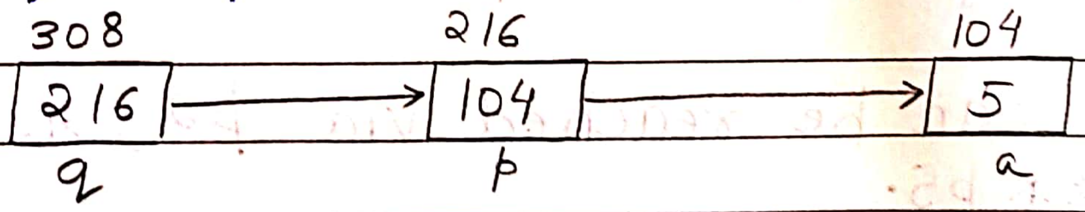
5/03/2023

```
int a = 5                          char ch = 'a';
int * ptr = &a;                    char * p = &ch;
int ** ptr1 = &ptr;
int *** ptr2 = &ptr1;
→ double pointer
```

We just have to add stars.

```
int a = 5;
int * p = &a;
int ** q = &p; // Pointer to pointer.
```



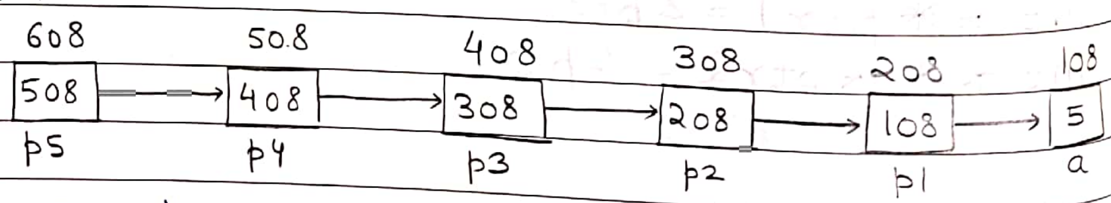
Pointer to pointer ⇒ double pointer.

cout << &a;	104
cout << a;	5
cout << p;	104
cout << &p;	216
cout << *p;	5
cout << q;	216
cout << &q;	308
cout << *q;	104
cout << **q;	5

104 → Address of a
216 → Address of pointer p
5 → Value of a
308 → Address of double pointer q

*q ⇒ Value present at address which is stored in q.
**q ⇒ Value present at address stored in *q i.e. block p value i.e address 104.

Ex→



1) a can be reached via a, *p1, **p2, ***p3, ****p4, *****p5.

2) p1 can be reached via p1, *p2, **p3, ***p4, ****p5

3) p2 can be reached via p2, *p3, **p4, ***p5.

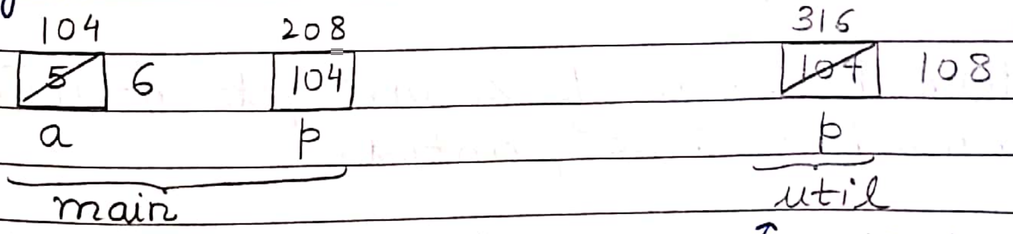
4) p3 can be reached via p3, *p 4, **p5

5) p4 can be reached via p4, *p5

6) p5 can be reached via p5.

Ex→
```
main() {
    int a = 5;
    int *p = &a;
    cout << "Before" << endl;
    cout << a;          //5
    cout << p;          // 104
    cout << *p;         // 5
    util(p);
    cout << "After" << endl;
    cout << a;          //6
    cout << p;          // 104
    cout << *p;         // 6
}
util (int *p) {
    p = p+1;
    *p = *p+1;
}
```

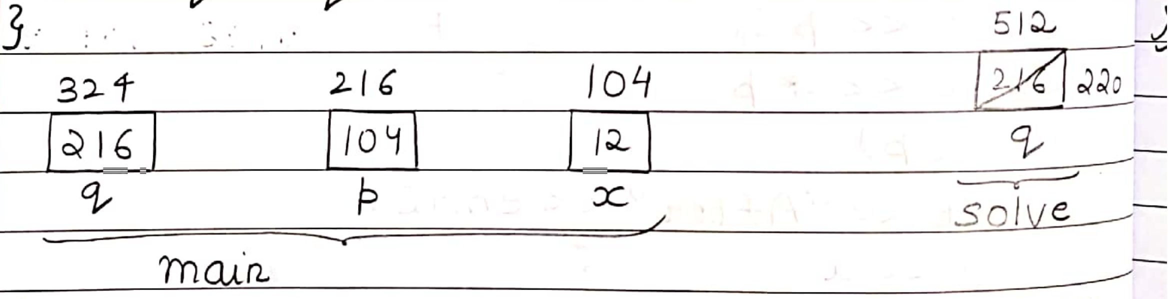
Here in function copy of pointer is created & hence the address stored in p will not get changed



As the util function is over, ↗ will be deleted & hence address of p is 208 & it has stored 104

& the value of a gets updated.

Ex→
```
main() {
        int x = 12;
        int *p = &x;
        int **q = &p;
        solve(q);
        cout << x;
}
solve( int **q) {
        q = q+1;
}
```



As solve is over, 512 address block will be deleted & hence x = 12.

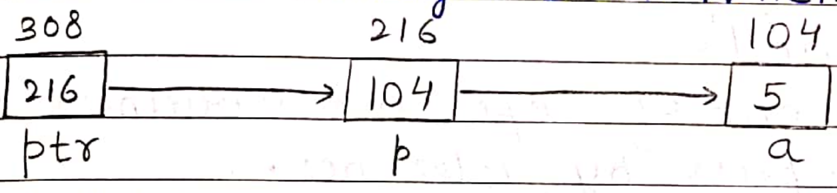
(ii)
```
solve (int **q) {
        *q = *q + 1;
}
```

Now the address stored in p will get updated but x will remain same.

(iii)
```
solve (int **q) {
        **q = **q + 1;
}
```

Here the value of x will get modified & hence 13 will be printed.

Note → If we try to do
**ptr = *ptr+1 , this will give us an error as we are trying to store address in integer block which is not possible.



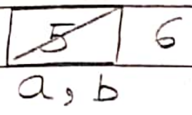
```
**ptr ⇒ block a
*ptr ⇒ address
```

Note → `int ** ptr = p ;` } Will give error
→ pointer

To replace the concept of pointers, reference variable concept came into picture as concept of pointers is basically difficult to understand.

Reference Variables
The concept behind this is that we can call the same memory location by different names.

```
int a = 5;
int &b = a ;  // This means that b is a reference
                 variable & pointing to same
                 memory location.
b++;
cout << a << endl;  // 6
cout << b << endl;  // 6
```



Here symbol table is also updated

Pointers vs Reference Variable

1) Reference variable is used because it can not be set to null whereas pointers can be set to null. Hence it is safe to use the concept of reference variable.

2) Concept of pointer is difficult to understand.

3) Also concept of reference variables is used to pass by reference.

Pass by reference

```
void solve (int & x) { //Pass by reference
           x++;                 concept
}
int main () {
        int a = 5;
        solve (a); //6 is printed
        cout << a;
}
```

In this concept copy is not created & original variable will be updated

Note →

```
void solve (int* x) {
        *x = *x +1;
      }
main() { int a = 5;
        solve (& a);
        cout << a; // 6 is printed
}
```

Passing pointers as reference

→ Pass by reference of pointer

```
void solve (int *& p) {
        p = p + 1; // Address will be changed
}
```

Note →

```
void solve (int & * p) {
        p = p + 1;
    }
```

This code will not work. First * and then & will come.