Arrays

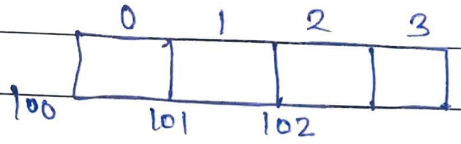
a[i] = i[a]

Date
Page No

One-dimensional array :- Array stored contiguously in memory.

Random access is possible.

Index started with 0.



A[i] = (100 + i) = (i*2) + 100 (for 2 bit)

Index started with 1.

A[i] = (100 + (i-1)) =

= (i-1)*2 + 100 (for 2 bit)

Row major and column major order in 2D :-
Looks different but stored in m/m differently.

Ex :-

A : 3×4 (Index start with 0)

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23

(Index start with 1)

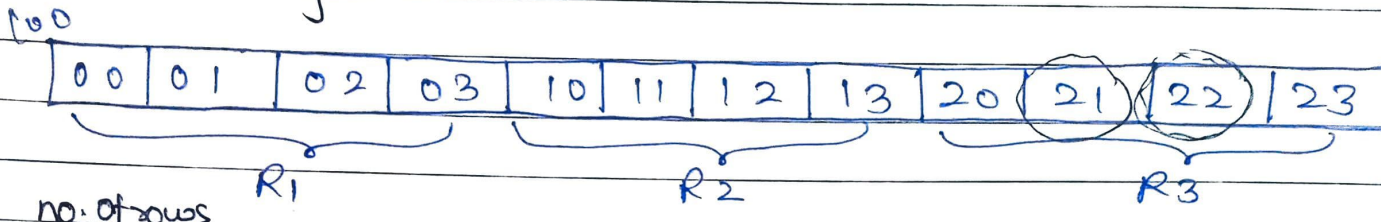
	1	2	3	4
1	11	12	13	14
2	21	22	23	24
3	31	32	33	34

To access the array randomly we can't store it in the same way. To store it into contiguous manner we need to convert it into 1-D representation using two approaches

① Row major
② Column major

Date :
Page No.

Row major order –



no. of rows to cross

$A[2][1] = ((2*4)+1) + 100$

Element size = 1 byte

no of colum to cross — elements to cross

If size of element is given multiply it with elements to cross

Generally

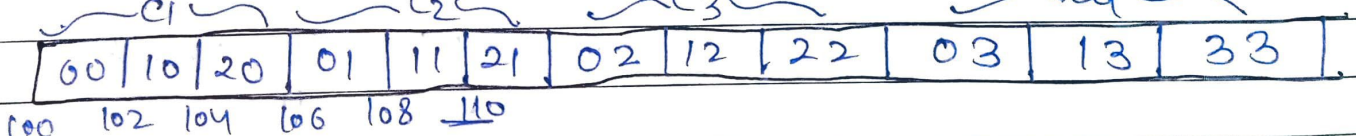
A: m x n then,

$A[i][j] = ((i*m)+j) * size + Base$

If index start with 1.

$A[i][j] : ((i-1)*n) + (j-1)) * size + Base$

Column major order



$A[2][1] = ((1*3)+2) * size + Base$

size = 2 Byte , Base = 100

= 110

→ index from 0.

m: m

$A[i][j] = ((j*m)+i) * size + Base$

If index start with 1 replace

$j \rightarrow (j-1)$, $i \rightarrow (i-1)$.

Program

Determine whether two arrays a and b have an element in common.

```
int isCommon (int a[], int b[], int n, int m) {
    int i, j;
for (i=0 ; i<n; i++) {
  for (j=0 ; j<m ; j++) {
   if ( a[i] == b[j])
      return 1;
  }
}

  return 0;
}
```

A: array [1---10][1---15]

Base = 100

address of element A[i][j] = ?

Gate (1998)

a) 15i + j + 84
b) 10i + j + 89
c) 15j + i + 84
d) 10j + i + 89

Row major order

$A[i][j] = ((i-1)*15 + j-1)*S + 100$] take size = 1

$= 15i + j + 84$ (a)

Q Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in RMO & CMO in contiguous memory locations. The time complexity of an "algorithm" to compute $M_1 \times M_2$ will be

Gate: 2004

a) Best if A is in RMO and B is in CMO
b) Best if both are in RMO
c) Best ——— CMO
d) Independent of storage scheme

Solution :-

Any element can be access in uniform time

Algorithm is independent of storage scheme because availability of element is same.

If Algorithm replaced by implemented then,

Row [→] [↓]

my (A) column M_2 (B)

हमें M_1 की Row और M_2 की Column से Multiply करना होता है so,

Store A in Row major and Store B in Colum major is better.

Additional info. regarding previous modules

Date :
Page No.

Imp. terms

Overloading → defining a set of similar functions.

friend → used to give a non-member function access to the private parts of an object

Constructor → a function which is automatically called when an object is created.

Protected → allows a derived class to have a access to the private parts of the base class

this → a pointer to the object associated with the current function.

Inheritence → allows to define a class have properties of another class.

Strings

i) Strings operation - <string.h>

Strcat (s,t) → Concatenate t to the end of s

Strncat (s,t,n) → concatenate n characters of t to th end of s.

Strcmp (s,t) → return negative, zero or positive for s<t, s==t, s>t

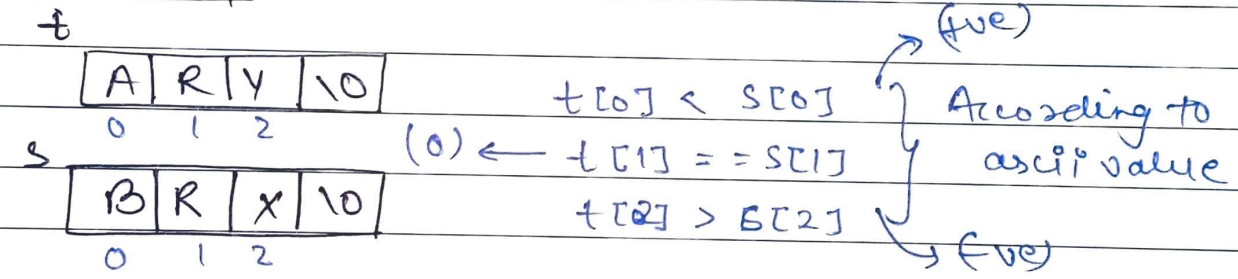
strncmp (s,t,n) → same as strcmp but only in first n charaters.

Strcpy (s,t) → copy to s from t

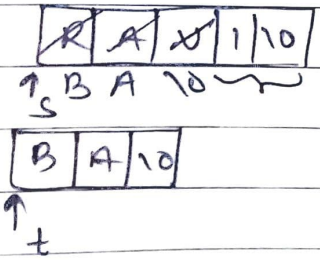
Strncpy (s,t,n) → copy at most n charater of t to s

Strrev (s) → Reverse the string.

In strcmp



In strcpy



Space in s must be greater than size of t.

Strlen (s) → return the length of s.

Strchr (s,c) → return pointer to first c in s or NULL if not present

Date :
Page No.

Ex strchr (S, 'A')

R	A	V	\0

↑ → Return the address of A.

Strchr (S, C) → Return the pointer to last character in S or NULL if not present.

A	R	V	A	N	\0

S ↑ → return address of last one.

Char strchr (S, A)

C program using strcpy().

Strcpy()

```
Char *Strcpy (char *s, Char *t) {
    Int i;
    i=0;
while ((s[i] = t[i]) != '\0')
    i++ ;
  return s;   # s is pointing the beggining of string.
}
```

without i" ...

```
while ((*s = *t) != '\0') {
    s++ ;
    t++ ; }
```

→ while ((*s++ = *t++) != '\0');

Write a C program using strcat().

```
Void strcat (char s[], char t[]) {
    int i, j;
    i = j = 0;
  while (s[i] != '\0')  /* find end of s */
        i++;
  while ((s[i++] = t[j++]) != '\0');
}
```

Write a C Program using strcmp().

```
int strcmp (char *s, char *t) {
for (; *s == *t; s++, t++)
    If (*s == '\0')
        return 0;
return *s - *t;
}
```

Ascii value of '\0' → NULL

Write C program to count characters,

```
# include <stdio.h>
void main() {
    int nc;
  for (nc=0; getchar() != EOF ; ++nc);
    printf ("%d", nc);
}
```

EOF can be perform using 'Ctrl+C'

Date :
Page No.

Reverse String :- strrev()

```
# include <string.h>
void reverse (char *s) {
  int c, i, j;                    (4-1)
for (i=0; j= strlen(s)-1 ; i<j ; i++, j--){
        c = s[i];
        s[i] = s[j];          } # swapping logic
        s[j] = c ;
    }
}
```

Ex

0	1	2	3	4
R	A	V	i	\0

↑ i ↑ j

→

0	1	2	3	4
i	V	A	R	\0

Imp Prog.

i to a : Convert int n to characters in string s

```
void itoa (int n, char s[]) {
   int i, sign ;
if ((sign = n)<0)
    n = -n;
 i = 0;
do{
 s[i++] = n%10+'0';   # Convert the number into character.
} while ((n/=10)>0);
if (sign <0)
  s[i++] = '-' ;
  s[i] = '\0';
  reverse(s);
}
```

Imp. Concept

Ex → n = -123

sign = -123
n = 123

3	2	1	

3	2	1	-	
3	2	1	-	\0

↳ | - | 1 | 2 | 3 | \0 | as a string.

atoi
itoa } → Sir, said predefined function in C.

May Check

Date :
Page No.

atoi: convert string to an integer.

```
int atoi (char s[]){
    int i, n;
    n=0;
for(i=0; s[i]>='0' && s[i]<='9'; ++i)
    n = 10*n + (s[i] - '0');
    return n;
}
```

n = 0+1 = 1
n = 10+2 = 12
n = 12x10+3 = 123

Ex

1	2	3	\0

↓ Not 1 but ascii value of 1 is saved here.

Write the output for following program-1.

Q.1
```
#include <stdio.h>
void main(){
    int n;
for( n=3 ; n!=0; n--)
printf ("n=%d", n--);
```

Range → $-2^{31} - 0 - 2^{31}-1$
for 32 bit compiler
It may vary for hardware.

O/p n = 3
n = 1
n = -1 and so on → untill range ends.
because we already skipped zero

Q.2
```
#include <stdio.h>
# define scanf "%s ravindra"
void main(){
    printf (scanf, scanf);
}
```

macros replace scanf everywhere by "%s ravindra" before compilation.

So, before entering in main() following changes occur as —

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

↳ space

%s denotes to print the string then

Output will be `%s ravindra ravindra`

↳ space

Q.3

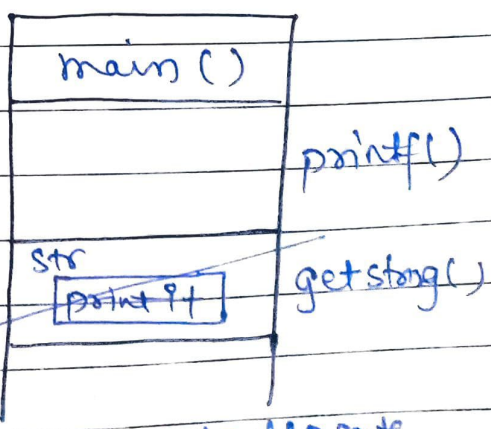
```
# include <stdio.h>
enum {false, true};
Int main() {
  int i=1;
do {
  printf("%d \n", i); → # print 1, returned value.
    i++;
    if (i<15)
    continue;  # no statement below so, not useful;
} while (false); # if anything inside while loop
                  condition is not (true) considers
                  as failure.
return 0;
}
```

o/p → 1

Q.4

```
char *getstring() {
    char str[] = "print it";
    return str; }
void main() {
  printf("%s", getstring());
}
```

Activation Record



After returning the value completed getstring() poped out so, space become deallocate.

So, printf() will not get any input and print any garbage values.

Output → Garbage undefined values

So, we can say that behaviour of a function is undefined.

Activation Record → Stack of functions processed.

Q. Write a c program to remove all occurences of all character c from the string.

```
void squeeze (char s[] , int c) {
    int i, j;
for (i=j=0 ; s[i] != '\0' ; i++) {
        if (s[i] != c)
        s[j++] = s[i]; }
  s [j] = '\0'
}
```

s

I/p | A | B | C | B | D | B | \0 |

O/p | A | C | D | \0 |

→ Deletion of c = 'B'

Storage Class

Examples

① int main() { → Activation record not formed for i.

register int i = 10; # store the variable in register.

int *a = &i;

printf ("%d", *a);

return 0;

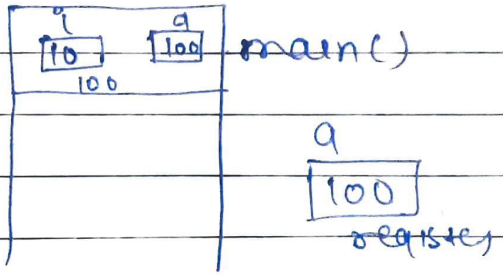
Note → Not a complete question, may show error, answer is compiler dependent

*a → stores the address but all the compiler not allowed to access the address of register.

So, answer will be compiler dependent

Date
Page No.

(2)
```
int main(){
    int i=10;
    register int *a = &i;
    printf("%d", *a);
    return 0;
}
```



o/p → 10

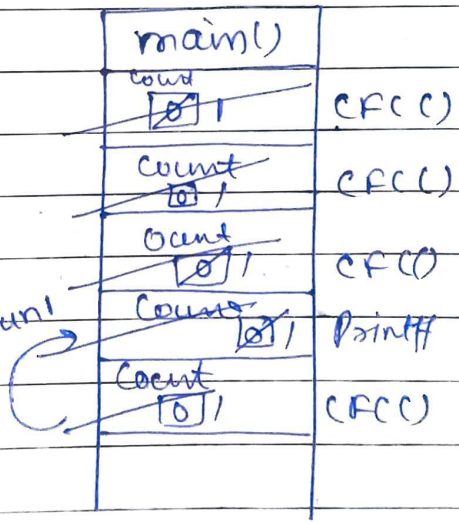
variable 'a' may be store in register or in activation record depends on compiler but output remains same

(3)
```
int main(){
    int i=10;
    register/static int i=10;
    printf("%d", i);
    return 0;
}
```

o/p → Compiler error bcoz a single variable can't be stored at two places.

Example of static int :-

```
int CFC(void){
    int count=0;
    return ++count;
}
int main(){
    CFC();
    CFC();
    CFC();
    printf("%d", CFC());
    return 0;
}
```



o/p → 1

If

Static int count=0;

then variable with create in data section instead of stack

so.

count

0 1, 2, 3 4

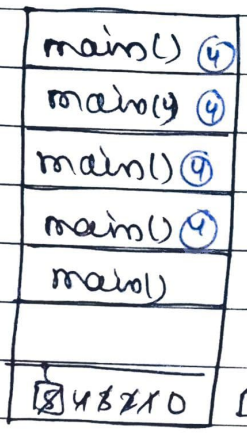
o/p → 4

Global and static returns a garbage value as 0.

Date :
Page No.

Example of static int -2

```
# include <stdio.h>
int main() {
    static int i=5;
    if(--i) {
    main();
    printf("%d", i);
}}
```



④ → execute the step ④ after returning back to main section.

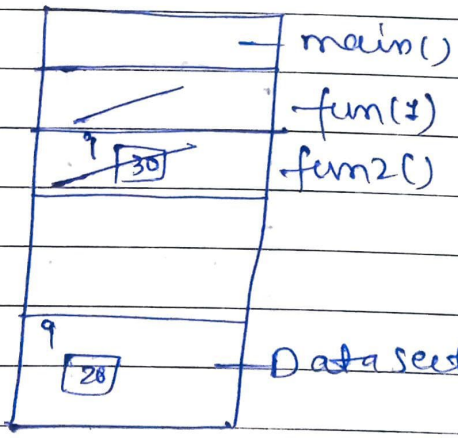
o/p → 0 0 0 0 → 4 times.

Either the i=5 initialize again and again but since it declares in data section, so its declaration occurs only once.

Example of global variable :-

If you dont initialize a global or static variable, automatically initialize to zero.

```
# include <stdio.h>
int i; # global
void fun1(){
    i=20;
    Pf("%d",i);}
void fun2(){
    int i=20;
Pf("%d",i);}
int main(){
    fun1()
    fun(2)()
return 0;}
```



o/p - 20 30

Scope of local variable is inside the function only. After completion variable diminished.

Module (5)

Structures

Date :
Page No.

Introduction → A type of data-structure Array is a collection of similar datatype. Structures are the collection of different data types

Ex→
```
struct {
    int i;
    char c;
} x,y,z;
```

Declaration

```
struct ex {
    int i;
    char c;
};
x,y,z;
```

Tag (optional)

Prototype

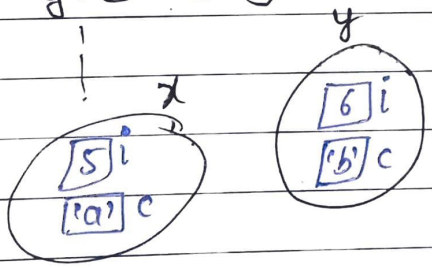
No need to declare this if tag is used.

```
int x,y,z;
x.i = 5
x.c = 'a'
y.i = 6
y.c = 'b'
```

Initialization

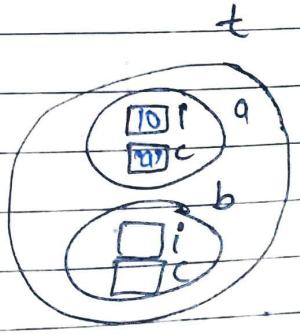
OR

```
struct ex x,y,z;
struct ex x = {5,'a'}, ... }
```



- **Nesting**

```
struct ex1 {
    struct ex a;
    struct ex b;
};
struct ex1 t;
t.a.i = 10
t.a.c = 'a'
```

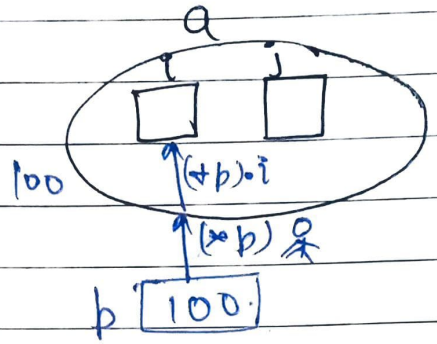


Declare of an structure can be take many types but you can choose anyone easier.

Date
Page No.

Examples on structures, arrays and pointers :-

①
```
struct node
{
  int i;
  int j; };
struct node a, *p;
  p = &a;
```



`a.i , (*p).i , p->i` Accessing of variable.

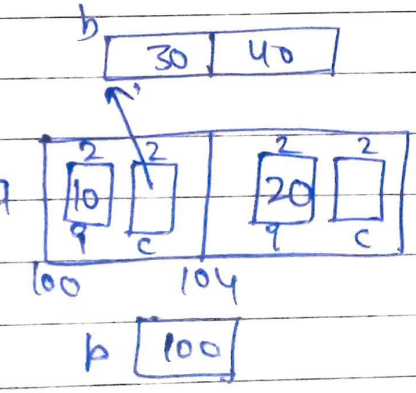
All three are similar but use of pointer is frequent.

② `struct node fun (struct node n1, struct node n2);`

```
struct node
{
  int i;
  int *c; };

struct node a[2], *p;
  int b[2] = { 30, 40 };
    p = &a[0];
  a[0].i = 10;
  a[1].i = 20;
  a[0].c = b;
```

`++p->i`	
`x = (++p)->i`	
`x = (p++)->i`	
`x = *p->c`	
`x = *p->c++`	
`x = (*p->c)++`	
`x = *p++->c`	



`a[0].i = 10` — Associativity

`a[0].c = b;` → means b contains will pointed by the location at `a[0].c`

We assume to restore the same value as previous after execution of each process.

Date :
Page No.

① (++(p→i)) a[0].i → Changes to 11 (10+1)
↳ Brackets, priority के according लगाए हैं।

② x = (++p)→i

p → changes from 100 → 104

Since p is pointing a structure node, so one(++) it will not increase by 1 but increment by size of structure (4 here).

p [100] ⟶ p [104] So, x=20.

③ x = (p++)→i

[x=10] and after that [p→104]

④ x = *p→c

[x=30]

⑤ x = *p→c++

x=30 and
c will move to point 40 instead of 30.

⑥ x = (*p→c)++

x=30 and
increment value at b[0] = 31

⑦ x = *p++→c

x=30 and then
p→ 104 from 100

Points to remember

① Apply the operators according to precedence.

② Increment or decrement will be done according to size of datatype.

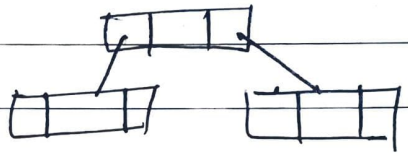
Ex → In previous case p [100] but p++ ≠ 101
p++ = 104.

Self referential structures :-

```
Struct ex
{
  int i;
  struct ex *link;
};
Struct ex abc;
```

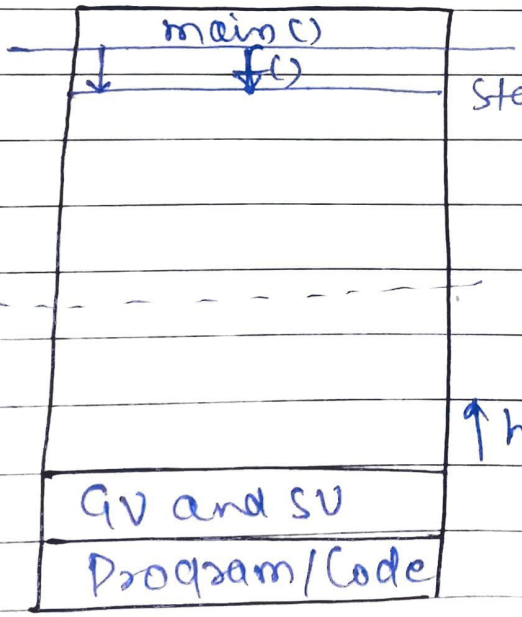
Ex Linked List

Tree (Binary)



If a pointer pointed to other structure of same type called self referential structure.

Malloc() :-



① dynamic memory take heap section

② Maximum size alloted to a process to use by operating system

③ sbrk(n) → command in Unix OS to increase the process spallo

④ C Library allot malloc().

void * malloc(int);] Not appropriate.

If int = 10;

malloc returns void *, so typecasting is necessary.

Date :
Page No.

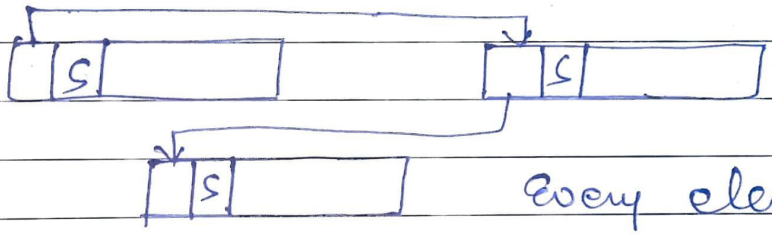
int *p = (int *) malloc (2)
↓ typecasting
↓ Not appropriate to pass an integer

because it will only works on platforms where integer is of size 2 byte and if integer will take 4 byte it shows an error.

To make it platform independent use sizeof()

int *p = (int *) malloc (sizeof(int)); ← Appropriate.
free (p); → free the space given to you.

malloc() might not get contiguous memory to allocated so, it creates a linked list



Every element of list have a header consist of two parts, size and addr of another element.

#
malloc() used first fit in OS.
↳ faster one

```
struct node {
    int i;
    struct node *l; };
struct node *p = < // >
(struct node *) malloc (size of (struct node))
```

If your program already have lot of space then, OS doesn't allowed to use more and malloc() returns NULL.

Input and Output

① formatted output - printf

int printf (char *format, arg1, arg2, ------)

Ex:-
```
void main(){
    printf("%d", printf("%s", "ravindra"));
}
```

printf("%s", "ravindra") → returns 8.

%d → format / Conversion specifier

ravindra 8

ravindra 8

② formatted input - scanf

int scanf (char *format, arg1, arg2, ----)

```
main(){
    int day, month, year;
    scanf("%d %d %d", &day, &month, &year);
}
```

int sscanf (char *string, char *format, arg1, ----)

Ex
```
sscanf("10 20 30", "%d %d %d", &a, &b, &c)
```

Store in a, Store in b, Store in c

↓

Input already given as string

③ Count number of set bits in x.

```
int bitcount(unsigned x){
    int b;
    for(b=0; x!=0; x>>=1)
        if(x&1)
            b++;
    return b;
}
```

set bits means no of bits are 1 in binary no.

Ex 1001 has 2 set bits.