Module (3)

Date :
Page No.

Functions / Pointers / Arrays

① functions

Introduction :-

function can be execute if call. (received Arguments)
function has return type and formal arguments.

```
int max (int x, int y){      } function definition
  return (x>y)? x:y; }
        ↑ formal arguments
```

```
void main() {
  int a=10, b=20
  int maximum = max (a,b);   # function call.
                ↑ Actual parameter
    printf ("maximum is", maximum);
  }
```

→ Actual and formal parameters.

If function is declared below the main() then declare the function above main() as

```
# include < >
  int max (int, int);   } Declaration
  void main()  or
```

you can declare the complete function above main().

We can't return an array because it contains multiple elements.

i) Call by value
ii) Call by reference

C provide only call by value. If we change the formal parameters the actual parameters will not changed.

Ex →

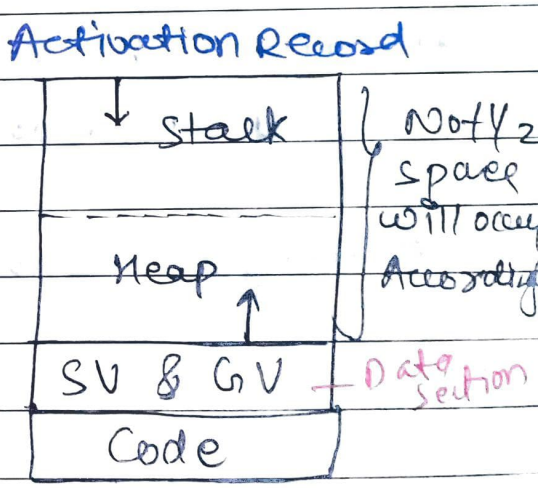
```
void fun (int y) {
    y = 30;
}
void main () {
    int x = 20;
    fun (x);
    printf ("%d", x);
}
```

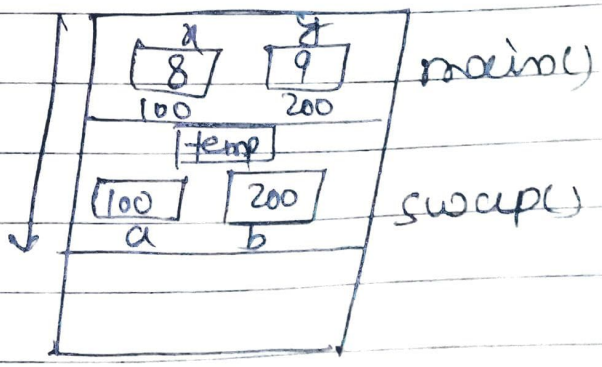
Call by value

o/p → 20

Swapping two variables :-

```
# include
void swap (int *, int *)
void main() {
    int x, y;
    scanf ("%d %d", &x, &y);
    Swap (&x, &y);
    printf ("After swapping x: %d
            y: %d", x, y);
}
Void swap (int *a, int *b) {
    int temp;
    temp = *b;
    *b = *a;
    *a = temp;}
```





Date
Page No.

In the previous case we passed the address value so, it is also a call by value not call by reference

Write a function to implement pow() function:

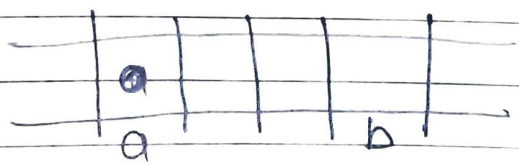
```
int power (int base, int n)
{
    int p;
for (p=1; n>0; --n){
        p=p* base;
    }
    return p;
}
```

```
# include <math.h>
pow(a, b)
a^b
```

(1) Always take care of the typecasting of variable and its format specifier.

(2) Always choose a suitable type for variable.

Pointers in C



In C memory accessed as a one-D array.

Char a (1 byte say)
int b (2 byte say)

Declare a variable make the access easy.

Date :
Page No.

Ways to access a variable -

i) By name → easy to store
ii) By address (fast to access) → fast to implement

If you access a variable by name then it converts into logical address and then physical address.

Dynamically created data structure accessed using pointers.

Pointer is a variable supposed to contain the address of another variable or entity.

Declaration - int *p;

p = &b; but p ≠ &(a+b*c)

You can get the address of variables only, not the expression because expressions are not declared in the memory.

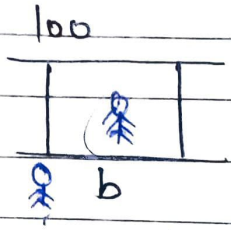
Pointer saved the starting address.

for any type of variable pointer variable always have a same size.

We can also declare as char *p; or float *pt but it arises a problem called address arithmatic.

* → dereferencing or indirection operator.

& → reference operator

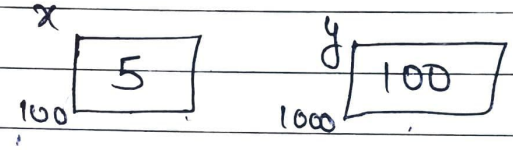



```
int *p;
p = &b;
```

then "p" means you are at the door of variable and "*p" means you are inside the variable.

Ex →

```
int x = 5;
int *y = &x;
print("%d", x);   → 5
print("%u", &x);  → 100
print("%u", y);   → 100
print("%d", *y);  → 5
print("%u", &y);  → 1000
```



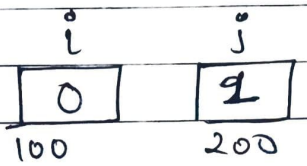
& → gives address , * → gives values at address.

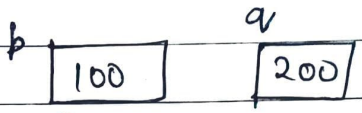
Q What will be the output [Gate - 2010)

```
void f(int *p, int *q)
{
  p = q;   — i)
  *p = 2;  — ii)
}

int i = 0, j = 1;
int main()
{
  f(&i, &j);
  Printf("%d %d", i, j);
  return 0;
}
```

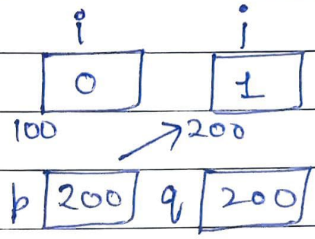
i) 2 2
ii) 2 1
iii) 0 1
iv) 0 2 ✓

Sol^{n} Initially

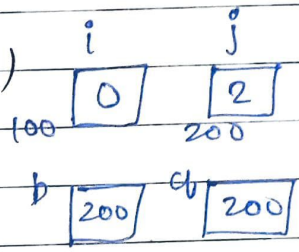




After –(i)



After (ii)



Ans i = 0, j = 2 (iv)

Pointers and function

```
main() {
int a=3, b=4;
Swap(a,b);
}
void swap (int x, int y){
  int temp;
  temp= x;
   x = y;
   y = temp; }
```

this code will swap only formal parameters x, y bcoz they are locally declard but a and b remains unswapped actual parameters.

Sol^{n} Sent → swap(&a, &b);

Receive → void swap(int *x, int *y);

```
int temp;
temp = *x;   (a)
*x = *y;     (b)
*y = temp;
```

*x → a
*y → b

so it will swap the value on a and b

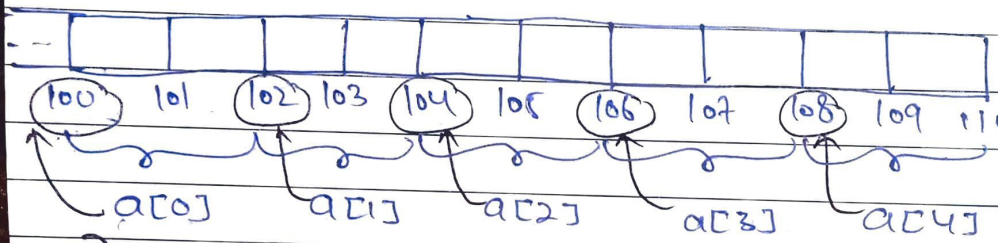
Date
Page No.

Pointers and Arrays -

a[5] Blog of contigous m/m space
System is byte addressable → Every byte is going to get a different addressable

Systems can be word addressable also.

Ex int a[5]; # a is not a variable just a mnemonic.
Byte addressable
int = 2 Byte, int *p; p = &a[0];



(I) "a" = 100
"*a" = a[0]
"a+1" = 102
"a+2" = 104
"a+3" = 106
"a+4" = 108

In general
a+i = a+(i x size of data type)
data is int in our case.

(II) *(a+1) = a[1]
*(a+2) = a[2] - - soon.

(III) # (a+3) = &a[3]
(a+1) = &a[1]

(IV) If p 100 then p = p+1 implies p 102

Date :
Page No.

So, delaration of datatype for a pointer as a int is appropriate due to arithmatic of addition.

$*(p+3) = a[3] = p[3]$

$(p+3) = a+3 = \&a[3]$

Difference between a and p

a is mnemonic but p is variable so,

```
a = a+1 ;
a++ ;
a = p ;
```
] Not possible

```
p = p+1 ;
p++ ;
p = a ;
```
] Possible

If we passed array through function, it accept to abbaut the base address and access as a pointer.

Pointer arithmatic and address arithmatic

Valid Pointers Operation

① Assignment of pointers of the same data type.

```
char *c;
int *p, *q, a[];
p = q;
q = p;
p = a;
a = p;   # Not possible
```

we can assign one pointer to other if they have same datatype otherwise do typecasting.

So,

```
p = (int *) c
```

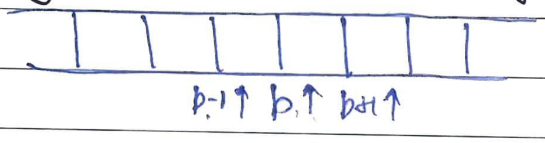
This typecasting don't be needed if c is a wide pointer.

Although, there is not much issue with declaration of pointer in any type but on increment or any other operation, it generates confusion of scalability also.

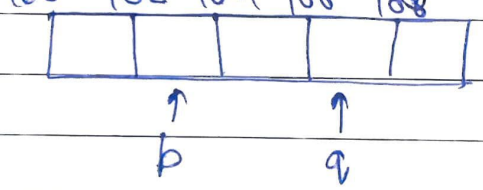
a+i = a + i(size of char) } Both will generate
a+i = a+i(size of int) }

But pointer have to opt the same size for every variables.

(2) Adding or Substracting a pointer an integers



(3) substracting or comparing two pointers of same array.




```
if (p<q)
printf ("p pointes no. lesser than q")
```

or,

number between 'p' and 'q'
" q-p+1
$= \left(\frac{106-102}{2}\right)+1 = 3$

3 numbers including p and q

We devide by 2 due to memory scaled up by 2.

④ Assigning or comparing to zero.

define NULL 0 ↳ Looks good to use NULL either 0.
then, (p==0) can be use

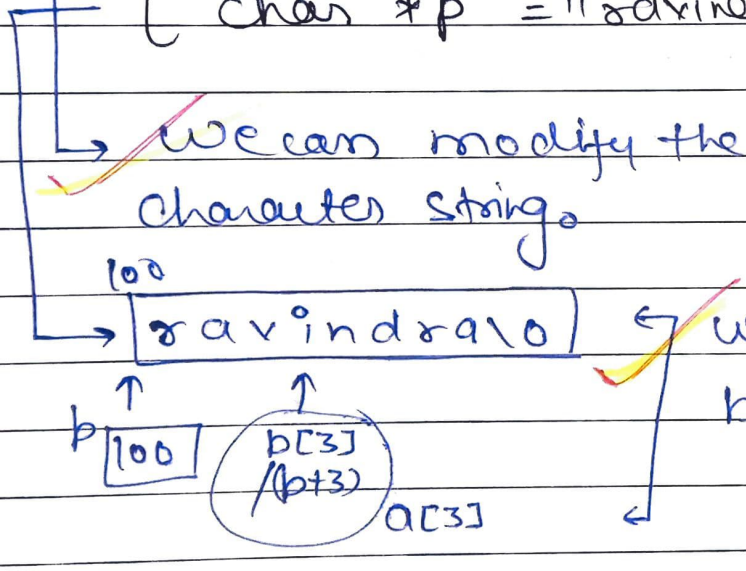
Character pointers and functions :-

printf ("aryan")
"aryan" is a character array but treated here as a string constant.

```
{ Char a[] = "ravindra";
  char *p  = "ravindra";
```

r	a	v	i	n	d	r	a	\0
0	1	2	3	4	5	6	7	8

↳ We can modify the character array but not character string.



↳ We can't modify the value but only access using pointers

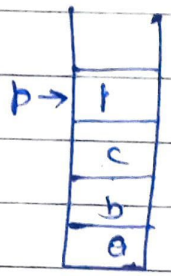
Ex

```
/* strcpy : copy t to s */
void strcpy (char *s, char*t)
{
  while ( *s++ = *t++);
}
```

Good Example of coding Technique

On assigning value NULL complete expression become NULL and move out from loop.

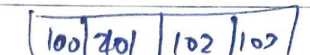
Example of stack




```
Push()
  *p = value;  }  *p++ = value
  p++;
Pop()
  p--;         }  value = *--p;
  value = *p;
```

good

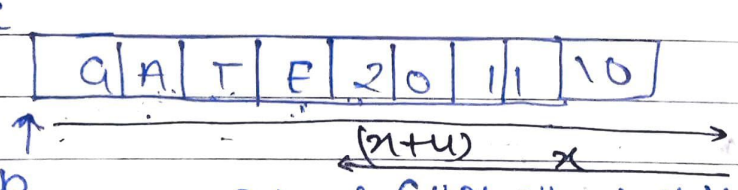
Q What does the following fragment of C-program print Gate-[2011]



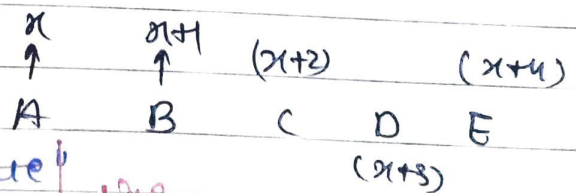

```
char C[] = "GATE2011";
char *p = C;          # pointing C[0]
printf("%s", p + p[3] - p[1]);
```

(A) GATE 2011 B) E2011 C) 2011 D) 011

Soln If printf("%s", p) it means start from p[0] till '\0'.




```
printf("%s", p+4);
o/p ATE 2011
```



p → G

p[3] → "E" # takes its "Ascii value"

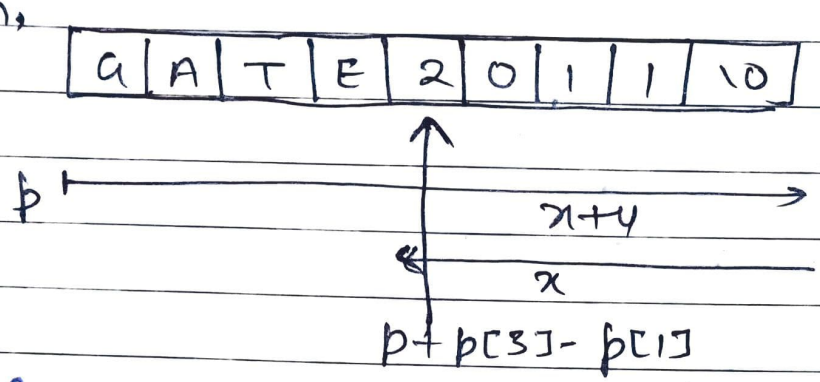
p[1] = "A" # take Ascii value

Use ASCII value

p + p[3] → moves the cursor (x+4) away from p.

p + p[3] - p[1] → come back the cursor by x.

then,



So, answer will be (2011) → (C)

Ex printf ("ravi");
printf ("%d", printf ("ravi"));
→ return the no. of characters.

o/p → ravi
ravi 4

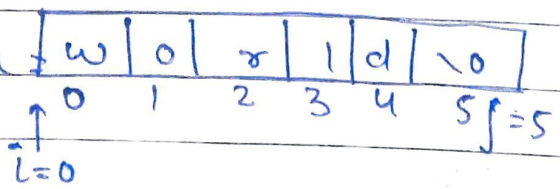
If you give (%s) then it will print till '\0' from the given starting address.

Q what is the o/p printed by following C-program (2008-IT)

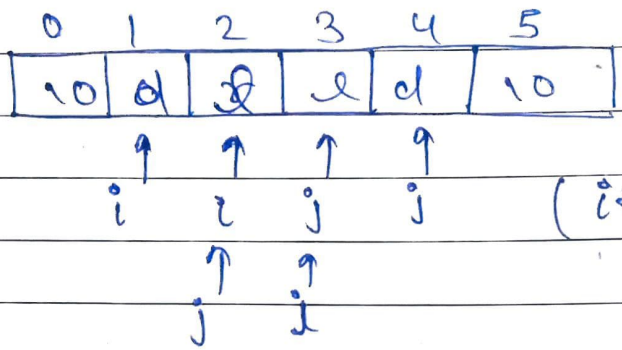
```
#include <stdio.h>
int main()
{
Char a[6] = "world";
Int i, j;
for (i=0, j=5 ; i<j ; a[i++] = a[j--]);
printf ("%s\n", a);
```

A) dlrow B) Nullstring C) dlrdl D) worow

Soln



before increment or decrement, do a[i] = a[j]

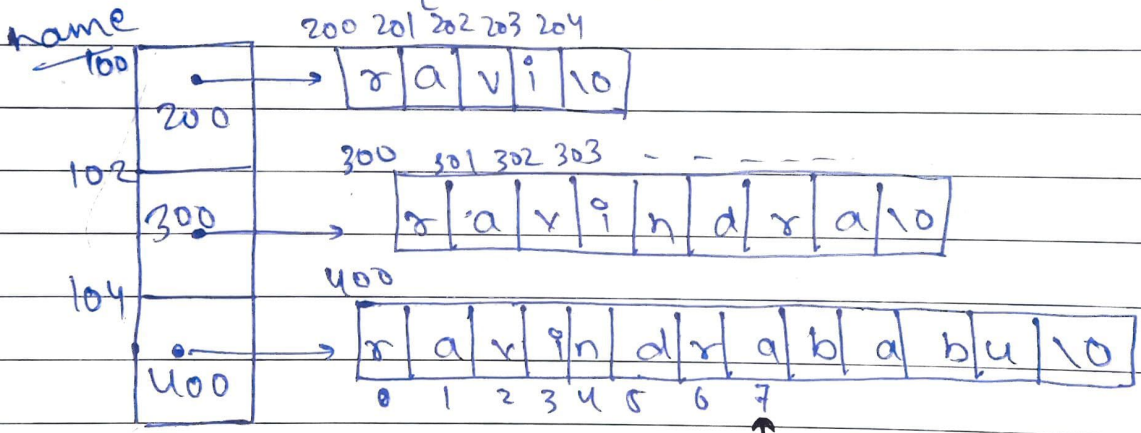


So a[0] = a = '\0'

So, output will be NULL String (B)

Array of pointers and multidimension array

Char *name[] = {"ravi", "ravindra", "ravindrababu"};



To print ravindra → printf ("%s", *(name+1))

if printf ("%s", *name+1)
→ print [avi]

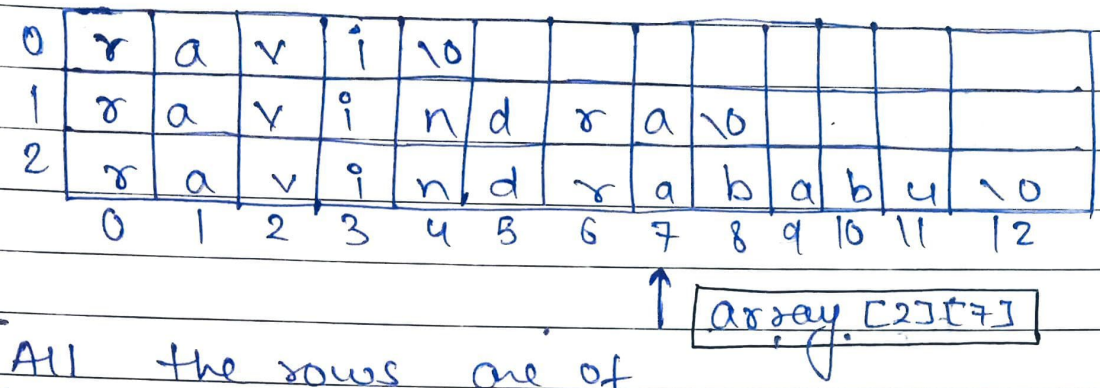
printf ("%c", (*(name+2)+7)) → It will print 'a' in third string.

name[2][7]

Using multidimension array —

Char array[3][13] = {"ravi", "ravindra", "ravindrababu"};

Date :
Page No.



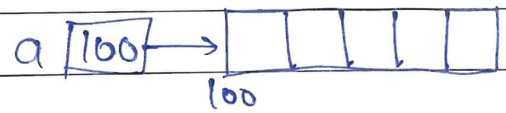
* [All the rows are of same size, either your space used or not.

* [But in array pointers we have liberty to take row size as needed. (Size may be different)

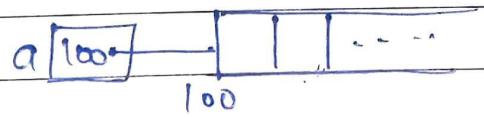
* [Array in pointer is more useful than multi-dimension array but little tough to access.

Multidimensional arrays, pointers and function calls :-

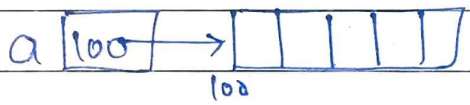
fun(int a[5])



fun(int a[])

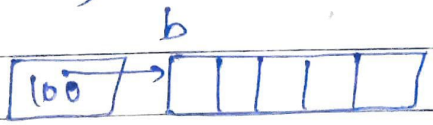


fun(int *a)



All are same.

int b[5]
fun(b)
fun(int a[5])



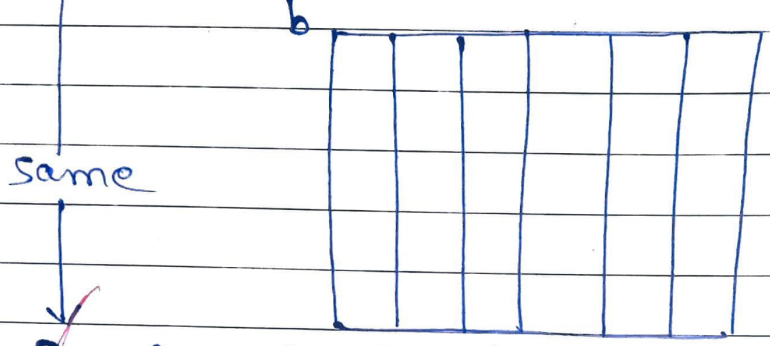
int b[5][6]

fun(b)

fun(int a[5][6]) → This is also fine

fun(int a[][6])

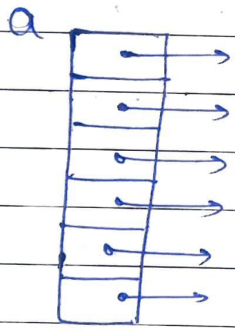
↳ But it is better approach.



fun(int (*a)[6])

↳ a is an pointer pointed to 6 elements array.

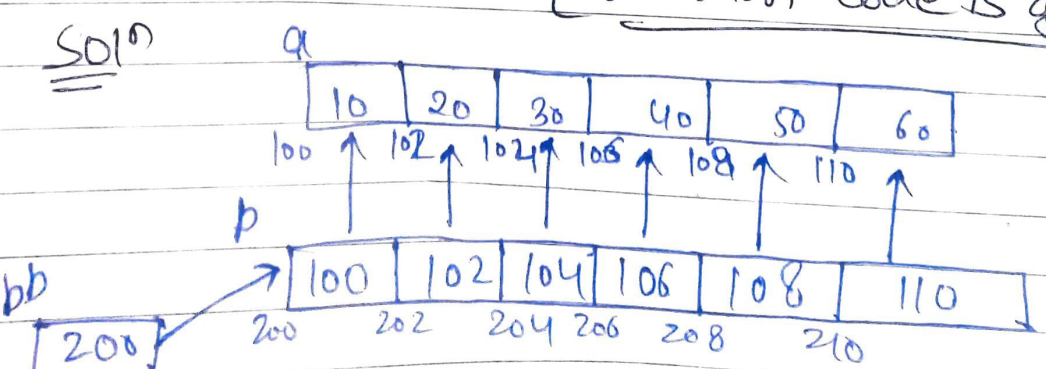
int *a[6] → a is an array pointed to 6 different integers.



Question on pointer to pointers and array of pointers?

[Question code is given after solution]

Soln



Date :
Page No.

$a[3] = *(a+3) = *(p[3]) = **(p+3)$

$= **(pp+3)$

After line (4) $p \rightarrow (p+1)$

Output :-

Line (5) $pp = 202$

$pp - p = \left(\frac{202-200}{2}\right) = 1$

$*pp - a = \left(\frac{102-100}{2}\right) = 1$

$**pp = 20$

Line 6 → $*pp++ = 104$ pp [204]

Line 7 → $pp - p = \frac{204-200}{2} = 2$

$*pp - a = \frac{104-100}{2} = 2$

$**pp = 30$

Line-8 → ~~Address~~ Value [104 → 106] in p[] and start pointing value 40 in a[].

Line 9 → $pp - p = \frac{204-200}{2} = 2$, $*pp - a = \frac{106-100}{2} = 3$

$**pp = 40$

Date :
Page No.

Line 10 → (++(*(*pp)))

value 40 → 41 in a[].

Line 11 pp-p = 2

*pp-a = 3

**pp = 41

Solution to previous Page

Problem Code

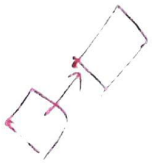
```
main() {
int a[] = {10, 20, 30, 40, 50, 60};          —(1)
int *p[] = {a, a+1, a+2, a+3, a+4, a+5, a+6}; —(2)
int **pp = p;                                 —(3)
(4)  pp++;
(5)  printf("%d %d %d", pp-p, *pp-a, **pp);
(6)          *pp++;
(7)  printf("%d%d%d", pp-p, *pp-a, **pp);
(8)          ++**pp;
(9)  printf("%d %d %d", pp-p, *pp-a, **pp);
(10)  ++***pp;
(11)  printf("%d%d%d", pp-p, *pp-a, **pp);
}
```

C1

C2

Solution of this question is discussed before the question.

```
#include <stdio.h>

(C1);        # In problem code on left Page.
int r = 10;
  int *q = &r;
  int **rq = &q;
print ("facts = = = = = = = = \n");
printf (" Value of r and adress of r is: %d %u\n", r, &r);
print (" Value of q and adrs of q is: %d %u \n", q, &q);
printf ("Value of rq and address of rq is: %d %u\n", rq, &rq);
   print (" - - - - - \n");

printf (" value of r ⇒ r = %d, *q = %d, **rq = %u\n",
                    r, *q, **rq);
printf ("Add. of r ⇒ &r = %u, q = %u, **rq = %u, &r, q,
                    *rq );
printf (" Value of q ⇒ q = %u, *rq = %u \n", q, *rq );
printf (" adress of rq ⇒ &rq = %u\n" &rq );
printf (" address of q ⇒ &q = %u, rq = %u \n", &q, rq);
  printf (" Value of rq ⇒ &rq = %u\n", &rq );

// printf (" Value of *q, *rq, **rq using name: %u
                    %u %u \n", *q, *rq, **rq);

printf (" = = = = = = = = = = = = \n");

(C2);
```



→ May be ask in product based.
→ Not imp or even ask in Gate Exam
→ Just for the sake of knowledge

Date :
Page No.

Pointers to function

int * fp (int, int) → fp is a function that returns a pointer to integer.

```
int (*fp)(int, int);
fp = sum;
```

```
S = sum (5, 6);
S = (*fp)(5, 6)
```

Both are equivalent

void *(*gp) (void *, void *) → gp is a pointer which points to a function of arguments type void * and return a void*

int * sum (int*, int*)

If we want to declare sum() using gp then we need to do typecasting.

So,

```
gp = (void * (void*, void*)) sum;
```

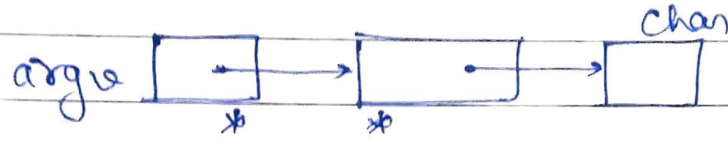
On assigning information from one type of pointer to another or do typecasting, there is no loss of information.

—x—

Some Complex declaration → Just for sake of knowledge, not for Gate exam

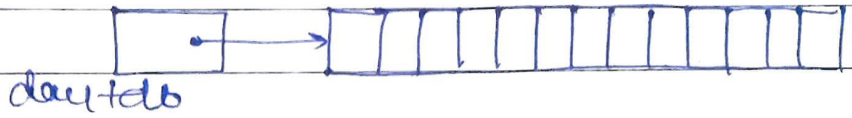
① char **argv

Passing a function to main() is not very easy because main() is not called by the users.



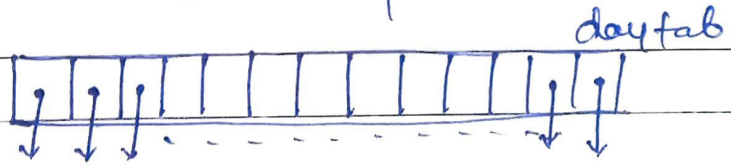
② int (*daytab)[13]

Daytab is a pointer to an array of 13 integers.



③ int *daytab[13]

daytab is an array of 13 integer which are all pointers to integers.



④ void *comp()

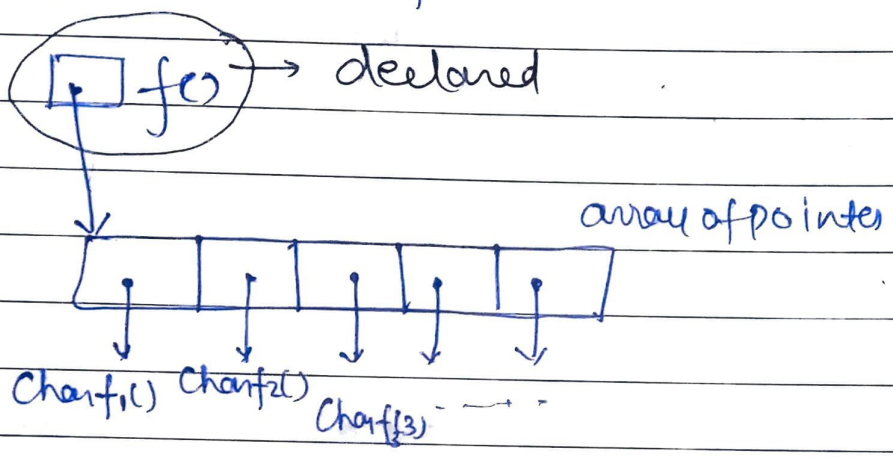
comp is function, returning a pointer to void.

⑤ char (*comp)()

comp is a pointer to a function, that returns a character.

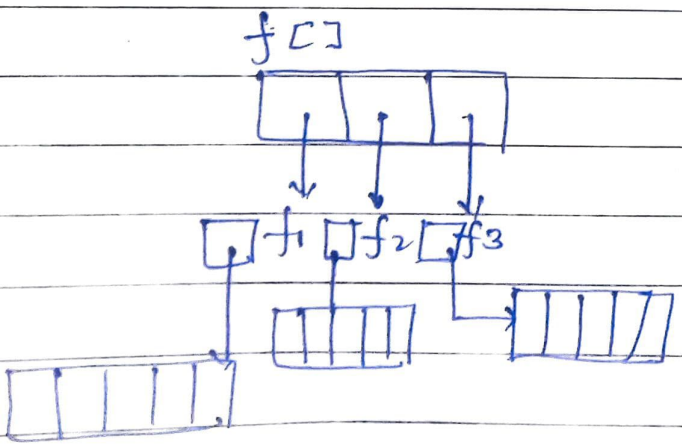
(6) char (*(*f())[])()

f is a fn returns a pointer to the array of pointers to a function returns a characters



(7) char (*(*f[3])())[5]

f is an array of 3 elements. which are pointers to functions returning pointer to array of 5 elements which are characters



Extra Information regarding

Date :
Page No.

((i++)%2) → In this instruction first modulus operator operate then increment will occur.

$\binom{m}{n}$ = mCn returns 1 when mC0 or mCm
→ (n==0) || (m==n)

Strings |00|00|0 and 0|00|00|0 are same.

Reason behind- declare a pointer normally as int* is pointer arithmatic

Specifier	Storage	Initial V.	Scope	Life
auto	Stack	Garbage	Within block	End of block
extern	Dataseg.	Zero	Global	Till end of prog.
Static	Dataseg	Zero	Within block	Till end of prog.
register	Register	Garbage	Within block	End of block.