Convert char to int → int a = (int) ch[i];

to take name that have space → cin.getline(arr, 50);
↓
this is max length of input

OR cin.getline(arr, 50, t)
↓
termination

to find length → strlen(name);

to compare → strcmp(a, b);

Convert lower case to upper case = char - 'a' + 'A';

Convert upper case to lower case = char - 'A' + 'a';

to take name that have space in string → getline(cin, str)

major used function → Compare
substr
find
pop_back
push_back
length
size
empty
erase

a.substr(start, how many);

Compare → a.(compare(b) == 0 so both are same
else not same

if output is 0 than both string are same else not same

to find word in string → a.find(abc)

if it exist it return index else it return npos (use this std::string::npos)

Convert int to char ⇒ num + '0';

Convert char to int ⇒ char - '0';

Comparision operator

Sort in ascending order

```
static bool myComparison(const pair<int,int>& a
        , const pair<int,int>& b)
{
    return a.first < b.first;
}
```

☆ String ki problem me agar kisi character ko delete karna h ya koi order maintain karna h to stack try karna chyea

to replace in string → a.replace(start, howmuch, str2);

to erase in string → a.erase(start, how much);

Convert string to int → int x = stoi(string);

eg:- {"23"} → 23

Palindrom in string :-

eg:- | n | o | o | n |

for odd length

| n | o | o | n | → n

i j

| n | o | o | n |

i j

| n | o | o | n | → o

i o

| n | o | o | n |

i o

| n | o | o | n | → o

i o

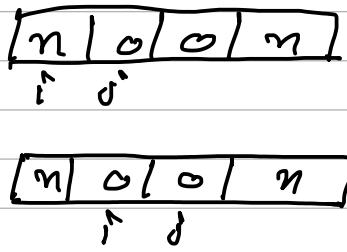
| n | o | o | n | → n

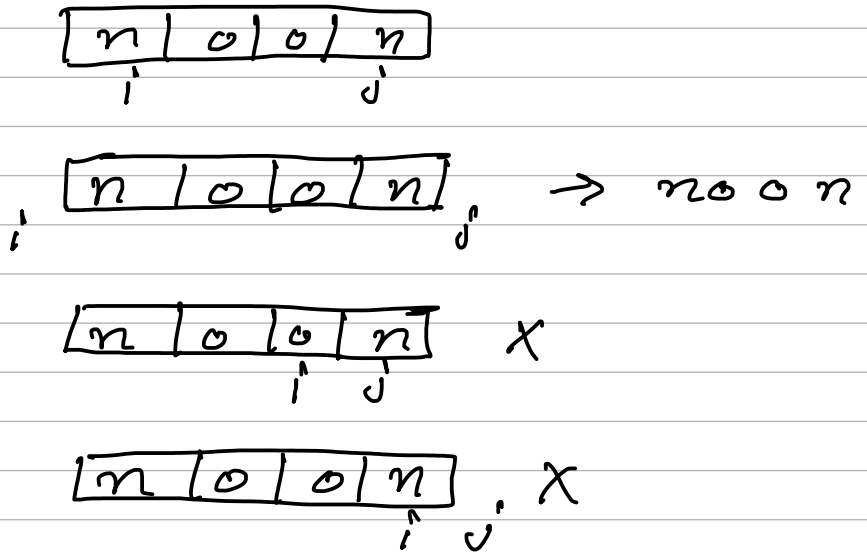
i j

| n | o | o | n | X

i o

for even length





longest palindromic substring

```
class Solution {
public:
    int find(string s, int i, int j)
    {
      int count = 0;
      while(i >= 0 && j < s.length && s[i] == s[j])
      {
        count++;
        i--;
        j++;
      }
     return count;
    }

  int countSubstrings(string s)
    {
      int count = 0;
      int n = s.length();

     for(int i=0; i<n; i++)
      {
        int odd = find(s, i, i);
        count = count + odd;

        int even = find(s, i, i+1);
        count = count + even;
      }

    return count;
```

242
917
14
345
Isomorphic strings
Reorganise strings
Group Anagrams
Longest palindromic substring
find index of first occurrence in a string
String to integer

String Compression — help Ramu
Integer to Roman — house Robber
Zig Zag Conversion

Largest no — 179
Custom sort string — 791
verify Alien Dictionary — 953
Longest word in Dictionary through deleting 524

sort ⇒ sort (s.begin(), s.end());

it give us in accending order

To make in decending order sort we make a compare function

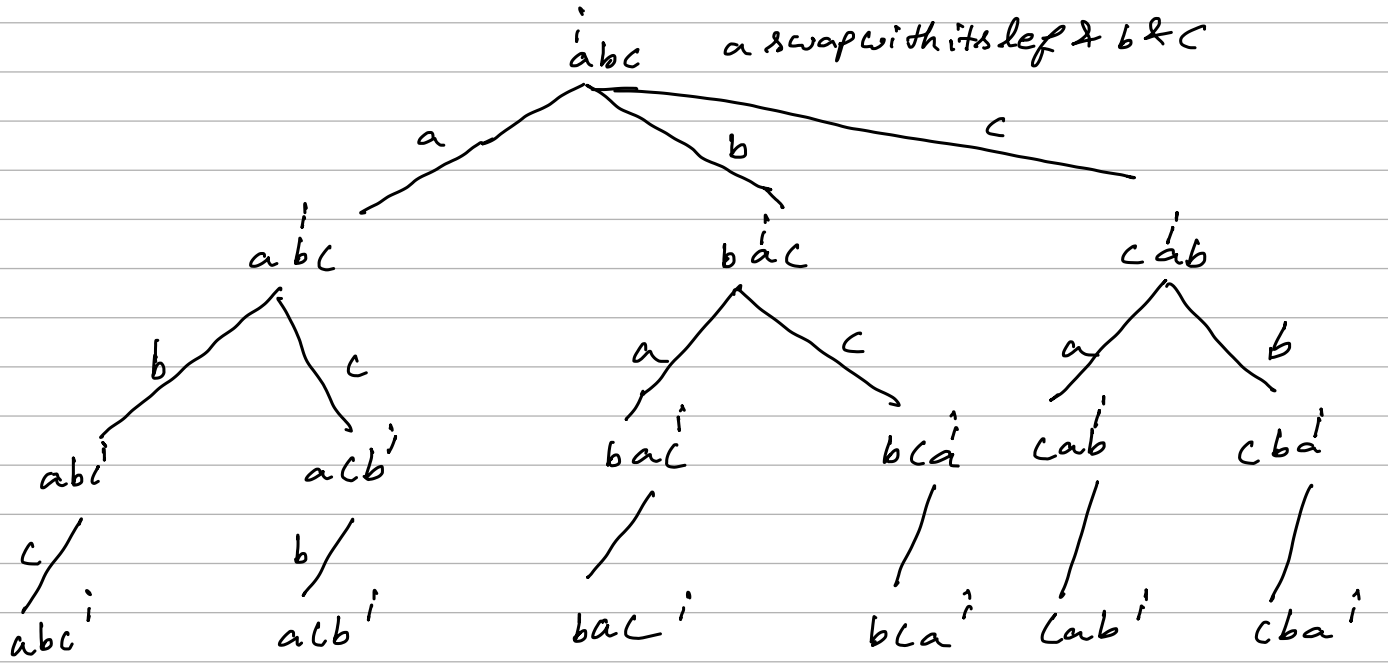
```
bool cmp( char first, char second)
{
    return first > second;
}

sort(s.begin(), s.end(), cmp);
```

Subsequence of a string

eg:- abc → a, b, c, ab, bc, ac, abc

```
void printSubsequences(string str, string output, int i)
{
    if(i >= str.length())
    {
        cout << output << endl;
        return;
    }
    printSubsequences(str, output, i+1);
    output.push_back(str[i]);
    printSubsequences(str, output, i+1);
}
```

Permutation of a String




```
void permutation (vector<int> nums, vector<vector<int>> &ans, int index)
{
    if(index >= nums.size())
    {
        ans.push_back(nums);
        return;
    }

    for(int i = index; i <= nums.size(); i++)
    {
        swap(nums[index], nums[i]);
        permutation (nums, ans, index+1); // for next index
        // backtrack
        swap(nums[index], nums[i]);
    }
};
```

if want all diff permutations use set their

```
void permutation( string nums, set<string>& set , int index)
{
    if(index >= nums.size())
     {
        set.insert(nums);
       return;
     }

    for( int i = index; i <= nums.size() ; i++)
     {
        swap(nums[index], nums[i]);
        permutation ( nums, ans, index+1); // for next index
        // backtrack
       swap(nums[index], nums[i]);
     }
}.
```

to iterate set

```
for( auto it = c.begin(); it != c.end(); it++)

   {
        cout << *it << endl;
    }
```

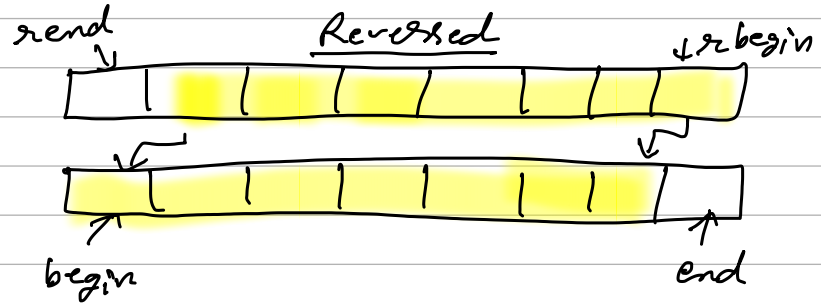
Map

if element found it give index of that else it give the end so not found

```
if( m.find() != m.end())
{

}
```

erase ⇒ `m.erase( key );`

Reverse ⇒



`m.rend();` return a reverse iterator to the end
`m.rbegin();` " " " " " beginning

size = `m.size()` ⇒ return no of element in map

clear ⇒ `m.clear()` → clear the content from the map

insert ⇒ `m.insert( {key, value}, {key, value} );`

swap ⇒ swap to map ⇒ `m1.swap(m2);`

at ⇒

empty ⇒

erase ⇒

count ⇒

find ⇒

To find diff or count no's in b/w iterators

`auto diff = std::distance(first, second);`

Priority Queue

```
class item {

  public:
        float devide;
         int num;
         int dem;
        item (float devide, int num, int dem)
        {
          this -> devide = devide;
          this -> num = num;
          this -> dem = dem;
         }

  };

// Comparison function

struct itemComparator {

bool operator() (const item& a, const item& b) const {

    return a.devide > b.devide;

} };

// Priority Queue of items

typedef priority_queue<item, vector<item>,
 itemComparator> PriorityQueue;

  // to push elements

  PriorityQueue pq;

pq.push(item( devide, num, dem ));
```

// get element from queue

```
auto topElement = pq.top;
topElement.devide;
topElement.num;
topElement.dem;
```

Remember:- $\frac{1}{2} = 0$

$\frac{1}{(float)2} = 0.5$

if no is in float and want 2 decimal points

```
double roundNumber = round(no*100)/100;
```

Right shift

$x >> k \Rightarrow \left(\frac{x}{2^k}\right)$

Sign

0 0 ----- 0 1 1 0 1

↑

31st

to check

0 - - - - - - - - 1 1 0 1

1 1 1 - - - - - - 0 0 1 0

+ 1

same 1 1 0 0 1 1

0 → positive

1 → -ve

Left shift

$x << k = x \times 2^k$

NOT

$x = \sim(x)$

1. flip

2. check -ve

Yes Stop

↓

2's

Precedence and Associativity of operators

Operator	Description	Precedence level	Associativity
( ) [ ] → .	Function call Array subscript Arrow operator Dot operator	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes	2	*Right to Left*
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right
<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
= = !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
? :	Conditional operator	13	*Right to Left*
= *= /= %= += - = &= ^= |= <<= >>=	Assignment operators	14	*Right to Left*
,	Comma operator	15	Left to Right

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Extended ASCII Codes

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	|	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	▉	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	n
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	2
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Exercise

What is the output of the following programs; assume stdio.h is included in all programs.

1.
```
int main(void)
{
    int a=-3;
    a = -a-a+!a;
    printf("%d\n",a);
    return 0;
}
```

2.
```
int main(void)
{
    int a=2,b=1,c,d;
    c = a<b;
    d = (a>b) && (c<b);
    printf("c=%d, d=%d\n",c,d);
    return 0;
}
```

3.
```
int main(void)
{
    int a=9,b=15,c=16,d=12,e,f;
    e = !(a<b || b<c);
    f = (a>b) ? a-b: b-a;
    printf("e=%d, f=%d\n",e,f);
    return 0;
}
```

4.
```
int main(void)
{
    int a=5;
    a=6;
    a=a+5*a;
    printf("a=%d\n",a);
    return 0;
}
```

5.
```
int main(void)
{
    int a=5, b=5;
    printf("%d,%d\t",++a,b--);
    printf("%d,%d\t",a,b);
    printf("%d,%d\t",++a,b++);
    printf("%d,%d\n",a,b);
    return 0;
}
```

6.
```
int main(void)
{
    int x,y,z;
    x = 8++;
    y = ++x++;
    z = (x+y)--;
    printf("x=%d, y=%d, z=%d\n",x,y);
```

```
    return 0;
}
```

7.
```
int main(void)
{
    int a=4, b=8, c=3, d=9, z;
    z = a++ + ++b * c-- - --d;
    printf("a=%d, b=%d, c=%d, d=%d, z=%d\n",a,b,c,d,z);
    return 0;
}
```

8.
```
int main(void)
{
    int a=14,b,c;
    a=a%5;
    b=a/3;
    c=a/5%3;
    printf("a=%d, b=%d, c=%d\n",a,b,c);
    return 0;
}
```

9.
```
int main(void)
{
    int a=15,b=13,c=16,x,y;
    x = a-3%2+c*2/4%2+b/4;
    y = a = b+5-b+9/3;
    printf("x=%d, y=%d\n",x,y);
    return 0;
}
```

10.
```
int main(void)
{
    int x,y,z,k=10;
    k+=(x=5, y=x+2, z=x+y);
    printf("x=%d, y=%d, z=%d, k=%d\n",x,y,z,k);
    return 0;
}
```

11.
```
int main(void)
{
    float b;
    b=15/2;
    printf("%f\t",b);
    b=(float)15/2 + (15/2);
    printf("%f\n",b);
    return 0;
}
```

12.
```
int main(void)
{
    int a=9;
    char ch='A';
    a = a+ch+24;
    printf("%d,%c\t%d,%c\n",ch,ch,a,a);
    return 0;
}
```

13.
```
int main(void)
{
    int a,b,c,d;
    a=b=c=d=4;
```

```
    a*=b+1;
    c+=d*=3;
    printf("a=%d, c=%d\n",a,c);
    return 0;
}
```

14.
```
int main(void)
{
    int a=5,b=10,temp;
    temp=a, a=b, b=temp;
    printf("a=%d, b=%d\n",a,b);
    return 0;
}
```

15.
```
int main(void)
{
    int a=10, b=3, max;
    a>b ? max=a : max=b;
    printf("%d\n",max);
    return 0;
}
```

16.
```
int main(void)
{
    int a=5, b=6;
    printf("%d\t",a=b);
    printf("%d\t",a==b);
    printf("%d %d\n",a,b);
    return 0;
}
```

17.
```
int main(void)
{
    int a=3,b=4,c=3,d=4,x,y;
    x = (a=5) && (b=7);
    y = (c=5) || (d=8);
    printf("a=%d, b=%d, c=%d, d=%d, x=%d, y=%d\n",a,b,c,d,x,y);
    x = (a==6) && (b=9);
    y = (c==6) || (d=10);
    printf("a=%d, b=%d, c=%d, d=%d, x=%d, y=%d\n",a,b,c,d,x,y);
    return 0;
}
```

18.
```
int main(void)
{
    int a=10;
    a=a++;
    a = a++ * a--;
    printf("%d\n",a);
    printf("%d\n",a++ * a++);
    return 0;
}
```

19.
```
int main(void)
{
    int a=2, b=2, x, y ;
    x = 4*(++a * 2 + 3);
    y = 4*(b++ * 2 + 3 );
    printf("a=%d, b=%d, x=%d, y=%d\n",a,b,x,y);
    return 0;
}
```

1. 6
2. c=0, d=1
3. e=0, f=6
4. a=36
5. 6,5 6,4 7,4 7,5
6. Error, Expressions like 8++, ++x++, (x+y)-- are not valid.
7. a=5, b=9, c=2, d=8, z=23
8. a=4, b=1, c=0
9. x=17, y=8
10. x=5, y=7, z=12, k=22
11. 7.000000 14.500000
12. 65,A 98,b
13. a = 20, c = 16
    Precedence of + operator is more than that of += . Associativity of compound assignment operators is from right to left.
14. a=10, b=5
15. Since the precedence of assignment operator is less than that of conditional operator, the conditional statement is

    interpreted as- (a>b?max = a : max) = b;
    Now this statement reduces to statement 10=b, which results in an error. So the solution is to put parentheses around the third operand as- a>b? max = a : (max = b);
16. 6 1 6 6
17. a=5, b=7, c=5, d=4, x=1, y=1
    a=5, b=7, c=5, d=10, x=0, y=1
18. Result of these types of expressions is undefined.
19. a=3, b=3, x=36, y=28

CHAPTER

MISCELLANEOUS CONCEPTS 21

21.1 Introduction

In this chapter we will cover the topics which are useful for interviews and exams.

21.2 Hacks on Bit-wise Programming

In C and $C++$ we can work with bits effectively. First let us see the definitions of each bit operation and then move onto different techniques for solving the problems. Basically, there are six operators that C and $C++$ support for bit manipulation:

Symbol	Operation
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive-OR
<<	Bitwise left shift
>>	Bitwise right shift
~	Bitwise complement

21.2.1 Bitwise AND

The bitwise AND tests two binary numbers and returns bit values of 1 for positions where both numbers had a one, and bit values of 0 where both numbers did not have one:

```
  01001011
& 00010101
----------
  00000001
```

21.2.2 Bitwise OR

The bitwise OR tests two binary numbers and returns bit values of 1 for positions where either bit or both bits are one, the result of 0 only happens when both bits are 0:

```
  01001011
| 00010101
----------
  01011111
```

21.2.3 Bitwise Exclusive-OR

The bitwise Exclusive-OR tests two binary numbers and returns bit values of 1 for positions where both bits are different; if they are the same then the result is 0:

```
  01001011
^ 00010101
----------
  01011110
```

21.2.4 Bitwise Left Shift

The bitwise left shift moves all bits in the number to the left and fills vacated bit positions with 0.

```
  01001011
<< 2
--------
  00101100
```

21.2.5 Bitwise Right Shift

The bitwise right shift moves all bits in the number to the right.

```
    01001011
>> 2
    --------
    ??010010
```

Note the use of ? for the fill bits. Where the left shift filled the vacated positions with 0, a right shift will do the same only when the value is unsigned. If the value is signed then a right shift will fill the vacated bit positions with the sign bit or 0, whichever one is implementation-defined. So the best option is to never right shift signed values.

21.2.6 Bitwise Complement

The bitwise complement inverts the bits in a single binary number.

```
    01001011
~
    ---------
    10110100
```

21.2.7 Checking Whether K-th Bit is Set or Not

Let us assume that the given number is n. Then for checking the K^{th} bit we can use the expression: $n \,\&\, (1 \ll K - 1)$. If the expression is true then we can say the K^{th} bit is set (that means, set to 1).

Example:

n	01001011	$K = 4$
$1 \ll K - 1$	00001000	
$n \,\&\, (1 \ll K - 1)$	00001000	

21.2.8 Setting K-th Bit

For a given number n, to set the K^{th} bit we can use the expression: $n \mid 1 \ll (K - 1)$

Example:

n	01001011	$K = 3$
$1 \ll K - 1$	00000100	
$n \mid (1 \ll K - 1)$	01001111	

21.2.9 Clearing K-th Bit

To clear K^{th} bit of a given number n, we can use the expression: $n \,\&\, \sim(1 \ll K - 1)$

Example:

n	01001011	$K = 4$
$1 \ll K - 1$	00001000	
$\sim(1 \ll K - 1)$	11110111	
$n \,\&\, \sim(1 \ll K - 1)$	01000011	

21.2.10 Toggling K-th Bit

For a given number n, for toggling the K^{th} bit we can use the expression: $n \,\hat{}\, (1 \ll K - 1)$

Example:

n	01001011	$K = 3$
$1 \ll K - 1$	00000100	
$n \,\hat{}\, (1 \ll K - 1)$	01001111	

21.2.11 Toggling Rightmost One Bit

For a given number n, for toggling rightmost one bit we can use the expression: $n \,\&\, n - 1$

Example:

n	01001011
$n - 1$	01001010
$n \,\&\, n - 1$	01001010

21.2.12 Isolating Rightmost One Bit

For a given number n, for isolating rightmost one bit we can use the expression: $n \,\&\, -n$

Example:

n	01001011
$-n$	10110101
$n \,\&\, -n$	00000001

Note: For computing $-n$, use two's complement representation. That means, toggle all bits and add 1.

21.2.13 Isolating Rightmost Zero Bit

For a given number n, for isolating rightmost zero bit we can use the expression: $\sim n \,\&\, n+1$

Example:

n	01001011
$\sim n$	10110100
$n+1$	01001100
$\sim n \,\&\, n+1$	00000100

21.2.14 Checking Whether Number is Power of 2 or Not

Given number n, to check whether the number is in 2^n form for not, we can use the expression: $if(n \,\&\, n-1 == 0)$

Example:

n	01001011
$n-1$	01001010
$n \,\&\, n-1$	01001010
$if(n \,\&\, n-1 == 0)$	0

21.2.15 Multiplying Number by Power of 2

For a given number n, to multiply the number with 2^K we can use the expression: $n \ll K$

Example:

n	00001011	$K = 2$
$n \ll K$	00101100	

21.2.16 Dividing Number by Power of 2

For a given number n, to divide the number with 2^K we can use the expression: $n \gg K$

Example:

n	00001011	$K = 2$
$n \gg K$	00000010	

21.2.17 Finding Modulo of a Given Number

For a given number n, to find the %8 we can use the expression: n & $0x7$. Similarly, to find %32, use the expression: n & $0x1F$

Note: Similarly, we can find modulo value of any number.

21.2.18 Reversing the Binary Number

For a given number n, to reverse the bits (reverse (mirror) of binary number) we can use the following code snippet:

```
unsigned int n, nReverse = n;
int s = sizeof(n);
for (; n; n >>= 1) {
    nReverse <<= 1;
    nReverse |= n & 1;
    s--;
}
nReverse <<= s;
```

Time Complexity: This requires one iteration per bit and the number of iterations depends on the size of the number.

21.2.19 Counting Number of One's in Number

For a given number n, to count the number of $1's$ in its binary representation we can use any of the following methods.

Method1: Process bit by bit with bitwise and operator

```
unsigned int n;
unsigned int count=0;
while(n) {
    count += n & 1;
    n >>= 1;
}
```

Time Complexity: This approach requires one iteration per bit and the number of iterations depends on system.

Method2: Using modulo approach

```
unsigned int n;
unsigned int count=0;
```

```
while(n) {
    if(n%2 ==1)
        count++;
    n = n/2;
}
```

Time Complexity: This requires one iteration per bit and the number of iterations depends on system.

Method3: Using toggling approach: $n \& n-1$

```
unsigned int n;
unsigned int count=0;
while(n) {
    count++;
    n &= n - 1;
}
```

Time Complexity: The number of iterations depends on the number of 1 bits in the number.

Method4: Using preprocessing idea. In this method, we process the bits in groups. For example if we process them in groups of 4 bits at a time, we create a table which indicates the number of one's for each of those possibilities (as shown below).

0000→0	0100→1	1000→1	1100→2
0001→1	0101→2	1001→2	1101→3
0010→1	0110→2	1010→2	1110→3
0011→2	0111→3	1011→3	1111→4

The following code to count the number of 1s in the number with this approach:

```
int Table = {0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4};
int count = 0;
for(; n; n >>= 4)
    count = count + Table[n & 0xF];
return count;
```

Time Complexity: This approach requires one iteration per 4 bits and the number of iterations depends on system.

21.2.20 Creating Mask for Trailing Zero's

For a given number n, to create a mask for trailing zeros, we can use the expression: $(n \& -n) - 1$

Example: n	01001011
$-n$	10110101
$n \& -n$	00000001
$(n \& -n) - 1$	00000000

Note: In the above case we are getting the mask as all zeros because there are no trailing zeros.

27.2.21 Swap All Odd and Even Bits

Example: n	01001011
Find even bits of given number (evenN) = n & 0xAA	00001010
Find odd bits of given number (oddN) = n & 0x55	01000001
evenN >>= 1	00000101
oddN <<= 1	10000010
Final Expression: evenN | oddN	10000111

21.2.22 Performing Average without Division

Is there a bit-twiddling algorithm to replace $mid = (low + high) / 2$ (used in Binary Search and Merge Sort) with something much faster?

We can use $mid = (low + high) >> 1$. Note that using $(low + high) / 2$ for midpoint calculations won't work correctly when integer overflow becomes an issue. We can use bit shifting and also overcome a possible overflow issue: $low + ((high - low)/ 2)$ and the bit shifting operation for this is $low + ((high - low) >> 1)$.

21.3 Other Programming Questions

Problem-1 Give an algorithm for printing the matrix elements in spiral order.

Solution: Non-recursive solution involves directions right, left, up, down, and dealing their corresponding indices. Once the first row is printed, direction changes (from right) to down, the row is discarded by incrementing the