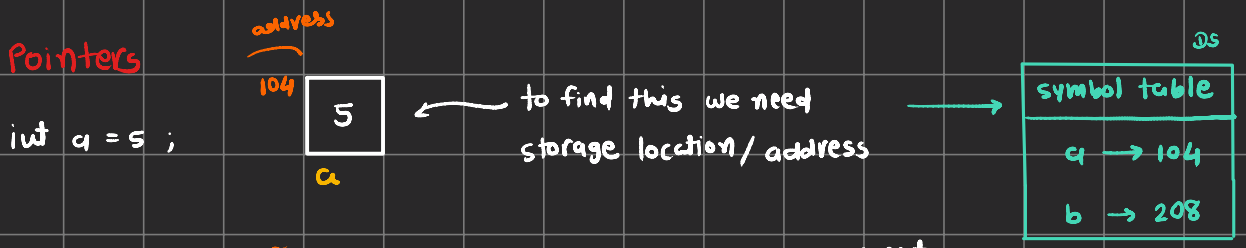
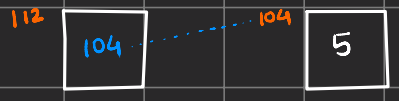
Pointers



→ cout << &a ; → shows memory address where a's value is stored



cout << a << &a ;
5 104

→ int a = 5 ; means you can store int type data under variable a

→ int * ptr = & a ;

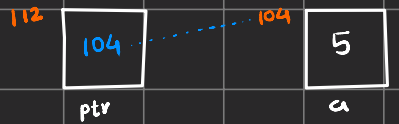
ptr is pointer to integer data
↳ is one datatype that can store address

cout << ptr << &ptr << * ptr ;
104 112 5
↳ value at location stored in ptr
value at (104) = 5
value stored at ptr
address of ptr

→ ptr is pointing to int, double, short However everytime the size of ptr is 4 byte in 32 bit architechure (4 byte = 4*8 bit = 32 bit)
8 byte in 64 bit architechure

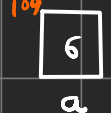
Why pointer needed !

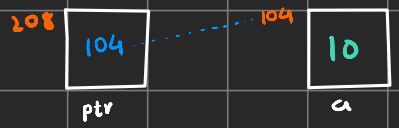
↳ dynamic memory allocation, memory management, hardware part access, pointer to function



ptr = ptr + 1 = 104 + 1 = 108
int → 4 byte 4 byte

*ptr = *ptr + 1 = 6
new value of a 5 + 1





a → 10 &a → 104

ptr → 104 *ptr → 10 &ptr → 208

→ *ptr * 2 → (*ptr)++ → ++(*ptr) → a = a + 1 → *p = *p + 2 → *p = *p * 2

10 * 2 10++ ++11 = 12+1 *p = *p + 2 *p = *p * 2

20 11 12 = 13 a = 13 + 2 a = 15 * 2

= 15 = 30

✱ copy pointer

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

✱ for array

```
int arr[5];
```

constant pointer

arr = arr + 1

→ user can never change value from ST.

cout << arr << &arr << &arr[0] ;

same → base address

symbol Table

arr → 104

} thats why ↑

*arr → 5 | *arr + 1 → 5 + 1 = 6 | *(arr) + 1 → 5 + 1 = 6 | *(arr+1) → *(104+1) → *(108) → 8 arr[1] | *(arr+2) → *(104+2) → *(104+2*4) → *(112) → 7 | arr[i] → *(arr+i) ; *(i+arr) → i[arr] ; Both ans is same

int * ptr = arr ;

ptr = ptr + 1 → now ptr pointing to arr[1]

* char Array

char ch[10] = "Babbar"

char * cptr = ch

cout << cptr → Babbar

cout << *cptr → B Here → *cptr → *(c+0) → *(104) → B

cout <<

ch → Babbar

&ch → 104

*(ch+3) → ch[3] → b

cptr → 104

&cptr → 216

*(cptr+3) → cptr[3] → b

*cptr → *(cptr+0) → *(104) → B

cptr + 2 → bbar (string from 2nd index)

cptr + 6 → ""

↳ print until null char

cout has behaviour for cstring is that it prints until gets null char

char ch = 'K' ; char * cptr = &ch

cout << cptr ; → KGIBGIBGIBValue

`char name[10] = "Babbar" ;` → ① temp storage ke andar "Babbar" dalenge ② then memory change : copy to name array ki storage

`char * c = "Babbar" ;` → ① temp storage ke andar "Babbar" dalenge ② here c pointer points to first element of that temporary storage
↳ Bad practise bcz temp storage can be changed / removed any time

`cout << c` ⇝ Babbar

⁂ array pass by reference

```
main()
{
  int arr[10];
  sizeof(arr)  → 10 * 4 = 40
  solve(arr)
}
```

```
solve(int arr[])
{
  sizeof(arr)  → 4 byte  so pointer is passed
  arr[0] = 12  → change into actual array
  cout << arr  → 104  Base address of actual array
  cout << &arr → 106  different than base address it means this is some new pointer
}
```

underhood pointer passed

```
main() {
  int a = 5
  int *ptr = &a
  solve(ptr)
}
```

112 | 5 | a

112 | ptr

```
solve(int * p) {
  *p = *p + 10
}
```

int * p = ptr = 112

*p pointing to a

112 | p

a → 15 now

⭐ Double pointer

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

p is pointer to int data

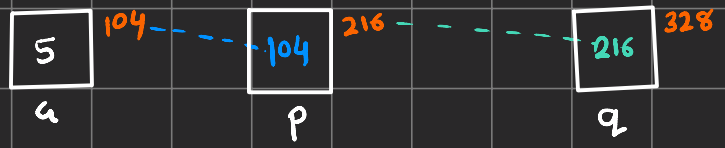
q is pointer to int* data

a → 5 &a → 104

p → 104 &p → 216 *p → 5

q → 216 &q → 328 *q → 104

**q → 5



**q

↓

value at location 216 → value 104

*104

value at address (104) → 5

5

how to get value of a →

* ptr1

** ptr2

*** ptr3

**** ptr4

How to reach to ptr2 (216) → * ptr3, ** ptr4

→ `int main()`

`int a=5`

`int *p= a`

`cout << a << p << *p` → 5, 104, 5

`util(p)`

`cout << a << p << *p` → 5, 104, 5 why no change?

because pass by value

`void util(int *p)`

`p = p + 1`

if we print inside util

`cout << p << *p`

108 GiB

`void util(int *p)`

`*p = *p + 1 ;`

a 5+1 = 6

After this in main

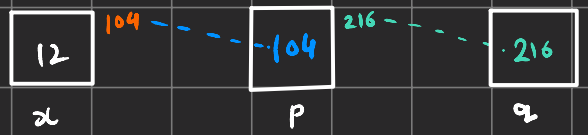
`cout << a << p << *p`

6 104 6

→

```
main()
  int x = 12
  int *p = &x
  int **q = &p
   solve(q)
  cout << x
```

```
solve ( int ** q )
  q = q + 1
    216 + 1
    = 220
```
does not affect on x as above



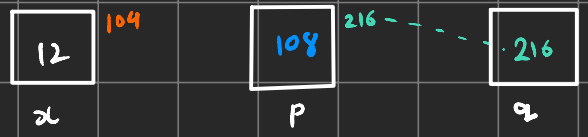

```
solve ( int ** ptr )
  * ptr = * ptr + 1
```
value of p

int ** ptr = q = 216

* (216)

= 104 + 1

= 108



how to change value of a ?

** ptr = ** ptr + 1

* p → value present at location stored in p

** q → value present at location stored in * q

means value present at location stored in q

** ptr = * ptr + 1

a

* (216)

108 + 1

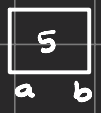
112 → address

not possible to store address inside int datatype

★ Reference Variable → Alternative of pointer → same memory location accessed by different name of variable

```
int a = 5
int &b = a
```

↳ b is ref variable and pointing to same memory location of a



symbol table

a	108
b	→

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

```
solve(int a)
{
    a = a + 2
}
```
→ pass by value } copy banti hai

actual a in main not affected

```
solve(int & num)
{
    num = num + 2
}
```
→ pass by reference } copy nahi hoti

can affect actual a in main()

→
```
main()

int a = 5
int * p = &a

cout << p      → 102
solve(p)
cout << p      → 106
```

```
solve(int* & p)

p = p + 1
```
→ pass by ref

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

```
main()
{ int * ptr = solve()
  cout << * ptr
}
```
→ GB bcz a તો delete હો ગયા હોગા After execution of solve function

(a → local variable for solve() function)

→ Return by referance → remaining topic

→ questions

```
int a[] = {1,2,3,4}
int *p = a++
cout << *p << endl
```

→ `int *p = (a = a+1)`

↳ cannot change memory address stored in Symbol Table

```
char b[] = "xyz"
char *c = &b[0]
cout << c << *c << b << b[0] << c[0]
```

a	y	z	\0
0	1	2	3

xyz x xyz a a

↳ in cstring if we give address it print until null pointer

```
char *ptr
char str[] = "abcdefg"
ptr = str
ptr += 5   → 100+5=105
cout << ptr → fg
```

ptr: ~~100~~ 105

str

a	b	c	d	e	f	g	\0
0	1	2	3	4	5	6	7
100	101	102	103	104	105	106	107

ptr → 105

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

i=0

st[i]	*(st)+i	*(i+st)	i[st]
A	'A'+0	*(0+st)	0[st]
	65+0	*(st)	st[0]
	65	A	A

cout gives INT when char + int

but

```
char ch = 'A' + 2
        = 65 + 2
        = 67
```

char = 'C'

```
float arr[5] = {1,2,3,4,5}
float *ptr1 = &arr[0]
float *ptr2 = ptr1 + 3
cout << *ptr2
cout << ptr2 - ptr1
```

100	104	108	112	116
1	2	3	4	5
0	1	2	3	4

ptr1 = 100

ptr2 = 112 → ptr1 + 3

100 + 3*4

112

Pointer arithmetic

112 − 100 = 12

4

= 3

but this is pointer arithmetic so we need to subtract by 4 byte of float

OR

= 112 − 100

= 3index − 0index

= 3

★ Basic Maths

① Prime no → n = 5

1, 2, 3, 4, 5 — do not take 1 and 5

n % 2 = 5 % 2 → 2 | 5 | 2 | 4 | 1 → 1

5 % 3 → 3 | 5 | 1 | 3 | 2 → 2

5 % 4 → 4 | 5 | 1 | 4 | 1 → 1

if mod is coming 0 means notPrime

→ count total prime no's less than n (Naive Approach)

```
bool isPrime(int num){
    for(int i=2; i<num; i++){
        if( num%i == 0 ){
            return false;
        }
    }
    return true;
}

int countPrimes(int n) {

    int count = 0;
    for(int i=2; i<n; i++){
        if( isPrime(i) ){
            count ++;
        }
    }
    return count;
```

→ This TC = n^2

So giving TLE for n = 499979

to solve this we have sieve of Eratosthenes

Sieve of Eratosthenes

		prime		prime		prime		prime	
~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
prime 11	~~12~~	13 P	~~14~~	~~15~~	~~16~~	17 P	~~18~~	19 P	~~20~~
~~21~~	~~22~~	23 P	~~24~~	25	~~26~~	~~27~~	~~28~~	29 P	~~30~~
31 P	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37 P	~~38~~	~~39~~	~~40~~

→ total 12 primes

TC = O(n * log(log n))

① Mark all number as Prime no

② Then go one by one from 2 ... if prime number hai then jo jo us number ke table me ayegy uska non prime mark kar do

(learn about segmented sieve)

In Naive approach, isPrime() function's TC is O(n), So let's make it better (sqrt Approach)

let assume N is nonPrime

$n = a \times b$ if $a > \sqrt{n}$, $b > \sqrt{n}$ → but $ab > n$ is not Possible

} if N is non Prime than at least one factor must be less than $\sqrt{n}$

1, 2, 3 . . . . n-1, N

```
bool isPrime(int n){
    for(int i=2; i<=sqrt(n); i++){
        if(n%2 == 0){
            return false;
        }
    }
    return true;
}
```

TC $O(\sqrt{n})$

→ GCD / HCF - factor that is maximum and that can divide both num so ans cannbe zero

a & b → 24 and 72

2	24
2	12
2	6
2	3
	1

2	72
2	36
2	18
3	9
3	3
	1

24 = 2×2×2×3
72 = 2×2×2×3×3
common

= 2×2×2×3
= 24 → HCF/GCD

gcd (a,b) = gcd (a-b, b)
↓
= gcd (a-1.b, b)

gcd (72,24) → gcd (72-24, 24)
= gcd (48, 24)
a b
→ gcd (48-24, 24)
= gcd (24, 24)
a b
→ gcd (24-24, 24)
= gcd (0, 24)
zero → Ans.

milte he
last me jo
dusra element
milega wo ans hoga

lcm (a,b) * gcd (a,b) = a * b

Modulo Arithmetic

1. (a % n) -> [0, ... , n - 1]
2. Generally, to avoid overflow while storing Integer we do modulo with a Large number.
   1. (a + b) % M = a % M + b % M
   2. a % M - b % M = (a - b) % M
   3. ((a % M) % M) % M = a % M
   4. a % M * b % M = (a * b) % M

Fast exponential

a^b → $(a^{b/2})^2$ → if b is even

a^b → $(a^{b/2})^2 \times a$ → if b is odd

ex. 2^{10}

2^b => b = 10 → even so $(2^{b/2})^2 = (2^{10/2})^2 = (2^5)^2$

Tc. O(log n)

ex $2^5 = 2^4 \cdot 2$

$= (2^2 \cdot 2^2) \cdot 2$

$= ((2^1 \cdot 2^1) \cdot (2^1 \cdot 2^1)) \cdot 2$

```
int fastExpo(int a, int b) {
    int ans = 1;

    while (b > 0) {
        // If b is odd, multiply ans by a
        if (b & 1) {
            ans = ans * a;
        }
        // Square the base
        a = a * a;
        // Divide b by 2
        b = b >> 1;
    }

    return ans;
}
```

Modular Exponentiation for large numbers

Medium Accuracy: 52.56% Submissions: 18K+ Points: 4

Don't Miss Out on the Chance to Work with Leading Companies! Visit the GFG Job Fair Now!

Implement pow(x, n) % M.

In other words, given x, n and M, find (x^n) % M.

Example 1:

Input:
x = 3, n = 2, m = 4
Output:
1
Explanation:
3^2 = 9. 9 % 4 = 1.

```
// } Driver Code Ends
class Solution
{
    public:
        long long int PowMod(long long int x,long long int n,long long int M)
        {
            long long int ans = 1;

            while(n>0){
                if(n&1){
                    ans = (ans * x) % M;
                }
                x = (x * x) % M;
                n>>=1;
            }
            return ans % M;
        }
};
// } Driver Code Ends
```

when we are playing with large numbers with multiply, division types of extensive operation that time mod helps to remove overflow