C-Programming and DS

Today Class Topics

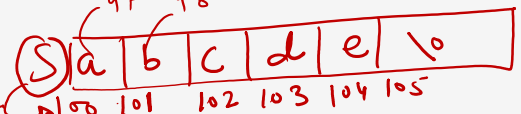
-Strings
-Array of pointers to strings
-Multidimensional Arrays on strings
-Pointer to function
-Structure and Union

Strings:- Array of character followed by null character('\0')

1) Character array is called as String. String always ends with null (\0) character

2) Null character representation is '\0' and ASCII value 0

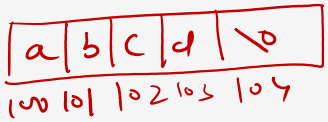
3) String name always gives base address



4) Every character of string stored in the memory sequentially in the continuous locations, in the form of their ASCII value

Ex:- char S[] = {'a','b','c','d','e','\0'};

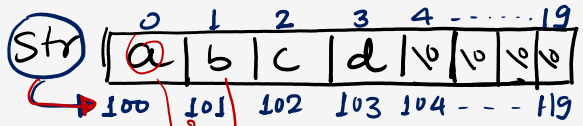
char t[] = "abcd";



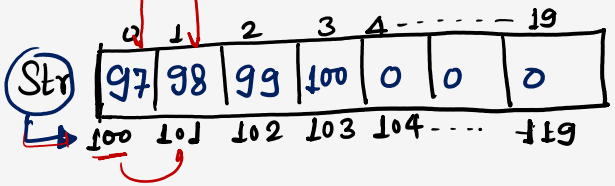
Ex:- char a[4] = "GATE"

Pf("%s",a); // Undefined behaviour

Ex:-
```
void main()
{
  char Str[20] = {'a','b','c','d','\0'};
  Pf("%c%c%c%c", Str[0], Str[1], Str[2], Str[3]);
}
```
a b c d



Stored in memory:



Str[0] = *(Str+0) = *(100) = 97 ⇒ a

Str[1] = *(Str+1) = *(101) = 98 ⇒ b

Pf("%c", Str[0]); =? a

Pf("%d", Str[1]); =? 98

Pf("%c", Str[15]); =? no print

Pf("%c", Str); =? Error

Str = 100, Str[0] = *100

Pf(Str) = 100

Pf(Str+1) = 101

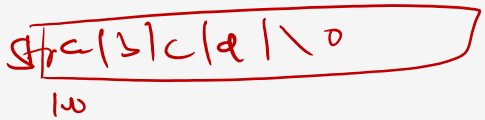
Ex:- main()

{

char str[20] = "abcd";

printf("%s", str);

}



abcd

o/p = ?

NOTE :-

1) for '%c', we need to provide ASCII value

2) Reading and printing str[0] by using

   i) %c → char

   ii) %d → ASCII value

3) for the %s we need to provide the Base Address, from the base Address provided it will print all the characters until the null characters

Ex:-

```
main()
{
char S[20]="abcdef";
```

① `Pf("%s",S);` // 100 // abcdef
`S[3]=97;`

② `Pf("%s",S);` // 100 // abcaef
`S[3]=0;`

③ `Pf("%s",S);` // 100 // abc
`S[3]='0';` //

④ `Pf("%s,S);` // abc0ef

⑤ `Pf("%s",*S);` // Error

⑥ `Pf("%c",S+3);` // Error

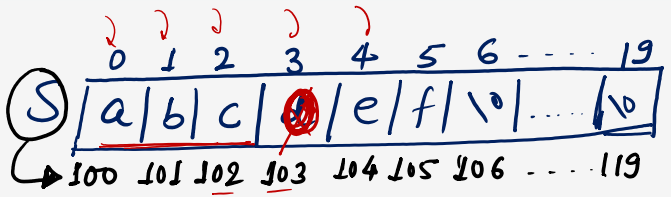
⑦ `Pf("%s",S+3);` // 0ef

⑧ `Pf("%c",*(S+3));` // 0

```
}
```

Ans

=>



97

0 → \0

48

S+3 = 103

0 ≠ '0'

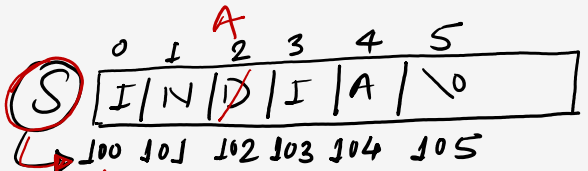
\0 → Ascii → 0

'0' → 48 '0' → Ascii → 48

0 →

Initialization of Strings:-

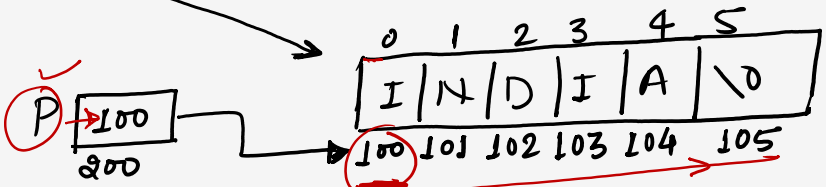
1) `Char S[6] = "INDIA";`
2) `Char S[] = "INDIA";`
3) `Char *P = "INDIA";`



`Pf("%s", S); =?` INDIA

`S[2] = 'A' // allowed`

The meaning of Statement (3) is, we are requesting the Compiler to allocate the memory for the String INDIA and Base address of that String will be Stored in the Pointer Variable P. it is Consider as String Constant and Contents of String Constant can't be modified



`Pf(P) = 100`

`Pf(P+1) = 101`

`Pf("%s", P); =?` INDIA

`P[2] = 'A'; // not allowed`

Const Keyword :-

1) `Char *a = "Success";`

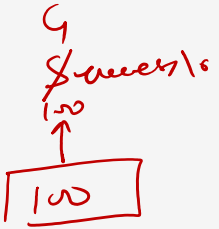
`*a = 'G'; // Error`

`a = "GATECSE"; // work`

2) `Const Char *b = "Success";`

`*b = 'G'; // Error`

`b = "GATECSE"; // work`



3) `Char * Const d = "Success"; //`

`*d = 'G'; // work`

`d = "GATECSE"; // Error`

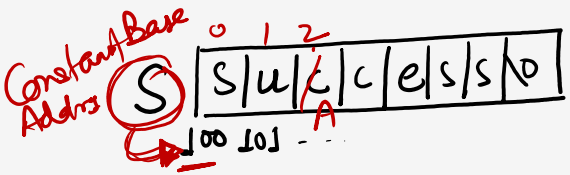
d++ ✗

4) `Const Char * Const e = "Success";`

`*e = 'G'; // Error`

`e = "GATECSE"; // Error`

1) Char S[] = "Success";

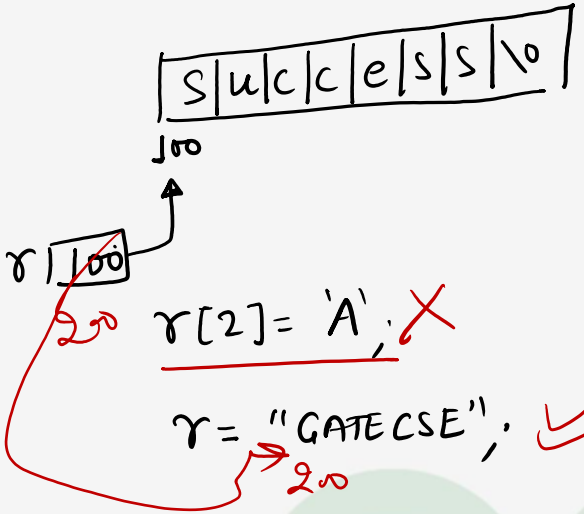


S[2] = 'A'

S = "GATECSE";

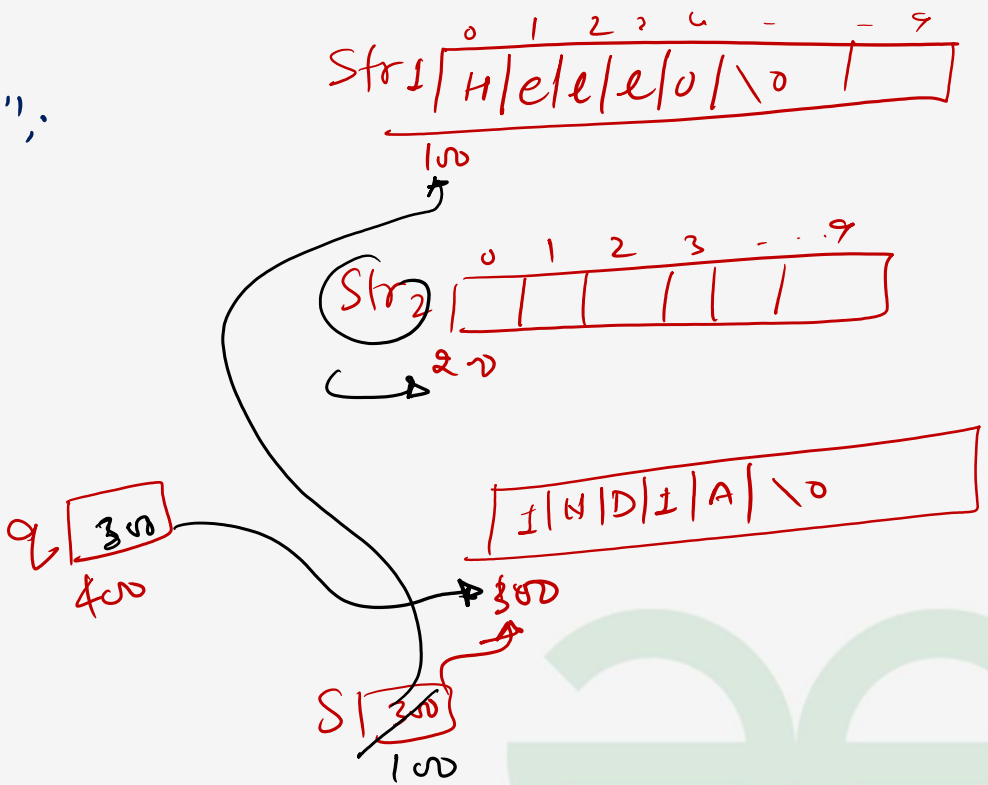
GATECSE
S → 200

2) Char *r = "Success";



Ex:-

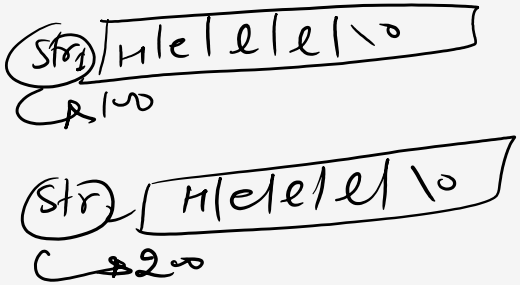
```
main()
{
Char Str1[10] = "Hello";
 Char Str2[10];
 Char *S = "INDIA";
 Char *q;
Str2 = Str1;  //Error.
 q = S;  // Work
}
```



S = Str1,

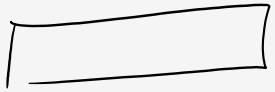
Ex:-

```
main()
{
 Char Str1[10] = "Hello";
 Char Str2[10] = "Hello";
 if (Str1 == Str2)
  Pf("Hi");
 else
    Pf("Bye");
}
```



Ex:-

```
main()
{
  if(*"Hello" == *"He")
    Pf("They are same");
  else
    Pf("They are different");
}
```



*100 == *200

H == H

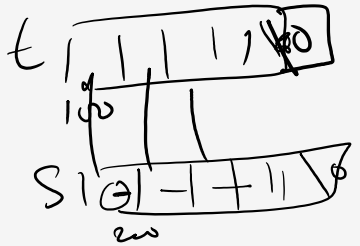
NOTE: if the String is given in the double quotes (" ") it means base address is given

General function available in the <String.h> header file:-

1) Strcpy(T, S) // it copies Source string to target string

Target string Base Address

Source string Base Address



Ex:- Char Source[] = "GATE 2025";
Char target[20];
Strcpy(target, Source);

Pf("%s", target); // GATE 2025

NOTE: targetsize ≥ Sourcesize

mystrcpy():

```
void mystrcpy(char *t, char *s)
{
    while(*t++ = *s++);
}
```

2) Strlen(s) // It return no. of character before null character

(s) → BaseAddress

Ex:- Char S[] = {'a', 'b', 'c', 'd', '\0'};

Char t[] = "GATE\0\0";

Char * r = "BCD\0\0EFG\0";

Strlen(S) = ? 4

Strlen(t) = ? 4

Strlen(r) = ? 3

mystrlen():-

```
int mystrlen (char *t)
{ int length=0;
  while (*t != '\0')
  { length++;
    t++;
  }
  return length;
}
```

3) Strcat(target, source); // Concatenate source string to target string

Ex:- Char source[] = "CSE 2025";
Char target[20] = "GATE";
Strcat(target, source);
Pf("%s", target); // GATECSE 2025

mystrcat():

```
Void mystrcat (char *t, char *s)
{
    t = t + strlen(t)
    while (*s != '\0')
    {
        *t++ = *s++;
    }
    *t = '\0';
}
```

4) Strcmp(S1, S2) :- // Compare strings S1 and S2 and return ASCII difference of first mismatch character, otherwise 0

```
Char S1[] = "GATE";
Char S2[] = "CSE";

x = Strcmp(S1, "GATE");  // 0
y = Strcmp(S2, S1)       // -4
z = Strcmp(S1, "GATECSE");

Pf(x, y, z);
   | | ↓
   0 -4 -32
```

C G

A - 65
B -
C
D -
E -
F -
G -

5) Strrev(s); // it reverse the given string

=>

↑ Base Address of string

6) Strupr(s); // Converts all characters to uppercase

7) Strlwr(s); // Convert all characters to lowercase

8) Strncpy(t, s, n);

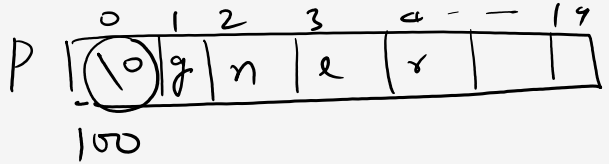
9) Strncat(t, s, n);

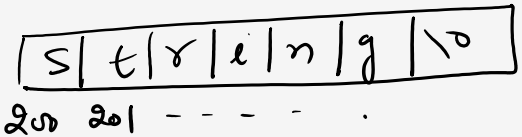
Ex:-

```
Char P[20];
Char *S = "String";
int length = Strlen(S);
for(int i=0; i<=length; i++)
  P[i] = S[length-i];
printf("%s", P);
```

O/p = ?

a) gnirts
b) nirts
c) String
d) No output





length = 6

S[6-0]
S[6]
\0

Ex:-

```
Char *P = "GATE2011";
printf("%s", P+P[3]-P[1]);
```

o/p = ?

a) GATE2011

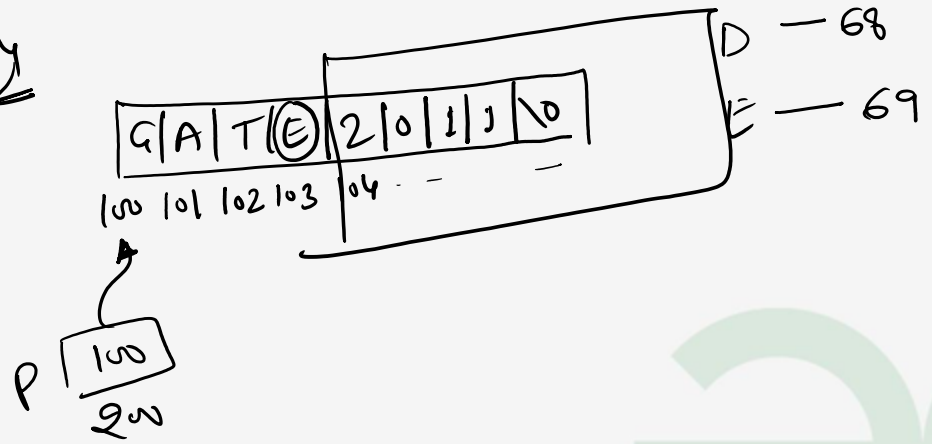
b) E2011

c) 2011

d) 011

100 + 69 - 65

104



A — 65

B — 66

C — 67

D — 68

E — 69

Ex:-

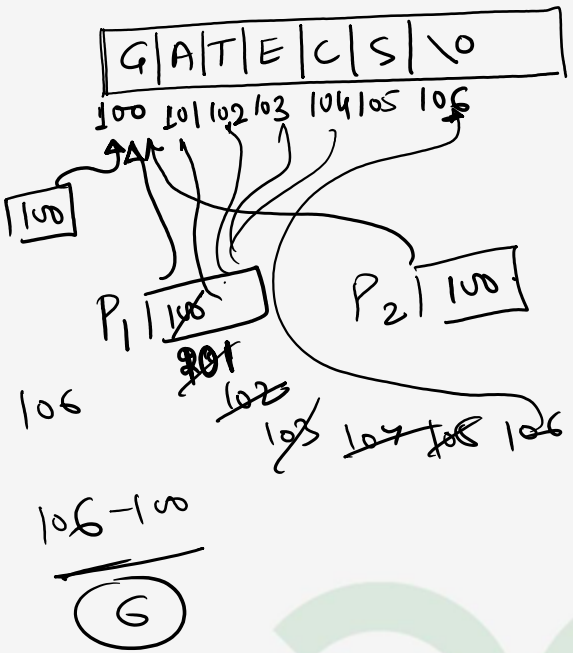
```
Void main()
{
  Char *Str = "GATECS";
  printf("%d", test(Str));
}
int test(Char *P1)
{
  Char *P2 = P1;
  while(*++P1);
  return (P1 - P2);
}
```

What is o/p printed 6 Ans ?

*ptr++

*ptr

ptr+1



Array of Pointers to strings:-

char *S[4]

Pointer takes 4 byte

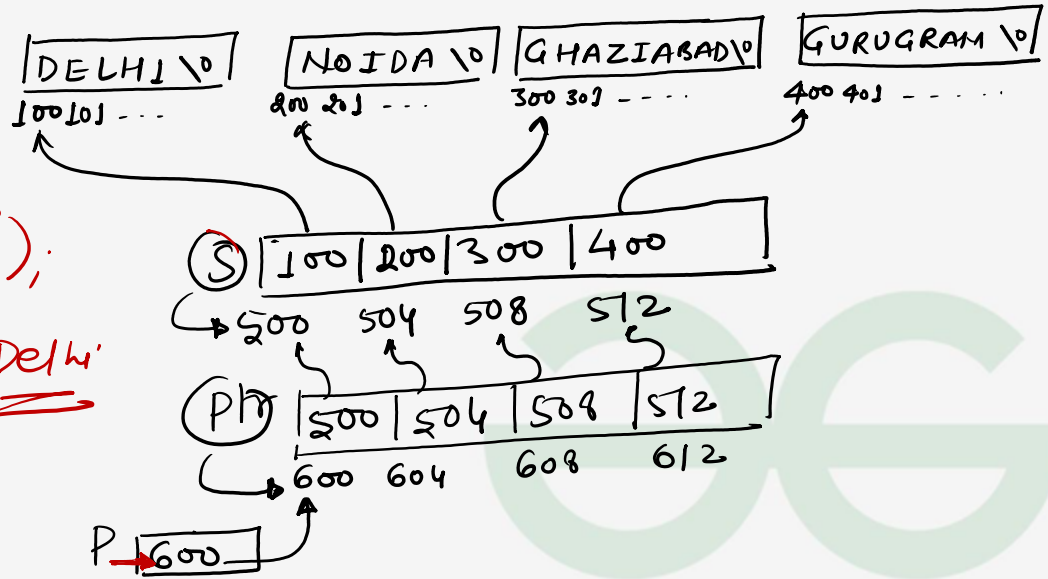
100 200 300 400

```
main()
{ char *S[4] = {"DELHI", "NOIDA", "GHAZIABAD", "GURUGRAM"};
  char **ptr[4] = {S, S+1, S+2, S+3};
  char ***P = ptr;
}
```

Pf(S) = ? 500
Pf(ptr) = ? 600
Pf(P) = ? 600
Pf(S+1) = ? 504
Pf(ptr+1) = ? 604
Pf(P+1) = ? 604
Pf("%s", *S); = ?

Pf("%s", *S);

O/P = Delhi

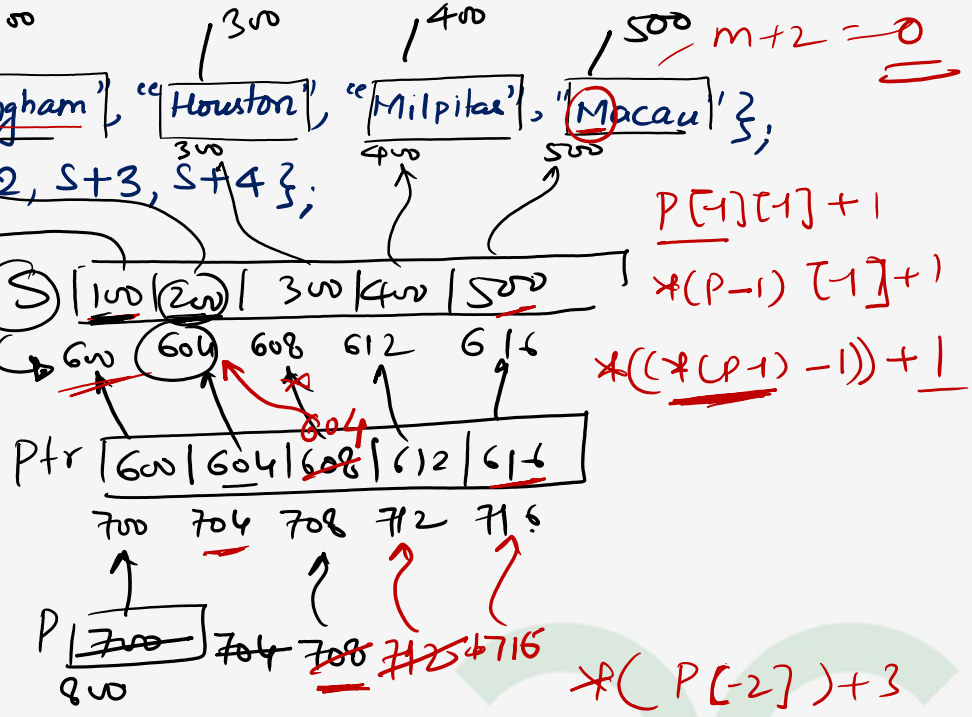


Ex: main()

{char *S[5] = {"London", "Birmingham", "Houston", "Milpitas", "Macau"};

char **ptr[5] = {S, S+1, S+2, S+3, S+4};

char ***P = ptr;



① Pf("%s", **++P); // Birmingham

② Pf("%s", *--*++P+3); // mingham

③ Pf("%s", *(P[-2])+3); // don

④ Pf("%s", P[-1][-1]+1); // ondon

⑤ Pf("%s", **P++); // Birmingham

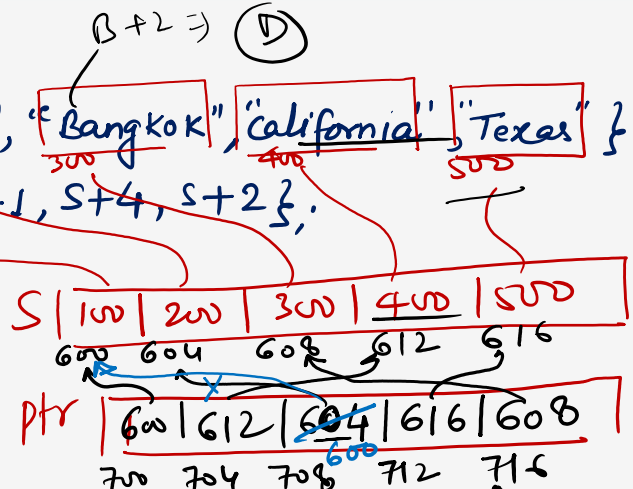
⑥ Pf("%c", ***++P+2); // 0 (oh)

⑦ Pf("%c", S[1][2]); // r

}

Ex!- main()

```
{ char *S[5] = {"Qatar", "Dubai", "Bangkok", "California", "Texas"};
  char **ptr[5] = {S, S+3, S+1, S+4, S+2};
  char ***P = ptr;
```



① Pf("%s", **P++); // Qatar

② Pf("%s", *--*++P+3); // ar

③ Pf("%s", **++P); // Texas

④ Pf("%s", P[-1][-1]+1); // Garbage or Error

⑤ Pf("%s", *(P[-2])+3); // efern

⑥ Pf("%c", ***++P+2); // D

⑦ Pf("%c", S[1][2]); // b ⇐ **700

```
}
```

P[-2] + 3

*(P-2) + 3

*(700) + 3 = 403

P[-1][-1]+1

((P-1) - 1) + 1

((712-1) - 1) + 1

((708) - 1) + 1

*(600-1) + 1

*(596) + 1

Multidimensional Arrays on strings:-

```
main()
{ char mess[6][30] = { "Koi bhi Lakshya",   — 0th 1D Array
                       "Bada nahi",         — 1st 1D Array
                       "Jeeta wahi",        — 2nd 1D Array
                       "Jo dara nahi",      — 3rd 1D Array
                       "Hara wahi")         — 4th 1D Array
                       "Jo lada nahi" };    — 5th 1D Array

Pf("%s", mess+3); // Jo dara nahi
Pf("%c", *(*(mess+2)+3)); // t
Pf("%s", *(mess+4)); // Hara wahi
}
```

Void Pointer :-

1) In the void Pointer we can store address of any type of variable while accessing the data we need to Perform the respective type casting accordingly

2) Type casting is required because void pointer doesn't know how many bytes it has to dereference

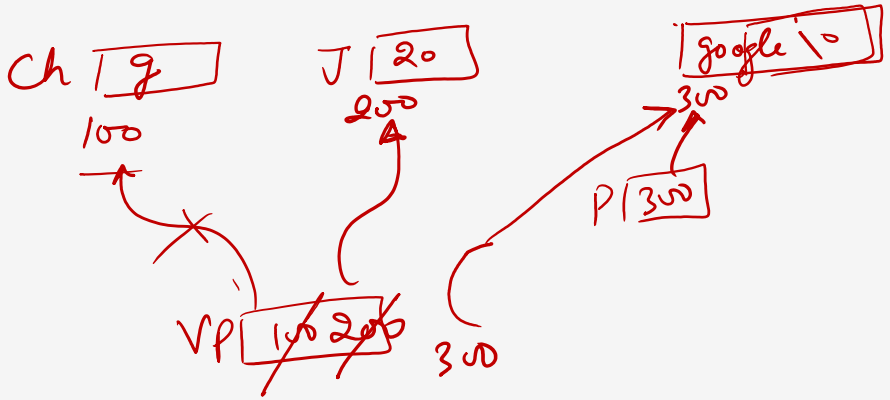
Ex:-

```
main()
{
  void *vp;
  char ch='g';
  int j=20;
  char *p="google";
  vp=&ch;
  Pf("%c",*(char*)vp); // g
  vp=&j;
  Pf("%d",*(int*)vp); // 20
  vp=p;
  Pf("%s",(char*)vp+3); // gle
}
```

Typecast

*vp

Typecast



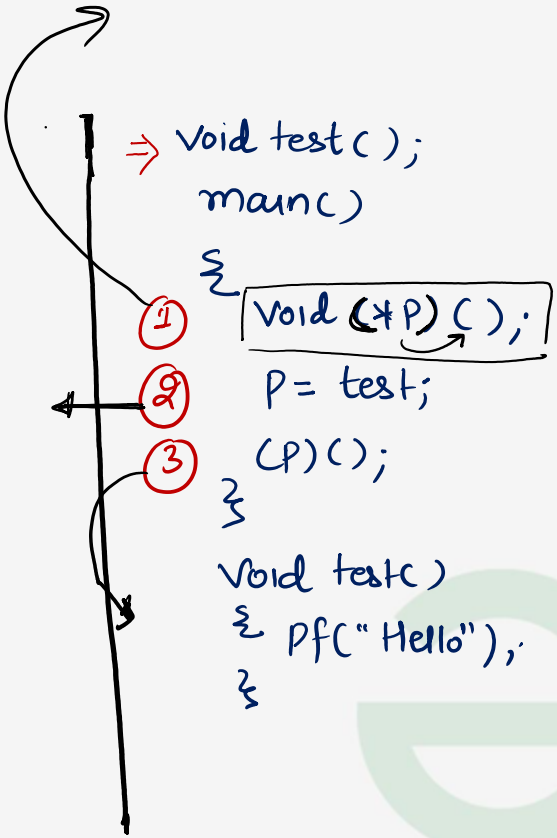
Pointer to function :-

① void (*p)(); //

② int (*f)(int, float); //

③ int *f(char); //

② ①



Terminology for various expressions:

1) `void (*p)();` //

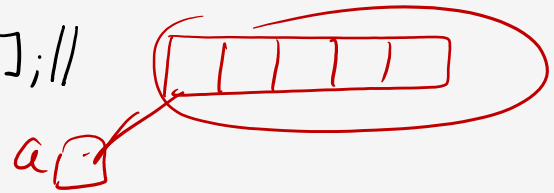
2) `int (*p)(int, int);` //

3) `int (*p)(int *);` //

4) `int a[5];` //

5> int *a[5]; //

6> int (*a)[5]; //



7> int (*a)(); //

=>

8> int *a(); //

=>

9) `char **(*a)(*,*); //`

10) `int (*a[5])(); //`

11) `void (*F[10])(int,char); //`

12) `float ***F(double ****); //`

10 minits Break

2:40

Structure :- it is collection of heterogeneous datatype

Keyword → Struct
Structure name → emp

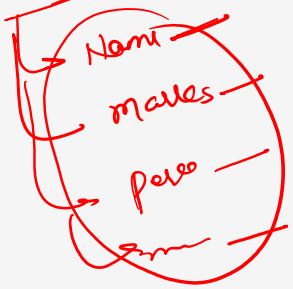
```
Struct emp
{
    int id;
    char name[40];
    int sal;
};
```

2 Byte → int id;
40 Byte → char name[40];
2 Byte → int sal;
member variable.
44 Byte

5 minute - 10 minute

2:40

Stud



Structure is used to create user defined datatype,
in structure the two operators are used

1) • Structure variable to member variable

2) → Pointer variable to member variable

→

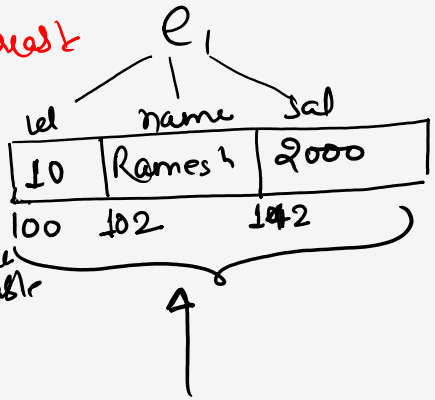
NOTE:-

1) Structure Contains member variables
2) Size of Structure is Sum of all member variables Size
3) We Can't define a Structure without any member variable
4) We can define the Structure inside the function
5) We can't define the function inside the Structure
6) We can create existing Structure as member to other Structure
7) We Can't create same Structure variable as member to the Structure because logically size is Undefined
8) We can create Same Structure Pointer as member to the Structure and that is called Self referential Structure (This is used in Linked list)

```
Struct emp
{ int id;      } 2Byte
  char name[40];
  int sal;
};
typedef Struct emp Emp;
```

type

⇒ membervariable




```
main()
{ emp e1;  // X not allowed
  Struct emp e1,e2;  // allowed
  Emp e3,e4;  // allowed
  e1.sal = 2000;
  e1.id = 10;
  Emp *ptr;
  e1.name = Ramesh;  // X
  e1.name = "Ramesh";  // X
```

ptr → 100

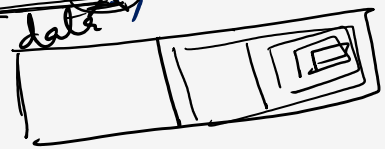
100 → 10

or

```
Strcpy(e1.name, "Ramesh");
ptr = &e1;
Pf(ptr+1);  // 144
e2 = e1+e3;
e2 = e1*e3;  X
e2 = e1/e3;
e2 = e1;  // Allowed
Pf(e1);  // Error
Pf(e1.sal);  // 2000
Pf(ptr);  // 100
Pf((*ptr).id);  // 10
Pf(ptr->id);  // 10
Pf(*ptr);  // error
}
```

Ex:

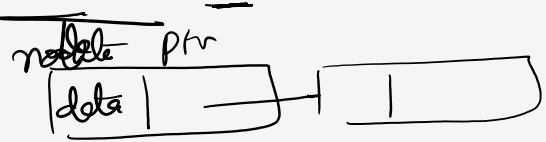
```
Struct node
{
    int data;
    Struct node *n1;
};
```



(1)

Ex:

```
Struct node
{
    int data;
    Struct node *n1;
};
```



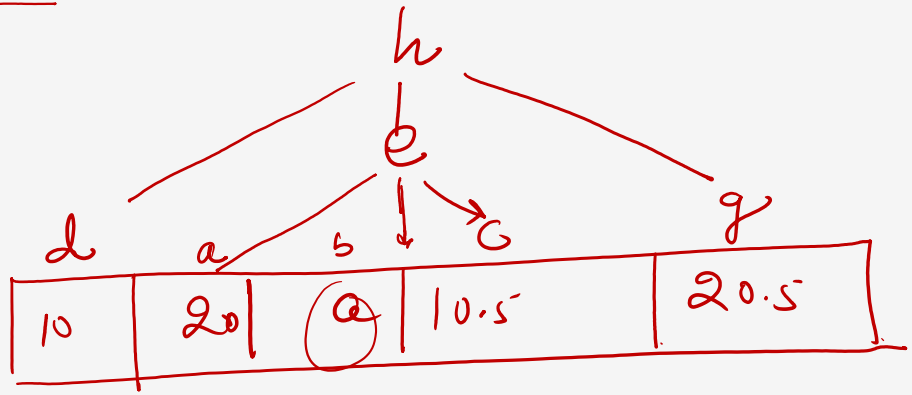

```
void main()
{
    Struct emp
    {
        int d;
        float b;
    };
    mec();
}
void mec();
{
    Struct emp e1, e2, e3;
}
```

Ex:- (6)

```
Struct S1
{ int a;
  char b;
  float c;
};

Struct S2
{ int d;
  Struct S1 e;
  float g;
};
```

```
Struct S2 h = {10, {20, 'a', 10.5}, 20.5};
```



pf(h.d) = 10

pf(h.g) = 20.5

pf(h.e.a) = 20

pf(h.e.b) = a

pf(h.e.c) = 10.5

Union:-

Linked list

1) The size of Union will be maxsize of member variable

2) In the Structure we can access all the member at any time but in case of Union we can access only one member variable and all the remaining members will contain garbage value

3) In the Union same memory space will be shared by all the member variable

Ex:-

```
Union data
{ int i;
  char Str[20];
  float f;
};
```

2B + 20B + 4By = 26Byte → Struct

Union → 20Byte

Thank You !