```
short int i=20;
void main()
{
 int *p;
 static int i=0;
 p=&i;
 pf("i=%d\n",i);
 pf("*p=%d\n",*p);
}
```

o/p i=10
     *p=10

But when

```
{
 int *p;
  p=&i;
 static int i=10;
 pf("*p=%d\n",*p);
}
```

→ for the same program

o/p *p=20 here

```
static int i=20;
int *abc(void);
void main()
{
 int *p;
 static int i=10;
 pf("i=%d\n",i);
 p=abc();
 pf("*p=%d\n",*p);
}
int *abc(void)
{
 return &i;
}
```

As local has the higher priority local 'i' is printed

here the global variable value is printed as the abc() returns the address of the global variables.

this fun() access the global variable 'i', (anyway it can't access the local var 'i' inside the main())

→ Stack variables are stored in data section.

static int i;
→ is stored in (BSS) (uninitialised data type)
Block started by symbol

stat int i=10;
→ is stored in initialised data section.

In hard disk

when the program is compiled, compiler will create an .exe file.

when ./a.out ↵ this .exe file is loaded into RAM, a data section, code, heap & stack section are allocated.

→ If a static variable is declared globally, it can be accessed in the entire file.

→ If we declare a local static, it can be accessed within a function.

→ Life of a static variable starts when the program starts execution & life ends when program execution completes

→ for static variable, re-initialization is not going to happen

```
static int i;
void abc(void);
void main()
{
  abc();
  abc();
  abc();
}
void abc()
{
static int i=10;
i++;
printf("i= %d\n", i);
}
```

o/p
i=11
i=12
i=13

Life of a static variable starts when the program is loaded in RAM and ends, when the program completes execution.

Session 58

26-5-20

→ Extern and multiple files compilation

→ when we supply multiple files together, each file is seperately pre-processed, translated and assembled.

→ After assembler stage, all '.o' file and libraries

get linked and compiler will generate a.out '1' execution file

To compile multiple files together

cc m1.c m2.c m3.c

m1.c

```
Void main()
{
  Printf("hello world\n");
}
```

Not printf

Printf function

m2.c

```
Void print(char *p)
{
printf("\n m2.c :%s\n",p);
}
```

→ These two files are not compiled simultaneously. Each files are compiled till the '.o' stage seperately & linked together at the end.

m1.c throws Warning, as there's no fun prototype

Now in m1.c,

```
void main()
extern void print(char *);
Void main()
{
  print("Hello World\n");
}
```

Now no warning is thrown

Even without 'extern' this program works.

This statement is to inform the translator that the prototype is present in another file.

```
extern void print(char *);
```

Note :- Default, the functions are extern

(Header file inclusion will not affect the file size

main.c

```
main()
{


}
```

Source code 1

read.c

```
void read_data()
{
  ======
  ======
}
void abc()
{
  ======
  ======
}
```

Source code 2

extern → apart from the same source code, other source codes can also use/call the fun definition.

→ When a function is extern, that fun. can be called by other function, in same source code and by the function in other source code also.

Session - 59

27-8-20

For:-

extern int i; → No memory is given

int i;
int i=10; } → memory is allocated.

extern int i; // is a declaration, no memory is allocated.
int i; // is a definition
int i=10; // is initialization } Here 2 memory is allocated.

	Translator	Linker
extern int i;	declaration	X
int i;	definition	weak symbol
int i=10;	initialization	strong symbol
void abc(void);	fun. declaration	weak symbol
void abc(void) { ====== }	fun. definition	strong symbol

→ Multiple strong statements are not possible in linkage

m1.c

```
#include<stdio.h>
int i;          // extern variable
static int j; // This is available throughout the program.
void main()     It can't be accessed by another program
{                          as the variable is static
  static int k; // local static (scope is within main)
  int l;  // auto (local) (scope within main)
}void ab()          (both l & k can't be accessed by
{                                main())
}
```

Note: By default for global variable, compiler will assign static properties with external linkage.

→ Default value of extern is **zero**

→ Scope is both inside & outside the program.

* m2.c can access the extern int i variable of m1.c if m2.c has declared extern int i globally

m1.c

```
int i;
void main()
{
  ===
}
```

```
extern int i;
void ab()→
{
  ===
}void de()
{
}
```

Both these fun can access i;

Local static } → No linkage
Auto }

Global static → Internal linkage

Extern → External linkage

Session 60 — Register storage class — 28-8-20

→ Default register storage value is unexpected (garbage value)
If the CPU register is not free (occupied by some other programs), the data is stored in stack frame in RAM.
Register data when stored in CPU register has faster accer bec. address is not needed to be resolved.

scanf("%d", &i) → Not possible

→ We can't use '&' operator on register storage class.
→ pointers can't be used to access register storage.
→ Can't be declared globally.
→ Compiler will decide the memory and OS will allocate memory.
→ Register (like auto) can be declared only locally

eg extern int i=0; ✗ invalid
Because initialization can't be done with extern keyword.

Recursive function

In one fun def., if the same fun. is called, it is called as recursive function.

```
void main()
{
  Pf("Hello");
  main();
  Pf("Hai!!");
}
```

This cause segmentation fault error

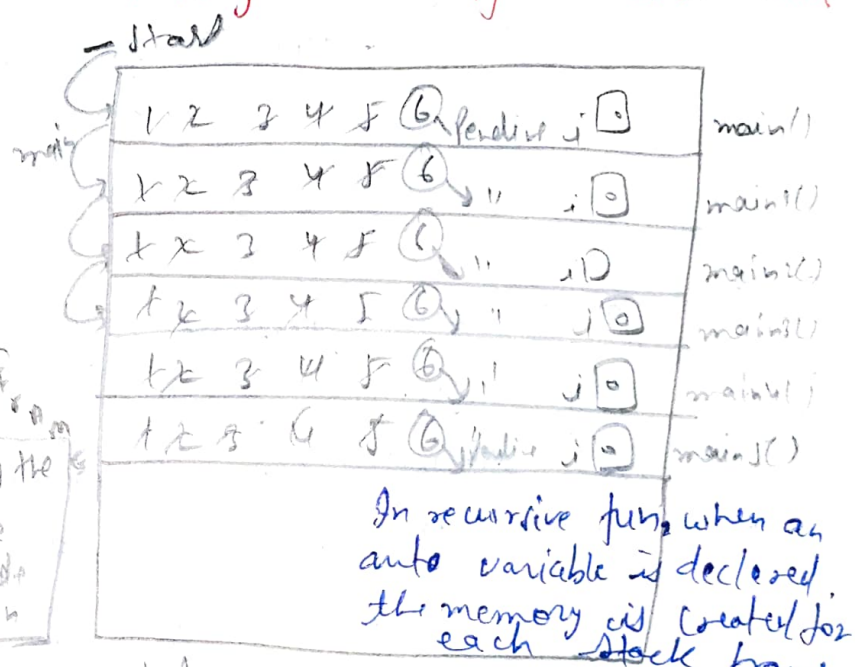
→ If a fun does not return any value, it returns control when the lastest fun. are executing, the previous function are also existing in the stack frame.

The main() calls itself until 8 MB of stack frame is occupied. Once, it is occupied, the control tries to access other memory, due to which segmentation fault error occures.

eg→

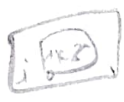
```
Void main()
{
①  int i=0;
②  Pf("Hello \n");
③  i++;
④  if(i<5)
⑤  main();
⑥  Pf("Hai\n");
}
```

→ segmentation fault error occure

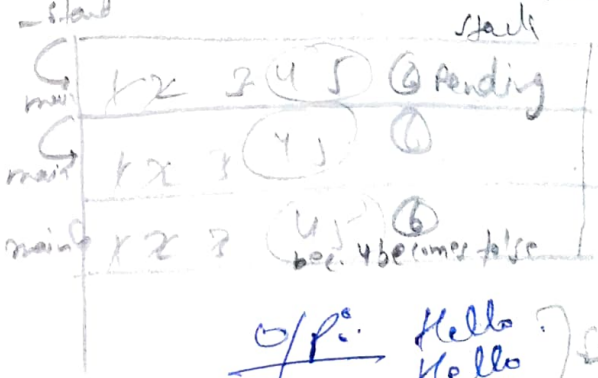


Now consider

```
Void main()
{
static int i=0;
Pf("Hello \n");
i++;
if(i<5)
main();
Pf("Hai \n");
}
```



No reinitialisation in stack frame



o/p: Hello, Hello, Hello, Hai, Hai, Hai

(stack: Last In First Out)

→ To control recursiveness either global variables have to be declared or local static variable has to be declared.

Session – 61

29-8-20

Conditions for a recursive function :-

(1) It should be possible to express the problem in recursive form.
(2) The statement must include stopping condition.

Things to be taken care in function recursion

① Try to avoid loops, as fun. recursion acts as loops. Compares to loops recursive fun. takes more time & occupies more space in stack frame. This is one of the reason why this is not encouraged in hardware programming.
② Use a conditional statement in recursion
③ Use proper return statements in conditional statements
④ Use Pf() during program development, so that mistakes if made can be watched.

Design a recursive function for finding the factorial of a given number.

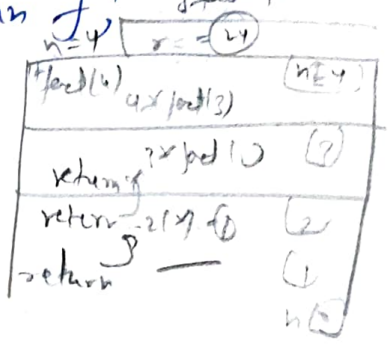
```
#include <stdio.h>
int fact(int);
void main()
{
  int n, r;
  Pf("Enter the number \n");
  Sf("%d",&n);  //n=4
  r = fact(n);
```

```
    Pf("factorial = %d \n", r);
}
int fact(int n)
{
  if(n)
  return n * fact(n-1);
  else
   return 1;
}
```

this 1 will return the control from where it is called.

Without recursive logic

```
int fact(int n)
{ int f=1
  while( n )
  { f = f * n;
    n--
  }
return f;
}
```



Now

```
int fact(int n)
{
  if(n)
  return n * fact(n--) // recursion is unable to control due to post dec.
  else
    return 1;
}
```

Causes segmentation fault error

Design a recursive function for printing a no. into binary form

```
#include <stdio.h>
 Void print(int);
 void main()
 {
   int num;
   Pf("Enter the number \n");
   Sf("%d", &num);
   Print(num); ← function call
  Pf("\n");
  Print(num); // ← if 'else' part is not written, this will
   Pf("\n");
}
```

not be executed as 'per' value would have become '-1'.

```
void print(int n)
{
  static int pos=31;
    if(pos>=0)
    {
      printf("%d", n>>pos&1);
       pos--;
      print(n); // function call for print
    }
      else
       pos=31;
  }
```

[recursion happens here]

31-8-20

Session-6.2

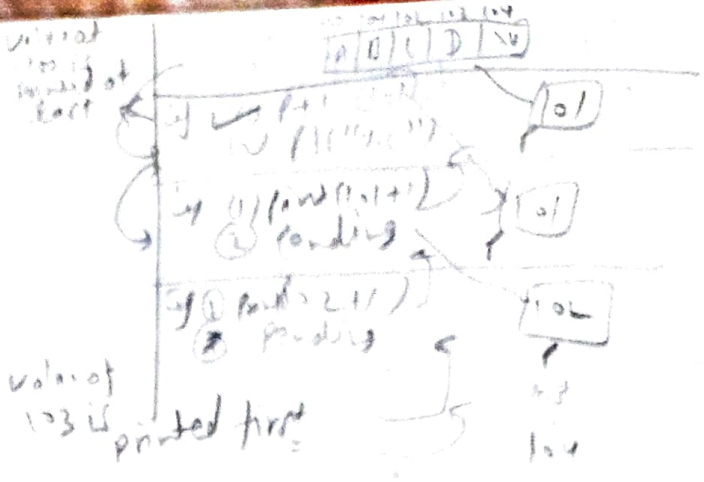
Design a recursive function to print a string char by char.

```
#include<stdio.h>
 void print(const char *);
 void main()
 {
  char s[]="ABCD";
   print(s);
   printf("\n");
 }
 void print(const char *p)
  {
      if(*p)
      {
       printf("%c_",*p);
        print(p+1);
      }
 }
```

when these two lines are exchanged, the o/p is reverse printed.

Pre inc works
Post inc causes segmentation fault (as the addr. of A is passed again & again)

```
if(*p)
{
  print(p+1);
  Printf("%c", *p);
}
```



Consider,

```
if(*p)
{
  print(++p);
  Printf("%c", *p);
}
```

Design a recursive fun for finding the length of the given string.

```
#include <stdio.h>
int my_strlen(const char *s);
Void main()
{
  int l;
  char s[] = "abcd";
  l = my_strlen(s);
  Printf("l = %d\n", l);
}
int my_strlen(const char *p);
{
    if (*p)
    return 1+my_strlen(p+1);
    else
    return 0;
}
```

Session 63 — 1-9-20

→ Recursive function → 2D array

Design a recursive fun for copy source string into destination.

```
Void strcpy_cpy(char *d, const char *s)
{
    if(*s)
    {
        *d = *s;
        strcpy_rec(d+1, s+1); // incrementing the address by
    }                         // '1' and calling the same
    else                      // function again.
      *d = *s;
}
```

Searching character in a given string

```
char * strchr_rec(const char *s, char ch)
{
    if(*s)
    { if (*s == ch)
        return s;
      else
        return strchr_rec(s+1, ch);
    }
    else
      return 0;
}
```

2-d array

2-d array:- collection of 1-D array or array of the arrays or matrix

2-d array declaration

syntax: datatype variable name [ ] [ ] ;

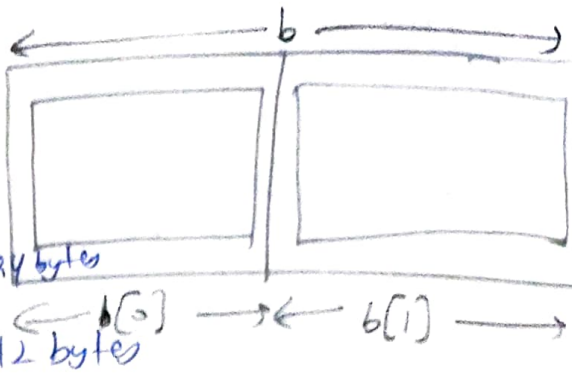
- [ ] (row) → No. of 1-d arrays
- [ ] (columns) → In each 1-D array no. of elements

eg →

```
#include<stdio.h>
Void main()
{
  int b[2][3];
  printf("%ld\n", sizeof(b));
  printf("%ld\n", sizeof(b[0]));
  printf("%ld\n", sizeof(b[0][0]));
}
```

b[0] and b[1] ← 2 elements are present ← There are 2 1-D array

2*3 = 6;



112 bytes

12 bytes

4 bytes

o/p:- 24 → entire 2-D array

12 → 1 array in 2-d array

4 → 1 element in the array

Note:- In 2-d array, the elements are 1-D array

eg →

```
#include<stdio.h>
Void main()
{
  int b[2][3]
  int r, c, i, j;
  r = sizeof(b)/sizeof(b[0]);
  c = sizeof(b[0])/sizeof(b[0][0]);
  for(i=0; i<r; i++)
  {
    for(j=0; j<c; j++)
      pf("%d ", b[i][j]);
    pf("\n");
  }
```

o/p 1 2 3
4 5 6

2-D array initialization example

```
b[2][3] = {{10,20,30},{40,50,60}};
  or
b[][3] = {{10,20,30},{40,50,60}};
```

Session 64

2-9-20

b[2][3] = {{10,20,30},{40,50,60}};

b[0][0]	b[0][1]	b[0][2]	b[1][0]	b[1][1]	b[1][2]
10	20	30	40	50	60
1000	1004	1008	1012	1017	1020

b[0] — 1-D array

b[1] — 1-D array

b (2d array)

* what is the size of b?
  24 bytes
* what is the type of b[0]?
  Integer 1-D array
* what is the size of b[0]?
  12 bytes.
* what is the type of b[0][0]?
  Integer type.
* what is the size of b[0][0]?
  4 bytes.
* How many elements are present in b & what are those?
  2. 1-D array
  There are two elements in b. They are b[0] & b[1]
* How many elements are present in b[1] & what are those?
  3 elements. Those elements are b[1][0], b[1][1], b[1][2].

b → 1000

b[0] → 1000

base address of 1-D array b[0]

b[0][0] → 10

b+1 → 1012

base address of 1-D array

b[0] → 1000

next element in that 1-D array

b[0][0]+1 → 11

$b \rightarrow 1000$ $\quad b+1 \rightarrow 1024$

$b[0] \rightarrow 1000$ $\quad b[0]+1 \rightarrow 1002$

$b[0][0] \rightarrow 1000$ $\quad b[0][0]+1 \rightarrow 1004$

$b \rightarrow 1000$ $\quad b+1 \rightarrow 1012$ $\quad b+2 \rightarrow 1024$

$b[0] \rightarrow 1000$ $\quad b[0]+1 \rightarrow 1002$ $\quad b[0]+2 \rightarrow 1004$ $\quad b[0]+3 \rightarrow 1006$

$b[0] \rightarrow 1000$

$b[0]+1 \rightarrow 1004$

↳ $b[0]$ is a 1-d array

(i.e. increases the next element in 1-D array $b[0]$)

when '&' is there, consider the complete array

$b++ \rightarrow$ error $\quad b[0]+1 \rightarrow$ error } bec. array is treated as a const pointer.

$b[0][0]++ \rightarrow$ ll

$*b[i][j] == *(*(b+i)+j)$

$b[i][j]$ let, $b[i]$ be m

$m[j]$

as per ①

$m[j] = *(m+j)$

$= *b[i]+j$

$b[i][j] = *(*b[i]+j)$ } again as per ① by substituting $m = b[i]$

Consider (universal formula)

$a[i] = *(a+i)$ — ①

$b \rightarrow$ 2d-array

$b \rightarrow 1000$

$*b \rightarrow 1000$ as b is 2d array, it has to be de-ref. two times

$*(b+0) \Rightarrow b[0] \rightarrow$ which represents the base address of $b[0]$

$\Rightarrow *b \rightarrow 1000$

$**b \rightarrow$

*b+1 | *(b+1) where b is 1d array

*(b+0)+1 | *(b+1) = b[1]

b[0]+1 | reason of ①

1-d array | a[i] = *(a+i) — ②

(In 1d array 1 element size gets incremented
b[0]+1 → 1004

*b can be represented as *(b+0)

**b+1 | *(*b+1) | *(*(b+1))
① convert all these pointers in arrays
b[0][0]+1 | *(b[0]+1)
1+1=11 | *(m+1) = m[1]
⇒ b[0][1] = 20

⇒ b[1][0] = 40

3-9-20

Session-65

*b+1 → b[0][0]+1
b+1 → 1012
b[0]+1 → 1004

Scanning & printing a 2-d array

```
#include<stdio.h>
void main()
{
  int b[2][3];
  int r,c,i,j;
  r=sizeof(b)/sizeof(b[0]);
  c=sizeof(b[0])/sizeof(b[0][0]);
  pf("Enter the elements \n");
  for(i=0;i<r;i++) // scanning
  for(j=0;j<c;j++)// scanning inner loop
```

```
        scanf("%d", &b[i][j]);
    printf("=========\n");
    for(i=0; i<2; i++)
    {
        for(j=0; j<j; j++)
            printf("%d", b[i][j]);
    printf("\n");
    }
}
```

→ `int b[2][3] = {{10,20,30},{40,50,60}};` ✓

`int b[][3] = {{10,20,30},{40,50,60}};` ✓

`int b[2][] = {{10,20,30},{40,50,60}};` ✗ error

Note:- Providing no of elements is optional but providing no. of sub-elements is compulsory during initialization.

`int b[2][3] = {{10,20},{30,40}};` ✓

`int b[2][3] = {10,20,30,40};` ✓

Write a program to scan 5 strings & print on the screen

```
#include<stdio.h>
#include<string.h>
void main()
{
    int s[5][10];
    printf("%ld\n", sizeof(s));
    printf("%ld\n", sizeof(s[0]));
    printf("%ld\n", sizeof(s[0][0]));
    int i, ele;
    ele = sizeof(s)/sizeof(s[0]);
```

o/p:
→ 50
10
1

```
for (i=0 ; i<ele ; i++)
 scanf(" %s", s[i]);
for( i=0 ; i<ele ; i++)
 Pf(" %s  %ld \n", s[i], strlen (s[i]));
}
```

Sorting the string in ascending order (bubble sort)

```
#include <stdio.h>
#include <string.h>
 void main()
 {
    char  s[5][10], t[10];
    int ele , i, j;
    ele = sizeof(s) / sizeof(s[0]);
    for(i=0;i<ele;i++)
    scanf("%s", s[i]);
    Pf("-befor--\n");
    for( i=0; i<ele; i++)
     Pf("%d \n", s[i]);
    for( i=0; i<ele-1; i++)
     {
       for (strcmp(s[j] ; s[j+1])>0)
         {
            strcpy( t, s[j]);
            strcpy(s[j], s[j+1]);
            strcpy (s[j+1], t);
         }
     }
}
   Pf(" After \n");
  for( i=0 ; i<ele; i++)
  Pf(" %s\n", s[i]);
```

Session-66

4-9-20

Array of Pointers:-

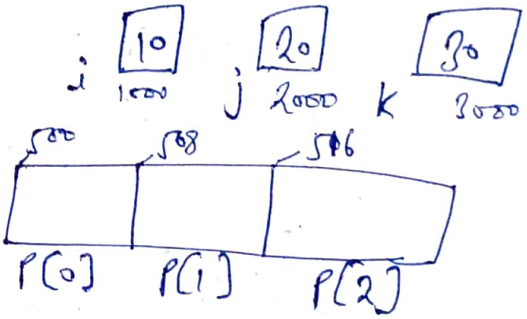
Syntax of dec. of an array of pointers

datatype *variablename[];

eg:- int *p[5];

→ p is an array of 5 int pointer.

```
#include<stdio.h
void main()
{
   int i=10, j=20, k=30;
   int *p[3];
   p[0]=&i;
   p[1]=&j;
   p[2]=&k;
}
```



① What is the type of p?

→ p is an array of 3 int. pointer.

② what is the size of p?

→ 8 X 3 = 24 bytes.

③ what is the size of p[0]?

→ 8 bytes.

④ p++ is possible or not?

→ No p is an array (const pointer)

⑤ p[0]++ is possible or not?

→ Yes, because p[0] is integer pointer (4 bytes inc)

⑥ p ⇒ 500 p+1 → 508

⑦ &p ⇒ 500

&p+1 ⇒ 524 (total array gets inc)

p[0] → 1000

p[0]+1 → 1004

*p[0] → 10 first [] index is done because index operator has more priority.

*p[1] → 20

*p[2] → 30

p[0] = 10 ✗ → it is wrong dealing with the pointer

Not a compile time error but runtime error bec. p[0] holds the addd of 10 and not 10

→ p is 500 bec. array name represents base address

→ *p[0], *p[1], *p[2] can also be written as p[0][0], p[1][0], p[2][0] resp...

for eg. consider *p[0] where m = p[0]

*m

(1) *(m+0) = m[0]

→ p[0][0] *p[0] = p[0][0]

value of p[0][1]

p[0][1] = *(*(p+0)+1)

500 holds 1000

((500)+1)

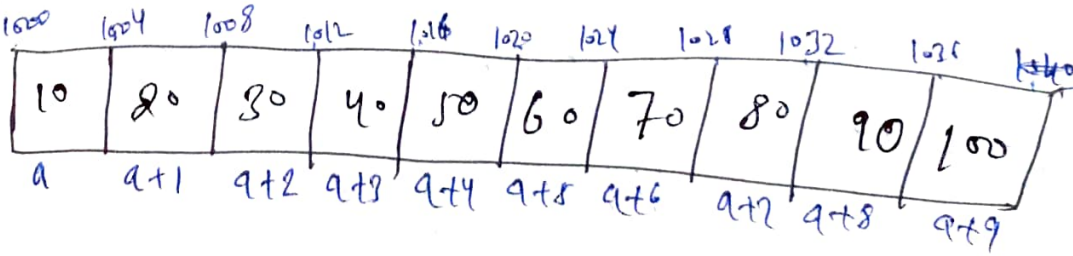
*(1000+1)

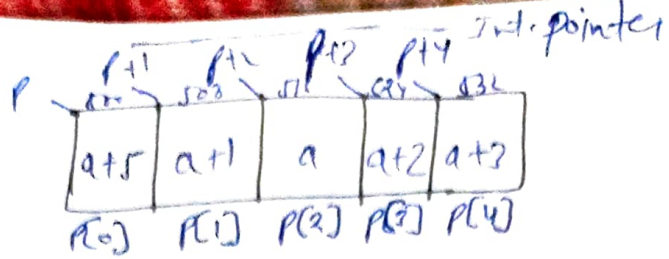
is an int pointer now

⇒ 1004 → 10

Example:-

```
void main()
{
int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
int *p[5] = {a+5, a+1, a, a+2, a+3};
}
```





→ Here p[1][1] = *(*(p+1)+1)
= *(a+1+1)
= *(a+2) → Value = 30

→ p[0][3] = *(a+5+3)
= *(a+8) → Value = 90

→ p[2][1] = *(*(p+2)+1)
= *(*(1016)+1)
= *(1004) = 20

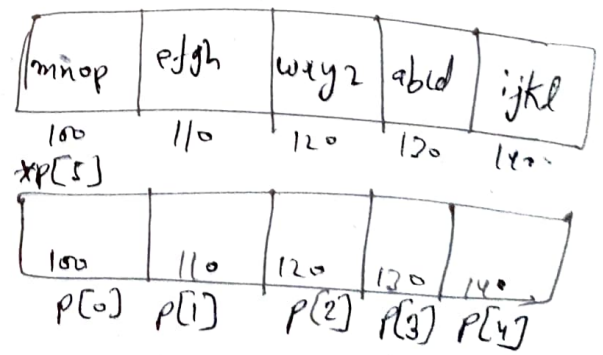
→ *(p+3) = *(524) = 1008

→ p[1][1] = *(*(p+1)+1)
= *(508+1)
= *(1004+1)
= *(1008)
= 30

Session-6]

5-9-2020

char s[5][10];
char *p[5];
when sorting is done in s[5][10], it is physical sorting.
But here we are not supposed to disturb s[5][10], so we are sorting through *p[5] this type of sorting is known as Virtual sorting
Here we sort the element in *p[5], w.r.t. the value of the s[5][10].



p[0] = s[0] ✓ p[0] is a pointer

s[0] = p[0] ✗ bcz s[0] is an array (constant)

the o/p should be

p[0]	p[1]	p[2]	p[3]	p[4]
130	120	140	100	120

```
# void main()
{
  char s[5][10], *t;
  char *p[5];
  ele = sizeof(s)/sizeof(s[0]);
  for(i=0; i<ele; i++)
    p[i] = s[i]
  for(i=0; i<ele i++)
  pf("%s", p[i]);      // s[i] can also be used.
  pf("Before array\n");
  for(i=0; i<ele; i++)
  pf("%s\n", s[i]);
  pf("Before pointer\n");
  for(i=0; i<ele; i++)
  pf("%s\n", p[i]);
  ////////// /////// main logic
  for(i=0; i<ele-1; i++)
  {
    for(j=0; j<ele-i-1; j++)
    {
      if(strcmp(p[j], p[j+1])> 0)
      { t = p[j];
        p[j] = p[j+1];
      } p[j+1] = t;
  } }
  ////////
```

```
Pf("After array \n");
for(i=0; i<ele; i++)
Pf("%s \n", s[i]);
Pf("After pointer \n");
for(j=0; j<ele; j++)
Pf("%s\n", p[j]); // Sorted value only appears here
}
```

eg:-

```
#include<stdio.h>
void main()
{
  char [][10] = {"ABCD", "EFGH", "IJKL"};
  char *p[] = {"abcd", "efgh", "ijkl"}; //
}
```

(s) is an array of array

(p) is a pointer of array

→ Size of s → 30 bytes

→ Size of p → 8*3 = 24 bytes

String name represents base address

→ If p[0] = s[0]
p[] will lose the values inside the arrays
This is called "memory leak"

Session – 68

07-9-20

* char s[][10] = {"ABCD", "EFGH", "IJKL"};

ABCD	EFGH	IJKL
1000	1010	1020

→ The elements are contigious

What is the type of s?

→ 2-D array

Qn) What is the size of s? → 3 × 10 = 30 bytes.

Qn) What is the type of s[0]? → 1-D array.

Qn) What is the size of s[0]? → 10 bytes.

→ s+1 = 1010

→ 45+1 = 1030

→ s++ ✗ not possible (bec. constant ptr.)

→ s[0]++ ✗ → Not possible error bec. (1-d array)

→ s[0][0]++ ✓ 'A' becomes 'B'

Qn) What is the size of each element in s[0]? → 10 bytes.

Qn) What is the size of each element in s? → 10 bytes

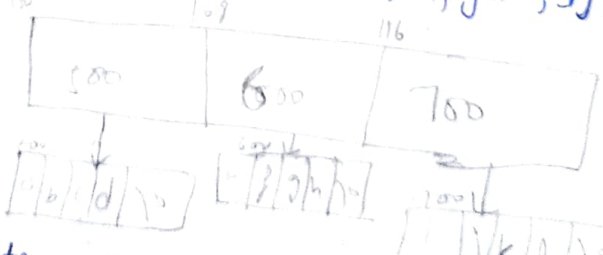
Qn) data type of s[0][0]? → char

→ To copy one string to another

```
char t[10];
strcpy(t, s[0]);
strcpy(s[0], s[1]);
strcpy(s[1], t);
```

ABCD	EFGH
1000	1010

★ `char *p[][] = {"abcd", "efgh", "ijkl"};`



→ These elements are present in code section.

→ The elements may/may not be contiguous.

Qn) size of p? → 8 × 3 = 24 bytes

Qn) How many ele in p? → 3 elements p[0], p[1], p[2]

Qn Type of p[0]?
→ char pointer

Qn what is p; what is p[0]?
p → 100 ; p[0] → 500
→ p+1 = 108
→ &p+1 = 184
→ p++ X No. bec p is an array
→ p[0]++ ✓ bec. p[0] is ptr
↳ 500
→ p[0][0]++ X (it is not syntactical error but a runtime error, bec. in gcc, when a char pointer is initialised it is stored in (read only) code section.
→ p=100 ; *p = 500
→ p is not in code section but its pointer points to code section

Qn size of p[0]
→ 8 bytes
→ ++*p ✓ = 500 becomes 501
→ *p = p[0]
→ *p++ X (Not possible bec. p is an array)

Qn what is size of p?
→ 24 bytes (3 ptrs)
→ *p[0] = p[0][0] = 'a'
→ p+1 = 108

→ s[0] = p[0] X bec s[0] is an array & p[0] is ptr.
→ p[0] = s[0] ✓
Initially p[0] holds 500, Now it holds 100.
The memory '500' can't be accessed now, so it becomes leak
⇒ If we lose the base address of the string then it is called as memory leak

Pointer to an array

```
void main()
{
    int a[5] = {10, 20, 30, 40, 50};
    int *p; // p is an int ptr. It can point to one integer.
```