

**TYPES OF PROGRAMMING LANGUAGES:**

Low level

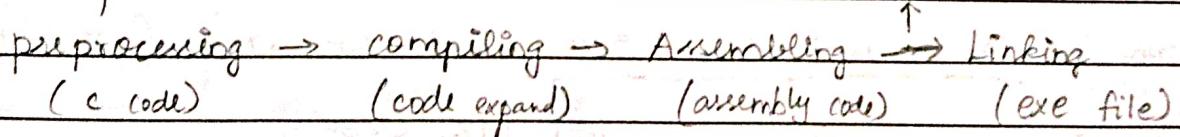
Middle level

High level

Assembly language:

C  $\Rightarrow$  structure oriented - procedural language (customization)OOPS  $\Rightarrow$  Non-procedural (divides the code into smaller parts & run)C, Java  $\Rightarrow$  static type language (mention datatype manually) [Fast]python  $\Rightarrow$  dynamic [slower than static]C  $\Rightarrow$  OS dependent language  $\xrightarrow{\text{modularity}}$  modularity  $\Rightarrow$  mother langctype.h  $\Rightarrow$  cstring.h  $\Rightarrow$ conio.h  $\Rightarrow$  console I/Ostdlib.h  $\Rightarrow$  memory allocation.

Compiler process:

**Data type:**Int  $\Rightarrow$   $-2^{31}$  to  $2^{31}-1$  (4 bytes)byte  $\Rightarrow$  -128 to 127 (1 byte)short  $\Rightarrow$  (2 bytes)long  $\Rightarrow$   $-2^{63}$  to  $2^{63}-1$  (8 bytes)float  $\Rightarrow$  (4 bytes)double  $\Rightarrow$  (8 bytes)

char  $\rightarrow$  ASCII key [2<sup>7</sup> to 2<sup>7</sup>-1] (1 byte) in C, in Java (2 bytes) because of UNICODE

### Variable:

name given to a value.  $\rightarrow$  one among identifier

### Format Identifier:

int  $\Rightarrow$  %d signed int  $\Rightarrow$  %i

char  $\Rightarrow$  %c long  $\Rightarrow$  %ld

float  $\Rightarrow$  %f short  $\Rightarrow$  %hd

double  $\Rightarrow$  %lf

### Variable:

Declaration, Initialization, Definition

2018

unsigned short, short, unsigned short, ushort int  $\Rightarrow$  2 bytes.

### Type casting:

change from one datatype to another datatype.

$\rightarrow$  Implicit TC

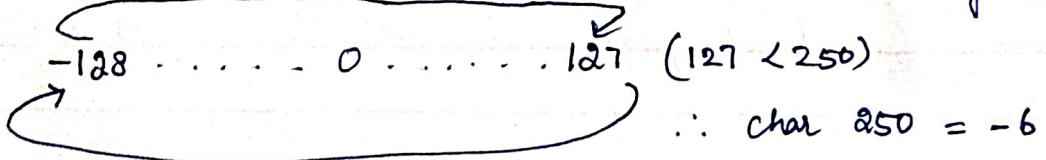
$\rightarrow$  Explicit TC

ASCII  
Value

A = 65	a = 97	Number 0 = 48
Z = 90	z = 122	Number 9 = 57

Total characters in ASCII is 256

Eg: consider a char value 250, but char range (-128 to 127)



$$\text{char } -130 = 126$$

unsigned char a = 130;

OUTPUT:

char s = 130;

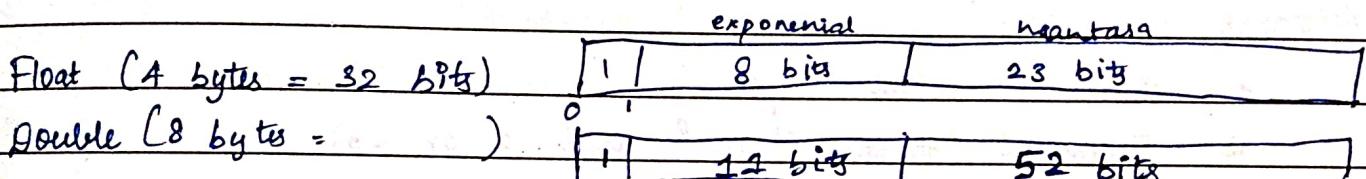
unsigned : 130

printf("Unsigned : %d\n", a);

signed : -126

printf("Signed : %d\n", s);

\* Though we declare a char in unsigned defaultly it will consider unsigned as int. So. size = 4 by b6



CODE : printf ("%x %o %b", 30, 30, 30);

OUTPUT : 1e 36 11110

1. 127 :

Binary :  $\begin{smallmatrix} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{smallmatrix} \Rightarrow 0111111$

Octal :  $\begin{smallmatrix} 8^3 & 8^2 & 8^1 & 8^0 \\ 1 & 1 & 1 & 1 \end{smallmatrix} \Rightarrow 0177$

Decimal :

Hexadecimal : D x 7F

700 base 9  $\Rightarrow$  857.

784 base 3  $\Rightarrow$   $\begin{smallmatrix} 3^7 & 3^6 & 3^5 & 3^4 & 3^3 & 3^2 & 3^1 & 3^0 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \end{smallmatrix}$  ~~2 3 0 1 3~~

$\Rightarrow 1002001$

7541 base 6  $\Rightarrow$   $\begin{smallmatrix} 6^5 & 6^4 & 6^3 & 6^2 & 6^1 & 6^0 \\ 1 & 2 & 4 & 1 & 2 & 1 \end{smallmatrix}$

5 4 4 8 6

$\begin{array}{r} 7541 \\ - 6480 \\ \hline 1061 \\ - 864 \\ \hline 197 \\ - 144 \\ \hline 53 \end{array}$

```

CODE: unsigned int a = -20;
      printf("%d", a);           // output : -20
      printf("%.2u", a);         // o/p : 4294967296
      printf("%s", "len of string"); // o/p : [ ] 10
      printf("%3d", 10);          // o/p : Hell
      printf("%.10.4s", "Hello"); // o/p : Hello [6 spaces]

```

#include <stdbool.h> [1 bit] Header file to use boolean.

Q18

### STRONG NUMBER:

1. Get the last digit
2. Factorial (Iteration & mul until 1 or >0)
3. add all factorial values
4. Remove the last digit

### FUNCTION:

Declaration [return-type] function-name (arg datatype);

Definition

Calling

function types:

(i) with arg with ret

```

int sum(int a, int b)
{
    int s = a + b;
    return s;
}

```

(ii) without arg with return type

```

int sum()
{
    int a = 5, b = 6; s = a + b;
    return s;
}

```

(iii) with arg without ret

```

void sum(int a, int b)
{
    int s = a + b;
    printf("%d", s);
}

```

(iv) without arg without ret

```

void sum()
{
    int a = 5, b = 6; s = a + b;
    printf("%d", s);
}

```

Parameters - value passed inside () in function definition.

Argument - value passed inside () during function call.

Local scope, global scope:

Eg: How much percentage 60 in 200  $\frac{60}{200} \times 100 = 30\%$ .

```
void percentage (int a, int b)
{
    float percent = ((float)a/b)*100;
    printf ("% .2f % .% .", percent);
}
```

### 2018/20 PATTERNS : (\*, 1, A):

- 1. Right triangle 
- 2. Left triangle 
- 3. Inverted right tri 
- 4. Inverted left tri 
- 5. straight triangle 
- 6. Inverted str. tri 
- 7. Hollow right triangle (for all \* format)
- 8. Diamond pattern, hollow, half diamond
- 9. Pyramid, hollow pyramid
- 10. Hour Glass pattern, hollow
- 11. Floyd's triangle
- 12. Numbers 
  
- 13. Rhombus pattern
  
- 1. Dutch National Problem
- 2. Quick, Merge, Insert sort are efficient sorts. (O, 1, 2)  
most efficient      2. approach.

28/8

Try:

1. Find second largest element
2. sort 0s, 1s, 2s in a 1D array
3. Find  $k^{\text{th}}$  largest element.  $\frac{3}{4}^{\text{th}}$   $k$  smallest.

Practice problems in program wiz.

4. Move (-ve) to the beginning
5. Perfect square without math.h
6. Frequency of vowel in a char array.

24/8

`int count[26] = {0}; // then all index stores value 0.`

In leetcode,

`int * nums  $\rightarrow$  int nums[]` is equal to

`arr1, arr2` are given. Have to find the median of them.

$$\rightarrow \text{size}(arr1) = l_1, \text{size}(arr2) = l_2; l_3 = l_1 + l_2;$$

`arr3[l3];`

$$\begin{bmatrix} 1, 2, 5, 8, 12 \\ 3, 6, 10, 20, 25 \end{bmatrix} \quad (i < l_1) \quad (j < l_2)$$

Right rotation array:

$$\begin{bmatrix} 1, 2, 3, 4, 5 \end{bmatrix}, r=3$$

$$\text{rev } \left( \underbrace{(r-1)}_{(2)} = 2 \right) \text{rev}$$

$$\underbrace{\begin{bmatrix} 2, 1 \\ 5, 4, 3 \end{bmatrix}}_{\text{rev}}$$

$$\Rightarrow 3, 4, 5, 1, 2,$$

Left rotation array:

`rotateRight(int arr[], int size int r)`

{  $r = r \% \text{size}$  ;

`rev(arr, 0, K-1)`

`rev(arr, K, size-1)`

`rev(arr, 0, size-1)`

}

## SERIES

### 1. Arithmetic series sum:

$$\text{sum} = \frac{\text{no.of. terms}}{2} \times (\text{2} \times \text{first term} + (\text{no.of. terms}-1) \times \text{com diff})$$

$$\text{sum} = \frac{n}{2} (2a + (n-1)d)$$

Edge case  $\Rightarrow n \leq 0 \Rightarrow \text{return } 0$ ;

### 2. Geometric series sum:

$$\text{sum} = \frac{\text{first term} \times (\text{common ratio}^{\text{no.of terms}} - 1)}{\text{common ratio} - 1} = \frac{a(r^n - 1)}{r - 1}$$

Edge case  $\Rightarrow n \leq 0 \Rightarrow \text{return } 0$ ;

### 3. Harmonic series sum:

Harmonic series = sum from ( $n=1$ ) to ( $n=\infty$ ) with terms  $\frac{1}{n}$   
 $= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{n}$

Edge case  $\Rightarrow n \leq 0 \Rightarrow \text{return } 0$ ;

### 4. n fibonacci series

i/p: 5 ; o/p : 0 1 1 2 3

## 5. Factorial series:

Factorial =  $1 \times 2 \times 3 \times \dots \times n$

Edge case  $\Rightarrow n=0 \Rightarrow$  return 1;

## Factorial sum series:

Factorial inside for loop.

## 6. Power series:

math.h  $\rightarrow$  power (base, exponent)

## 7. Taylor series for $\sin(x)$ :

$$\text{sum} \left( \frac{(-1)^n \times x^{2n+1}}{(2n+1)!} \right)$$

double taylor(double x, int n)

{ if ( $n < 0$ ) return 0.0;  
double sum = 0.0;

for (int i=0; i<n; i++) {  
double term = (pow(-1, i) \* pow(x, 2\*i+1)) / factorial(2\*i+1);  
sum += term; }  
return sum;

## 8. MacLaurin series (for $e^x$ ):

It is a power series that allows one to calculate an approximation of function  $f(x)$  for i/p values close to 0.

$$\text{Sum} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

double macLaurin(double x, int n)

{ if ( $n < 0$ ) return 1.0;

double sum = 1.0, term = 1.0;

for (int i=0; i<n; i++)

{ term \*= x/i;

sum += term;

g

return sum;

3

## 9. Alternating series

```

int altSum(int firstSum, int commratio, int numTerms)
{
    if (numTerms <= 0) return 0;
    int sum = 0, term = firstSum;
    for (int i = 0; i < numTerms; i++)
    {
        sum += term;
        term *= commratio;
    }
    return sum;
}

```

## 10. Infinite series: (Harmonic series with tolerance)

double infSeries(double tolerance)

Series:  $1 + \frac{1}{2} + \frac{1}{3} + \dots$ 

{ double sum = 0.0, term = 1.0;

Tolerance : 0.001

int n = 1;

O/P : 14.39273

while (term &gt; tolerance)

{ sum += term;

n++;

term = 1.0/n;

} return sum;

}

## PATTERN

1)

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

3)

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

5)

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

\* \* \*

2)

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

4)

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

6)

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

1 6 15 20 15 6 1

1 7 21 35 35 21 7 1

1 8 28 56 70 56 28 8 1

1 9 36 84 126 126 84 36 9 1

1 10 45 120 210 252 210 120 45 10 1

1 11 55 165 330 462 462 330 165 55 11 1

1 12 66 220 495 880 1320 1320 880 495 220 66 12 1

1 13 78 252 630 1320 2310 2310 1320 630 252 78 13 1

1 14 91 315 840 1820 3430 3430 1820 840 315 91 14 1

1 15 105 360 1050 2520 5040 5040 2520 1050 360 105 15 1

1 16 120 420 1260 3150 6300 6300 3150 1260 420 120 16 1

1 17 136 480 1470 3780 7560 7560 3780 1470 480 136 17 1

1 18 153 540 1710 4590 9180 9180 4590 1710 540 153 18 1

1 19 171 600 1970 5910 11820 11820 5910 1970 600 171 19 1

1 20 190 660 2250 7200 14400 14400 7200 2250 660 190 20 1

1 21 210 720 2520 8400 16800 16800 8400 2520 720 210 21 1

1 22 231 780 2820 9600 19200 19200 9600 2820 780 231 22 1

1 23 253 840 3150 11200 22400 22400 11200 3150 840 253 23 1

1 24 276 900 3500 12800 25600 25600 12800 3500 900 276 24 1

1 25 300 960 3900 14400 28800 28800 14400 3900 960 300 25 1

1 26 325 1020 4300 16000 32000 32000 16000 4300 1020 325 26 1

1 27 351 1080 4700 17600 35200 35200 17600 4700 1080 351 27 1

1 28 378 1140 5100 19200 38400 38400 19200 5100 1140 378 28 1

1 29 406 1200 5500 20800 41600 41600 20800 5500 1200 406 29 1

1 30 435 1260 5900 22400 44800 44800 22400 5900 1260 435 30 1

1 31 465 1320 6300 24000 48000 48000 24000 6300 1320 465 31 1

1 32 500 1380 6700 25600 51200 51200 25600 6700 1380 500 32 1

1 33 535 1440 7100 27200 54400 54400 27200 7100 1440 535 33 1

1 34 571 1500 7500 28800 57600 57600 28800 7500 1500 571 34 1

1 35 608 1560 7900 30400 60800 60800 30400 7900 1560 608 35 1

1 36 646 1620 8300 32000 64000 64000 32000 8300 1620 646 36 1

1 37 685 1680 8700 33600 67200 67200 33600 8700 1680 685 37 1

1 38 725 1740 9100 35200 70400 70400 35200 9100 1740 725 38 1

1 39 766 1800 9500 36800 73600 73600 36800 9500 1800 766 39 1

1 40 808 1860 9900 38400 76800 76800 38400 9900 1860 808 40 1

1 41 851 1920 10300 40000 80000 80000 40000 10300 1920 851 41 1

1 42 895 1980 10700 41600 83200 83200 41600 10700 1980 895 42 1

1 43 940 2040 11100 43200 86400 86400 43200 11100 2040 940 43 1

1 44 986 2100 11500 44800 89600 89600 44800 11500 2100 986 44 1

1 45 1033 2160 11900 46400 92800 92800 46400 11900 2160 1033 45 1

1 46 1081 2220 12300 48000 96000 96000 48000 12300 2220 1081 46 1

1 47 1130 2280 12700 49600 99200 99200 49600 12700 2280 1130 47 1

1 48 1180 2340 13100 51200 102400 102400 51200 13100 2340 1180 48 1

1 49 1231 2400 13500 52800 105600 105600 52800 13500 2400 1231 49 1

1 50 1283 2460 13900 54400 108800 108800 54400 13900 2460 1283 50 1

1 51 1336 2520 14300 56000 112000 112000 56000 14300 2520 1336 51 1

1 52 1390 2580 14700 57600 115200 115200 57600 14700 2580 1390 52 1

1 53 1445 2640 15100 59200 118400 118400 59200 15100 2640 1445 53 1

1 54 1500 2700 15500 60800 121600 121600 60800 15500 2700 1500 54 1

1 55 1556 2760 15900 62400 124800 124800 62400 15900 2760 1556 55 1

1 56 1613 2820 16300 64000 128000 128000 64000 16300 2820 1613 56 1

1 57 1670 2880 16700 65600 131200 131200 65600 16700 2880 1670 57 1

1 58 1728 2940 17100 67200 134400 134400 67200 17100 2940 1728 58 1

1 59 1786 3000 17500 68800 137600 137600 68800 17500 3000 1786 59 1

1 60 1844 3060 17900 70400 140800 140800 70400 17900 3060 1844 60 1

1 61 1902 3120 18300 72000 144000 144000 72000 18300 3120 1902 61 1

1 62 1960 3180 18700 73600 147200 147200 73600 18700 3180 1960 62 1

1 63 2018 3240 19100 75200 150400 150400 75200 19100 3240 2018 63 1

1 64 2076 3300 19500 76800 153600 153600 76800 19500 3300 2076 64 1

1 65 2134 3360 19900 78400 156800 156800 78400 19900 3360 2134 65 1

1 66 2192 3420 20300 80000 160000 160000 80000 20300 3420 2192 66 1

1 67 2250 3480 20700 81600 163200 163200 81600 20700 3480 2250 67 1

1 68 2308 3540 21100 83200 166400 166400 83200 21100 3540 2308 68 1

1 69 2366 3600 21500 84800 169600 169600 84800 21500 3600 2366 69 1

1 70 2424 3660 21900 86400 172800 172800 86400 21900 3660 2424 70 1

1 71 2482 3720 22300 88000 176000 176000 88000 22300 3720 2482 71 1

1 72 2540 3780 22700 89600 179200 179200 89600 22700 3780 2540 72 1

1 73 2598 3840 23100 91200 182400 182400 91200 23100 3840 2598 73 1

1 74 2656 3900 23500 92800 185600 185600 92800 23500 3900 2656 74 1

1 75 2714 3960 23900 94400 188800 188800 94400 23900 3960 2714 75 1

1 76 2772 4020 24300 96000 192000 192000 96000 24300 4020 2772 76 1

1 77 2830 4080 24700 97600 195200 195200 97600 24700 4080 2830 77 1

1 78 2888 4140 25100 99200 198400 198400 99200 25100 4140 2888 78 1

1 79 2946 4200 25500 100800 201600 201600 100800 25500 4200 2946 79 1

1 80 3004 4260 25900 102400 204800 204800 102400 25900 4260 3004 80 1

1 81 3062 4320 26300 104000 208000 208000 104000 26300 4320 3062 81 1

1 82 3120 4380 26700 105600 211200 211200 105600 26700 4380 3120 82 1

1 83 3178 4440 27100 107200 214400 214400 107200 27100 4440 3178 83 1

1 84 3236 4500 27500 108800 217600 217600 108800 27500 4500 3236 84 1

1 85 3294 4560 27900 110400 220800 220800 110400 27900 4560 3294 85 1

1 86 3352 4620 28300 112000 224000 224000 112000 28300 4620 3352 86 1

1 87 3410 4680 28700 113600 227200 227200 113600 28700 4680 3410 87 1

1 88 3468 4740 29100 115200 230400 230400 115200 29100 4740 3468 88 1

1 89 3526 4800 29500 116800 233600 233600 116800 29500 4800 3526 89 1

1 90 3584 4860 29900 118400 236800 236800 118400 29900 4860 3584 90 1

1 91 3642 4920 30300 120000 240000 240000 120000 30300 4920 3642 91 1

1 92 3700 4980 30700 121600 243200 243200 121600 30700 4980 3700 92 1

1 93 3758 5040 31100 123200 246400 246400 123200 31100 5040 3758 93 1

1 94 3816 5100 31500 124800 249600 249600 124800 31500 5100 3816 94 1

1 95 3874 5160 31900 126400 252800 252800 126400 31900 5160 3874 95 1

1 96 3932 5220 32300 128000 256000 256000 128000 32300 5220 3932 96 1

7) \* \* \* \* \*

\* \*

\*

\* \*

\* \* \* \*

9)

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

8) 1  
2 3  $i/p = 5$   
4 5 6  
7 8 9 10  
11 12 13 14 15

10) \* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

\* \* \* \* \*

27/8/22

Leetcode 121 - Best time to buy & sell stock

```
int maxProfit(int* prices, int pricesSize){  
    {  
        int min = prices[0];  
        int profit = 0;  
        for (int i=1; i<=pricesSize; i++)  
        {  
            if (prices[i] < min)  
                min = prices[i];  
            else if (prices[i]-min > profit)  
                profit = prices[i] - min;  
        }  
    }  
    return profit;  
}
```

Leetcode 55 - Jump game

```
for {  
    if (i > max) return false;  
    if (max > i + nums[i]) max = i + nums[i];  
    return true;  
}
```

Leetcode 70 - Climbing stairs

Climbing stairs ( $n$ ) = fiboSeries ( $n+2$ )

```
int climbStairs(int n)  
{  
    int a=0, b=1, c;  
    for (int i=3; i<=n+2; i++)  
    {  
        c = a+b;  
        a = b;  
        b = c;  
    }  
    return c;  
}
```

Steps
1 → 1
2 → 2
3 → 3
4 → 5
5 → 8
6 → 13
7 → 21

**Task:**

1. Union, intersection of 2 1D arrays.
2. Frequency char (256 char)  $\rightarrow$  most occurring element.

**2-D ARRAY**

\* Arrays of array

\*  $\Rightarrow$  int arr[3][3];

$\rightarrow$  int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

$\rightarrow$  int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

$\rightarrow$  int arr[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};

**Task:**

1. sum of all elements

2. search

3. transpose

4. addition

5. Multiplication

6. Diagonal sum

7. Antidiagonal sum

8. For 256 chars - freq array - find most occurring element

**Binary search in 2D array:**

```
while(i < row && j >= 0)
{ if (arr[i][j] == target)
    return 1;
```

```
else if (arr[i][j] < target)
```

```
i++;
```

```
else
```

```
j--;
```

```
3
```

```
return 0;
```

Antidiagonal sum of elements in a matrix:

```
void antidiagonalSum ( int r1 , int c1, int mat [r1][c1])
```

```
{ int sum = 0 ;
```

```
    for ( int i = 0 ; i < r1 ; i++)
```

```
        sum += matrix [i] [r1 - i - 1];
```

```
} printf ("sum : %d ", sum);
```

SEARCHING (3<sup>rd</sup> approach - fully sorted 2d array/matrix :  
FLATTENED APPROACH :

STEP 1 : 1D index calculation

2d position (i,j) = 1d position ( $i * n + j$ )

index =  $i * n + j$

STEP 2 : ACCESSING ELEMENTS.

Use flattened index  $i * n + j$  in 1d array.

Spiral traversal of a matrix:

```
while ( l <= r && t <= b )
```

```
{ for ( i = l ; i <= r ; ++i)
```

```
    printf ("%d ", a[t][i]);
```

```
    top++;
```

```
    for ( i = t ; i <= b ; ++i)
```

```
    printf ("%d ", a[i][r]);
```

```
r--;
```

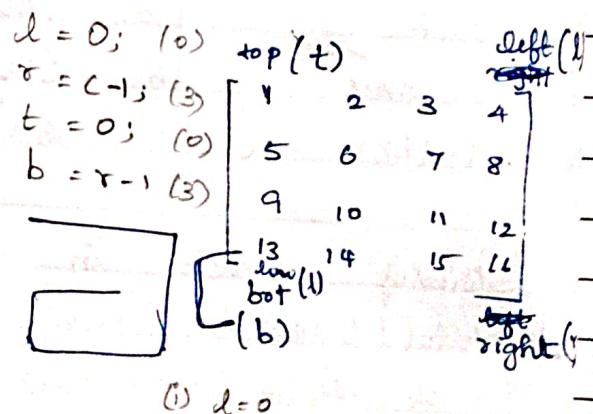
```
if ( t <= b )
```

```
{ for ( i = r ; i >= l ; --i)
```

```
    printf ("%d ", a[b][i]);
```

```
b--;
```

```
}
```



(i) l = 0

(ii)

if ( l <= r )

(iv)

```
{ for ( i = b ; i >= t ; --i)
```

```
    printf ("%d ", a[i][l]);
```

```
l++;
```

```
}
```

## STRING : ( reverse )

1. If user programming - gningmargorp erel g  
g . " " - g user gningmargorp
2. " " - Programming loose g.

To get 1st ifp : scanf("%[^\n]%.c ", str);

## OPERATORS

associativity

1 <sup>st</sup>	priority - primary operators -	( ) (left to right)
2 <sup>nd</sup>	priority - unary operator - ++, --, &, *, ~, ? (right to left)	associativity
3 <sup>rd</sup>	priority - Type casting -	(right - to left)
4 <sup>th</sup>	priority - Multiplicative ope - *, /, %.	(left to right)
5 <sup>th</sup>	Priority - Additive ope - +, -	(left to right)
6 <sup>th</sup>	- Shift ope - <<, >>	(left to right)
	left shift $\rightarrow a \ll b = a * 2^b$	(left to right)
	right shift $\rightarrow a \gg b = a / 2^b$	
7 <sup>th</sup>	Priority - Relational ope - >=, <=, >, <, !=	(left to right)
8 <sup>th</sup>	Priority - Equality ope - ==, !=	(left to right)
9 <sup>th</sup>	pri - Bitwise and (&) - &	( " )
10 <sup>th</sup>	pri - Bitwise XOR - ^	( " )
11 <sup>th</sup>	pri - Bitwise OR -	( " )
	Task : $33 \& 11 \wedge 9$	
12 <sup>th</sup>	pri - Logical AND - &&	( " )
13 <sup>th</sup>	pri - Logical OR -	( " )
14 <sup>th</sup>	pri - Ternary operator - (condition ? true : false)	
15 <sup>th</sup>	pri - Assignment ope - =, +=, -=, *=, /=, %=, <=, >=, &=,  =, ^= (right - to left)	(right to left)
16 <sup>th</sup>	pri - Cstrima operator - ,	(left to right).

1) To convert 'oh' to upper case  $\rightarrow \underline{ch = ch \hat{\wedge} 32;}$

**STRING:**

→ char str[50] = "Hello World";  
 → char str[50] = {'H', 'e', 'l', 'l', 'o', ' ', 'o'};  
 → char str2[50];  
 scanf("%s", str2); printf("%s", str2);

⇒ To give multiple words in input (sentence)

- (i) scanf("%[^\\n]", str2);
- (ii) gets(str2); [dangerous]
- (iii) fgets(str2, 5, stdin);

⇒ Different methods to print string.

- (i) printf("%s", str2);
- (ii) puts(str2);
- (iii) fputs(str2, 5); stdout default - stdcout

⇒ str = "Hello"; sizeof(str)/sizeof(str[0]); O/p = 6

Q13 Anagram : Program

str1 = "listen" , str2 = "silent".

All char in str1 also present in str2 , then  
its anagram.

use count[256] ; if count() = 0 then anagram.

**PROGRAM:**

str = "machine" → No char in this word is repeated.

Logic : int count[26] = {0};  
 for(int i=0; str[i] != '\0'; i++)  
 { if(isalpha(str[i]))  
 { int index = tolower(str[i]) - 'a';  
 if(count[index] > 0) return 0;  
 count[index]++; } }

## PANGRAM:

→ A sentence containing all letters in alphabet.

Logic: int alphabet[26] = {0};

index of alphabets present in sentences are '1'.

if any index in alphabet[] is 0, then not pangram.  
else pangram.

## STRING ROTATION: K rotations

right rotation → (i) temp[i] = str[size - K + i];

(ii) str[i] = str[i - K];

(iii) str[i] = temp[i];

left rotation → (i) temp[i] = str[i];

(ii) str[i] = str[i + K];

(iii) str[size - K + i] = temp[i];

## VALID PARENTHESES: (using stack in array):

Logic: char stack[1000], int top = -1, int size = strlen(str)

for (i = 0; i < size)

{ char curr = str[i]

    if (curr == '(' || curr == '[' || curr == '{') stack[++top] = curr;

    else if (curr == ')' || curr == ']' || curr == '}')

        if (top < 0) return false;

        char topchar = stack[top--];

        if ((curr == ')') && topchar == '(') || ((curr == '}') && topchar == '{') || ((curr == ']') && topchar == '['))

            return false;

    3 3

    return top == -1;

**INBUILD FUNCTIONS :**

`string.h → string header`

- (i) `strlen (char [ ]) → length of char [ ]`
- (ii) `strcpy (char1, char2) → char2 value is copied in char1`
- (iii) `strncpy (char1, char2, length) →`
- (iv) `strcat (c1, c2) → c2 value is concatenated to c1`
- (v) `strncat (c1, c2, z) → concatenates the limited length`
- (vi) `strcmp (c1, c2) → returns the difference btw 2 strings.`
- (vii) `strcmp (c1, c2; 4) → compares first 4 letters only.`
- (viii) `strchr (r, 'u') → prints 'u' in the string.`
- (ix) `strchr (r, 'u') → print 'u' from the last occurrence`
- (x) `strstr (str, "sth") → prints the occurrence of substring in string`

**REMOVE DUPLICATES IN STRING:**

```

for (i to size)
    for (j to i)
        if (str[i] == str[j]) → break;
        if (j == i) → str[index++ = str[i]];
        str[index] = '0';
    
```

30/8

<sup>num of elements</sup>  
**SUBSTRING & SUBSEQUENCE :**

all substrings are subsequence but not all  
subsequence are substring

## RECURSION :

factorial  $\Rightarrow$  if( $n == 1 \text{ || } n == 0$ ) return 1;  
return  $n * \text{factorial}(n-1)$ ;

Power( $a^b$ )  $\Rightarrow$  if( $b == 0$ ) return 1;  
return  $b * \text{pow}(b, e-1)$ ;

## Task:

Fib sequence

sum of all elements in array

Palindrome (string)

solve any pattern in recursion.

## POINTERS:

of variable that stores the address of another variable.

Format specifier - "%p" - hexadecimal ~~addr~~ address.

contains data

LITERALS : constant values that are assigned to the variables.

INTEGER PROMOTION : some datatypes like char, short int takes less number of bytes than 'int', these datatypes are automatically promoted to int or unsigned int, ~~this~~ when an operation is performed on them.

Q: When an operation is performed b/w two short values  
the size will be ④  $\rightarrow$  integer promotion.

Q: size of pointers for all datatypes is 8 bytes

Q: Difference between null and void pointer.

## TYPES OF POINTERS

### (i) Primitive datatype pointer:

`int* p, float* q, double* r, long* v;`

### (ii) array pointers:

array base address = its first element address

Eg: `int arr[] = {1, 2, 3, 4, 5};`

`int * f = arr // no '8' used because array is a pointer`

`for(int i=0; i<5; i++)`

`printf("%d ", arr[i]); // O/P: 1 1 1 1 1`

### (iii) arithmetic pointers:

copy → Performs arithmetic operations in pointers.

Eg: `printf("%d ", *f); f++; // O/P: 1 2 3 4 5`

### (iv) Double pointers:

cd pointer that stores address of another pointer.

Used while 2D array ↗ using

### (v) Null pointer:

- \* When the null pointer is de-referenced then the system will crash
- \* Can be assigned to any pointer
- \* Used to identify the termination

\* Syntax: type pt\_name = NULL; type pt\_name = 0;

### (vi) Wild pointer:

No value is assigned to the pointer.

### (vii) Void pointer:

Can store the address of values of all datatypes.

Cannot be de-referenced directly, it should be typecast

Eg: `p = &a, *p; (Wrong)`

`printf("%d", *(int *)p); (Correct)`

### (vi) Function pointer:

`&function-name` — is called code reference.

Eg: `int add(int a, int b); // function definition`  
`return a+b;`

`main()`

```
{ int (*ptr)(int,int)=&add; //using code ref to pointer  
(*ptr)(4,3); // function call  
}
```

### (vii) Dangling pointer:

Pointer that points to a memory location that has been deallocated or is no longer valid.

Eg: `int *p = (int *) malloc(sizeof(int));`  
`printf("%d", *p);`  
`free(p);`  
`printf("%d", *p); // Dangling pointer.`

### (viii) Constant pointer: (Address - constant)

the address once assigned it cannot be reassigned.

Eg: `int * const p = &a;`

### (ix) Pointers to constant: It points to the constant variable value.

Eg: `const int c = 5;`

`const int *q; // pointer used to store const value`  
`int *r;`

### (Value - constant)

### Storage class :

Auto , static , extern , register.

Auto - local variable (default storage)

- variables can be accessed only within the block.

- stored in stack memory.

### register :

static : - Initialized only once & exist till the termination

No new memory is allotted because they aren't redeclared

- Restricting the usage of function [in function]

- Used in

### Extern :- Global variable

- Only declaration occurs definition will be in some other file.

### STRUCTURE :

→ User-defined datatype

→ used to group different datatypes into a single type.

→ Only declaration is done, the memory is allocated when only when it is undefined.

→ Can see initialize the value in 3 methods.

(i) dot operator `stu.name = "ABC";`

(ii) Using initializer list `stu = {"ABC", 12, 12345}`

(iii) Designated initializer list `stu.name = "ABC"`

→ `typedef` - keyword - provide some meaningful names to the already existing variables in a program.

### Nested structure:

→ structure within structure.

→ One structure can be declared inside another structure in the same way structure members are declared inside a structure.

**structure pointer:** the pointer which points to the address of the memory block that stores a structure.

Eg: struct node

{ int data;

    struct node \* next; } // structure pointer declaration

Task:

Write code to add, mul, & complex numbers using struct.

### PADING:

In C it adds one or more empty bytes between the memory addresses to align the data in memory.

### UNION: (Shared memory allocation)

Similar to structure. (user defined data type).

But all the members in the C union are stored in the same memory location.

so only one member can be accessed at a time.

## DYNAMIC MEMORY ALLOCATION :

→ stdlib.h → stored memory in heap.

Malloc - as per the given length of datatype the memory is allotted random.

- Return type void.

Calloc - also void return type. similar to malloc

- But continuous memory allocation.

- Initialize 0 in all memory not garbage value.

Realloc - → void pointer return type.

Used to extend / shrink the size of the memory created by malloc , calloc .

free - clear the memory created before.

## POINTER ARITHMETIC :

→ Increment , decrement can be performed in pointer

→ Operations which can be performed.

- (i) addition of an integer or Increment

- (ii) subtraction of an integer or Decrement

- (iii) subtraction of one pointer from another pointer  
(same type).

- (iv) Relation operators can be used btw 2 pointers.