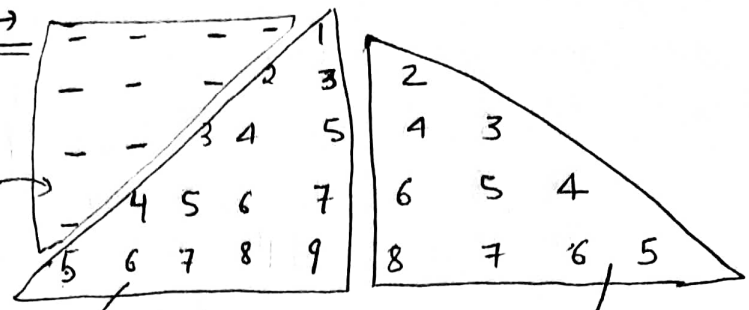
Lec-5

04/Feb/2023

⊙ Numeric Full Pyramid →

row	space
0	4
1	3
2	2
3	1
4	0

n = 5



n-1-row count

row	no of element
0	1 (start from 1)
1	2 (start from 2)
2	3 (start from 3)
3	4 (start from 4)
4	5 (start from 5)

start from row+1
and print row+1 elements

reverse count

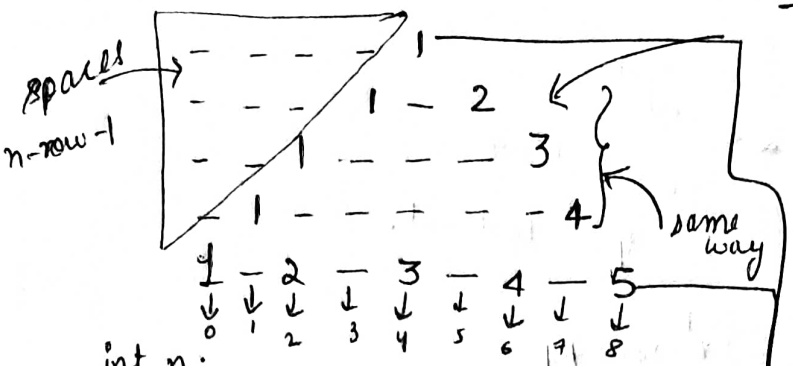
row	
0	0 elements ~~start~~
1	1 (start from 2)
2	2 (starts from 4)
3	3 (starts from 6)
4	4 (starts from 8)

starts from 2* row.

⇒ We know how to print spaces.

```
int n;
cin >> n;
for (int row = 0; row < n; row++){
    // spaces
    for(int col = 0; col < n-row-1; col++){
        cout << " ";
    }
    // numbers
    for (int col = 0; col < row+1; col++){
        cout << row+col+1;
    }
    // reverse counting
    int start = 2*row;
    for (int col = 0; col < row; col++){
        cout << start;
        start = start - 1;
    }
    cout << endl;
}
```

(2) Numeric Hollow Pyramid →



row	1 element
0	1
1	3
2	5
3	7
4	9

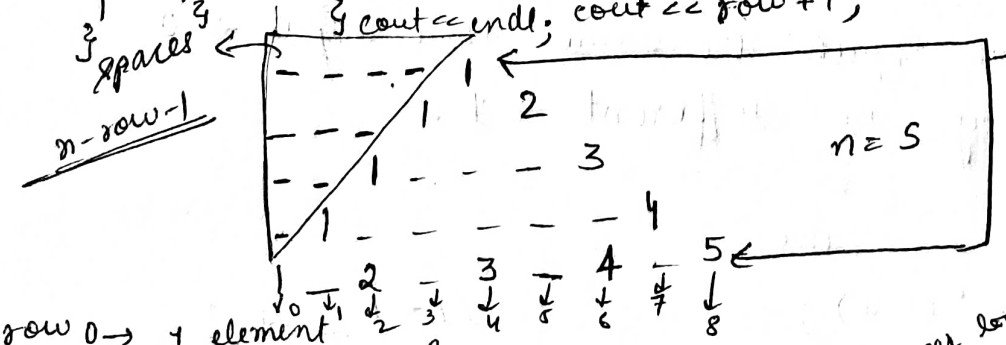
2* row + 1

→ same way.

```
int n;
cin >> n;

for (int row = 0; row<n; row++){
    //spaces
    for (int col = 0; col < n-row-1; col++){
            cout << " ";
    }
    int start = 1;
    // numbers with spaces in between.
    for (int col = 0; col < 2* row+1; col++){
        // first or last row.
        ~~if (col = 0 || col == 2*row+1-1)~~
        if (row == 0 || row == n-1) {
            //even
            if (col %2 == 0 ){
                cout << start;
                ~~cout <<~~
                start = start +1;
            }
            else {
                cout << " ";
            }
        }else{ if (col == 0)
                cout << 1;
            else if ( col = 2* row + 1 -1)
    }cout<<endl; cout << row+1;
}
```

even no → number
odd no → space

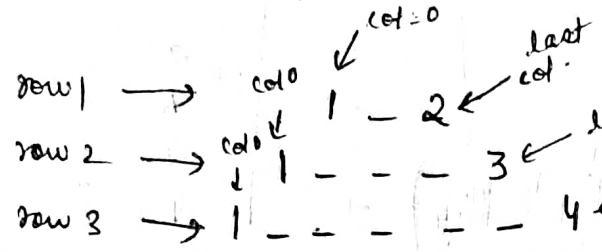


→ different from other rows. (counting is printed in these rows).

row 0 → 1 element
row 1 → 3 elements
row 2 → 5 elements
row 3 → 7 elements
row 4 → 9 elements

odd no. → 2* row + 1 ← inner loop.

only for 1st and last row →
if even col → no.
if odd col → space



if total col = 2 * row + 1
and first col = 0 ⟶ print 1
so last col = 2row+1-1
↳ print row+1.

⊙ BITWISE OPERATORS → AND, OR, NOT, XOR

Bits → 0 or 1.

Operators that are used in bit level are called Bitwise operator.

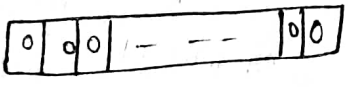
a	b	a & b (AND)	a | b (OR)	a ^ b (XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

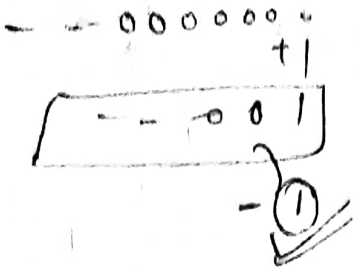
a	not a (~a) (tilda)
0	1
1	0

AND → The result is true only if all the inputs are true.
OR → Even if one input is true the result will be true.
NOT → flip the bits (a: 0→1, 1→0; ~a: b→nb).

XOR → If two inputs are same then output → 0
If they are different → 1.

```
bool a = false;
cout << (~a);        // -1 ? why?
```

⇒ a → 



① cout<<(5 & 10) → o/p = 0

5 → 0101
10 → 1010
0010

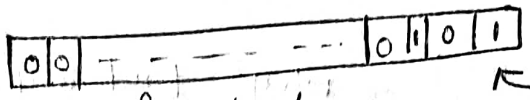
• cout << (3 | 4) → o/p → 7

3 → 011
4 → 100
111 → 7

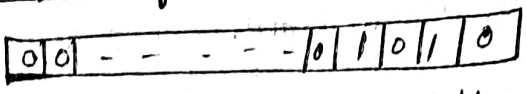
① Left & Right Shift Operator

• Left shift (<<) →

int a = 5;



→ left shift by 1



So we can say that by left shift value is multiplied by 2.

a = 1

0000 ... 0001 → a = 1

<<1

000 ... 0010 → a = 2

<<1

000 ... 0100 → a = 4

<<1

000 ... 1000 → a = 8

• -12 << 1 → o/p = -6

• Right shift (>>) →

What happens when we try to shift negatively?

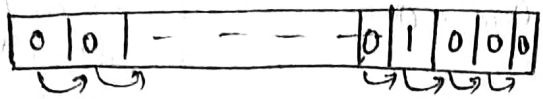
eg, a = 2

a = a << -1;

This will give us garbage value and we will get a warning that "left shift is negative".

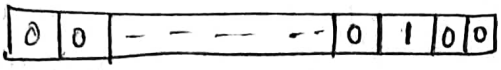
- Right shift (>>) ~~value is divided by 2.~~

```
int a = 8
a>>1;
```



a = 8

>>1



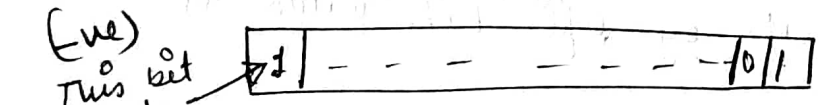
a = 4

Q → Can I say that by shifting right value is divided by 2? Find out the reason if no.

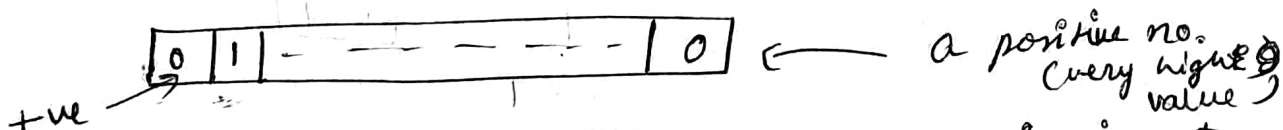
↳ NO.

⊙ -24 >>1 ⟶ -14 ⟶ why! Find out.

⇒ if we have a -ve no.



Let's right shift by 1

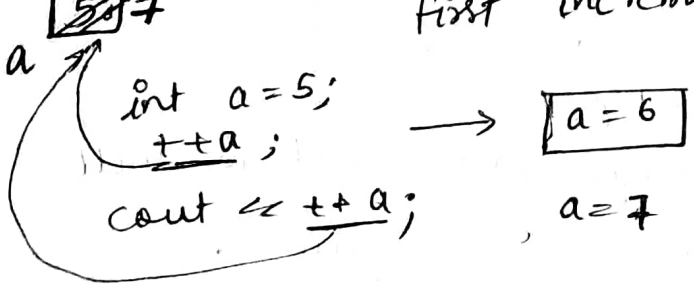


it becomes a positive no. But this is not true as if we divide a -ve no. by 2 then it doesn't become a high +ve no.

⊙ Pre / Post Increment / Decrement Operator →

- Pre increment (++a)

first increment a then you can use it.



① Post Increment →

```
int a = 5;
cout << a++;        // o/p → 5
```

(a++) → first use then increment.

a [5 6]

→
```
int a = 6;
int c = ++a + 1;
cout << c;          // o/p → 8
```

a [6 7] [8] ✓
c = a + 1
7 + 1 = 8

→
```
int a = 6;
int c = a++ + 1;
cout << c;          // o/p → 7
```

a [6 7] [7]
c = 6 + 1 = 7

② Pre decrement (--a) →
↳ first decrement then use.

③ Post Decrement (a--) →
↳ first use then decrement.

•
```
int a = 3;
int b = 4;
int c = (++a) * (--b);
cout << c;          // o/p = 12
```
a [3 4] b [4 3] c [12] ✓

•
```
int c = (--a) * (b++);
cout << c;          // o/p = 8
```
a [3 2] b [4 5]
c [8]
c = 2 × 4 = 8

•
```
int a = 5
cout << (++a) * (++a);
```
↳ It's output is compiler dependent.
correct ans is 49.

① Break and Continue keyword →

→ **Break** –

```
for (int i = 0; i<n; i++){
        cout << "Babbar";
}
cout << "Love";
```

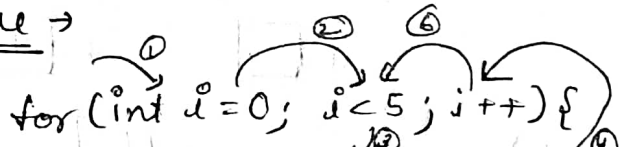
This loop print Babbar 5 times and at last Love.
let's add break.

```
for (int i = 0; i<n; i++){
        cout << "Babbar";
        break;
}
cout << "Love";
```

o/p → Babbar Love

⇒ So if we are traveling in a loop and we find break statement that means to terminate that loop and come out of it.

→ **Continue** →



without continue o/p should be → 0 1 2 3 4
with continue o/p nothing is printed.

continue skips the iteration.

```
for (int i = 0; i < 5; i++){
    if (i == 2)
        continue;
    cout << i;
}
```

o/p → 0 1 3 4

⊙ Variable Scoping →

```
int a;          // declared a integer variable a
int b = 5;      // initialized b as 5.
b = 10;         // update b's value.
// int b = 6;   // error → redefinitation
```

```
if (true){
    int b = 15;
    cout << b;      // no error.
}                   → o/p = 15.
```

This means the scope of b is between → there braces.

```
cout << b;          o/p → 10
```

These are local variables.

- Global variable → Anywhere can be used in a file.

```
int a = 10;  ——————→ // global variable.
int main(){
    cout << a;      // o/p = 10
}
```

It is BAD Practice to use global variable. Also they can be updated anywhere but the updatation scope is limited.

⊙ Operator Precedence →

2 + 3 * 1 / 3 - 1	2 + (3*1)/3 - 1
5 * 1 / 3 - 1	2 + (3/3) - 1
5/3 - 1	2 + 1 - 1
1 - 1 = 0	= 2

different results of same equation.
That's why in computer precedence of operators is decided.

Btw we will use bracket to avoid this collision.

For eg → (a+((b*c)/d) - a)

⊙ Switch case - Alternative of if-else statement.

```
switch (val)   ip
{                 → expression
   case 1: cout << "Love";
           break;
   case 2: cout << "Babbar";
           break;
   default: cout << "Suresh";
}
```

↑ readable

break statement after each case.

if-else ↓ not that readable

i/p → 1 → o/p → Love

2 → o/p → Babbar

any random value → 5, 500,

o/p → Suresh.

break is important.

H.W → Read about switch case and find out?

① Can we apply these in switch expression →

- → A character
- → "Babbar"
- → expression
- → 2 same cases.
- → -ve
- → float
- → "This is me"
- → bool

⊙ can we use continue in place of break?