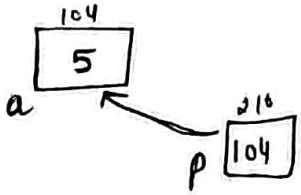
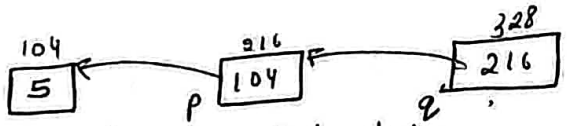
Pointers - 3

```
int a = 5;
int* p = &a;

char ch = 'd';
char* cp = &ch;

int** ptr = &p;        → double pointer.
```



Symbol Table
a → 104
p → 216

```
int a = 5;
int* p = &a;     → p is a pointer to int data.
int** q = &p;    → q is a pointer to int* data.
```



cout → &a → address of a (here - 104)

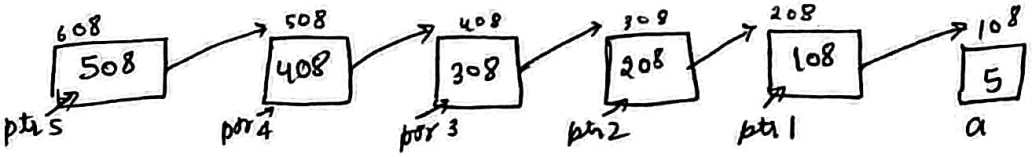
a → 5

p → 104 | q → 216

&p → 216 | &q → 328

*p → 5 | *q → 104

**q → 5



a → *ptr 1, **ptr 2, ***ptr 3, ****ptr 4, *****ptr 5, a

ptr 1 → *ptr 2, **ptr 3, ***ptr 4, ****ptr 5, ptr 1

ptr 2 → *ptr 3, **ptr 4, ***ptr 5, ptr 2

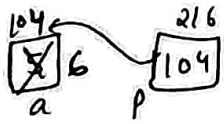
ptr 3 → *ptr 4, **ptr 5, ptr 3

ptr 4 → ptr 4, *ptr 5

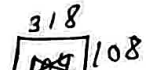
ptr 5 → ptr 5

①

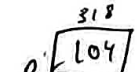
```
main(){
    int a = 5;
    int* p = &a;
    print a, p, *p  => 5, 104, 5
    util(p);
    print a, p, *p  => 5, 104, 5
    solve(p);
}
```




```
util(int* p){
    p = p+1;
}
```




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



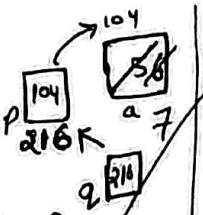
*p = 5

*p = 5+1 = 6

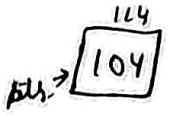
o/p → 6, 104, 6

```
main(){
    int a = 5;
    int *p = &a;
    print a;  o/p = 5
    util(p);
    print a;  → o/p → 6
    int *q = &p;
    solve(q);
    print a;  → o/p → 7
}
```

*q = value present at location stored in q.

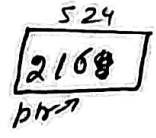



```
util(int *ptr){
    *ptr = *ptr + 1;
}
```



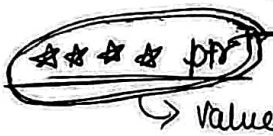
*(104) = *(104) + 1
= 5 + 1
*(104) = 6

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

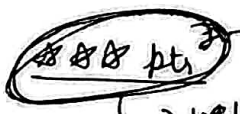


**(104) = **(104) + 1
= 6 + 1 = 7

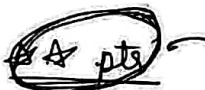
*ptr = *ptr + 1
*(216) = *(216) + 1
= 104 + 1 = 108
↖ address



↳ value present at location stored in ***ptr

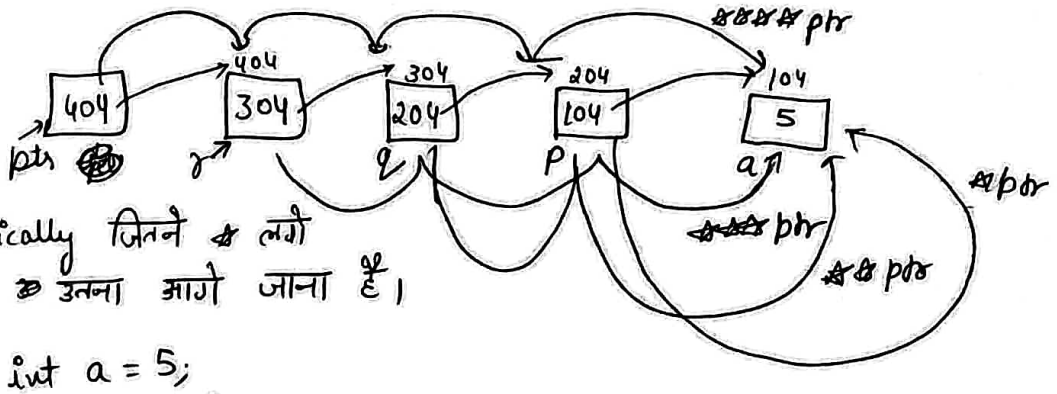


↳ value present at location stored in **ptr



↳ value present at location stored in *ptr

↳ value present at location stored in ptr.
⇒ 404



Basically जितने * लगे हैं ** उतना आगे जाना है।

```
int a = 5;
int *p = &a;
int **ptr = &p;
```

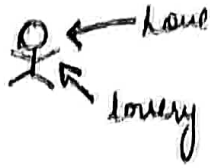
**ptr = *ptr + 1; ⟶ Error. ⟵ either do this

**ptr = **ptr + 1

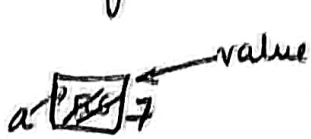
or

*ptr = *ptr + 1

⊙ Reference Variable →



same person two names.



same memory location different names.

a++ → 6

value++ → 7

Ques → Why we need reference variable when we have pointers?

int a = 5;

int &b = a; → b is a reference variable & pointing to same memory location as a.

cout << a; → o/p → 5

cout << b; → o/p → 5

a++;

cout << a << b << endl; → o/p → 6 6

b++;

cout << a << b << endl; → o/p → 7 7

Symbol Table
a → 104
b → 104

reference variable doesn't take extra memory, just their entries is stored in symbol table.

⊙ Why to use Reference variables?

→ We can't set reference variable to null but pointers can point to Null, so there is more safety in reference variables. Reference variables always points to a valid memory location.

→ Pointers are difficult to understand but reference variable have more readability.

⊙ Function → Pass by Value and Pass by Reference → pass by Value

```
main(){
    int a = 5;
    solve(a);
    cout << a;
}
```

1000 | 5 | a

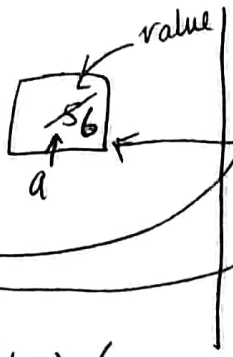
```
solve(int a){
    a++;
    cout << a;
}
```

2000 | 6 | a

solve(int a) ← 0/p → 6

main() ← 0/p = 5

```
main(){
    int a = 5;
    solve(a);
    cout << a;
}
```

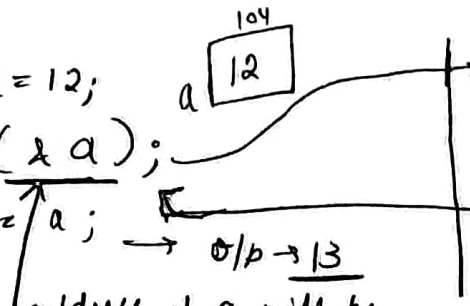



```
solve(int& value){
    value++;
}
```

pass by Reference

o/p → 6

```
int a = 12;
solve(&a);
cout << a;
```



address of a, will be stored in a pointer.

```
solve(int* val){
    *val++;
}
```

104
val

o/p → 13

*(104) = *(104) + 1
= 12 + 1 = 13

Pass by value & Reference in pointer →

```
int main(){
    int a=5;
    int* p = &a;
    cout << "Before" << p;
    solve(p);
    cout << "After" << p;
}
```

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

In this func nothing will be changed

← Pass by value.

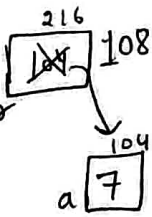
```
solve(int*& p){
    p = p+1;
}
```

← Pass by reference.

Because this is pass by reference so p(pointer) will be incremented by 1.

⊙ Pass by reference →

```
main(){
    int a = 7;
    int* ptr = &a;
    print ptr → 104
    solve(ptr);
    print ptr → 108
}
```




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

H.W
1. Explore return by refernce.

#2→ Output ?

very famous

```
int * solve ( ){
    int a = 5;
    int * ans = &a;
    return ans;
}
```