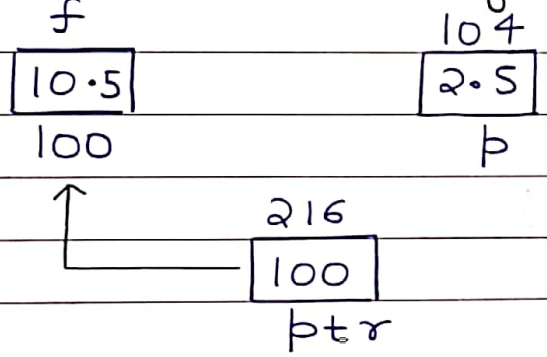
Questions on Pointers

① float f = 10.5;
float p = 2.5;
float *ptr = &f;
(*ptr)++;
*ptr = p;

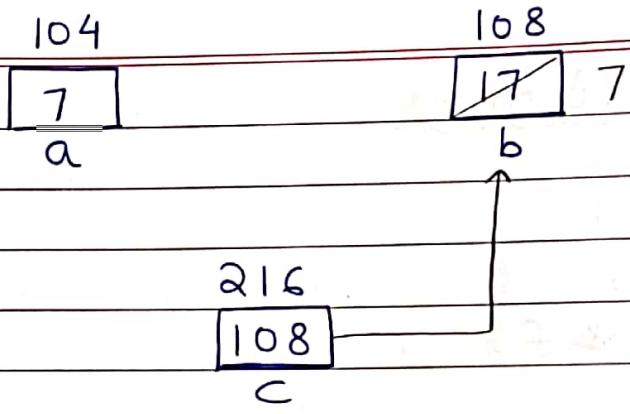
What are the values of *ptr, f and p?



(*ptr)++ ⇒ 10.5 + 1 = 11.5 } f
(*ptr) = p
*ptr = 2.5
This means value present at address stored in ptr is changed to 2.5

f = 2.5
p = 2.5
*ptr = 2.5
So all the values are equal to 2.5

② int a = 7;
int b = 17;
int *c = &b;
*c = 7
What are values of a & b?



*c = 7
means we have to change the value of b.

a = 7
b = 7

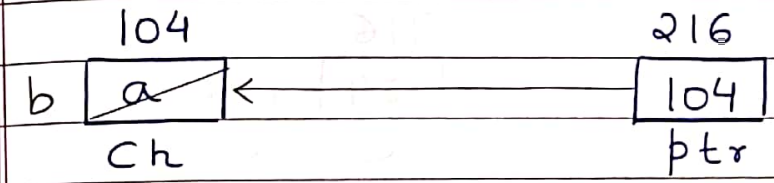
(3) int *ptr = 0;
int a = 10;
*ptr = a;
What is the value of *ptr?

*ptr = 0 ∴ null pointer
Dereferencing null pointer will give the run time error.

(4) Which of the following gives the memory address of variable b pointed by pointer a i.e
int b = 10;
int *a = &b;

a is the pointer which has stored the address of b.

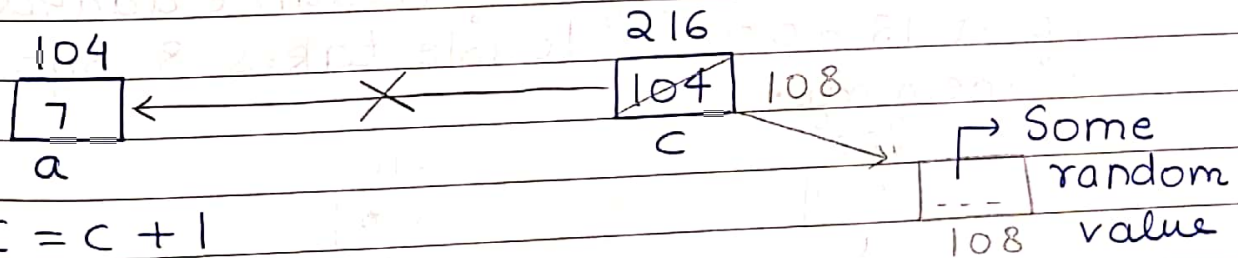
(5) char ch = 'a';
char *ptr = &ch;
ch++;
What is the value of *ptr?



ch++ will be b as a has ASCII value of 97 and hence 97+1 = 98 is ASCII value of b.

*ptr = b

(6) int a = 7;
int *c = &a;
c = c+1;
What is the value of a and *c?

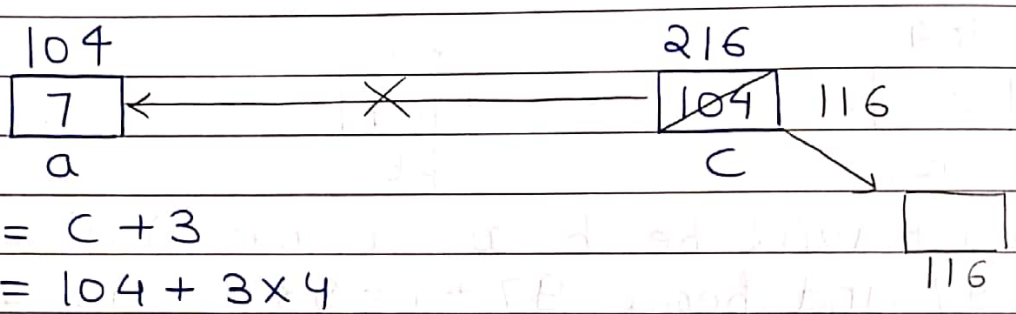


c = c+1
= 104 + 1×4
= 104+4
= 108

Now address 108 is not pointing to a & is pointing to some random / garbage value.

a = 7
*c = garbage value.

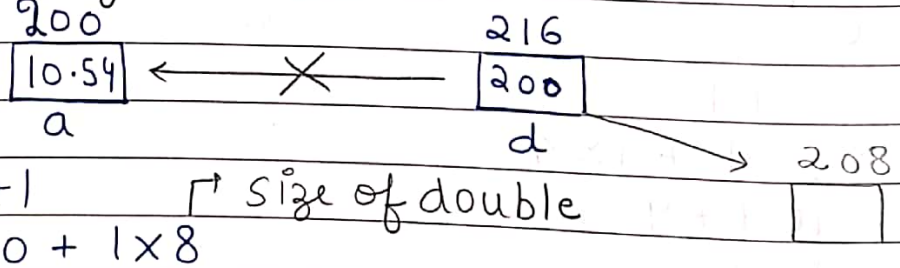
(7) int a = 7;
int * c = &a;
c = c + 3;
What is the value of c?



c = c + 3
c = 104 + 3 × 4
c = 104 + 12
c = 116

Ans = 116 } Here we are not printing * c and hence garbage value is not coming.

(8) double a = 10.54;
double * d = &a;
d = d + 1;
What is value of d? Assume address of a is 200 & double takes 8 bytes of memory.



d = d + 1 ⌜ size of double
d = 200 + 1 × 8
d = 200 + 8
d = 208

Ans = 208

(9) int a [5]; Array (Contiguous blocks)

int *c;
sizeof(a) and sizeof(c) will have what values? Assume integer takes 4 bytes size & integer pointer takes 8 bytes.

5 × 4 = 20 bytes
pointer has size 8 bytes

sizeof(a) → 20
sizeof(c) → 8

(10) int a[] = {1, 2, 3, 4};
What is value of *a and *(a+1)

1	2	3	4
104	108	112	116

a → base address of array
*a → value at address 104 i.e 1
*(a+1) → value at address (104 + 1×4)
(104 + 4)
(108) i.e 2

*a → 1
*(a+1) → 2

(11) int a[3] = {1, 2, 3};
what is the value of *(a+2)?

*(a+n) ⇒ value at index = n
Here n = 2 i.e a[2] i.e 3

*(a+2) = 3

(12) int a[] = {1, 2, 3, 4};
int *p = a++;
What is value of *p ?

Here post increment is used. Here what happens is that a is the constant pointer & hence a++ can't be done & hence we face an error here. This is constant pointer as array is created in stack & not heap. We can do a++ if we have created the array on the heap.

(13) int arr[] = {4, 5, 6, 7};
int *p = (arr+1);
What is the value of *arr + 9 ?

p is pointing to the address of 1st index of array.

*arr ⇒ value present at base address i.e. 4

*arr + 9 ⇒ 4 + 9 = 13

(14) char b[] = "xyz";
char *c = &b[0];
What is value of c assuming address of 0th index of array is 200?

c is pointing to base address & hence 200 is stored in c. The implementation of cout is different in case of char

arrays. Value get printed until we get null character.

cout << b → xyz (→ pointing to base address)

cout << c → xyz (→ pointing to base address)

(15) char s[] = "hello";
char *p = s;
What is value of s[0] & p[0] ?
p is pointing to base address & s is the base address

s[0] → h
p[0] → h

(16) char arr[20];
int i;
for (i=0; i<10; i++)
*(arr+i) = 65+i;
*(arr+i) = '\0';
What is value of arr?

*(arr+i) ⇒ arr[i]
arr[0] = 'A' → 65+0 = 65
arr[1] = 'B' → 65+1 = 66
arr[2] = 'C' → 65+2 = 67
arr[3] = 'D' → 65+3 = 68
arr[4] = 'E' → 65+4 = 69
arr[5] = 'F' → 65+5 = 70
arr[6] = 'G' → 65+6 = 71
arr[7] = 'H' → 65+7 = 72
arr[8] = 'I' → 65+8 = 73
arr[9] = 'J' → 65+9 = 74
arr[10] = '\0'

cout << arr; →
ABCDEFGHIJ
↓ Prints until we get null character.

(17)
```
char *ptr;
char str[] = "abcdefg";
ptr = str;
ptr += 5;
```
What is the value of ptr?

ptr is pointing to base address of str.
p = p+5
Now p is pointing to f value. So cout << ptr will print until we get the null character

O/p → fg

(18)
```
int numbers[5];
int *p;
p = numbers;
*p = 10;
p = &number[2];
*p = 20;
p--;
*p = 30;
p = numbers + 3;
*p = 40;
p = numbers;
*(p+4) = 50;
```

```
for (int i=0; i<5; i++)
        cout << numbers[i] << " ";
```

Initially p points to the base address of array.

$*p = 10 \Rightarrow p[0] = 10$

p is now pointed to address of 2nd index

$*p = 20 \Rightarrow p[2] = 20$

$p--; \Rightarrow p = p - 1 \times 4$

$p = p - 4$

Now p points to 1st index of array.

$p[1] = 30$ by $*p = 30$

$p = numbers + 3 \Rightarrow p = \&numbers[3]$

Now p points to 3rd index

$*p = 40 \Rightarrow p[3] = 40$

$p = numbers \Rightarrow$ p points to base address

$*(p+4) \Rightarrow p[4] = 50$

indexes	values
0 →	10
1 →	30
2 →	20
3 →	40
4 →	50

(19)
```
char st[] = "ABCD";
for (int i=0; st[i] != '\0'; i++)
    cout << st[i] << *(st) + i << *(i+st)
    << i[st] << endl;
```

i = 0

A 65 A A

↳ typecasting due to implementation of cout

i = 1

B 66 B B

Similarly for $i = 2$ & $i = 3$

O/p → A 65 A A
B 66 B B
C 67 C C
D 68 D D

Note → cout << ('A' + 0);
↳ 65 + 0 = 65 is o/p.

(20) float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
float *ptr1 = &arr[0];
float *ptr2 = ptr1 + 3;
What is the value of

(i) *ptr2

ptr1 is pointing to base address & hence ptr1 + 3 will point to 3rd index.

ptr2 will point to 3rd index & hence
*ptr2 = arr[3] = 90.5

(ii) ptr2 - ptr1

200	204	208	212	216
12.5	10.0	13.5	90.5	0.5
↑			↑	
ptr1			ptr2	

ptr2 - ptr1 = 212 - 200 = 12
Divide 12 by 4 as 4 is the size of float

$\frac{12}{4} = 3$ ↳ Pointer arithemetic

(21) void changeSign (int *p) {

```
    *p = (*p) * (-1)
}
main() {
  int a = 10;
  changeSign(&a);
}
```

What is value of a?
Here value is updated as the address is passed in the function.

*p = (10) × (-1) = -10

Hence a becomes -10.

(22) void fun (int a[]) { ← a+1 was passed.

```
    cout << a[0] << " ";
}
main() {
  int a[] = {1, 2, 3, 4}
  fun (a+1);
  cout << a[0];
}
```

0	1	2
2	3	4

0	1	2	3
1	2	3	4

a+1 is basically the address of 1st index of array. Also array is passed by reference only.

o/p → 2 1

(23) void square (int *p) {

```
    int a = 10;
    p = &a;
```

```
*p = (*p) * (*p);
}
main() {
    int a = 10;
    square(&a);
}
```

What is value of a ?

Here in function we have basically changed the address stored inside it to the address of local variable of function. Hence original a will remain as it is.

O/p → Value of a = 10.

(24)

```
void Q(int z){
    z = z + z;
    cout << z << " ";
}
void P(int *y){
    int x = *y + 2;
    Q(x);
    *y = x - 1;
    cout << x << " ";
}
main() {
    int x = 5;
    P(&x);
    cout << x;
}
```

In function Q, actual value is not updated due to pass by value concept

$x = 5$ ⌐ x of P

P(&x) → $x = 5 + 2 = 7$

Q(x) → $z = 7 + 7 = 14$

$*y = 7 - 1 = 6$

$x = 7$

↳ of P

due to this

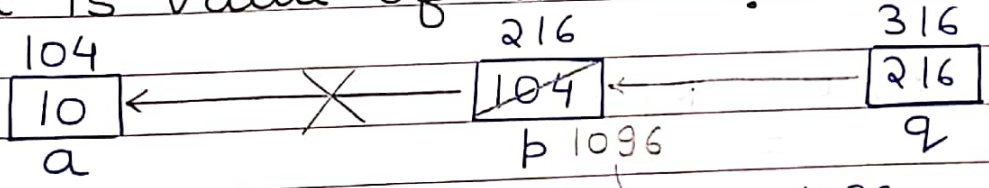
$x = 6$

↳ main

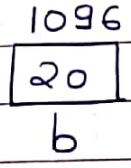
O/p → 14 7 6

(25)
```
int a = 10;
int *p = &a;
int **q = &p;
int b = 20;
*q = &b;
(*p)++;
```

What is value of a & b ?



*q = address of b.



(*p)++ ⇒ 20+1 = 21

a = 10
b = 21
} Ans

(26)
```
int f(int x, int *py, int **ppz) {
    int y, z;
```

```
    **ppz += 1;
    z = **ppz;
    *py = *py + 2;
    y = *py;
    x = x + 3;
    return x+y+z;
}
main(){
    int c, *b, **a;
    c = 4;
    b = &c;
    a = &b;
    cout << f(c, b, a);
}
```

4
c

```
**ppz = **ppz + 1
      =   4 + 1
      = 5
```

5	z = 5
c	

```
*py = *py + 2
    = 5 + 2
    = 7
```

7	y = 7
c	

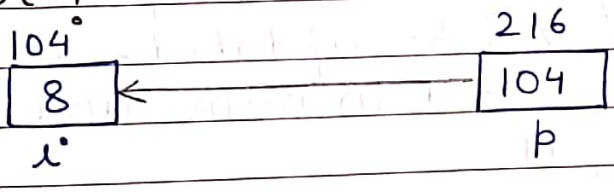
```
x = 4 + 3 = 7
```

↳ Here c was not passed by reference

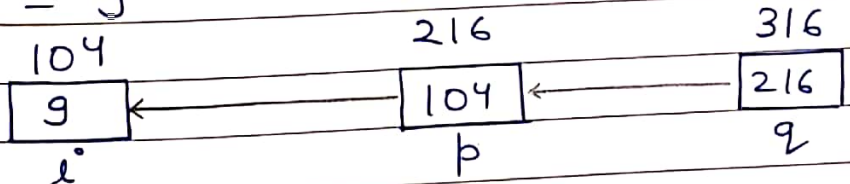
$x + y + 2 = 7 + 7 + 5 = 14 + 5 = 19$

(27)
```
main() {
  int ***r, **q, *p, i = 8;
  p = &i;
  (*p)++;
  q = &p;
  (**q)++;
  r = &q;
}
```

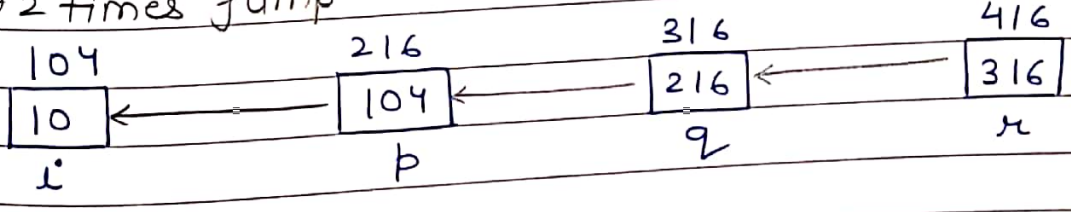
What are the values of *p, **q & ***r ?



*p = *p + 1
= 8 + 1
= 9



(**q)++ ⇒ i = 10
↳ 2 times jump



*p
**q
***r
} All are i i.e 10

O/p → 10 Ans
↳ for all *p, **q, ***r

(28)
```
void increment (int **p) {
        (**p)++;
}
main() {
int num = 10;
int *ptr = &num;
increment (&ptr);
}
```

What is the value of num ?

Here pass by reference concept is used & hence value will be updated. Hence 10+1 = 11 will be stored in num and hence 11 is the output.