -->char -->%c
   // 10.33 --> double --> %lf

   return 0;
}
```

```
#include<stdio.h>
int main()
 {  // i want to print 10
   // 10 --> int
   // int --> %d
   //printf("%d",10);
   // 10
   //printf("num = %d",10);
   //      num = 10
   // I want to print char = A
   //printf("char = %c",'A');
   //      char = A
    // I want to print d1 = 10.33
   //printf("double = %lf",10.33);
       //      double = 10.33
  //printf("%d %c %lf",10,'A',10.33);
     printf("int=%d char=%c double=%lf",10,'A',10.33);
     //      int=10 char=A  double=10.33

   return 0;
}
```

```
#include<stdio.h>
int main()
{
   //Data is hold in memory locations
```

```
   //identified by names called as variables.
   // 10 --> %d
   // what if i want to store data 10 in memory
   // i need to declare the variable

      int num;
      // variable declaration
      // variable --> whose value can change
      /*
            variable declaration
            datatype variable-name
             int     num
      */
      num=10;
      /*
            Assignment
            variable-name = value
               num      = 10
      */
      //assignment
   // what is i want to print 10
   // 10 is stored in variable num
   // 10 is identified by the name num
   // can i use num to print 10
      printf("num = %d",num); //10
      //     num = 10
   return 0;
}
```

```
#include<stdio.h>
int main()
{//int num;  // variable declaration
   int num = 10; // variable initialization
   //int num = 100; // ERROR  // Init can be done only once
   /* variable initialization ==>
    */ variable-declaration = value
         printf("num = %d",num); // 10
         return 0;
          }
```

```
#include<stdio.h>
int main()
{
   //int num; // variable declaration
```

```
    int num = 10; // variable declaration + Initialization
    // variable init can be done only once
    printf("prev num = %d\n",num); // 10
    num = 100;  // assignment
    printf("after updated num = %d\n",num); // 100
    num = 1000; // assignment
    printf("after second updated num = %d\n",num); // 1000
    // assignment can be done multiple number of times
    return 0;
}
```

Data type Size concept

```
#include<stdio.h>
int main()
{
    int num = 100 ;
    // 4 bytes space is getting reserved in memory
    // inside that 4 bytes 100 is getting stored
    // binary of 100 is getting stored in 4 bytes --> 32 bits

     char ch = 'A';
    //1 bytes space is getting reserved in memory
    double d1 = 10.33;
    //8 bytes space is getting reserved in memory

    // sizeof( ) --> sizeof operator returns size in bytes
    // sizeof( ) --> operator

    printf("%d\n",sizeof(num)); // 4 bytes
    // 4 bytes --> 32 bits
    printf("%d\n",sizeof(ch)); // 1 bytes
    // 1 bytes --> 8 bits
    printf("%d\n",sizeof(d1)); // 8 bytes
    // 8 bytes --> 64 bits

    return 0;
}
```

FORMAT SPECIFIERS

- char - %c
- int - %d,%i
- float - %f
- double - %lf
- long int - %ld
- short int - %hd
- unsigned long - %hu
- unsigned short - %hu
- string type - %s
- Pointer type - %p

Width Specifier concept

```
#include<stdio.h>
int main()
{
   //Width specifier
   int num = 10;

   //printf("%d",num); //10
   //printf("%6d",num);  //Right justified
   // - - - - - -
   //       1 0

   //printf("%-6d",num); // left justified
   //- - - - - -
   //1 0

   //printf("%06d",num); // left justified
   // - - - - - -
   // 0 0 0 0 1 0

   float fvar =12.50;
   //printf("%f",fvar);// 12.500000
   //printf("%6.2f",fvar);
   // - - - - - -
   //   1 2 . 5 0

   //printf("%-6.2f",fvar);
   // - - - - - -
   // 1 2 . 5 0

   return 0;
}
```

Escape sequence concept

```
#include<stdio.h>
int main()
{
// Escape sequence
//printf("Hello world");
//printf("\"Hello world\"");
//printf("\'Hello world\'");
```

```
// \n --> new line
//printf("Hello world\n");
//printf("OM25 batch ");
//Hello world
//OM25 batch

//printf("Hello world\nOM25 batch");
//printf("Hello\nworld\nOM25\nbatch");

//printf("OM25 batch,");
//printf("OM25 batch,\b.");
// printf("OK25 batch\b\b\b\b\b\b\b\b\bM");
//printf("OK35 batch\b\b\b\b\b\b\b\b\bM2");

//printf("12345678\n");
//printf("\tSunbeam");
// \t --> tab

//printf("\tSunbeam\tcom\tInfo\n");
//            S u n b e a m   c o m
//- - - - - - - - - - - - - - - - - - - - - - - -

//printf("KM25 batch\rO");
//printf("\\n is used to print on new line");

// I want to print
// Discount is 10%
printf("Discount is 10%%");
 return 0;
}
-------------------------------------------------------------------------
#include<stdio.h>
int main()
{
char ch ='A'; // 65
// char ch = 65;
// 65 --> Ascii value
// character are internally integral const
printf("%c\n",ch); // A
// character representation of ch                    //A -Z --> 65 to 90
// decimal representation of ch                      //a-z --> 97 to 122
printf("%d\n",ch); // 65        return 0;
}-------------------------------------------------------------------------------------------------------
```

```c
#include<stdio.h>
int main()
{
// what if i want to take i/p from user
// scanf() --> library function

// int num; // var decl
// printf("Enter the number"); //10
// scanf("%d",&num);   //10
// // & --> address of operator

// printf("%d",num); // to print data
//  int num1,num2;
//  printf("Enter the number1");
//  scanf("%d",&num1); // 10

//  printf("Enter the number2");
//  scanf("%d",&num2); // 100

//  printf("%d %d",num1,num2); // 10 100
return 0;
}
```

SIZE OF OPERATOR

```c
#include<stdio.h>
int main()
{
   int num;
   char ch;
   float fvar;
   short int num1;
   long int num2;
   long long int num3;
   double d1;

   // printf("int = %d\n",sizeof(num)); // 4
   // printf("char = %d\n",sizeof(ch)); // 1
   // printf("float = %d\n",sizeof(fvar)); // 4
   // printf("double = %d\n",sizeof(d1)); // 8
   // printf("short int = %d\n",sizeof(num1)); //2
   // printf("long int = %d\n",sizeof(num2));
   // printf("long long int = %d\n",sizeof(num3)); // 4
```

```
   // printf("%d\n",sizeof(65));  //4
   // printf("%d\n",sizeof('A')); //4
   // // A --> 65 --> int --> 4 byte
   // printf("%d",sizeof(12.33)); // 8
   // // 12.33 --> double --> 8 bytes
   // printf("%d",sizeof(12.33f)); //4
   // // 12.33 --> double
   // // 12.33f --> float

    printf("%d\n",sizeof(12L)); // 4
    printf("%d\n",sizeof(12LL));// 8
    printf("%d\n",sizeof(12l)); // 4
    printf("%d\n",sizeof(12u)); // 4
    printf("%d\n",sizeof(12U)); // 4
}
```

Escape sequences

- \' - Single quotation mark — prints '
- \" - Prints Double quotation mark — prints"
- \\ - Backslash Character — Prints\
- \a - Alert — Alerts by Generating beep
- \b – Backspace — Movers cursor one postion to the left of its current position
- \f - Form Feed — Moves cursor to the beginning of next page
- \n - New line — Moves cursor to the beginning of next line
- \r - Carraige return — Moves the cursor to the beginning of current line
- \t - Horizontal tab — Moves the cursor to next horizontal tab stop
- \v - Vertical tab — Vertical tab

Operators

- Classification of operators
- Unary Operators – Unary Operator operates on only one operand for example in the expression -3 – is a unary minus operator examples of unary operator are &,sizeof,!(logical negation),~(bitwise negation),++(increment),--(decrement) operator
- Binary Operators – Binary operator operates on 2 operands for example expression 2-3, - acts as a binary minus operator as it operates on 2 operands 2 and 3 for exampe *,/,<>(Right shift),Logical And(&), Bitwise And(&)
- Ternary Operator – A ternary operator operates on three operands for example Condtional operator (?:) is the only ternary operator in C

- Classification Based on operator Based on there role operators are classified as
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

```c
#include<stdio.h>
int main()
{
   int num;
   char ch;
   double d1;
   short int s1;

  // sizeof is evaluated at compile time

   printf("%d\n",sizeof(num+ch+d1));//8
   // 8 bytes
   printf("%d\n",sizeof(num+ch));//4

   printf("%d\n",sizeof(num+d1));//8

   printf("%d\n",sizeof(10+2)); //4

   printf("%d\n",sizeof(12.33+'A'));//8

   printf("%d\n",sizeof(12.33f+'A'));//4

   printf("%d\n",sizeof(s1+ch));  //4
         // short + char --> int
   return 0;
}
```

Reminder (%) And Quotient (/)

```c
#include<stdio.h>
int main()
{
   int num = 1234;
   int ans;
   ans = num % 10; // Remainder
   printf("%d\n",ans);
   ans = num / 10; // Quotient
   printf("%d",ans);
   return 0;
}
```

```
1. #include <stdio.h>
int main( void )
{
printf("\n ans1=%d ans2=%x ans3=%o", 064,064,064);
return 0;
}
```

A. ans1=52 ans2=34 ans3=64
B. ans1=52 ans2=52 ans3=64
C. ans1=52 ans2=64 ans3=34
D. ans1=34 ans2=52 ans3=64

solution:
=> 064
= (64)8=(?)10
= 6x8^1 +4x8^0
= 48 + 4
=52

=> 064
= (52)10=(?)16
= 16|52
= | 3 4
= | 3
= 34

=> 064=064
= 64

Ans. 52,34,64

```
2. #include<stdio.h>
int main( void )
{
char num1='A';
short int ch=101;
double num2=10.24;
int value =sizeof(num1)+sizeof(ch)+sizeof(num2);
printf("value=%-8d", value);
int Value =sizeof(1)+sizeof('A')+sizeof(1.2);
Value+= sizeof(10.2F);
printf("Value=%-8d", Value);
printf("Result=%-8.2f",value + Value /8.0f);
```

```
return 0;
}
```
A. value=11 Value=20 Result=13.50
B. value=13 Value=20 Result=15.50
C. value=13 Value=17 Result=13.50
D. value=11 Value=17 Result=15.50

solution:

```
#include<stdio.h>
int main( void )
{
char num1='A';
short int ch=101;
double num2=10.24;

int value =sizeof(num1)+sizeof(ch)+sizeof(num2);
        //       1  +  2 + 8 --> 11

printf("value=%-8d", value); // 11

int Value =sizeof(1)+sizeof('A')+sizeof(1.2);
        //  4 + 4 + 8 // 16

        //a+=b
        // a = a + b

Value+= sizeof(10.2F);
// Value = Value + sizeof(10.2f)
//          16   + 4 // 20
printf("Value=%-8d", Value);
//                       20
printf("Result=%-8.2f",value + Value /8.0f);
//                       11   + 20/8.0 --> 13.5
return 0;
}
```

Ans. 11,20,13.5

3.
```
#include<stdio.h>
int main(void)
```

```
{
int value1=0x32;
int value2=064;
int value3 = 0x8 + 016 + 128 - value1 + value2;
printf("value1=%d \t", value1 ); // decimal rep of 0x32(hex)
printf("value2=%d \t", value2 ); // decimal rep if 064(oct)
printf("value3=%d \n", value3 ); //
return 0;
}
```

A. value1=50 value2=52 value3=152
B. value1=32 value2=64 value3=164
C. value1=50 value2=64 value3=152
D. value1=50 value2=64 value3=164
Answer:
Solution:
=> value1=0x32
= (32)16=(?)10
= 3x16^1 + 2x16^0
=48+2
=50

=> value2=064
= (64)8=(?)10
= 6x8^1 +4x8^0
= 48+4
= 52

=>0x8
= (8)16=(?)10
= 8x16^0
=8

=>016
=(16)8=(?)10
=1x8^1+6x8^0
=8+6
=14

value3=> 8 + 14 +128 - 50 + 52
= 152
Ans.A--> 50,52,152

Polling Questions

1.#include <stdio.h>
int main()
{
   int data = 65;
   printf("%c\n", data);
   return 0;
}
A) A
B) 65
C) Compile Time Error
Answer : A

2.The format specifier should be start by ____ symbol.
(A) +
(B) /
(C) %
(D) -
Answer :C

3.#include <stdio.h>
int main(void)
{
  printf("%d", sizeof('a'));
  return 0;
}
a) 4
B) 1
C) Error
D) 97
Answer : A

4.#include <stdio.h>
int main(void)
{
  printf("%d", sizeof(98.3));
  return 0;
}
A) 4
B) 8
C) 2

D) 1
Answer : B

```
#include<stdio.h>
int main()
{
   //short hand operators OR short hand Assignment operators  = right to left associvity
   // num = num + 2
   int num =4 ;
   //num+=2;  // num = num + 2
   //num-=2;  // num = num - 2  // 4 - 2 --> 2
   //num+=5; // num = num + 5 --> 4 + 5 --> 9
   // num%=2; // num = num % 2 --> 4 % 2 --> 0

   num+=8;   //num=num+8 = 4+8=12
   num-=16;   //num=num-16 =12-16=-4
   num+=32;   //num=num+32 =-4+32= 28
   num%=5;   //num=num%7 = 28%5=3
   printf("%d",num);

   /*
         num = num + 5 // same
         num+=5 // same
   */

   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int a = 1;
   int b = 2;
   int c = 3;
   a+=b+=c+=1;  //associvity right to left
   // // c = c + 1 // 3 + 1 --> 4
   // // b = b + c // 2 + 4 --> 6
   // // a = a + b // 1 + 6 --> 7

   printf("a = %d b = %d c = %d\n",a,b,c);
   int d,e,f;
```

```
   d=e=f=-3;
   // = --> assignment operator  right to left associvity

   printf("d = %d e = %d f = %d",d,e,f);

   return 0;
}
```

```
#include<stdio.h>
int main()
{
   // , --> comma operator
   //int a = 1,2,3;// NOT OK // compiler error

    int a; // variable declaration

    //a = 1;  //assignment
     //a = 1,2,3;   // a =1  left most value hi copy hoti hai.
     //a = (1,2,3);   // a =3  bracket case mai right most value hi copy hoti hai.
     a = ((1,2),3);   // a =3
     //  (( 1,2),3 )
     //     ( 2 , 3 )
     //       3
    printf("%d",a);

   return 0;
}
```

```
#include<stdio.h>
int main()
{
   // a = a + 1// valid
   // a+=1 // a = a + 1 // valid
   // ++ and -- // unary operator // only one operand

   // int a = 2;
   // int ans;
   // ans = ++a;
   // ++ --> increment operator ( pre increment )
   // pre-increment operator
   // ++a ==> a = a + 1
   //  ++a --> 1st increment and copied into answer
```

```
   // a = a + 1 --> 2 + 1 --> a =3
   //  ans = 3
   //  a = 3
   // printf("a=%d ans=%d",a,ans);

   // int a = 2;
   // int ans;
   // ans = a++;
   // ++ --> increment operator ( post increment )
   // post-increment operator
   // a++ ==> a = a + 1   1st copied into answer then increment
   // printf("a=%d ans=%d",a,ans);

   // int a = 2;
   // int ans;
   // ans = --a;
   // -- --> Decrement operator ( pre decrement )
   //  pre-decrement operator
   //  --a ==> a = a - 1   1st decrement and copied into answer
   // printf("a=%d ans=%d",a,ans);

   // int a = 2;
   // int ans;
   // ans = a--;
   // -- --> posDecrement operator ( pre decrement )
   // post-decrement operator
   // a-- ==> a = a - 1   1st copied into answer and then decrement.
   // printf("a=%d ans=%d",a,ans);

   // return 0;

}
```

```
#include<stdio.h>
int main()
{
   int ans;
   int a = 1;

   ans = a++,a++,a++;  //bcoz of , operator left most value copied post increment so first copiedd then increment ans=1 a=4
   printf("ans=%d a=%d",ans,a);
```

```
   ans = ++a,++a,++a;
   printf("ans=%d a=%d",ans,a); //bcoz of , operator left most value copied pre increment
so
// first increment then copied ans=5 a=7
}
```

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

   a = ( 1,2,3);
   // a = 3

   b = ( ++a , ++a , ++a);
   // b = 6
   // a = 6

   // b = 6
   c = ( b++ , b++ , b++);
   //c=  6   , 7   , 8
   // b=9
   printf("a=%d b=%d c=%d",a,b,c);
                                 // 6   9   8
}
```

---

relational operator

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

!= --> not equal to

== --> relational operator

= --> assignment operator

---

relational operator --> output --> 0 or 1

0 --> False

1 --> true

any nonzero is considered as true (-ve)

0 --> false

```
printf("%d\n",10>20); // false --> 0
10 > 20 --> false --> 0

printf("%d\n",30>20);// True --> 1
30 > 20 --> true --> 1

printf("%d\n",30>=20);// True --> 1
>= --> greater than or equal to

printf("%d\n",9==9);// True --> 1
9==9 --> true --> 1

printf("%d\n",9==-9);// false
9==-9 --> false --> 0
```

```
! --> logical negation
non-zero ( negative )  ---> true
zero --> false

printf("%d",!100);
100 --> non zero --> true
!true --> false
false -- > 0

printf("%d",!-100);
-100 --> non zero --> true
!true --> 0

printf("%d\n",10!=10); // 0
!= --> not equal to
!true=flase -->0

printf("%d\n",10!=11); // true--> 1
!flase= true--> 1

printf("%d\n",3>=3); // 1

printf("%d",!0);// 1
0--> zero--> false
!false-->true-->1
```

```
    printf("%d",!!-22);
    ! !T -->  ! F --> T --> 1
```

```
#include<stdio.h>
int main()
{
 */   Logical operators
      && --> logical AND
      || --> Logical OR
      !  --> logical negation

      logical operator --> 1 or 0
      T and T
      Nonzero ( -ve ) --> TRUE
      0 --> False
/*

Example:1
    int ans;
    int a = 1;
    int b = 2;
    ans = a && b; // LOGICAL AND
         1 && 2
       TRUE && TRUE
          TRUE
     ans = TRUE
     ans = 1
     printf("ans=%d a=%d b=%d",ans,a,b);
                                ans=1 a=1 b=2

Example:2
   int ans;
   int a = 1;
   int b = 0;
   ans = a && b; // LOGICAL AND
   //    T && F
   //      F --> 0
   printf("ans=%d a=%d b=%d",ans,a,b);
                               ans=0 , a=1 , b=0
```

```
Example:3
   int ans;
   int a = 1;
   int b = 22;
   ans = a || b; // LOGICAL OR
   //   T  || XXX
   // ans = true --> 1
   printf("ans=%d a=%d b=%d",ans,a,b);
                                //      1      1     22

Example:4
   int ans;
   int a = 0;
   int b = 22;
   ans = a || b; // LOGICAL OR
   //    F || T
   //      T --> 1
   printf("ans=%d a=%d b=%d",ans,a,b);
                //                          1    0    22

Example:5
  int ans;
   int a = 0;
   int b = 0;
   ans = a || b; // LOGICAL OR
   //    F || F
   //      F --> 0
   printf("ans=%d a=%d b=%d",ans,a,b);
                           //               0    0    0
   return 0;
}
-----------------------------------------------------------------------------------------------------------
#include<stdio.h>
int main()
{
Example:1
   int a = 1;
   int b = 2;
   int ans;
   ans = a++ || b++;
   //    T++ || XXX
   //    T --> 1
```

```
   printf("ans=%d a=%d b=%d",ans,a,b);
      // ans=1 a=2 b=2
```

Example:2

```
   int a = 0;
   int b = 0;
   int ans;
   ans = a++ || ++b;
   //    F++ || T
   //      T
   printf("ans=%d a=%d b=%d",ans,a,b);
                                  //  ans=1 a=1 b=1
```

Example:3

```
   int a = 0;
   int b = 0;
   int ans;
   ans = a++ || b++;
        F   || F
         F
    printf("ans=%d a=%d b=%d",ans,a,b);
                          //ans=0 a=1 b=1
```

Example:4

```
   int a = 1;
   int b = 2;
   int ans;
   ans = a++ || b;
        T   || xxxx
            T
   printf("ans=%d a=%d b=%d",ans,a,b);
   //       ans=1 a=2 b=2
   return 0;
}
```

```
#include<stdio.h>
int main()
{
Example:1
      int a = 1;
      int b = 2;
      int ans;
      ans = a++ && b;
      //     T && T
      //      T
      printf("ans=%d a=%d b=%d",ans,a,b);
                                 //    ans=1 a=2 b=2

Example:2
   int a = 0;
   int b = 2;
   int ans;
   ans = a++ && b;
      //   F++ && XXXX
      //  F++ --> post incre
   printf("ans=%d a=%d b=%d",ans,a,b);
                               //   ans=0 a=1 b=2

Example:3
   int a = 0;
   int b = 2;
   int ans;
   ans = a++ && ++b;
         F  &&  xxx
          F
   //ans=0 a=1 b=2
   printf("ans=%d a=%d b=%d",ans,a,b);

Example:4
   int a = 0;
   int b = 2;
   int ans;
   ans = ++a && ++b;
   //   ++F && ++T
   //    T && T
   //     T
   // ans --> T --> 1
```

```
   //ans=1 a=1 b=3
   printf("ans=%d a=%d b=%d",ans,a,b);

   return 0;
}
```

```
#include<stdio.h>

int main()
{
Example:1
   int a = 1;
   int b = 2;
   int c = 3;
   int ans;
   ans = a || b && c;
   //    T || XXXX
   //     T --> 1
   // ans -> 1
   //    a || ( b && c )
   // || --> logical OR
   // && --> logical AND
   printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                        // ans=1 a=1 b=2 c=3

Example:2
   int a = 1;
   int b = 2;
   int c = 3;
   int ans;
   ans =  a || b++ && c++;
      // a || (b++ && c++);
      // T || XXXXX
      // T
      // ans --> T
      // ans --> 1
   printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                          //ans=1 a=1 b=2 c=3
```

Example:3

```
int a = 0;
int b = 2;
int c = 3;
int ans;
ans = a || b++ && c++;
//  a || (b++ && c++);
//  F || (T++ && T++ )
// F  || T
//  T
// ans --> T -->1
//ans=1 a=0 b=3 c=4
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
```

Example:4

```
int a = 0;
int b = 2;
int c = 3;
int ans;
ans = a++ || b++ && c++;
//   a++ || (b++ && c++);
//   F++ ||  T++ && T++
//   F++ ||  T
//     T
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                              // ans=1 a=1 b=3 c=4
```

Example:5

```
int a = 0;
int b = 0;
int c = 3;
int ans;
ans = a++ || b++ && c++;
//    a++ || (b++ && c++);
//    F++ || (F++ && XXXX);
//    F++ || F++
//      F
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                    // ans=0 a=1 b=1 c=3
```

Example:6

```
int a = 0;
int b = 0;
int c = 3;
int ans;
ans = a++ && b++ || c++;
//   (a++ && b++) || c++;
//    F++ && XXXX || T++
//     F++ || T++
//        T
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                    //ans=1 a=1 b=0 c=4
```

Example:7

```
int a = 0;
int b = 0;
int c = 3;
int ans;
ans = ++a && b++ || c++;
   //(++a && b++) || c++;
   // ++F && F++  || T++
   // (T && F) || T++
   //   F || T++
   //  T++ --> post
   // ans --> 1
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                    //ans=1 a=1 b=1 c=4
```

Example:8

```
int a =1;
int b = 2;
int c = 3;
int ans;
ans = a++ || b++ || c++;
//   T++ || XXXX || XXXX
//    T++ --> 1
// ans = 1
printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                    //ans=1 a=2 b=2 c=3
```

Example:9

```
   int a = 0;
   int b = 2;
   int c = 3;
   int ans;
   ans = a++ && b++ && c++;
   //  ( a++ && b++) && c++;
   //  ( F++ && XXXX)&& c++
   //   F++ && XXXX
   //  F--> ans --> 0
   printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                    //    ans=0 a=1 b=2 c=3
```

Example:10

```
   int a = 0;
   int b = 2;
   int c = 3;
   int ans;
   ans = a++ || b++ && ++c && a++ || b++;
   //  a++ || ((b++ && ++c) && a++) || b++;
   //  F++ || ((T++ && ++T) && a++) || b++
   //  F++ || ( T && T) || b++
   // F++ || T || b++
   //   T || XXXX
   printf("ans=%d a=%d b=%d c=%d",ans,a,b,c);
                                      //   ans=1 a=2 b=3 c=4
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int count;
   count = printf("Hello S");
   // 7 char --> including space
   printf("%d",count);
   return 0;
}
```

Twisster-

```
1. #include <stdio.h>
int main( void )
{
int num1,num2,num3, num4;
num1 = 100 ; num2 = 200 ;
num2= ++num1;
num3= num2--/2;
num4 = printf(" %-10d ", ++num1 ) + ++num2;
printf("%10d\t",num3);
printf("%-10d",num4);
return 0;
}
```

A. 101 51 113
B. 102 50 113
C. 102 50 112
D. 200 51 113
Answer: B
solution:-

```
int num1,num2,num3, num4;
num1 = 100 ; num2 = 200 ;
num2= ++num1;
// num2 = 101
// num1 = 101

num3= num2--/2;
num3= 101 / 2 --> 50 --> num3 = 50
num2--> num2-- => 100

// num2 = 100 num3=50 num1=101
// -%10d --> - - - - - - - - - - + 2 spaces --> 12

num4 = printf(" %-10d ", ++num1 ) + ++num2; // num1 --> 102
//num4 = 12 + ++num2-> 12 + 101 --> 113
//     (10+2spaces)
//++num1 --> 102
// ++num2 --> 101
printf("%10d\t",num3);// 50
printf("%-10d",num4); // 113
```

Twistter-

2. #include<stdio.h>
int main(void)
{
int num1 = 0, num2 = -1 , num3 = -2, num4 = 1, ans;
ans = num1++ || num2++ && ++num4 || num3++;
printf("%d %d %d %d %d",num1,num2,num3,num4,ans);
return 0;
}
A. 0 0 -2 2 0
B. 0 0 -2 2 1
C. 1 0 -1 2 0
D. 1 0 -2 2 1
Answer: D

solution:-

```
int num1 = 0, num2 = -1 , num3 = -2, num4 = 1, ans;
ans = num1++ || num2++ && ++num4 || num3++;

// num1++ || (num2++ && ++num4) || num3++;

//  num1++-> post incr --> num1 value 1

// num2++ --> post incr --> -1 is non zero so true --> -1 becomes 0 ( post)
// ++num4 --> pre incr --> 2 -->
// exp (num2++ && ++num4) becomes true

// num3 --> no change

printf("%d %d %d %d %d",num1,num2,num3,num4,ans);
//                      1   0   -2  2   1
```

Twister:-
3. #include<stdio.h>
int main(void)
{
int val=0;
int value = val++ || ++val==1 || val--;
printf(" val=%d value=%d", val, value);
return 0;
}

A. val=1 value=1
B. val=0 value=0
C. val=1 value=0
D. val=0 value=1
Answer: A

```
Solution:-      int val=0;
                int value = val++ || ++val==1 || val--;
                //          0(post) || 2 == 1 || val-- --> 1
                                    false    True

                printf(" val=%d value=%d", val, value);
                //                  1    1
```

Twisster:-

```
4. #include<stdio.h>
int main( void )
{
int x1=1, x2=2, x3=3;
int val=!(!((x1+x2)<(x3+1)));
printf(" val=%d ", !val);
return 0;
}
```

A. val=0
B. val=1
C. val=-1
D. garbage value
Answer: A

```
solution:-      int x1=1, x2=2, x3=3;
                  int val=!(!((x1+x2)<(x3+1)));

                  //  (1 + 2) < ( 3 + 1 )
                  //  ( 3 ) <  ( 4 ) --> T --> 1
                  //    !T --> F
                  //    !F --> T

                  printf(" val=%d ", !val);
                                  // !T --> F -->0
```

Day6 poll questions

Example:1

```
int main()
{
   int i = -5;
   int k = i % 4;
   printf("%d\n", k);
}
```

A. Compile time error
B. -1
C. 1
D. None
Answer : B

Example:2

```
int main()
{
   int i = 7;
   i = i / 4;
   printf("%d\n", i);
   return 0;
}
```

A. Run time error
B. 1.6
C. 1
D. Compile time error
Answer : C

Example:3
which is correct with respect to data type?
A. char > int > float
B. int > char > float
C. char < int < double
D. double > char > int
Answer : C

Example:4
what will be the output?

```
int main(void)
{
        signed char chr;
        chr=128;
        printf("%d",chr);
        return 0;
}
```

A.0
B.Depends on compiler
C.-128
D.128
Answer :C

Bitwise AND Bitwise OR Bitwise XOR

```
printf("%d\n",10&5);  // Bitwise AND
 // 10--> 0000 1010
 // 5 --> 0000 0101
 //     -------------
 //      0000 0000
 // 0--> so output-->0

printf("%d\n",10|5);// Bitwise OR
// 10--> 0000 1010
 // 5 --> 0000 0101
 //     -------------
 //      0000 1111  ---> 15
 // so output is 15

printf("%d\n",10^5);// Bitwise XOR
 // 10--> 0000 1010
 // 5 --> 0000 0101
 //     -------------
 //      0000 1111 ---> 15
 //   so output is 15
```

i/p		O/p
A	B	
0	0	0
0	1	0
1	0	0
1	1	1

i/p		O/p
A	B	
0	0	0
0	1	1
1	0	1
1	1	1

i/p		O/p
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

#Bitwise NOT ~

```
int num = 10;
printf("%d",~num);  // -11
// ~ --> -(n+1) -->shortcut formula--> -(10 + 1) --> -11
```

#Left shift operator <<

```
int ans;
   // Left shift operator << ( bitwise )
   // ans = 10 << 2;
   /*
         0000  1010  --> Binary of 10
           << 2
         0010  1000  -->output 40
   */
   printf("%d",ans); //40
  /*      shortcut formula for Left shift
         multiply by 2 to the power n

         10 * 2 to the power 2
         10 * 4
         40
  */
```

#Right shift operater >>

```
// Right shift operater >> (Bitwise)
      //ans = 10 >>2 ;
      /*
         Bitwise ( binary of 10 )

         0000 1010

           >>2
         0000 0010
      */
    // printf("%d",ans); //2

      /*     shortcut formula for right shift operator
            divide by 2 to the power n

            10 / 2 to the power n
            10 / 2 to the power 2
            10 / 4
            2
      */
```

Example-1

```
// a = 5(00000101), b = 9(00001001)
   unsigned char a = 5, b = 9;

   // The result is 00001010
   printf("a<<1 = %d\n", a<<1);
```

By formula-a= 5*2^1 = 10

```
   // The result is 00010010
   printf("b<<1 = %d\n", b<<1);
```

By formula-b =9*2^1

Example-2

```
// a = 5(00000101), b = 9(00001001)
   unsigned char a = 5, b = 9;

   // The result is 00000010

   printf("a>>1 = %d\n", a >> 1);
```

By formula- a= 5/2^1= 2

```
   // The result is 00000100
   printf("b>>1 = %d\n", b >> 1);
```

By formula b=9/2^1= 4

Example-3

```
int x = 19;
   unsigned long long y = 19;
   printf("x << 1 = %d\n", x << 1);
```

By formula- x=19*2^1=38

```
   printf("x >> 1 = %d\n", x >> 1);
```

By formula x=19/2^1=9

Example-4

```
int i = 3;
  printf("pow(2, %d) = %d\n", i, 1 << i);
```

By formula- 1*2^3=8

```
  i = 4;
  printf("pow(2, %d) = %d\n", i, 1 << i);
```

By formula- 1*2^4=16

Example-5

```
unsigned int a = 60;        /* 60 = 0011 1100 */
  unsigned int b = 13;      /* 13 = 0000 1101 */
  int c = 0;

  c = a & b;       /* 12 = 0000 1100 */
  printf(" c is %d\n", c );

  c = a | b;       /* 61 = 0011 1101 */
  printf(" c is %d\n", c );

  c = a ^ b;       /* 49 = 0011 0001 */
  printf(" c is %d\n", c );

  c = ~a;          /*-61 = 1100 0011 */ By formula -(n+1) = -(60+1) = -61
  printf(" c is %d\n", c );

  c = a << 2;     /* 240 = 1111 0000 */ By formula- 60*2^2= 60*4=240
  printf(" c is %d\n", c );

  c = a >> 2;     /* 15 = 0000 1111 */  By formula- 60/2^2= 60/4= 15
  printf(" c is %d\n", c );
```

Example-6

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a&b);
   return 0;
}
```

solution:-

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

```
 00001100
& 00011001
 ________
 00001000  = 8 (In decimal)
```

Example-7

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a|b);
   return 0;
}
```

solution:-

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
 00001100
| 00011001
 ________
 00011101  = 29 (In decimal)
```

Example-8

```
#include <stdio.h>
int main()
{
   int a = 12, b = 25;
   printf("Output = %d", a^b);
   return 0;
}
```

Solution:-

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

```
 00001100
^ 00011001
 ________
 00010101  = 21 (In decimal)
```

Example-9

```
#include <stdio.h>
int main()
{
   printf("Output = %d\n",~35);
   printf("Output = %d\n",~-12);
```

```
   return 0;
}
```

solution:-

~35 = -(n+1) = -(35+1) = -36

~-12= -(n+1) = -(-12+1) = -(-11) = 11

Example-10

```
#include <stdio.h>
int main()
{
   int num=212, i;
   for (i=0; i<=2; ++i)
      printf("Right shift by %d: %d\n", i, num>>i);

    printf("\n");

    for (i=0; i<=2; ++i)
      printf("Left shift by %d: %d\n", i, num<<i);

    return 0;
}
```

solution:-

By formula

For i=0 212/2^0 =212

For i=1 212/2^1 =106

For i=2 212/2^2= 212/4 = 53

For i=0 212*2^0= 212

For i=1 212*2^1= 424

For i=2 212*2^2= 212*4 = 848

Example-11

```
int main()
{
char a = 255;
char b = 127;
b = ~b;
a = a ^ b;
printf("\n%d,%d",a,b);
return 0;
}
```

Solution:-

Bitwise complement(~) of 127 is -128 (o/p of b is -128)
255 ^ -128
Binary equivalent of 255-> 1111 1111
Binary equivalent of 128-> 1000 0000
As the number 128 is negative we perform the 2's complement on 128 to gets its binary equivalent

0111 1111 --->(1's compliment of 128)
+ 1
1000 0000 -- > (2's compliment of 128 i.e -128 binary)
Know we perform ex-or operations of 255 and -128
1111 1111 ----> Binary Equivalent of 255
1000 0000 ----> Binary Equivalent of -128
----------> ^
// 0111 1111 ---> 127
so O/P is 127

TWISTERS

```
1. #include <stdio.h>
int main(void)
{
int a,b,c;
a=printf("\t \"SunBeam\" - \t'IT Park' \t");
b=printf("\t \'Pune\' \t");
c=printf("\n a = %d",a)+ ++b;
printf(" c = %d",c);
return 0;
}
```

A. "SunBeam" - 'IT Park' 'Pune'
a = 26 c = 19
B. "SunBeam" - 'IT Park' 'Pune'
a = 32 c = 14
C. "SunBeam" - 'IT Park' 'Pune'
a = 23 c = 19
D. Compile time error
Answer: A
solution:-

```
#include <stdio.h>
int main(void)
{
int a,b,c;
        a=printf("\t \"SunBeam\" - \t'IT Park' \t");// 26
```

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


b=printf("\t \'Pune\' \t"); // 10
//printf("%d",b);

c=printf("\n a = %d",a)+ ++b;   (6 char + %d ke 2 char that 26 =8 )

// 8 + 11
// 19
printf(" c = %d",c);
return 0;
}
```

TWISTERS

```
2. #include <stdio.h>
int main( void )
{
int num1 = 1, num2 , num3 = 5;
num2=--num1;
int ans1 = num1-- || num2++ || num3++;
int ans2 = ++num1 && num2++ && num3++;
printf("num2=%d num3=%d ", num2,num3);
printf("(ans1-ans2)=%d ", ans1-ans2);
return 0;
}
```

A. num2=1 num3=6 (ans1-ans2)=1
B. num2=0 num3=5 (ans1-ans2)=0
C. num2=1 num3=5 (ans1-ans2)=0
D. num2=1 num3=6 (ans1-ans2)=0
Answer: A

solution:-

```
#include <stdio.h>
int main( void )
{
int num1 = 1, num2 , num3 = 5;

num2=--num1;
//num2 = 0
//num1 = 0
```

```
int ans1 = num1-- || num2++ || num3++;
//0 ( post --> -1)   || 0 (Post->1) || 5 ( post --> 6)
//  0 || 0 || T
//    T
// ans = 1

int ans2 = ++num1 && num2++ && num3++;
//         0    && num2++ && num3++
//         0     && skip && skip
// ans2 = 0 ;
printf("num2=%d num3=%d ", num2,num3);
//       1      6

printf("(ans1-ans2)=%d ", ans1-ans2);
//                      1 - 0 = 1
return 0;
}
```

Twisster

```
3. #include<stdio.h>
int main(void)
{
int one, two = 1, three;
if(three = (two == 0)); one = 5; two = 3;
printf("%d %d %d\n", three, two, one);
return 0;
}
```

A. 1 1 5
B. 0 3 garbage value
C. 0 1 5
D. 0 3 5
Answer: D

solution:

```
if(three = (two == 0)); one = 5; two = 3;
if(three = (0)); one = 5; two = 3;
if(three = 0); one = 5; two = 3;

printf("%d %d %d\n", three, two, one);
                       0     3    5
```

Twisster

```
4. #include<stdio.h>
int main(void)
{
int a = 1, b = 1, c;
if( c = b < 0) a = 5;
else if( b == 0) a = 7;
else a = 9;
printf("%d %d\n", a, c);
return 0;
}
```

A. 9 0
B. 7 0
C. 5 0
D. Compiler error
E. None of the above
Answer:A

solution:-

```
if( c = b < 0)  // if (c=1<0) false
                // if (c=0)
a = 5;
else if( b == 0) // else if (1==0) false
                 //else if (0)
 a = 7;
else a = 9;      // a=9

printf("%d %d\n", a, c);
                    9  0
```

Poll questions
Q.1

```
int main()
{
   int x = 2, y = 5;
   (x & y) ? printf("True ") : printf("False "); // False
   (x && y) ? printf("True ") : printf("False "); // True
   return 0;
}
```

A) False True
B) True False

C) True True
D) False False
Answer: A

2 -- 0000 0010
5 -- 0000 0101

0000 0000 --> 0 -- False condition

2 && 5 ---> T && T ---> True condition

Q.2
```
#include <stdio.h>
void main()
{
	int x = 0;
	if (x == 0) // True
		printf("hi"); // executed -- hi
	else
		printf("how are u"); // skipped
		printf("hello"); // out-side if-else -- will be executed irrespective of condition is
true/false
}
```
A) hi
B) how are you
C) hello
D) hihello
Answer: D

Q.3 What will be the output? (entered the value 1)
```
void main()
{
	int ch;
	printf("enter a value between 1 to 2: ");
	scanf("%d", &ch); // ch = 1
	switch (ch) {
		case 1: printf("1");
		default: printf(" 2\n"); // since break is missing, next case will also be executed.
	}
}
```

A) 1
B) 2
C) 1 2
D) Run time error
Answer: C

Q.4
```
#include <stdio.h>
void main()
{
   if(!printf("")) // ! 0 -- True
            printf("Okkk"); // executed
   else
      printf("Hiii");
}
```
A) Okkk
B) Hiii
C) Error
D) None
Answer: A

// printf("") -- returns number of chars printed = 0.
// ! 0 --> 1 (True condition)

// if there is space in printf.
   printf(" ") --> return 1
   ! printf(" ") --> 0 -- False condition

Poll Questions
1.
```
#include <stdio.h>
void main()
{
   int x = 1, y = 0, z = 5;
   int a = x && y || z++;
   printf("%d", z);
}
```
A)6
B)5
C)0
D)None
Answer : A

a = 1 && y || z++
a = 1 && 0 || z++
a = 0 || z++
a = 0 || 5++
a = 1
z = 6

2.#include <stdio.h>
void main()
{
int x = 1, y = 0, z = 5;
int a = x && y && z++;
printf("%d", z);
}
A)6
B)5
C)0
D)None
Answer: B
E1 && E2 && E3
1 && 0
0 && E3
0

3.Which among the following is NOT a logical or relational operator?
A)&&
B)||
C)!
D)=
Answer : D

4.#include <stdio.h>
int main()
{
int k = 8;
int x = 0 == 1 && k++;
printf("%d%d\n", x, k);
return 0;
}
0 9
0 8
1 8

1 9
Answer : B
//int x = 0 == 1 && k++;
// x = flase && skip
// x= 0
printf("%d%d\n", x, k);
0 8

6.Which of the following is bit wise operator?
A. && operator
B. & operator
C. || operator
D. ! operator
Answer : &

7.What will be output?
```
int main() {
        unsigned char a = 5, b = 9;
        printf("%d, %d\n", a & b, a && b);
        return 0;
}
```
A. 1, 1
B. 0, 1
C. 5, 0
D. 9, 1
Answer: A

5 --> 0 0 0 0 0 1 0 1
9 --> 0 0 0 0 1 0 0 1
---
(1) --> 0 0 0 0 0 0 0 1

5 (T) && 9 (T) --> 1 (T)

8.What will be output?
```
int main()
{
        int a = 20;
      int b = 21;
        int c = 0;
        c = a ^ b;
        printf("C= %d\n", c );
```

```
        return 0;
}
```

A) C=1
B) C=20
C) C=21
D) C=0
Answer: A

20 ---> 0 0 0 1 0 1 0 0
21 ---> 0 0 0 1 0 1 0 1
---
        0 0 0 0 0 0 0 1

IF ELSE STATEMENTS

```
#include<stdio.h>
/*
        Corona Pandamic
      1. Variable declaration
      2. Input the patient count
            a. if patient count >=1000 ( Impose the lockdown )
*/
int main()
{
   //If statement
   int count;
   // 1. Input the count
   printf("Enter the patient count ");
   scanf("%d",&count);  // 2000

   //2. processing
   // >= --> relation operator --> 1(True) or 0(false)
   //     2000>=1000 --> 1 --> True
   //  count>=1000 --> expression / condition
   //  if(1) --> if(true)
      if(count>=1000)
      {
         printf("Impose the lockdown");
      }
      //If ke bad else likhana neceessary nahi hai par else ke pahle if hona hi chahiye
   return 0;
}
```

```c
#include<stdio.h>
/*
        Corona Pandamic
      1. Variable declaration
      2. Input the patient count
           a. if patient count >=1000 ( Impose the lockdown )
           b  else Release the lockdown
*/
int main()
{
   //If statement
   int count;
   // 1. Input the count
   printf("Enter the patient count ");
   scanf("%d",&count);  // 500

   //2. processing
   // >= --> relation operator --> 1(True) or 0(false)
   //     500>=1000 --> 0 --> False
   //  count>=1000 --> expression / condition
   //  if(0) --> if(False)
      if(count>=1000) // 500 >=1000--> F->0 if(0)
      {
         printf("Impose the lockdown");
      }
      else
      {
         printf("Release the lockdown ");
      }
//If ke bad else likhana neceessary nahi hai par else ke pahle if hona hi chahiye
   return 0;
}
```

```c
#include<stdio.h>
// I/P --> Ammount
int main()
{
   double ammount,discount;
   printf("Enter the ammount");
   scanf("%lf",&ammount);
   // 4000>=5000 --> false
   // False --> 0
```

```
   // if(0) -> zero(False )
   if(ammount>=5000) // 5000
   {
      discount = ammount * 0.10;
      ammount = ammount - discount;
   }

   printf("Final Ammount %lf",ammount);
   return 0;
}
```

Nested If Else statements

```
#include<stdio.h>
/*
   I need to find greatest of 3 numbers
   num1 num2 num3
*/
int main()
{
   int num1 = 100;
   int num2 = 200;
   int num3 = 30;
   // greatest of 3 numbers
   if(num1 > num2)
   {
      if(num1 > num3)
      {
         printf("%d is greatest",num1);
      }
      else
      {
         printf("%d is greatest",num3);
      }
   }
   else
   {
         if(num2>num3)
         {
            printf("%d is greatest",num2);
         }
         else
         {
```

```
            printf("%d is greatest",num3);
        }
    }
    return 0;
}
```

```
#include<stdio.h>
// I/P --> character ( %c )
// char --> upper case , lower case , digit , special char
// ASCII --> 65 - 90 --> upper case
// 97 - 122 --> lower case
// 48 - 57 --> digit
// special char

int main()
{
    char ch;
    printf("Enter the character");
    ch = getchar ( );  // it will scan the char
    //scanf("%c",&ch);
    // --> 0 --> 48

    if(ch>=97 && ch<=122)
        printf("Lower case \n");
    else
    {
        if(ch>=65 && ch<=90)
          printf("upper case");
        else
        {
            if(ch>=48 && ch<=57)
                printf("Digit \n");
            else
                printf("Special symbol\n");
        }
    }
}
```

Ternary Operator(if else)

```
// int main()
// {
//     int a=100;
//     int b=12;
//     int max;
//     // if(a>b)
//     //    max = a;
//     // else
//     //   max =  b;

//     // printf("%d",max);

//     //condition ? expression1 : expression2

//     max = a   >   b   ?   a   :   b ;
//     //       condt   ?  exp1 :  exp2
//     printf("max = %d",max);

//     return 0;
// }
```

```
#include<stdio.h>
int main()
{
   int num1=100;
   int num2=20;
   int num3=300;
   int max;

   //condition ? expression1 : expression2

max = num1 > num2 ? ( num1 > num3 ? num1 : num3) : ( num2 > num3 ? num2 :num3);
   printf("%d",max);
return 0;
}
```

Break Jump statement

```
#include<stdio.h>
// jump statment --> break

int main()
{
   int choice;
   printf("Enter the choice");
   scanf("%d",&choice);

   switch(choice)
   {
      case 2:
      printf("One\n");
      break;

      case 1:
      printf("Two\n");
      break;

      case 3:
      printf("Three\n");
      break;

      default:
      printf("Invalid");
      break;
   }
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int num  = 2;
   switch (num)
   {
      case 1:
      printf("One");
      break;
```

```
      // case 2>1:  // --> 2 > 1 --> 1 --> case 1: duplicate case is not aloud.
      case 2-1+1:    // 2-1+1 --> case 2
       printf("Two");
       break;

   default:
      break;
   }
}
```

While Loop

```
#include<stdio.h>
int main()
{
   int i = 1;
   while(i<=5)
   // 1<=5 --> T
   // 2<=5 --> T
   // 3<=5 --> T
   // 4<=5 --> T
   // 5<=5 --> T
   // 6<=5 --> F
   {
       printf("%d",i); // 1 2 3 4 5
       i++;
   }
return 0;
}
```

```
#include<stdio.h>
int main()
{
int i = 1;
       while(i<=5)
       // 1<=5 --> T
       // 2<=5 --> T
       // 3<=5 --> T
       // 4<=5 --> T
       // 5<=5 --> T
       // 6<=5 --> F
       {
          printf("%d",i); // 12345
```

```
        i++;
    }
    printf("\nOutside the loop : %d",i);   // 6
```

```
#include<stdio.h>
int main()
{
 int i = 1;
    while(i<=5) // 6<=5
    {
        printf("%d",++i); // 23456

    }
    printf("\nOutside the loop : %d",i);//6
```

```
#include<stdio.h>
int main()
{
// i=1;
    // j=1;
    // while(i<=5,j<=10)
    // {
    //     printf("i:%d j:%d\n",i,j);
    //     i++;
    //     j++;
    // }
```

Output:- i = 1,2,3,4,5,6,7,8,9,10
J = 1,2,3,4,5,6,7,8,9,10

```
     int i,j;
    i=1;
    j=1;
    while(i<=10,j<=5)
    {
        printf("i:%d j:%d\n",i,j);
        i++;
        j++;
    }
```

Output:- i = 1,2,3,4,5
J = 1,2,3,4,5

```
  return 0;
}
```

For Loops

```
#include<stdio.h>
int main()
{
   // while --> entry controlled loop
   /*
         while(condtn)
         {
            // Statements
         }
   */
  // for loop --> entry controlled loop
   /*
         for(step1;step2;step4)
         {
            step3;
         }
         step 1 --> executed only once
   */

   // int i;
   // for(i=1;i<=5;i++)
   // {
   //    printf("\n%d",i); //1 2 3 4 5
   // }
   // printf("\nOutside the loop %d",i);
   //
         // for(step1;step2;step4)
         // {
         //    step3;
         // }
         // step 1 --> executed only once


   // int i=1;

   // for(;i<=5;i++)
   // {
   //    printf("\n%d",i); //1 2 3 4 5
   // }
   // printf("\nOutside the loop %d",i);  //6
```

```
   int i=1;
   //for(  ; i<=5  ;  )
   for( ; i<=5 ;)
   {
      printf("\n%d",i++); //1 2 3 4 5
   }
   printf("\nOutside the loop %d",i);
output:- 1 2 3 4 5 6
            Outside the loop 6

   return 0;
}
```

Do-While Loops

```
#include<stdio.h>
int main()
{
   // while and for --> entry controlled
   // do-while --> exit controlled
   // exit controlled --> condtn checked at the end
   int choice;
   do
   {
       printf("Enter the choice");
       scanf("%d",&choice);
   } while (choice<=10 && choice>=1);

   printf("loop exited");
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   // jump statment --> break
   // break , return , continue , goto
      // int i = 3;
      // if(i==3) //break statement not within loop or switch
      //   break;
```

```
    int i;
    for(i=1;i<=5;i++)
    {
        printf("Inside %d\n ",i);//123
        if(i==3) // 3==3
         break;
    }
    // break is taking the control outside the loop
    printf("\nOutsideside %d ",i); //3
  return 0;
}
```

Goto Statements

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

  START:
  printf("Enter the rank\n");
  scanf("%d",&rank); // 6

  if(rank>=1 && rank<=5)
     goto LABEL;
  else
     goto START;

  LABEL:
  printf("Excellent rank!!\n");
  printf("All the best");
  return 0;
}
```

Continue Jump statement

```
#include<stdio.h>
int main()
{
  int i=1;
  // continue --> jump statement
  // if(i==1)
  //   continue; // Error continue statement not within a loop
```

```
 /*
      1. continue statement should be within a loop
      2. continue takes the control to nearest enclosing loop
   */
   for(i=1;i<=10;i++)
   {
      if(i%2==0) //5%2==0
       continue;
      printf("%d",i); //135
   }
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   int num = 5;
   switch(num) {
   case 1 + 2 + 2: // case 5
      printf("1 + 2 + 2\n");
      break;
//  case 5: // error: duplicate case
//     printf("5\n");
//     break;
   case '5' && 'A': // T && T --> 1 ====> case 1:
      printf("5 && A\n");
      break;
   case NULL: // NULL is 0 --> case 0: // C++
            // NULL is (void*)0 --> // C -- error cannot use pointers.
      printf("NULL\n");
      break;
//  case 0: // error: duplicate case
//     printf("0\n");
//     break;
   }
   return 0;
}
```

Nested for Loops

```
int main()
{
    // nested loops
    /*
            for(step1;step2;step4)
                 step3
            for(i=1;i<=5;i++)
                printf("%d",i);
        i=1 --> only once
    */
    int i,j;

        // nested loops

        for(i=1;i<=5;i++) // i =6
        {
            printf("For each i :%d    ",i); //2
            for(j=1;j<=5;j++)//6
            {
                printf("j:%d",j);//12345
            }
            printf("\n\n");
        }
        /* output:-
            i==>1
                  j=1,2,3,4,5
            i==>2
                  j=1,2,3,4,5

            i==>5
                  j=1,2,3,4,5
        */
    return 0;
}
```

```
void main() {
   int a = 3, b = 6;
   printf("%d\n", a != b ? 0 ? b : a : b);
}
// a != b ? (0 ? b : a) : b
// a != b ? a : b
// a != b --> 3 != 6 --> true --> first expression = "a" --> 3
```

```
-----------------------------------------------------------------------
#include<stdio.h>
int main()
{
    int i,j,k;

        for(i=1;i<=5;i++) // i =2  2<=5
        {
            printf("For each i :%d    ",i); //2
            for(j=1;j<=5;j++)// j =1 1<=5
            {
                 printf("For each j :%d    ",j);
                 for(k=1;k<=5;k++) // k=6 6<=5
                 {
                     printf("k:%d",k);// 12345
                 }
                 printf("\n\n");
            }
            printf("\n\n");
        }
/* Output:-
For each i :1
For each j :1    k:1k:2k:3k:4k:5
For each j :2    k:1k:2k:3k:4k:5
For each j :3    k:1k:2k:3k:4k:5
For each j :4    k:1k:2k:3k:4k:5
For each j :5    k:1k:2k:3k:4k:5

For each i :2
For each j :1    k:1k:2k:3k:4k:5
For each j :2    k:1k:2k:3k:4k:5
For each j :3    k:1k:2k:3k:4k:5
For each j :4    k:1k:2k:3k:4k:5
For each j :5    k:1k:2k:3k:4k:5


For each i :3
For each j :1    k:1k:2k:3k:4k:5
For each j :2    k:1k:2k:3k:4k:5
For each j :3    k:1k:2k:3k:4k:5
For each j :4    k:1k:2k:3k:4k:5
For each j :5    k:1k:2k:3k:4k:5
*/
-----------------------------------------------------------------------------------------------------------
```

Enum Data type Concept

```c
#include<stdio.h>
enum color // creating the datatype
{
    RED,BLUE,GREEN
};
int main()
{
    // int , float ,double ,char ==> primitive / builtin
    // int --> int num
    // int -->datatype
    // num --> variable-name

      int i,j,k;// datatype variable-name
      // int --> datatype
      // i,j,k --> variable-name

      enum color c1,c2,c3;
      // enum color --> datatype
      // c1,c2,c3--> variable names

    return 0;
}
```

```c
#include<stdio.h>
enum color // creating the datatype
{
    RED,BLUE,GREEN,VIOLET
};
//enum constant values by default start from 0 and assigned
sequentially
int main()
{
    printf("%d\n",RED); //0
    printf("%d\n",BLUE);//1
    printf("%d\n",GREEN);//2
    return 0;
}
```

```c
#include<stdio.h> // enum ==> user defined datatype
enum color // creating the datatype
{
    RED,BLUE,GREEN,VIOLET
};
```

```c
int main()
{
    int num;
    // num++ --> yes
    //printf("%d\n",sizeof(num)); // 4
    //printf("%d\n",sizeof(enum color)); // 4
    enum color c1,c2,c3,c4;
    //c1,c2,c3 --> variable names

    c1 = RED;  //0
    c2 = BLUE; // 1
    c3 = GREEN; //2
    c4 = VIOLET;  //3

    printf("%d %d %d %d\n",c1,c2,c3,c4);
    c1++;   // c1 = c1 + 1 //  c1 = 0  + 1
    c2++;
    c3++;
    c4++;
    printf("%d %d %d %d",c1,c2,c3,c4);

    //RED++; //lvalue required as increment operand
    //0++ ==> 0 = 0 + 1
    // 3 = 2+1 // lvalue error
    // num = 2 + 1
    return 0;
}
```

```c
#include<stdio.h>
enum color // creating the datatype
{
    RED=1,BLUE=1,GREEN=2,VIOLET
};

int main()
{
    int num;
    // num++ --> yes
    //printf("%d\n",sizeof(num)); // 4
    //printf("%d\n",sizeof(enum color)); // 4
    enum color c1,c2,c3,c4;
    //c1,c2,c3 --> variable names
```

```
    c1 = RED;  //0
    c2 = BLUE; // 1
    c3 = GREEN; //2
    c4 = VIOLET;  //3

    printf("%d %d %d %d\n",c1,c2,c3,c4);
    c1++;   // c1 = c1 + 1 //  c1 = 0  + 1
    c2++;
    c3++;
    c4++;
    printf("%d %d %d %d",c1,c2,c3,c4);

    //RED++; //lvalue required as increment operand
    //0++ ==> 0 = 0 + 1
    // 3 = 2+1 // lvalue error
    // num = 2 + 1
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    // enum ==> user defined datatype
    enum color // creating the datatype
    {
        RED=1,BLUE=1,GREEN=2,VIOLET
    };
    int num;
    // num++ --> yes
    //printf("%d\n",sizeof(num)); // 4
    //printf("%d\n",sizeof(enum color)); // 4
    enum color c1,c2,c3,c4;
    //c1,c2,c3 --> variable names

    c1 = RED;  //0
    c2 = BLUE; // 1
    c3 = GREEN; //2
    c4 = VIOLET;  //3

    printf("%d %d %d %d\n",c1,c2,c3,c4);
    c1++;   // c1 = c1 + 1 //  c1 = 0  + 1
    c2++;
    c3++;
```

```
    c4++;
    printf("%d %d %d %d",c1,c2,c3,c4);

    //RED++; //lvalue required as increment operand
    //0++ ==> 0 = 0 + 1
    // 3 = 2+1 // lvalue error
    // num = 2 + 1
    return 0;
}
```

```
#include<stdio.h>
enum color
{
    RED=1,BLUE,GREEN
};
int main()
{

    enum color choice;

    printf("Enter the choice");
    scanf("%d",&choice);

    switch (choice)
    {
      case RED:
      printf("Red color");
      break;

      case BLUE:
      printf("Blue color");
      break;

      case GREEN:
      printf("Green color");
      break;
     default:
        break;
    }
```

Typedef concept

```c
#include<stdio.h>
int main()
{
    // user-defined datatype
    enum color
    {
        RED=1,BLUE,GREEN
    };
    int num;
    //datatype variable-name

    //typedef --> creating an alias
    // alias --> nickname

    typedef int INT;
    // INT is alternative/nickname given to int
    INT num3,num4,num5;

    enum color c11;
    //datatype variable-name

    typedef enum color e_c;
   //e_c is alias/alternative name for enum color
   // enum color c1,c2,c3

   e_c c1,c2,c3;
    return 0;
}
```

Functions

```c
#include<stdio.h>
double addition( double x , double y);
// function declaration
int main()
{
    double num1 = 12.00;
    double num2 = 4.00;
    double res;

    // num1 and num2 ==> actual arguments
    res = addition( num1 , num2 ); // function call
    //              12.00   4.00
    printf("result = %.2lf\n",res);//16.00

    res = addition(20.00,5.00 ); //function call

    printf("result = %.2lf\n",res);//25.00

    return 0;
}

// addition function will do the addition for me
// function defination / Implementation
//             12.00       4.00
// return-type function-name ( formal arguments  );
double addition( double x , double y)
{
      double res;
      res = x + y; // 12.00 + 4.00 // processing
      return res; // 16.00
 // i am returning the value of type double
 // res ==> double
 // function is returning the value of type double

}
```

```c
#include<stdio.h>
// function declaration
double addition( double x , double y);
void multiplication( double a , double b  );
double division(int num , int den);
void subtract( void );
int main()
{
    double num1 = 12.00;
    double num2 = 4.00;
    double res;
    int num,den;

    res = addition( num1 , num2 ); //function call
    printf("result = %.2lf\n",res);
    res = addition(20.00,5.00 ); //function call
    printf("result = %.2lf\n",res);

    //function call
    multiplication(2.0,4.0);

    printf("Enter the num and den\n");
    scanf("%d%d",&num,&den); // 4 2
    res = division(num ,den); // function call
    //                4    2
    printf("result = %.2lf\n",res);


    subtract( ); // function call
    return 0;
}
// void ==> no taking any I/P as well as
// not returning any value
void subtract( void )
{
    int p,q,r; // variable declaration
    printf("\nEnter the two numbers");
    scanf("%d%d",&p,&q);
    r = p - q;
    printf("\nResult = %d",r);
}
```

```
// function defination
//                    5          2
double division(int num , int den)
{
        double r;
        r = (double)num / den;
        return r;
}


// calling function ==> main
// function defination
double addition( double x , double y)
{
      double res;
      res = x + y;
      return res;
}
// void --> if function is not returning anything
// then we write it as void
void multiplication( double a , double b  )
{
        double c;
        c = a  * b;
        printf("\nresult = %.2lf\n",c);
 //printing result inside the function
 // no need to return
}
```

Test Questions:-

Q.1

```
int main()
{
 int a,b,c,sum;
 sum = (a=3,b=4,c=5,~(a+b)+c);

 //    a=3 b=4 c=5 ~(3+4)+5
 //          ~(7) + 5
 //           -8 + 5
 //            3 4 5 -3
 //printf("%d %d %d %d",a,b,c,sum);

  a = a=2,~a++,a+10; // NO lvalue
  //   2, 3, 13   // a = 3
```

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

   b=(--a,b++,(c=b |a?1:b&a?1:0));
  //   2 ,4, (c=5 | 2?1:5&2?1:0)
  //   2, 4 , (c=(5 | 2)?1:5&2?1:0)-->(c=7?1:5&2?1:0)--> 1
  //   2 , 5 , 1
  //  a=2 b=1 c=1 ( b =1 as right-most value is assigned due to bracket )
   printf("%d %d %d",a,b,c);

  //c=sizeof(a++ + ~b--),sizeof(a/(b+=c++)),c--;
  //printf("%d %d %d %d",a,b,c,sum);

 return 0;
}
```

Q.2

```
#include<stdio.h>
int main()
{
  int i =10;
  int j;
  j = !i < 9;  // !i --> !10 --> !T --> F --> 0 < 9 --> 1
  // j --> 1
  // i = 10;
  printf("%d %d",i, j);
                  10  1
}
```

Q.3

```
#include<stdio.h>
int main()
{
 int a,b,c,sum;
 sum = (a=3,b=4,c=5,~(a+b)+c);

 //     a=3 b=4 c=5 ~(3+4)+5
 //              ~(7) + 5
 //               -8 + 5
 //                3 4 5 -3
 //printf("%d %d %d %d",a,b,c,sum);

  a = a=2,~a++,a+10; // NO lvalue
```

```
  //    2, 3, 13   // a = 3
  //printf("%d",a);

   b=(--a,b++,(c=b |a?1:b&a?1:0));
  //   2 ,4, (c=5 | 2?1:5&2?1:0)
  //   2, 4 , (c=(5 | 2)?1:5&2?1:0)-->(c=7?1:5&2?1:0)--> 1
  //   2 , 5 , 1
  //  a=2 b=1 c=1 ( b =1 as right-most value is assigned due to bracket )
   printf("%d %d %d\n",a,b,c);

  c=sizeof(a++ + ~b--),sizeof(a/(b+=c++)),c--;
  // c = sizeof(a++ + ~b--),sizeof(a/(b+=c++)) ==> 4
  // but rightmost c-- makes it as 3
  printf("a=%d b=%d c=%d sum=%d",a,b,c,sum);
  //                    a=2 b=1 c=3 sum=-3
 return 0;
}
```

Poll questions

Q.1

What will be the output of following program ?

```
#include <stdio.h>
void main()
{
   char val=1;
   if(val--==0) // post decrement -- consider old value ===> 1 == 0 - false
      printf("TRUE");
   else
      printf("FALSE");
}
```

A) FALSE
B) Error
C) TRUE
D) None
Answer: A

Q.2

What will be the output of following program ?

```
#include <stdio.h>
void main()
{
        int num = 24;
```

```
        num % 2 == 0 ? goto end1 : goto end2;
                // condition ? expression1 : expression2;
                // expression1 & 2 --> MUST BE EXPRESSIONS -- NOT STATEMENT.
                        // EXPRESSION -- evaluate to some value.
                        // STATEMENT -- command/action -- goto, return;
                // compiler error.
end1:
        printf("Even ");
end2:
        printf("Odd ");
}
```

A) Even
B) Odd
C) Even Odd
D) None of these
Answer: D

Q.3
What will be the output of following program ?

```
#include <stdio.h>
int main() {
        int i, j;
        for(i=1; i<=3; i+=2) {
                for(j=1; j<=3; j=+2)
                        printf("%d ", i + j);
        }
        return 0;
}
```

A) 2 4 4 6
B) 2 3 3 3 ... (infinite loop)
C) Compiler error
D) None of these
Answer: B

Q.4
Which of the following is exit controlled loop?
A) for
B) while
C) do-while
D) if-else

Answer: C

TWISTERS

1. #include <stdio.h>
int num;
int function(int n)
{
int num = 10;
return num;
}
int main(void)
{
printf("%d %d\n", num, function(20));
return 0;
}
A. 10 0
B. 20 0
C. 0 10
D. 0 20
Answer: C

2. #include <stdio.h>
int no1 = 17, no2 = 71;
void swapping(void)
{
int temp = no2;
no2 = no1;
no1 = temp;
}
int main(void)
{
int no1 = 17, no2 = 71;
printf("%d %d ", no1 , no2);
swapping();
printf("%d %d\n", no1, no2);
return 0;
}
A. 17 17 17 17
B. 17 71 17 71
C. 71 17 71 17
D. 71 71 71 71
Answer: B

3. #include <stdio.h>
int sunbeam()
{
int a=3;
return a * a;
}
int main(void)
{
int a=3;
printf("%d ", sunbeam(a));
return 0;
}
A. 9
B. garbage
C. compiler error
D. runtime error
Answer: A

4. #include <stdio.h>
int testDemo(int, int);
int main(void)
{
int you = 64, me = 32;
int we = testDemo(you, me);
printf("%d %d %d\n", me, you, we);
return 0;
}
int testDemo(int me, int you)
{
me = me + you;
return me - you;
you = you - me;
return me + you;
}
A. 32 64 32
B. 64 64 32
C. 64 32 64
D. 32 64 64
Answer: D

Local , Global , Static variable storage class

```
#include<stdio.h>
// static
// created when program is started==> even before main () is called
    // created in datasectin ==> default value is 0
    // life ==> program
    // scope ==>program
    void fun();
int main()
{
    int num = 100; // local variable
    static int num2 = 200; // static variable
    printf("%d",num2); // 200
    fun();
}

void fun()
{
    int num3;
    static int num2 = 300;
    printf("%d",num2); //300
}
```

```
#include <stdio.h>
static variable should be init at the time of declaration
int fun() {
    static int num;
    num = 10;
    num++;
    printf("%d\n", num); //11 11 11
}
int main() {
    int num;
    fun(); // 11
    fun(); // 12
    fun(); // 13
    printf("%d\n", num); // error: num is not in scope
    return 0;
}
```

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

```
    static int num = 10; // static  -- initilized only once -- when
program started (like global)
    num++;
    printf("%d\n", num);  //11 12 13
}
int main() {
    //printf("%d\n", num); // error: num is not in scope
    fun(); // 11
    fun(); // 12
    fun(); // 13
    return 0;
}
```

```
#include <stdio.h>
int num = 10; // global variable

int fun() {
    num++;
    printf("%d\n", num);
}
int main() {
    printf("%d\n", num); // 10
    fun(); // 11
    fun(); // 12
    fun(); // 13
    return 0;
}
```

```
#include <stdio.h>
void fun() {
    int num = 10; // local variable
    num++;
    printf("%d\n", num);
}
int main() {
    //printf("%d\n", num); // error: num is not in scope
    fun(); // 11
    fun(); // 11
    fun(); // 11
    return 0;
}
```

Register storage class

```c
#include<stdio.h>
int main()
{
        register a = 10;
        {
            register a = 15;
            printf("%d\n",a);
        }
        printf("%d\n",a);
}
// #include<stdio.h>
// //register int num;//NOT OK
// int main()
// {
//     register int i;
//     // storage ==> cpu register ( fast accessible)
//     printf("%d\n",i);  // garbage
//     {
//         register int i = 100;
//         printf("%d\n",i);//100
//     }

//     int x; // local/auto --> stack --> process--> RAM
//     register int y;
//     printf("%u",&x); //address of x
//     // & --> addressof operator
//     //printf("%u",&y); //address of y // not OK
// }
```

->Register strorage class ko memory CPU register se milti hai.
->Register ko hum globaly defined nhi kr sakte.register ka address print nhi kr sakte bcoz of its stored on CPU.scanf bhi nhi used kr sakte.
->baki sab local/auto jaise hi rule hai.
->by defalut register int kaha jata hai.

->global variable ko memory data section par milti hai default value 0 hai aur uska scope program tk hi hai.
->global variable dusre file mai bhi access kr sakte hai.

-> static ko memory data section pr milti hai dafalut value 0 hai aur uska scope limited hai.
->static varible sirf same file mai hi access kar skte hai.

TWISTERS

1. #include <stdio.h>
int ext = 30;
int main(void)
{
extern int ext;
printf("Ext = %d ", ext);
extfun();
return 0;
}
int ext = 10;
int extfun(void)
{
int ext = 20;
printf("%d\n",ext);
}
A. Ext = 10 20
B. Ext = 30 20
C. Compile time error
D. Run time error
Answer: C

2. #include <stdio.h>
static char char1 = 'A';
extern char char2 = 'B';
register char char3 = 'C';
void mystorage(void)
{
printf("%c %c %c\n", char1, char2, char3);
}
int main(void)
{
printf("%c %c %c\n", char1, char2, char3);
mystorage();
return 0;
}
A. A B C A B C
B. Compile time error -static variable cannot be declared globally
C. Compile time error -extern variable cannot be declared globally
D. Compile time error -register variable cannot be declared globally
Answer: D

3. #include <stdio.h>
int demo(char p1, char p2)
{
char p3;
p3 = ~p1 + ~p2;
return p3;
}
int main(void)
{
char p1 = 255, p2 = 256;
char p3 = demo(~p1++, ~p2--);
printf("%d %d %d\n", p1, p2, p3);
return 0;
}
A. -1 -1 0
B. -1 0 -1
C. 0 -1 -1
D. None of the above
Answer: C

4.
#include <stdio.h>
int i = 0;
int main(void)
{
auto int i = 1;
printf("%d ", i);
{
int i = 2;
printf("%d ", i);
{
i += 1;
printf("%d ", i);
}
printf("%d", i);
}
printf("%d", i);
return 0;
}
A. 0 1 2 2 0
B. 1 2 3 2 1

C. 1 2 3 3 1
D. 0 1 2 1 0
Answer: C

5. #include <stdio.h>
int my = 0;
int myset(int my)
{
printf("%d ", my++);
return my = my <= 2 ? 5 : 0;
}
int main(void)
{
int my = 5;
myset(my/2);
printf("%d ", my);
myset(my=my/2);
printf("%d ", my);
my = myset(my/2);
printf("%d ", my);
return 0;
}
A. 3 5 3 2 2 5
B. 2 5 2 2 1 5
C. 2 3 2 2 2 5
D. 3 3 3 2 1 5
Answer: B

Pointer

Pointer

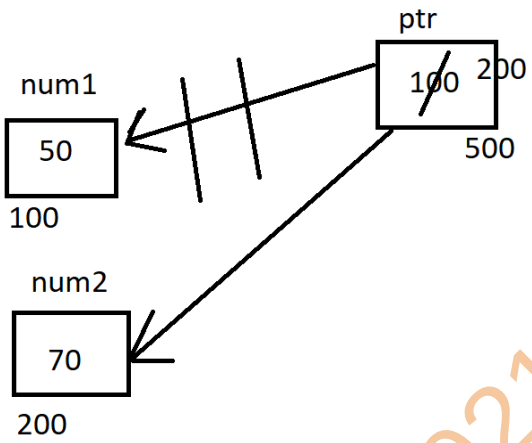
- ->pointer is variable which holds the address.
- ->pointer is variable which also has address.
- ->if variable is integer then pointer is also integer,if float then pointer also float same as with char.
- int *ptr ->ptr is pointer to integer.and internally is unsiged data type.

```
int num=10;
*ptr=&num; //this is (& is referancing operation.)
printf("Num=%d",num); // 10

printf("&Num=%u",&num); //address of num
printf("&ptr=%u",&ptr); //address of ptr
printf("ptr=%u",ptr); //address of num
```

- %u is used for print address bcoz pointer is unsiged.
- ->to read the value of num through ptr by (*)dereferencing operation

```
printf("ptr=%d",*ptr); //10
* --> value at operator or indirection operator
/*      *ptr
        valueat(ptr)
        valueat(100)
         10
*/

now i want to change num value by ptr
*ptr=1000;
/*
    *ptr
    valueat(ptr)
    valueat(100)
    valueat(100) = 1000
    through pointer i have updated the value of num
*/
```

```
#include<stdio.h>
int main()
{
   int num1 = 50;
   int num2 = 70;
   int *ptr = &num1;
   // here ptr init stored the address of num1
   printf("num1 = %d\n",num1);// 50
   printf("*ptr = %d\n",*ptr);// 50

   ptr = &num2;

   // now ptr is storing the address of num2( updated )
   printf("num1 = %d\n",num2);// 70
   printf("*ptr = %d\n",*ptr);// 70
   return 0;
}
```



```
#include<stdio.h>
int main()
{
   // int  ==> 4
   // float ==> 4
   // char ==> 1
   // double ==> 8
   int *ptr ;
   char *cptr;
   float *fptr;
   double *dptr;                8 byte    8 byte     8 byte       8 byte
   printf("%d %d %d %d",sizeof(ptr),sizeof(cptr),sizeof(fptr),sizeof(dptr));
  //size of pointer is always same in all data type.
   // 32 bit--> 4 byte
   // 64 bit--> 8 byte
   return 0;
}
```

- -->But when *ptr is reading num value that time size of ptr is according data type.

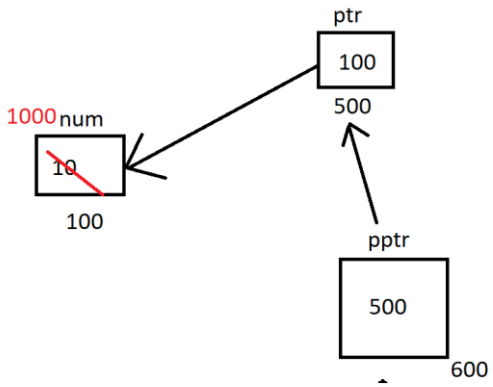
```
#include<stdio.h>
int main()
{
   int num = 10;
   int *ptr;  // integer pointer
   char *cptr; // character pointer
   double *dptr; // double pointer
   printf("%d\n",sizeof(*ptr));    //4 byte
   printf("%d\n",sizeof(*cptr));  //1 byte
   printf("%d\n",sizeof(*dptr)); //8  byte
}
```

- ->Ek pointer ka address dusre pointer mai rakh sakte hai uss concept ko pointer to pointer kahte hai.

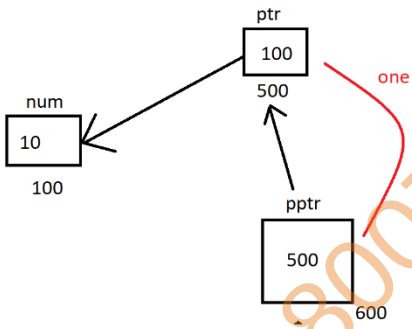
```
#include<stdio.h>
int main()
{
   int num = 10;
   int *ptr = &num; // referencing

   // pointer to pointer

   int **pptr = &ptr;
   //pptr is a pointer which is storing the
   // address of a pointer to a integer

   // int ***ppptr = &pptr ;

   printf("num = %d\n",num); // 10
   printf("&ptr = %u\n",&ptr); // address of ptr
   printf("ptr=%u\n",ptr); //address of num
   printf("&pptr=%u\n",&pptr); //address of pptr
   printf("pptr=%u\n",pptr);//address of ptr
   return 0;
}
```

- Double dereferncing krake num ki value tak ja sakta hai.

```
#include<stdio.h>
int main()
{
   int num = 10;
   int *ptr = &num; // ref
   // pointer to pointer
   int **pptr = &ptr;
   printf("num = %d\n",num); //10
   printf("*ptr=%d\n",*ptr); //10 // dereferencing
   printf("**pptr=%d",**pptr); //10
   /*
           **pptr
           valueat(valueat(pptr))
           valueat(valueat(500))
           valueat(100)
           10
   */

//now you can change the num value through pointer
        **pptr = 1000;

   printf("num = %d\n",num); //1000
   printf("*ptr=%d\n",*ptr); //1000 // dereferencing
   printf("**pptr=%d",**pptr); //1000
   /*
           **pptr

           valueat(valueat(pptr))
           valueat(valueat(500))
           valueat(100)
           1000
   */
   return 0;
}
```



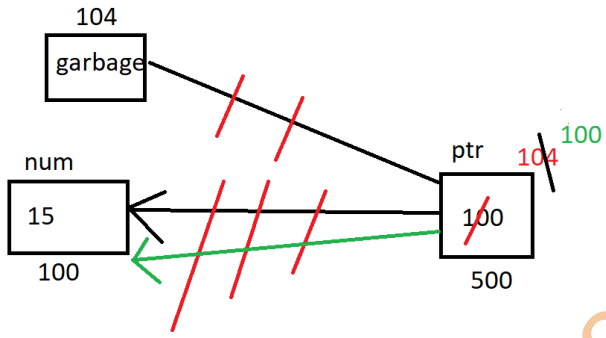
```
#include<stdio.h>
int main()
{
   int num = 10;
   int *ptr = &num; // referencing
   int **pptr = &ptr;

   printf("&num=%u\n",&num); //address of num
   printf("ptr=%u\n",ptr);   //address of num
   printf("*pptr =%u\n",*pptr);
   // *pptr --> state of ptr
   // state of ptr --> address of num
   /*
         *pptr
         valueat(pptr)
         valueat(500)
         100
   */
  return 0;
}
```



- ptr++ ; ptr ka type konsa hai us hisab se ohh aage jayenga. char ,float,int ,double ki size ke anusar.

```
#include<stdio.h>
int main()
{
   int num = 15;
   int *ptr = &num;

   printf("&num=%u\n",&num); // address of num
   printf("num = %d\n",num); // 15
   printf("*ptr=%d\n",*ptr); // 15
   printf("ptr=%u\n",ptr); // address of num
   // pointer arithmatic
   ptr = ptr + 1;
   //    100 + 1
   //    100 + 1 * 4 ( int* --> 4 bytes )( if char*-->1byte)
   //    100 + 4
   //    104
   printf("\n\n");
```

```
   printf("ptr=%u\n",ptr);  //garbage

   ptr = ptr - 1;
   //ptr = 104
   //    104 - 1
   //    104 - 1 * 4
   //    104 - 4
   //    100

   printf("ptr=%u\n",ptr);    //address of num
   printf("*ptr=%u\n",*ptr);  //15
   return 0;
}
```



Note:-

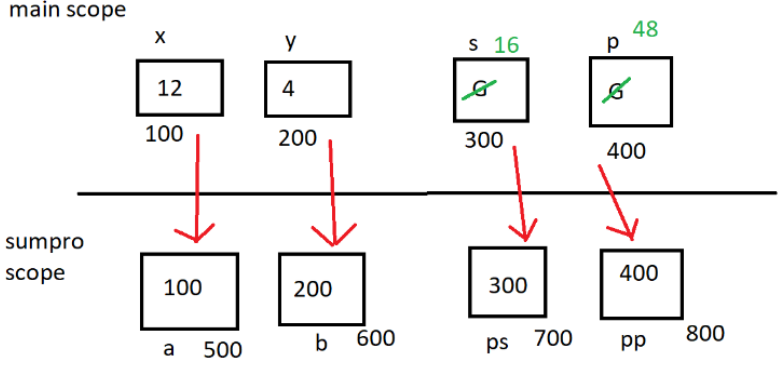
```
   /*
      ptr + n
      ptr + n * scale factor of ptr
   */
```

scale factor of ptr ->size of data type pointing to pointer.

-->When pointer is increment or decrement by 1 it changes by the scale factor.
-->When integer n is added or subtracted from a pointer it changes by n*scale factor.
-->Multiplication or division of any integer with two pointer is not allowed.
-->Addition,multiplication and division of two pointer is not allowed.
-->subtraction of two pointer gives number of location in between its useful in arrays.

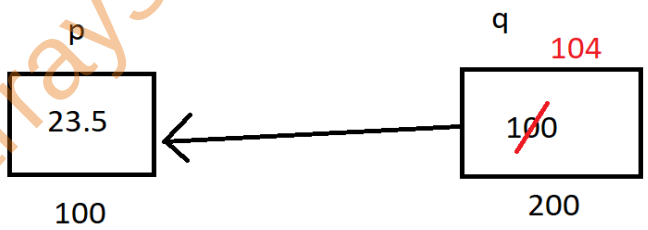
```
#include<stdio.h>
void sumpro(int *a, int *b, int *ps, int *pp)
{
 *ps = *a + *b;
 /*
     valueat(100) + valueat(200)
        12     +   4

  *ps --> valueat(ps)
       valueat(300)
       valueat(300) = 12 + 4
            = 16
 */

 *pp = *a * *b;
```

pass by address



```
/*
      valueat(a) * valueat(b)
      valueat(100 ) * valueat(200)
        12  * 4
 *pp
  ---> valueat(pp)
       valueat(400)
       valueat(400) = 12 * 4
             = 48
 */
}
// by using pointers we can return more than one value indirectly

int main() {
 int x = 12, y = 4, s, p;
 sumpro(&x, &y, &s, &p);
 printf("%d %d", s, p);
 return 0;
}
```

```
Example:- main( )
{
float.jamboee ( float*);
float P = 23.5, *q;
q=&p;
printf("q before call = %u", q ) ;
q=jamboee(&p);
printf ( "\q after call = %u", q ) ;
}
float *jamboee ( float *r )
{   r=r+1;
    return(r);
}
```



printf("q before call = %u", q) ; //100

printf ("\q after call = %u", q) ; //104

r=r+1;
=100+1*4
= 104

```
solution:-
printf("q before call = %u", q );//address of p i.e 100
// r=r+1;
//  =100+1*4
//  = 104
printf ( "\q after call = %u", q );// address of q update 104
```

Arrays

- int arr[5]; //array declaration
- int-->it will store only int type data
- arr-->is the name of the array
- [5]-->it can store 5 integer values
- [ ]-->this is subcript operator.
- Array is collection of similar data elements in contiguous memory locations.
- Elements of array share the same name i.e. name of the array.
- They are identified by unique index/subscript. Index range from 0 to n-1.
- Array indexing starts from 0.
- Checking array bounds is responsibility of programmer (not of compiler).(if arr size is 5 if you give 10 for printing its meaning less and print some garbage value)
(Also if your arr size is 5 and intialize krte waqkt bhi sirf 5 hi value chahiye)
- Size of array is fixed (it cannot be grow/shrink at runtime).

```
#include<stdio.h>
int main()
{
   // int num1,num2,num3,num4,num5
   int arr[5]= {1,2,3,4,5};// init list
   // [ ] --> subscript operator
   /*
         ( 0 to n-1 )
         ( 0 to 4   )
            arr
         1    2    3    4    5
         [0]   [1]    [2]    [3]    [4] (index/subscript)
        100   104   108   112   116 (address)
   */
  // if i want to print 1
  // i will use the name of the array --> arr
  // arr --> arr[0] --> 1

      // subscript operation
   // printf("%d",arr[0]); // 1
   // printf("%d",arr[1]); //2
   // printf("%d",arr[2]);// 3
   // printf("%d",arr[3]);//4
   // printf("%d",arr[4]);//5
```

```
    int i;
      // 0 to 4
    for(i=0;i<5;i++)
    {
        printf("%d",arr[i]);
    }
   return 0;
}
```

- // int arr[5]= {1,2,3,4,5};// init list

- //int arr[5]; // local (garbage) if it write in main scope
- //int arr[5]; //0 0 0 0 0 if its write in golbal scope
- //int arr[5]= {1,2};//partial init //1 2 0 0 0
- //int arr[5]= {1};//partial init //1 0 0 0 0
- //int arr[];// NOT OK
- //int arr[] = {1,2,3,4,5}; //its ok intialize kiya hai

```
#include<stdio.h>
int main()
{
   int arr[5]= {1,2,3,4,5};// init list
   printf("%d\n",sizeof(arr));// 20
   printf("%d\n",sizeof(arr[0]));//4
   /*
        1    2    3    4    5
       [0]  [1]  [2]  [3]  [4] (index/subscript)
       100  104  108  112  116 (address)
   */
      printf("%d",arr[0]); // 1

      printf("%u",&arr[0]); //address of [0]-> 100

      int i;
      for(i=0;i<5;i++)
      {
         printf("%u ",&arr[i]);
         // &arr[0] &arr[1] ...&arr[4]
      }
 // name of the array represents address of first element
 //  arr --> address of first element
```

```
    printf("%u \n",&arr[0]);    //100
    // address of 1st elements
    printf("%u \n ",arr);
    // address of 1st elements   //100
}
```

•If array is initialized partially at its point of declaration rest of elements are initialized to zero.

• If array is initialized partially at its point of declaration, giving array size is optional. It will be inferred from number of elements in initializer list.

• The array name is treated as address of 0th element in any runtime expression.

• Pointer to array is pointer to 0th element of the array.

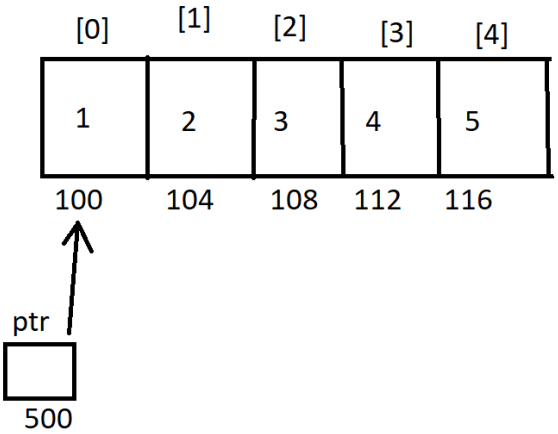
```
#include<stdio.h>
int main()
{
   int arr[5]= {1,2,3,4,5};// init list
   /*
         1     2     3     4     5
        [0]   [1]   [2]   [3]   [4] (index/subscript)
       100   104   108   112   116 (address)
   */
      printf("%u \n",arr);
      // arr --> name of array --> address of 1st element

      printf("%u \n",arr+1);
      /*
            arr + 1
            100 + 1
            100 + 1 * 4
            104
       */
     printf("%u \n",arr+2);
     /*
            arr + 2
            100 + 2
            100 + 2 * 4
            108
     */
```

```
 printf("%u \n",arr+3);
    /*
          arr+3
          100 + 3
          100 + 3 * 4
          112
    */
    printf("%u \n",arr+4);
    /*
          arr + 4
          100 + 4
          100 + 4 * 4
          116
    */

return 0;
}
```

```
#include<stdio.h>

int main()
{
   int arr[5]= {1,2,3,4,5};// initialization list

  /*
        1    2     3     4     5
       [0]  [1]   [2]  [3]   [4] (index/subscript)
      100  104  108  112  116 (address)
  */
   printf("%u \n",arr); // base address
   printf("%d \n",*arr);
   /*
        *arr
        valueat(arr)
        valueat(100)
         1
   */
   printf("%d\n",*(arr+1)); // 2
   /*
        *(arr+1)
        *(100 + 1)
        *(100 + 1 * 4)
```

```
        *(104)
         2
    */
    printf("%d\n",*(arr+2));
    /*
            *(arr+2)
            *(100 + 2 )
            *(100 + 2 * 4 )
            *(108 )
            3
    */
    printf("%d\n",*(arr+3));
    /*
            *(arr+3)
            *(100 + 3 )
            *(100 + 3 * 4 )
            *(112)
            4
    */
   printf("%d\n",*(arr+4));
    /*
            *(arr+4)
            *(100 + 4 )
            *(100 + 4 * 4 )
            *(116)
            5
    */
   printf("%d\n",*(4+arr));  //its also ok
   printf("%d\n",*(arr+2+2)); //its also ok

   /*
        arr[1] --> array notation
        1[arr] --> array notation
        arr[2-1] --> arr[1]

internally arr[1] is doing this
        *(arr+1) --> pointer notation
        *(1+arr) --> pointer notation
   */
   return 0;
}
```

```
#include<stdio.h>

int main()
{
   int arr[5] ={1,2,3,4,5};
   /*
        1      2      3      4      5
       [0]    [1]    [2]    [3]    [4] (index/subscript)
       100    104    108    112    116 (address)
   */
   int *ptr = arr;
   //int *ptr = &arr[0];
   // ptr is holding the base address

   printf("%u ",ptr);// base address of arr
   printf("%u \n",arr);  //base address of arr
    printf("%u ",*ptr);
    /*
            *ptr
            valueat(ptr)
            valueat(100)
            1
    */
     printf("%u\n",ptr); // 100
     printf("%u\n",ptr+1);
     // ptr + 1 --> 100 + 1 --> 100 + 1 * 4 --> 104
     printf("%u\n",ptr+2);
     // ptr + 2 --> 100 + 2 --> 100 + 2 * 4 --> 108
     printf("%u\n",ptr+3);
     printf("%u\n",ptr+4);

    printf("%u ",*ptr);
    /*
            *ptr
            valueat(ptr)
            valueat(100)
            1
    */
      printf("%d ",*(ptr+1));
      /*
            *(ptr+1)
            valueat(100 + 1)
```



```
            valueat(100 + 1 * 4 )
            valueat(104)
            2
       */
      printf("%d ",*(ptr+2));
      /*
            *(ptr+2)
            valueat(ptr + 2)
            valueat(100 + 2 * 4 )
            valueat(108)
            3

       */
      printf("%d ",*(ptr+3));
      /*
          *(ptr+3)
          valueat(100 + 3)
          valueat(100 + 3 * 4)
          valueat(112)
          4

       */
     printf("%d ",*(ptr+4)); //5
     /*    *(ptr+4)
          valueat(100 +4)
          valueat(100 + 4 * 4)
          valueat(116)
          5
          */
    printf("%d ",*(4+ptr)); //5
    return 0;
}
```

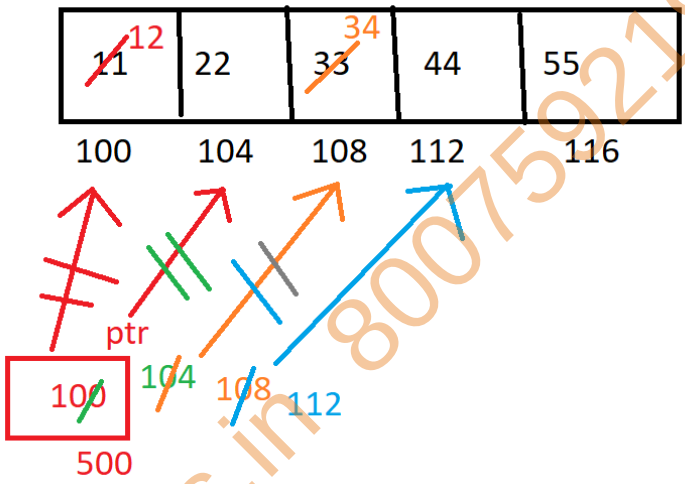
```
#include<stdio.h>
int main()
{
    int arr[5]= {1,2,3,4,5};// init list
    /*
          1      2     3      4      5
         [0]   [1]   [2]   [3]   [4] (index/subscript)
        100   104   108   112   116 (address)
```

```
   */
   // arr++; //lvalue  error
   // arr --> name of the array --> base address
   // baseaddress  = baseaddress + 1
   // 100 = 100 + 1
   //arr[0]++  its possible
    int *ptr1 = &arr[4]; // 116
    int *ptr2 = &arr[0]; // 100
    int ans;
    ans = ptr1 - ptr2;
    /*
        (ptr1 - ptr2)/(scale factor of ptr1)
        (116 - 100 ) / 4
         16 / 4
         4
    */
   // printf("%d",ans);  //4
     int i;
     for(i=0;i<5;i++)
     {
        printf("%u ",&arr[i]);
     }
     //printf("%u\n",arr); // base address
     printf("\n%u",&arr+1);// ye pure arr scale factor lenga
                    // 20 byte lenga
    printf("\n%u",&arr+2); //40 byte se aage jayenga
    printf("\n%u",&arr+3); //60byte se aage jayenga

    // arr + 1; // 4 bytes  imp concept
   //&arr+ 1;  //20 bytes   imp concept
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int arr[5]= {1,2,3,4,5};// init list
   /*
        1      2     3      4      5
       [0]    [1]   [2]    [3]    [4] (index/subscript)
       100   104   108   112   116 (address)
```

```
   */
   int *ptr = arr;
   //printf("%d",*ptr); // 1
   printf("%d",ptr[-2]);// garbage

   //array notation
   /*
           ptr[-2]
           *(ptr + -2)
           *(ptr - 2)
           *(100 - 2)
           *(100 - 2 * 4)
           *(100 - 8)
           *(92)
           Garbage
   */
     ptr++; // ptr = ptr + 1
     /*
           ptr++;
           ptr = ptr +  1
              = 100 + 1
              = 104
     */
     printf("\n%d",*ptr);
     //*(104) ==> 2

     ptr--;
     /*
           ptr--;
           ptr = ptr -  1
              = 104 - 1*4
              = 100
     */
     printf("\n%d",*ptr); //*(100) ==> 1
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int arr[5]= {11,22,33,44,55};// init list

   /*
         11    22    33    44    55
        [0]   [1]   [2]   [3]   [4] (index/subscript)
        100   104   108   112   116 (address)
   */
   int *ptr = arr;
   printf("%d\n",*ptr); // 11
   printf("%d\n",arr[0]);//11
   printf("%d\n",++*ptr);
   /*
         ++*ptr
         ++valueat(ptr)
         ++valueat(100)
         ++ ==> 11
         12
   */
    //printf("%d\n",arr[0]); //12

   printf("%d\n",*++ptr); // 22
   /*1
         *++ptr
         * ==> ptr = ptr + 1
         *       = 100 + 1
         *       = 104
          *(104) ==> 22
   */

   printf("%d\n",*ptr++);// 22
   /*
         *ptr --> (1 step ) --> 22
         ptr ++
         ptr = ptr + 1
            = 104 + 1
            = 108
   */
      printf("%d",*ptr); // 33
```



```
        //++*ptr++;// Homework
        printf("%d\n",++*ptr++);//34
        /*
              ++*ptr++
              ++33
              34
        */
       printf("%d",*ptr);//44

}
```

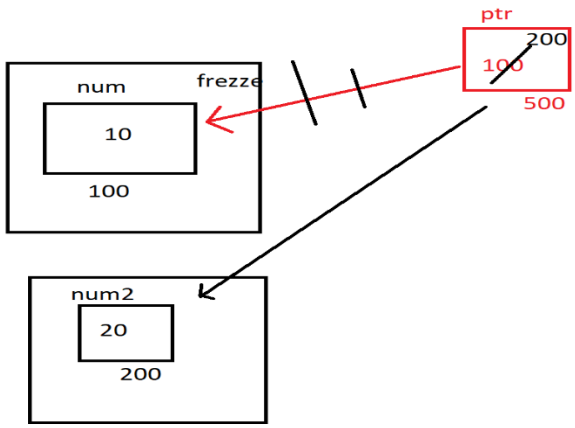
```
#include<stdio.h>
void readArray(int arr[],int size);
void readArray(int *ptr,int size);
void printArray(int arr[],int size); //array notation
int main()
{
   int arr[5];
   printf("Enter the elements of the array\n");
   readArray(arr,5);
   // arr --> name of array --> base address
   printf("Elements of the array are \n");
   printArray(arr,5);
   //arr--> name of array --> base address
   return 0;
}
// if i want to pass array to the function
// i need to pass the baseaddress
// array <--> pass by address
void readArray(int arr[],int size) //array notation
{
       int i;
       for(i=0;i<size;i++)
       {
          scanf("%d",&arr[i]);
       }
}
```

```
void readArray(int *ptr,int size) //pointer notation
{
	int i;
	for(i=0;i<size;i++)
	{
	   scanf("%d",&ptr[i]); // 1 2 3 4 5
	}
}
void printArray(int arr[],int size) //array notation
{
	int i;
	for(i=0;i<size;i++)
	{
	   printf("%d",arr[i]);
	}
}
```

```
#include<stdio.h>
void printArray(int *ptr,int size);
int main()
{
   int arr[5]= {1,2,3,4,5};
   //printf("%d\n",arr[1]); // 2
   //printf("%d\n",arr[arr[0]]);// arr[1] --> 2
   int i;
   printf("Elements of the array are \n");
   printArray(arr,5);
   //arr--> name of array --> base address
   printf("\n\n");
   for(i=0;i<5;i++)
   {
      printf("%d \n",arr[i]);
   }
}
void printArray(int *ptr,int size) //array notation
{
	int i;
	for(i=0;i<size;i++)
	{
	   printf("%d",ptr[i]);
	}
	for(i=0;i<size;i++)
```

```
    {
        ++ptr[i];
        // ptr[0]
        // ptr[0] ==> *(ptr+0)
        // *(100 + 0 )
        // ++*(100)==>++1
        //  2
        //after printing 2 3 4 5 6
    }
}
```

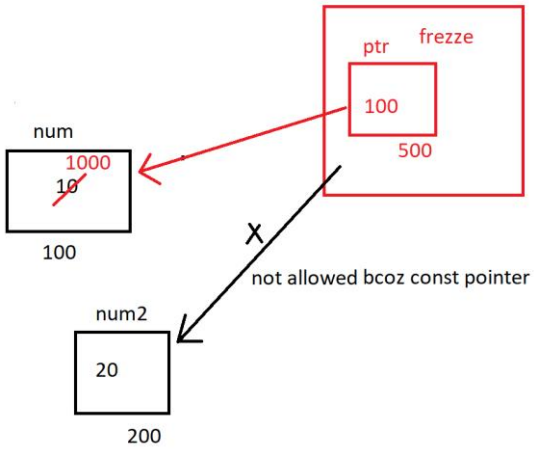
```
->Const Keyword Concept:--
#include<stdio.h>
// int main()
// {
//       int num = 10; // init
//       printf("%d",num); // 10
//       num = 100;
//       printf("%d",num); // 100
//    return 0;
// }

int main()
{
   // const --> type qualifier
   //const is  keyword.
     const int num = 10; // init
     printf("%d",num); // 10
     //num = 100; // NOT OK  bcoz of the value gets frezze.
     printf("%d",num); // 10
   return 0;
}
```



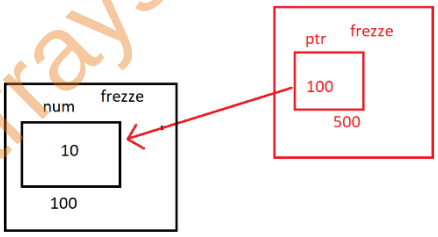
```
#include<stdio.h>
int main()
{
   const int num = 10;
   const int num2 =20;

   const int *ptr = &num;    //case 1
   // const int const *ptr = &num; //case 2
   //int const *ptr = &num;  // case3
   // * ke pahle const hai tho pointer constant nhi hai
   // value const hai theno case mai value hi const hai
   //ptr is non constant pointer pointing to constant integer variable

   printf("%d\n",num);// 10
   printf("%d\n",*ptr);// 10
   //*ptr = 100; NOT OK

   ptr = &num2;
   printf("%d\n",num2);// 20
   printf("%d\n",*ptr);// 20
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int num = 10;
   int * const ptr = &num;
   //* ke bad const hai tho pointer constant hai
   //int * ptr const = &num;
   //ptr = &num; // ERROR   agar apka pointer const hai tho uska
   //initialize vahi dena patata hai like int * const ptr = &num;
//ptr is constant pointer pointing to non constant int variable
   // int *ptr const = &num; this syntax is invalid
   printf("%d\n",num);//10
   printf("%d\n",*ptr);//10
//we can change the num through const pointer but we can cannot
//hold the anthoer num address.
   *ptr = 1000;
```

```
   printf("%d\n",num); // 1000
   printf("%d\n",*ptr);// 1000

   //int num2 = 20;
   //ptr = &num2; //ERROR
   return 0;
}
```



```
#include<stdio.h>

int main()
{
   const int num2 = 20;
   const int num = 10;
   const int * const ptr =&num; // allowed
   //const int const * const ptr =&num;
   //const int const * const ptr const =&num;//NOT OK
//ptr is constant pointer pointing to constant int variable

   printf("%d\n",num);// 10
   printf("%d\n",*ptr); //10
   //*ptr = 200; //NOT OK
   //ptr = &num2;//NOT OK
}
```



```
*/ Q.why arrays index start with 0 not 1
    1   2   3   4   5
   [1] [2] [3] [4] [5]
   100 104 108 112 116
   arr[1]=>1
   *(arr+1)
   *(100+1)
   *(104)
   2
   here we reached directly 104 not 100 i.e why 4 byte memory
   loss thats why we arrays index start with 0 not 1
*/
```

String

String

- String is character array terminated with '\0' character.
- '\0' is character with ASCII value = 0.

String input/output

- %s format specifer is used for string.
- char str[20];
- scanf("%s",str); /*user Input*/
- printf("%s",str); /*print Output*/
- gets(str); /*user Input*/
- puts(str); /*print Output*/
- scanf("%[^\n]", str); // scan whole line

```
#include<stdio.h>
int main()
{
  int i;
  // init list
  char str1[5]={'A','B','C','D','E'}; //this is not string bcoz not terminated by \0
  // 5 char array --> 5 bytes

  char str2[5] ={'a','b','c','d','\0'};
  // 5 char array terminated with \0 char = string

   char str3[5] ={'I','n','f','o'};
   // 5 char array terminated with \0 char=string
   //when array is init partially at the point of
   // declaration rest element are zero

   char str5[] ={ 'S','u','n','b','e','a','m'};
   // array of 7 chars ==> 7 bytes

  for(i=0;i<5;i++)
     putchar(str1[i]);
     // putchar --> to print single character
    for(i=0;str2[i]!='\0';i++)
     putchar(str2[i]);

 //print all char until \0 is encountered
   for(i=0;str3[i]!='\0';i++)
     putchar(str3[i]);
```

```
   for(i=0;i<7;i++)
     putchar(str5[i]);
  return 0;
}
```

```
#include<stdio.h>
int main()
{
   // using string literal
   int i;
   char str4[5]="Tech";
   // tech is string const/literal
   //short hand decl --> 'T' 'e' 'c' 'h' '\0'=> string

   char str6[] ="Aditya";
   // "Aditya " is a string literal/constant
   // 'A' 'd' 'i' 't' 'y' 'a' '\0'
   // 7 chars ==> ( 6 char + 1 '\0' )

   char str7[4]="Pune";
   // just a char array
   // str7 is not a string

   //print all char until \0 is encountered
   for(i=0;str4[i]!='\0';i++)
     putchar(str4[i]);

   for(i=0;str6[i]!='\0';i++)
     putchar(str6[i]);

   for(i=0;i<4;i++)
     putchar(str7[i]);
   return 0;
}
// char str1[5] ="Hell" ==> string
// char str2[4]="ABCD"--> char arrary
```

```
include<stdio.h>
int main()
{
   char str1[] ="";
   char str[10] ="";

   printf("%d\n",sizeof(str1)); //1
   printf("%d\n",sizeof(str)); //10
   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int i;
   char str[] = "Sunbeam Infotech";
     char name[20];
   // printing a string

   for(i=0;str[i]!='\0';i++)
    putchar(str[i]);   //Sunbeam Infotech

    printf("%s",str);   //Sunbeam Infotech
    puts(str);         // Sunbeam Infotech

  //  printf("Enter the name");   //ashok pate
  //  scanf("%s",name);
  //  printf("%s",name);        //ashok

   printf("Enter the name2");  //ashok pate
   scanf("%[^\n]", name);
   printf("%s",name);        //ashok pate
   return 0;
} note:- name1 after space not printing.
         name2 whole line print by using this format specifier %[^\n]
```

```
 #include<stdio.h>
int main()
{
   char name[80],city[40],job[60];
//while scanning string from user no need to use addressof operator
// because string name itself is based address of array
   printf("Enter the city");
   scanf("%s%*c",city);
//%s will read only upto white-space(space or tab or newline)
       // if i/p is newyork --> newyork
       // if i/p new york --> new
   //scan set

   printf("Enter the name");
   scanf("%[^\n]%*c",name);
   //%[^\n] will read upto \n
   // it can read alphabet,digit,space,tab,digit
   //scanf("%s",name);

   printf("Enter the job");
   //scanf("%s",job);
     gets(job);
// gets() will scan the string upto \n


 printf("city = %s\n Name=%s\n job=%s\n",city,name,job);
   return 0;
}
```

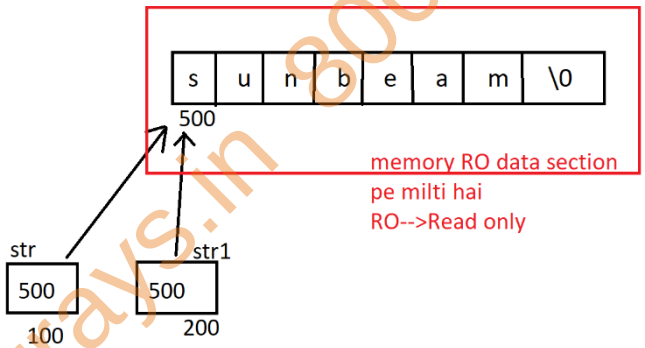
```
#include<stdio.h>
int main()
{
   char str[20];
   printf("Enter the name");
  //scanf("%s",str);
//    scanf("%[a-z]",str);   //only small case hi read krenga upto first capital
  //scanf("%[A-Z a-z]",str); //small case n upper case both work
  scanf("%[^a-z]",str);   //without a-z stop reading at a-z
   printf("%s",str);
scanf("%[0-9]",str); //only 0-9 numbers
   return 0;
}
```

```
/*
scanf("%[a-z]",str);
// get user input till char in range a-z(small case)
scanf("%[A-Z a-z]",str);
get user input till char in range a-z(small case)
get user input till char in range A-Z(upper case)
scanf("%[^a-z]",str);
// get user i/p until any char a-z is found
//(stop reading at a-z)
*/
```

```
#include<stdio.h>
#include<string.h>
int main()
{
   char str1[] ="Sunbeam Infotech";
//str1 is array of char storing 16 + 1 (\0)

   char *str2 = "Sunbeam Infotech";
   // str2 is a pointer to string constant

   char str3[] = "Sunbeam\0Infotech";
//str3 is array of char storing 16 + 1 (\0)

//sizeof()  is a compile time operator that counts number of bytes used

  printf("sizeof(str1)=%d\n",sizeof(str1)); //17
  printf("sizeof(str2)=%d\n",sizeof(str2));// 32 bit=>4 byte 64bits=>8byte
  printf("sizeof(str2)=%d\n",sizeof(str3));//17

//strlen() is a function that count number of chars
//until \0 is encountered
// strlen() --> length

printf("strlen(str1)=%d\n",strlen(str1)); //16
printf("strlen(str1)=%d\n",strlen(str2)); //16
printf("strlen(str1)=%d\n",strlen(str3)); //7

printf("%s\n",str1); //Sunbeam Infotech
printf("%s\n",str2); //Sunbeam Infotech
printf("%s\n",str3); //Sunbeam
}
```

```
#include<stdio.h>
int main()
{
   // char str1[20] = "Sunbeam";
   // char str2[20] = "Sunbeam";
   // // str1 --> name of array ( base address)
   // // str2 --> name of array ( base address )
   // if(str1==str2)
   //    printf("same");
   // else                  // ans:not same bcoz of base address compare
   //    printf("not same");
return 0;
}
```

```
#include<stdio.h>
int main()
{
char *str = "sunbeam";
   char *str1 = "sunbeam";
   // pointer to the string

   // printf("%u\n",&str);  //100 address
   // printf("%u\n",&str1);  //100 adress

   // printf("%u\n",str); //500 address of string
   // printf("%u\n",str1);  //500 address of string


   if(str==str1)
      printf("same");
   else                        //ans:same
      printf("not same");

    //    char *str = "Sunbeam";
   //    // RO data section
   //    // read only
   //     int i = 0;
   //   //printf("%s",str);
```



```
//     str[i] = 'A'; // runtime error boz memory allocated by read only data section.
//     // str[0] = 'A';
//     printf("%s",str);

       char str2[20] ="Sunbeam";
       //printf("%s",str2); // Sunbeam

       printf(str2); //sunbeam
       printf("\n");
       printf(str2+1);//unbeam
       printf("\n");
       printf(str2+2); //nbeam
       printf("\n");
       printf(str2+3); //beam
return 0;
}
```

C library have many string functions.

- They are declared in string.h
- strlen() – size_t strlen(const char *s);
- strcpy() – char* strcpy(char *dest, const char *src);
- strcat() – char* strcat(char *dest, const char *src);
- strcmp() – int strcmp(const char *s1, const char *s2);
- strcmpi() - int strcmpi(const char *s1, const char *s2);
- strchr() – char* strchr(const char *s, int ch);
- strstr() – char* strstr(const char *s1, const char *s2);
- strrev() – char* strrev(char *s);

```
// #include<stdio.h>
// #include<string.h>
// // typedef unsigned int --> size_t
// int main()
// {
//     char name[50] ="Reader";

//     printf("%d",strlen(name)); //6
//     return 0;
// }
```

->Now i want to write my own function

```
#include<stdio.h>
int mystrlen(const char *s);
int main()
{
   int len;
   char name[50]="Reader";
   len = mystrlen(name); //base address
   printf("%d",len);   //6
   return 0;
}
// R e a d e r \0
// user-defined function
int mystrlen(const char *s)//pointer notation
{
      int i = 0;

   // while(*(s+i)!='\0)
   while(s[i]!='\0')
   {
       i++;
   }
   return i;
}
```

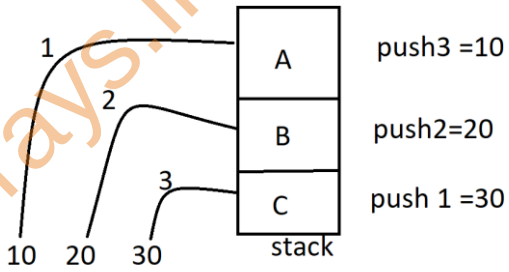
```
#include<stdio.h>
// int main()
// {
//     int num = 10;
//     int *ptr = &num;

//     printf("%d",*ptr); // 10;
// }

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

   int num = 10;
   void *ptr = &num; // point to int
   // void ==> generic pointer
   // void ==> dereferencing nhi hota you have to tell.
```

```
   // void --> nothing
   //printf("%d",*ptr); // NOT OK
    printf("%d\n",*(int*)ptr);  //10 bocz of void need to told.

   float fptr = 10.33;
   char ch ='A';
    ptr = &fptr;  // point to float
    printf("%f\n",*(float*)ptr);  //10.330000

   ptr = &ch;   // point to char
   printf("%c\n",*(char*)ptr);    // A

}
```

note://int *ptr;// uninitialize pointer Wild pointer hota hai.To avoid wild pointer assign as NULL

//int *ptr = NULL; //isliye agar pointer assign nhi kar rahe ho tho use null kardo null means 0 (void)

```
#include<stdio.h>
int main()
{
   int a =10;
   int b =20;
   int c =30;
   printf("%d %d %d",a,b,c);
   // printf --> right to left
   // calling convention --> cdecl-> c declarator
   // 10 20 30
   return 0;
}
```

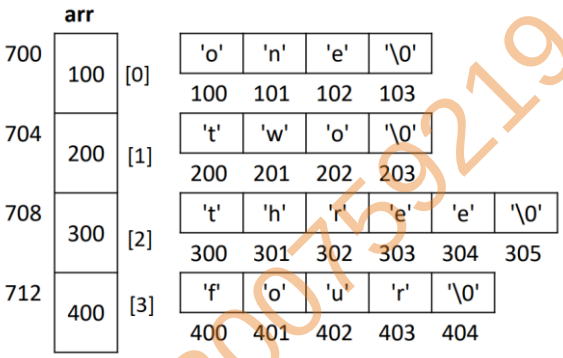


Null pointer

- If pointer is uninitialized, it will hold garbage address (local pointer variables).
- Accessing such pointer may produce unexpected results. Such pointers are sometimes referred as wild pointers.
- C defined a symbolic const NULL, that expands to (void*)0.
- It is good practice to keep well known address in pointer (instead of garbage).
- NULL is typically used to initialize pointer and/or assign once pointer is no more in use.
- Many C functions return NULL to represent failure.
- strchr(), strstr(), malloc(), fopen(), etc

Array of Pointer

```
#include<stdio.h>
// i/p a single digit number from user and print it nin words
int main()
{
   char* numbers[] ={"zero","One","two","three","Four","Five"};//32 bits 6*4 = 24
                                                              // 64bits 6*8=48
                //            0    1    2    3     4     5
   int num;
   printf("sizeof(numbers)=%d\n",sizeof(numbers));
   printf("Enter the single digit number");
   scanf("%d",&num); //0
   printf("%s\n",numbers[num]);
   //numbers[0]--> *(numbers + 0 )
   return 0;
}
```



- The name of exicutable(.\a.exe)is itself the first argument.
- What is used of this->with out using scanf we can give arguments to the function.
- used for console independent programming
- The last value of this argument is zero.This is system defined value.

```
#include<stdio.h>
int main(int argc, char const *argv[])
{
   //argv ==> argument vector
   printf("%d\n",argc);  //5
   //printf("%s\n",argv[0]); // name of executable .\a.exe
   //printf("%s\n",argv[1]); //  1
   int i;
   for(i=0; i < argc; i++)
       puts(argv[i]);    //.a\.exe 1 2 3 4
   return 0;
}
input:- .a\.exe 1 2 3 4
output:- 5
       .a\.exe 1 2 3 4
```

2D Array

```
#include<stdio.h>
int main()
{
   //     r  c
   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns

   /*
              matrix
          [0]    [1]     [2]
   [0]     1      2       3
          100    104     108

   [1]     4      5       6
          112    116     120

   [2]     7      8       9
          124    128     132
   */

   // array notation
    printf("%d",arr[0][0]); //1
    // 0th row and 0th col
    printf("%d",arr[0][1]);//2
    printf("%d",arr[0][2]);//3

    printf("%d",arr[1][0]);//4
    printf("%d",arr[1][1]);//5
    printf("%d",arr[1][2]);//6

    printf("%d",arr[2][0]);//7
    printf("%d",arr[2][1]);//8
    printf("%d",arr[2][2]);//9
   return 0;
}
```

```
#include<stdio.h>

int main()
{
   //     r  c
   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
   int j;
   /*
              matrix
          [0]     [1]      [2]
   [0]     1       2        3
          100     104      108

   [1]     4       5        6
          112     116      120

   [2]     7       8        9
          124     128      132
   */
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
       {
            printf("%4d",arr[i][j]);  //print array value
       }
       printf("\n");
    }

    printf("\n\n");
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
       {
            printf("%u ",&arr[i][j]);  //print array value address
       }
       printf("\n");
    }

   return 0;
}
```

```
#include<stdio.h>
//int arr[3][3];   // bcoz of golbal 0 0 0 0 0 0 0 0 0
int main()
{
   // int arr[3][3] = {1,2,3,4,5,6,7,8,9}; // 1   2   3   4   5   6 7   8   9
   //  3 rows and 3 columns
   // int arr[3][3] = {{1,2,3},{4,5,6},{7,8,9}};// 1   2   3   4   5 6 7   8   9
   //int arr[3][3] = {{1},{4,5},{7,8,9}};// 1 0 0 4 5 0 7 8 9
   int arr[3][3]; //garbage value
   // int arr[][3] = {1,2,3,4,5,6,7,8,9}; // 1   2   3   4   5   6 7   8   9
   //int arr[][3]; // NOT OK initialize karna row nhi dena tho
   // int arr[3][] = {1,2,3,4,5,6,7,8,9};  //not ok must enter columns complusury
   // int arr[][] = {1,2,3,4,5,6,7,8,9}; //not ok must enter columns
   int i;
   int j;
   /*
              matrix
         [0]      [1]     [2]
   [0]     1        2       3
         100     104     108

   [1]     4        5       6
         112     116     120

   [2]     7        8       9
         124     128     132
   */
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
       {
           printf("%4d",arr[i][j]);
       }
       printf("\n");
    }

       return 0;
}
```

```
#include<stdio.h>
int main()
{
   int arr[][4]={1,2,3,4,5,6,7,8,9};
   int r,c;
   /*
         [0]    [1]    [2]    [3]

    [0]   1      2      3      4

    [1]   5      6      7      8

    [2]   9      0      0      0
   */

   for(r=0;r<3;r++)
   {
         for(c=0;c<4;c++)
         {
            printf("%4d",arr[r][c]);    // 1 2 3 4 5 6 7 8 9 0 0 0
         }
         printf("\n");
   }
}
```

```
#include<stdio.h>
int main()
{
   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
      arr
               matrix
          [0]     [1]      [2]
   [0]     1       2        3
          100     104      108

   [1]     4       5        6                  //matrix form
          112     116      120

   [2]     7       8        9
          124     128      132
```

```
  int j;
          0                1                2
  | [0]     [1]  [2]   |  [0]    [1]   [2]    | [0]     [1]   [2]    |
  |  1       2     3    |   4     5      6     |  7       8     9     |
  | 100     104   108   |  112   116    120   |  124     128   132   |   //contiguous form
  | 100                 |  112                |  124                |
*/

   printf("\n\n");
   for(i=0;i<3;i++)
   {
      for(j=0;j<3;j++)
      {
          printf("%u ",&arr[i][j]);
      }
      printf("\n");
   }
   printf("\n");

   //printf("%u\n",&arr[0][0]); // base address
   //printf("%u\n",arr); // base address

    //printf("%u\n",arr+1); //
    /*
       arr + 1
       100 + 1*4*3
       100 + 1 * 12
       100 + 12
       112
    */

    //printf("%u\n",arr+2);
    /*
       arr + 2
       100 + 2*4*3
       100 + 2 * 12
       100 + 24
       124
    */
```

```
    printf("%u\n",*(arr+0)); //single deferincing
     /*
           *(arr+0)
           *(100+0)
           *(100)
            100
     */
     printf("%u\n",*(arr+1)); //single dereferincing
     /*
           *(arr+1)
           *(100+1*4*3)
           *(112)
            112
     */
     printf("%u\n",*(arr+2)); //single dereferincing
     /*
        *(arr + 2)
        *(100 + 2*4*3)
        *(100 + 2 * 12 )
        *(100 + 24)
        124
     */
   printf("%u\n",*(*(arr+0));  //1  Double-derefrencing
   printf("%u\n",*(*(arr+1));  //4  Double-derefrencing
   printf("%u\n",*(*(arr+2));  //7  Double-derefrencing

    // note:-In 2D arrays value tak pahuchne tak 2 bar derefrencing karna
   // padta hai single derefrincing kiya tho sirf address tak hi pahuchenge.
   return 0;
}
-----------------------------------------------------------------------------------------------------------
#include<stdio.h>

int main()
{

   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
   int j;
```

```
/*
      arr     //matrix form
                matrix
            [0]     [1]      [2]
   [0]       1       2        3
            100     104      108

   [1]       4       5        6
            112     116      120

   [2]       7       8        9
            124     128      132
   */
             0                1                2
   | [0]      [1]  [2]   |   [0]    [1]   [2]    | [0]      [1]   [2]    |
   |  1       2     3    |    4     5       6    |  7       8     9      |
   | 100     104   108   |  112    116    120    |  124     128   132    |   //contiguous form
   | 100                 |  112                  |  124                  |
 */
    printf("\n\n");
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
       {
            printf("%u ",&arr[i][j]);  //print address
       }
       printf("\n");
    }
    printf("\n");

    printf("%d\n",*(*(arr+0)));
     /*
            *(*(arr+0))
            *(*(100+0))
             *(*(100))
               *(100)
                 1
      */
    printf("%d\n",*(*(arr+1)));
     /*
            *(*(arr+1))
            *(*(100+1*4*3))
```

```
          *(*(100+12))
           *(112)
             4
 */
printf("%d\n",*(*(arr+2)));
 /*
        *(*(arr+2))
        *(*(100+2*4*3))
         *(*(100+24))
          *(124)
            7
 */
printf("%d\n",*(*(arr+0)+0));//1
 /*
        *(*(arr+0)+0)
        *(*(100+0)+0)
         *(*(100+0)
          *(100)
            1
 */
printf("%d\n",*(*(arr+0)+1));//2
 /*
        *(*(arr+0)+1)
        *(*(100+0)+1)
         *(*(100+1*4)
          *(104)
            2
 */
printf("%d\n",*(*(arr+0)+2));//3
 /*
        *(*(arr+0)+2)
        *(*(100+0)+2)
         *(*(100+2*4)
          *(108)
            3
 */
printf("%d\n",*(*(arr+1)+0)); //4
  /*
        *(*(arr+1)+0)
        *(*(100+1*4*3)+0)
         *(*(100+12)+0)
          *(112+0)
```

```
                4
     */
    printf("%d\n",*(*(arr+1)+1)); //5
    /*
          *(*(arr+1)+1)
          *(*(100+1*4*3)+1)
           *(*(100+12)+1*4)
            *(112+4)
            *(116)
              5
     */
    printf("%d\n",*(*(arr+1)+2)); //6
     /*
          *(*(arr+1)+2)
          *(*(100+1*4*3)+2)
           *(*(100+12)+2*4)
            *(112+8)
            *(120)
              6
     */
      printf("%d\n",*(*(arr+2)+0)); //7  as above method
      printf("%d\n",*(*(arr+2)+1)); //8  as above method
      printf("%d\n",*(*(arr+2)+2)); //9  as above method

      printf("%d\n",arr[2][2]); // array notation
      printf("%d\n",*(*(arr+2)+2));// pointer notation

   return 0;
}
```

```
#include<stdio.h>

int main()
{

   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
   int j;
```

```
/*
      arr
                matrix
              [0]    [1]     [2]
   [0]         1      2       3
             100    104     108

   [1]         4      5       6
             112    116     120

   [2]         7      8       9
             124    128     132
   */

    printf("\n\n");
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
       {
            printf("%u ",&arr[i][j]);
       }
       printf("\n");
    }
    printf("\n");


   //printf("%u\n",*arr); // base address 100
   //printf("%u\n",*arr+1);
   //*arr+1
   // 100 + 1
   // 100 + 1 * 4
   // 100 + 4 ==> 104

   //printf("%u\n",*arr+2);
   // 100 + 2
   // 100 + 2 * 4
   // 100 + 8 ==> 108

   //printf("%d\n",**arr); //1
   //*arr+1
   // 100 + 1
   // 100 + 1 * 4
```

```
   // 100 + 4 ==> *104==> 1

   //printf("%d\n",*(*arr+1));//2
    //*arr+2
   // 100 + 2
   // 100 + 2 * 4
   // 100 + 8 ==> *108 => 2

   //printf("%d\n",*(*arr+2));//3
    //*arr+3
   // 100 + 3
   // 100 + 3 * 4
   // 100 + 12 ==> *112 => 3

     printf("%d\n",**arr+1); //2
   /* = **arr + 1
      =   1  + 1
      =   2
   */
     printf("%d\n",*(*arr+1)+2);//4
     /* = *(*arr+1) + 2
      =   2  + 2
      =   4
   */
     printf("%d\n",*(*arr+2)+3);//6
     /* = *(*arr+2) + 3
      =   3  + 3
      =   6
   */


   return 0;
}
```

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

   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
   int j;
```

```
/*
   arr
            matrix
         [0]    [1]      [2]
[0]       1      2        3
        100    104      108

[1]       4      5        6
        112    116      120

[2]       7      8        9
        124    128      132
*/

 printf("\n\n");
 for(i=0;i<3;i++)
 {
    for(j=0;j<3;j++)
    {
        printf("%u ",&arr[i][j]);
    }
    printf("\n");
 }
 printf("\n");

 int *ptr[3] ={arr,arr+1,arr+2};

printf("%u\n",ptr[0]);  //address 100
printf("%u\n",ptr[1]);  //       112
printf("%u\n\n",ptr[2]);  //     124

   printf("%d\n",*ptr[0]);  // 1
   printf("%d\n",*ptr[1]);  // 4
   printf("%d\n\n",*ptr[2]);   //7

   printf("%d\n",*(*(ptr+0)));  //1
   printf("%d\n",*(*(ptr+1)));  //4
   printf("%d\n\n",*(*(ptr+2))); //7
```

```
    //pointer notation
      printf("%d\n",ptr[0][0]); //1   array notation
      printf("%d\n",*(*(ptr+0)+0));//1
      printf("%d\n",*(*(ptr+0)+1));//2
      printf("%d\n\n",*(*(ptr+0)+2));//3

      /*
      printf("%d\n",ptr[0][0]); //array notation
      printf("%d\n",*(*(ptr+0)+0));// pointer notation
      */

      printf("%d\n",ptr[1][0]); //4 array notation
      printf("%d\n",*(*(ptr+1)+0));//4
      printf("%d\n",*(*(ptr+1)+1));//5
      printf("%d\n",*(*(ptr+1)+2));//6

   return 0;
}
```

```
#include<stdio.h>
int main()
{
   int arr[3][3] = {1,2,3,4,5,6,7,8,9};
   //  3 rows and 3 columns
   int i;
   int j;
   /*
      arr
                matrix
            [0]    [1]      [2]
   [0]       1      2        3
           100    104      108

   [1]       4        5       6
           112    116      120

   [2]       7       8        9
           124    128      132
   */

    printf("\n\n");
    for(i=0;i<3;i++)
```

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

    int *ptr[3] ={arr,(int*)arr+1,(int*)arr+2};
   //          100    104      108
   //Explicitly we told take scale faactor of 4 bytes
    printf("%u\n",ptr[0]); //100
    printf("%u\n",ptr[1]); //104
    printf("%u\n",ptr[2]); //108

      printf("%d\n",*ptr[0]); //1
      printf("%d\n",*ptr[1]);//2
      printf("%d\n",*ptr[2]); //3

      printf("%d\n",*(*(ptr+0)));//1
      printf("%d\n",*(*(ptr+1)));//2
      printf("%d\n",*(*(ptr+2))); //3

      //pointer notation
      printf("%d\n",ptr[0][0]); //1 array notation
      printf("%d\n",*(*(ptr+0)+0));//1
      printf("%d\n",*(*(ptr+0)+1));//2
      printf("%d\n",*(*(ptr+0)+2));//3

      /*
            printf("%d\n",ptr[0][0]); //array notation
            printf("%d\n",*(*(ptr+0)+0)); // pointer notation
      */

      printf("%d\n",ptr[1][0]); //2 array notation
      printf("%d\n",*(*(ptr+1)+0));//2
      printf("%d\n",*(*(ptr+1)+1));//3
      printf("%d\n",*(*(ptr+1)+2));//4

   return 0;
}
```

```
#include<stdio.h>
void readArray(int arr[3][3],int row,int col);
void printArray(int arr[3][3],int row,int col);
int main()
{
   int arr[3][3];
   printf("%d\n",sizeof(arr));  // 36 byte   12+12+12=36
   printf("%d\n",sizeof(arr[0][0])); //4 byte
   readArray(arr,3,3);
   printArray(arr,3,3);
   return 0;
}

void printArray(int arr[3][3],int row,int col)
{
      int r,c;
      for(r=0;r<row;r++)
      {
         for(c=0;c<col;c++)
         {
             printf("%4d",arr[r][c]);
         }
         printf("\n");
      }
}
void readArray(int arr[3][3],int row,int col)
{
      int r,c;
      for(r=0;r<row;r++)
      {
         for(c=0;c<col;c++)
         {
             scanf("%d",&arr[r][c]);
         }
      }

}
```

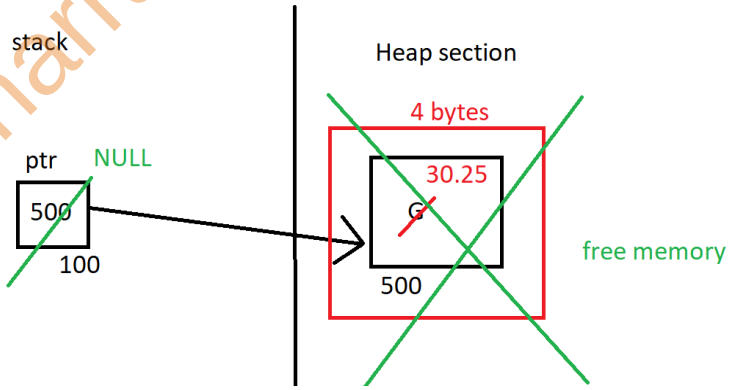
Dynamic memory allocation

• Dynamic memory allocation allow allocation of memory at runtime as per requirement.
• This memory is allocated at runtime on Heap section of process.
• Library functions used for Dynamic memory allocation are
• malloc() – allocated memory contains garbage values.
• calloc() – allocated memory contains zero values.
• realloc() – allocated memory block can be resized (grow or shrink).
•The declaration of this three library is in <stdlib.h> header files.
• All these function returns base address of allocated block as void*.
• If function fails, it returns NULL pointer

---

1)Malloc->

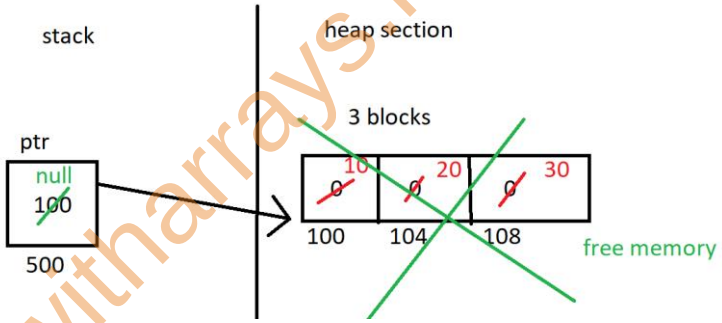
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  // void *malloc(size_t size);
    //why void->it is  genric pointer karke rakha kyuki
   //pata nhi user int,float,char kya used krena isliye.

   //malloc will return me the address
   float *ptr;

   ptr = (float*)malloc(sizeof(float));
   //request to the memory

   if(ptr==NULL)
   {
     printf("Unable to allocate memory\n");
     return 0;
   }

   *ptr = 30.25;
   printf("The value within the block %f",*ptr);
   free(ptr); // To avoid memory leakage
   ptr = NULL; // To avoid dangling pointer

}
```



---

2) Calloc

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

int main()
{
   // void*malloc(size_t size);
   // void* calloc(size_t n,size_t size);
   // n --> no of blocks.
    // calloc mai initial value 0 hoti hai
    // malloc mai initial value garbage hoti hai.
    //both syntax will be changed
   int *ptr;

   ptr = (int*)calloc(3,sizeof(int));
   if(ptr==NULL)
   {
     printf("Unable to allocate memory\n");
     return 0;
   }
   int i;
   for(i=0;i<=2;i++)
      *(ptr+i) = 10 * (i+1);

    for(i=0;i<=2;i++)
      printf("%d\n",*(ptr+i));

   free(ptr); // TO avoid memory leakage

   ptr = NULL; // TO avoid dangling pointer
   return 0;
}
```

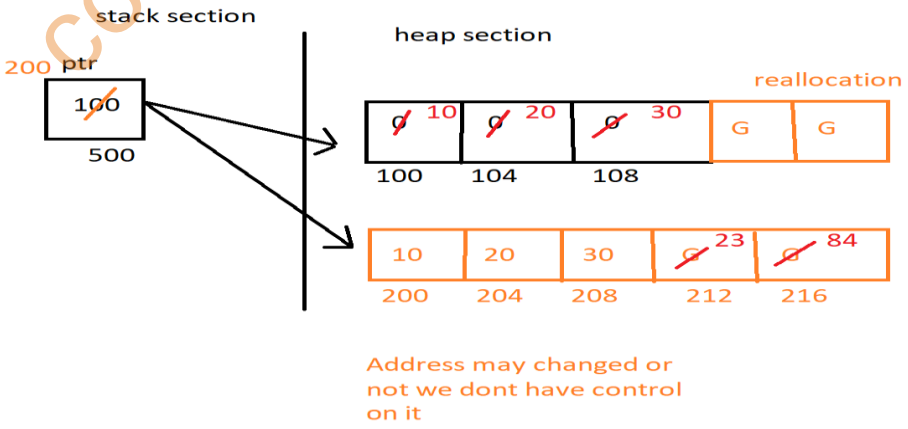


---

3)Realloc

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

int main()
{
```

```
   // firse memory allocation karna hai tab realloc use hota hai
   // suppose first 3 block memory allocated after that i want agin
   // extra 2 block memory then i used realloc function.
   // realloc mai initial value garbage hoti hai.

   // void realloc(void *ptr,size_t size);
   // void*malloc(size_t size);
   // void* calloc(size_t n,size_t size);
   int *ptr;

   ptr = (int*)calloc(3,sizeof(int));
   if(ptr==NULL)
   {
     printf("Unable to allocate memory\n");
     return 0;
   }
   int i;
   for(i=0;i<=2;i++)
      *(ptr+i) = 10 * (i+1);

   ptr = (int*)realloc(ptr,5 * sizeof(int));

   ptr[3] = 23;
   ptr[4] = 84;

   for(i=0;i<=4;i++)
      printf("%d\n",*(ptr+i));

   free(ptr);
   ptr = NULL;
   return 0;
}
```



Memory leakage

• If memory is allocated dynamically, but not released is said to be "memory leakage".
• Such memory is not used by OS or any other application as well, so it is wasted.
• In modern OS, leaked memory gets auto released when program is terminated.
• However for long running programs (like web-servers) this memory is not freed.
• More memory leakage reduce available memory size in the system, and thus slow down whole system.
• In Linux, valgrind tool can be used to detect memory leakage.

Dangling pointer

•Pointer keeping address of memory that is not valid for the application, is said to be "dangling pointer".
• Any read/write operation on this may abort the application. In Linux it is referred as "Segmentation Fault".
• Examples of dangling pointers
• After releasing dynamically allocated memory, pointer still keeping the old address.
• Uninitialized (local) pointer
• Pointer holding address of local variable returned from the function.
• It is advised to assign NULL to the pointer instead of keeping it dangling.

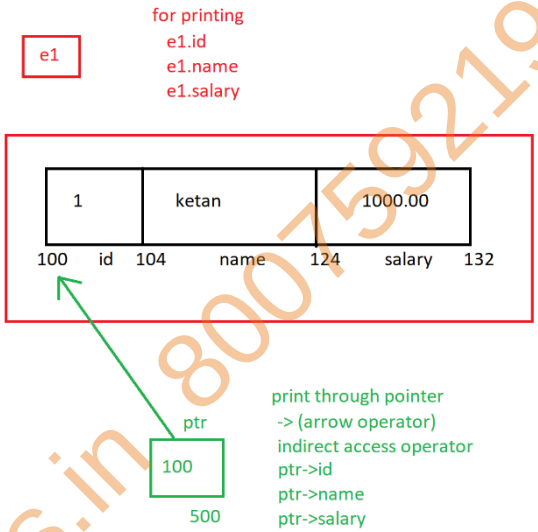
Structure

- Structure is a user-defined data type.
- Structure stores logically related (similar or non-similar) elements in contiguous memory location.
- Structure members can be accessed using "." operator via struct variable.
- Structure members can be accessed using "->" operator via struct pointer.
- Size of struct = Sum of sizes of struct members.
- If struct variable initialized partially at its point of declaration, remaining elements are initialized to zero.

```
#include<stdio.h>
int main()
{   //structure --> user defined data type
   // without struct how we doing
    //name , id , salary
    char name[32];
    int empid;
    float salary;

    printf("Name    :   ");
    scanf("%s",name);
    printf("empid    :   ");
    scanf("%d",&empid);
    printf("salary    :   ");
    scanf("%f",&salary);

    printf("%s %d %f",name,empid,salary);
    return 0;
}
```

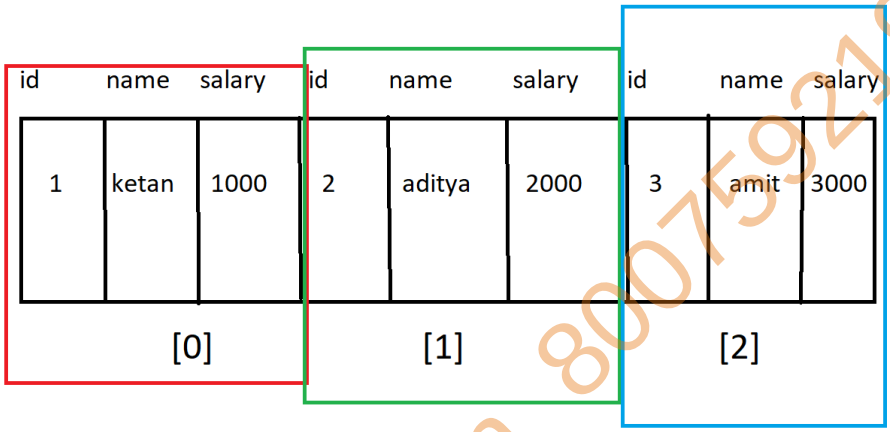
```
#include<stdio.h>
#include<string.h>
// user-defined datatype
// created my datatype
// blue-print hai isko memory nhi milte.
// global structure
```

```
struct emp
{
   int empid; // 4 bytes
   char name[20];// 20 bytes
   double salary; // 8 bytes
};
// we can create the variable of structure inside any user-defined global function
int main()
{
   // local structure -> we can create the variable structure only inside main
   struct student
   {
      int rollnumber; // 4 bytes
      char name[20]; // 20 bytes
   };

   struct emp e1 = { 1,"Ketan",1000.00};
   // e1 --> variable / object
   struct emp e2 = { 2,"Amit"};
   struct emp e3; // only declaration
   struct emp e4; // only declaration
   // e1,e2,e3,e4 are variables of structure emp
   printf("%d %s %f",e1.empid,e1.name,e1.salary);

   struct emp *ptr = &e1;
   printf("%d %s %f",ptr->empid,ptr->name,ptr->salary);
   // i am printing the structure members using pointer
   // use -> operator ( arrow operator )

   e3.empid = 2;
   strcpy(e3.name,"Aditya");
   e3.salary = 1000.00;
   printf("%d %s %f",e3.empid,e3.name,e3.salary);

   //struct emp e4; //==>  id name salary
   printf("Enter empid name and salary: ");
   scanf("%d%s%lf",&e4.empid,e4.name,&e4.salary);
   printf("%d %s %lf",e4.empid,e4.name,e4.salary);

   return 0;
}
```



```
#include<stdio.h>

// structure declaration
struct emp
{
   int empid; // 4  bytes
   char name[20]; // 20 bytes
   double salary; // 8 bytes
};
void accept_emp( struct emp *p);
void print_emp(struct emp e);
int main()
{
   struct emp e1;
   accept_emp(&e1); // pass by address / call by address
   print_emp(e1); // pass by value / call by value
   return 0;
}

void accept_emp( struct emp *p)
{
      printf("Enter the empid");
      scanf("%d",&p->empid);

      printf("Enter the name");
      scanf("%s",p->name);

      printf("Enter the salary");
      scanf("%lf",&p->salary);
}
void print_emp(struct emp e)
{
   printf("%d %s %.2lf",e.empid,e.name,e.salary);
}
```

```
#include<stdio.h>

// structure declaration
struct emp
{
   int empid; // 4  bytes
   char name[20]; // 20 bytes
   double salary; // 8 bytes
};

int main()
{
     // int arr[3] = {1,2,3};
   // struct emp e1;
   // struct emp e2;
   // struct emp e3;
   // struct emp e4;
   // struct emp e5;
```

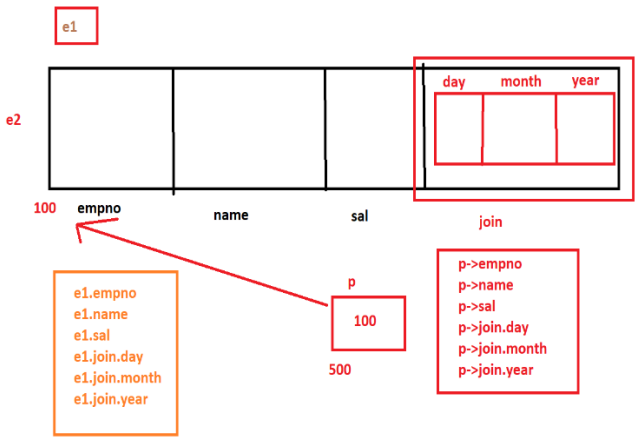



```
   // array of structure
   struct emp arr[3]={
                {1,"Ketan",1000.00},
                {2,"Aditya",2000.00},
                {3,"Amit",3000.00}
   };

    //arr[0] ==> id name salary
    //arr[1]==> id name salary
    //arr[2]==> id name salary

      int i;

      for(i=0;i<3;i++)
      {
         printf("%d %s %.2lf\n",arr[i].empid,arr[i].name,arr[i].salary);
      }
    return 0;
}
```

structure arrays with user defined value(scanf)

```
#include<stdio.h>
struct emp // global structure
{
   int empno; //4
   char name[20];//20
   double sal;// 8
};
void accept_emp(struct emp a[], int n);
void display_emp(struct emp a[], int n);
int main()
{
   struct emp arr[3];
   //array of structure
   accept_emp(arr,3);
   display_emp(arr,3);
}
void display_emp(struct emp a[], int n)
{
      int i;

      for(i=0;i<n;i++)
      {
         printf("%d %s %lf\n",a[i].empno,a[i].name,a[i].sal);
      }
}
void accept_emp(struct emp a[], int n)
{
      int i;
      for(i=0;i<n;i++)
      {
         printf("Enter empid name and sal");
         scanf("%d%s%lf",&a[i].empno,a[i].name,&a[i].sal);
      }
}
```

```
#include<stdio.h>

struct date
{
   int day; //4
   int month; //4
   int year; // 4
};
struct emp
{
   int empno; //4
   char name[20]; // 20
   double sal; // 8
   struct date join; //12
};
int main()
{
   struct emp e1 = {1,"Ketan",2000.00,{1,1,2000}};
   struct emp e2;
printf("%d %s %lf, %d-%d-%d",
                    e1.empno,e1.name,e1.sal,e1.join.day,e1.join.month,e1.join.year);

printf("Enter empno name and sal, Joindate(day,month,year");
scanf("%d%s%lf%d%d%d",&e2.empno,e2.name,&e2.sal,&e2.join.day,&e2.join.month,&e2.j
oin.year);

printf("%d %s %lf, %d-%d-%d\n",
                              e2.empno,e2.name,e2.sal,e2.join.day,e2.join.month,e2.join.year);
   struct emp *p;  y
   p = &e2;

   printf("%d %s %lf, %d-%d-%d\n",p->empno,p->name,p->sal,p->join.day,p->join.month,p-
>join.year);

   return 0;
}
```



```
#include<stdio.h>
struct book
{
   int id; //4
   char name[20];//20
   char author[20];// 20
};

int main()
{
  struct book b1 = {1,"Cprogram","Ritchie"};
  struct book b2;

  // int a = 10
  // int b;
  // b = a

  b2 = b1;
  printf("%d %s %s",b2.id,b2.name,b2.author);

   // // relational operator cannot be used
   // if(b2==b1)
   //    printf("Same");
   // else
   //    printf("Not same");

}
```

Struct padding or Slack bytes

- For efficient access compiler may add hidden bytes into the struct called as "struct padding" or "slack bytes".
- On x86 architecture compiler add slack bytes to make struct size multiple of 4 bytes (word size).
- These slack bytes not meant to be accessed by the program.
- Programmer may choose to turn off this feature by using #pragma.
- #pragma pack(1)

Structure size important concept->>

Example-1

```
#include<stdio.h>
#pragma pack(1)
struct test
{
   int a; // 4
   char b;// 1
};

int main()
{
   printf("%d",sizeof(struct test));
                  //8 without pragma->for efficient complier give size of multiple 4
                  // 5 with pragma
   return 0;
}
```

Example-2

```
#include<stdio.h>
#pragma pack(1)
struct test
{
   int a; // 4
   char b;// 1
   int c; // 4
};

int main()
{
   printf("%d",sizeof(struct test));
                //12 without pragma->for efficient complier give size of multiple 4
                //9 with pragma
   return 0;
}
```

Example-3

```
#include<stdio.h>
// #pragma pack(1)

struct test
{
   int a; // 4
   char b; // 1
   double d1; // 8
   int c;  // 4
};

int main()
{
   printf("%d",sizeof(struct test));
        //24 without pragma->for efficient complier give size of multiple 8.
       //it will take larger size of data type i.e after 17 he will take 8 byte size and became 24
       //17 with pragma
   return 0;
}
```

Example-4

```
#include<stdio.h>
#pragma pack(1)
struct test
{
   int a; // 4
   char b; // 1
   double d1; // 8
   int c;  // 4
   long L;  //4
};
int main()
{
   printf("%d",sizeof(struct test));
                //24 without pragma->for efficient complier give size of multiple 4
               // 21 with pragma
   return 0;
}
```

Union

- Union is user defined data-type.
- Like struct it is collection of similar or non-similar data elements.
- All members of union share same memory space i.e. modification of an member can affect others too.
- Size of union = Size of largest element
- When union is initialized at declaration, the first member is initialized.
- Application:
- System programming: to simulate register sharing in the hardware.
- Application programming: to use single member of union as per requirement

---

```
#include <stdio.h>
#include <stdlib.h>
// union test
// {
//       int a;
//       float b;
//       double c;

// };

// int main(void)
// {
//       union test t1;

//       printf("%d",sizeof(t1)); // 8 size of union= size of largest element
//       return 0;
// }
```

```
#include<stdio.h>
union testUnion
{
short int num;
char ch[2];
};
// //       ch[2]
// //
// // 00001000  00000100
// //      2 byte
// //        num
```

```
int main()
{
union testUnion ut;
ut.ch[0]=4;
ut.ch[1]=8;
printf("%d", ut.num); //2052
return 0;
}
/*
        4-->  00000100
        8-->  00001000
       2048 1024 512 256 128 64 32 16 8  4 2 1
0000  1      0     0    0     0    0  0   0  0  1 0 0 // 2052
*/
```

```
#include<stdio.h>
#pragma pack(1)
struct
{
        short s[5]; // 5 * 2 = 10
        union
        {
                char x; //1
                float y;  //4
                long z;    //4
                short int z1; //2
        }u;// 4 bytes
}t;
int main(void) {

        printf("\n Size of Structure =%u",sizeof(t)); // 14 with pragma  16 without pragma
        printf("\n Size of Union =%u",sizeof(t.u)); // 4
        printf("\n Size of Structure + union=%d",sizeof(t)+sizeof(t.u));
   //                              14    +   4 --> 18  with pragma
                                    16     +  4 -- > 20 without pragma
        return 0;
}
```

```
#include<stdio.h>
/*
   standard 1 to 4 --> Grade('A','B'...)
   standard 5 to 10--> marks(60,70,...)
*/

struct student
{
      int roll; // 4
      char name[20]; //20
      int std;  // 4

      union
      {
         char grade;
         int marks;
      }result; // 4
};

int main()
{
   struct student s1;
   printf("Enter roll name and std");
   scanf("%d%s%d%*c",&s1.roll,s1.name,&s1.std);

   printf("roll = %d name = %s std = %d",s1.roll,s1.name,s1.std);
   if(s1.std<=4)
   {
         printf("Enter the grade");
         s1.result.grade= getchar();
         printf("Grade = %c\n",s1.result.grade);
   }
   else
   {
         printf("\nEnter the Marks");
         scanf("%d",&s1.result.marks);
         printf("Marks : %d\n",s1.result.marks);
   }

   return 0;
}
```

File IO
• File is collection of data and information on storage device.
• Each file have data (contents) and metadata(information).
• File IO can enable read/write file data.
• File Input Output
• Low Level File IO
• Explicit Buffer Management. Use File Handle.
• High Level File IO
• Auto Buffer Management. Use File Pointer.
• Formatted (Text) IO
• fprintf(), fscanf()
• Unformatted (Text) IO
• fgetc(), fputc(), fgets(), fputs()
• Binary File IO
• fread(), fwrite()
Sunbeam Infotech www.sunbeaminfo.com

High Level File IO
• File must be opened before read/write operation and closed after operation is completed.
• FILE * fp = fopen("filepath", "mode"); – to open the file
• File open modes:
• w: open file for write. If exists truncate. If not exists create.
• r: open file for read. If not exists, function fails.
• a: open file for append (write at the end). If not exists create.
• w+: Same as "w" + read operation.
• r+: Same as "r" + write operation.
• a+: Same as "a" + append (write at the end) operation.
• Return FILE* when opened successfully, otherwise return NULL.
• fclose(fp);
• Close file and release resources.
Sunbeam Infotech www.sunbeaminfo.com

File IO
• Character IO • fgetc() • fputc() • String (Line) IO
• fgets() • fputs() • Formatted IO • fscanf() • fprintf()
• Binary (record) IO • fread() • fwrite() • File position
• fseek() • ftell()

Passing arguments: Call by value vs Call by address/reference

Call by value

- Formal argument is of same type as of actual argument.
- Actual argument is copied into formal argument.
- Any change in formal argument does not reflect in actual argument.
- Creating copy of argument need more space as well as time (for bigger types).
- Most of data types can be passed by value – primitive & user defined types.

Call by address

- Formal argument is of pointer type (of actual argument type).
- Address of actual argument is collected in formal argument.
- Actual argument can be modified using formal argument.
- To collect address only need pointer. Pointer size is same irrespective of data type.
- Array and Functions can be passed by address only

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/cceesept2023

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project