4/02/2023

Pattern-8 Numeric full pyramid

```
_ _ _ _ 1
_ _ _ 2 3 2
_ _ 3 4 5 4 3
_ 4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

Pattern-1 Pattern-2

In every row counting is started from row+1 and counting is till row+1 only.

Step-1 First we have to print the spaces

$i=0 \rightarrow$ 4 spaces
$i=1 \rightarrow$ 3 spaces
$i=2 \rightarrow$ 2 spaces
$i=3 \rightarrow$ 1 space
$i=4 \rightarrow$ 0 space

Formulae for space $= n-i-1$

Code

```
int space = n-i-1;
for ( ; space>=0; space--) {
        cout << " ";
    }
```

This code will be printing the spaces.

Step-2 Now we have to build the pattern-1

```
1                 → i+1 times = 0+1 = 1 time loop
2 3               → i+1 times = 1+1 = 2 times loop
3 4 5
4 5 6 7
5 6 7 8 9
```

Now we will have build some formulae

$i=0 \rightarrow$ number starts from $i+1$ $(0+1)$.

$i=1 \rightarrow$ counting starts from $i+1$ $(1+1=2)$

Hence we have developed a formulae for this which is $i+1$. Also we have to run this loop till $i+1$ only.

Code

```
int start = i+1;
for (int j=0; j<i+1 ; j++) {
        cout << start;
        start= start +1;
}
```

Till now we have built the following pattern

```
        1
      2 3
    3 4 5
  4 5 6 7
5 6 7 8 9
```

Step-3 Now we have to do the reverse counting and build pattern-2

2				1 time loop	$i=0+1=1$
4	3			2 time loop	$i=2$
6	5	4		3 time loop	$i=3$
8	7	6	4	4 time loop	$i=4$

Run the loop till i only. In every row, number is starting from $2*i$ and we have to run the for loop till i only.

Code

```
Start = 2* i;
for (int j=0 ; j<i ; j++){
        cout << start;
        start = start -1; → Reverse counting
}
```

Now the pattern has been successfully printed.

Pattern-9 Numeric hollow pyramid

```
- - - - 1
- - - 1 - 2
- - 1 - - - 3
- 1 - - - - - 4
1 - 2 - 3 - 4 - 5
```

Step-1 First we will have to print the spaces & for that we need to find a formulae.

$i = 0$ → row-1 has 4 spaces
$i = 1$ → row-2 has 3 spaces
$i = 2$ → row-3 has 2 spaces
$i = 3$ → row-4 has 1 space
$i = 4$ → row-5 has 0 space

Formulae for space $= n - i - 1$

Code

```
int space = n-i-1;
```

```
for (; space >=0; space--) {
        cout <<"  ";
}
```

Till now we have printed the spaces & now we have to print numbers & then space & then numbers.

Step-2 In $i=0$, only 1 number is getting printed
$i=$ 1, 2 number + 1 space (3)
$i=2$, 2 number + 3 space (5)
$i=3$, 2 number + 5 space (7)
$i=4$, 5 number + 4 space (9)

So the loop will be run for odd number of times.

We can develop a formulae i.e $2*i+1$, till this the loop will be run.

Also from observation for row-1 ($i=0$) & row-5 ($i=4$) are different. Hence we need an if condition here.

Condition will be $i==0$ || $i==n-1$
↳ 1st row ↳ Last row

Also if the col is an even number, then only number will be printed

Now if it is not the 1st and also not the last row we can observe that 1st number

is always 1 & there are some spaces in between & the last number is one more than the row number.

Code

```
int start = 1;
for (int j=0; j<2*i+1; j++) {
                 1st row        last row
        if (i==0 || i==n-1) {
            if (j%2==0) { → col is even
                cout << start << " ";
                start = start + 1;
            }
        }
        else {
            if (j==0) { → 1st column
                cout << 1;
            }
            else if (j==2*i) { → last column
                cout << i+1;
            }
            else {
                cout << " ";
            }
        }
}
```

Bitwise operators

First we need to understand what are bits. Computer understand 0s & 1s & these are known as bits.
Bitwise operators can be applied on these bits.

(i) Bitwise AND

Symbol of AND is &.

This operator returns true (1) only when both a & b is 1. If any of bit is 0, then result of a & b is 0.

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

(ii) Bitwise OR

Symbol of OR is |.

This operator returns true (1) if any of the bit is 1. It returns false when both the bits are 0.

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

(iii) Bitwise NOT

Symbol of NOT is ~ (Tilda)

This will return the opposite of the input bit.

a	!a
0	1
1	0

(iv) Bitwise XOR

The symbol of XOR is ^.

If the bits are same, then the result will be 0 but if they are different, then the result will be 1.

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

Note → We can find out that 2 numbers are equal with the help of XOR operator.

a ^ a = 0 } By this we can find whether 2 numbers are equal or not.

Understanding NOT

```
bool a = false ;
cout << (~a) << endl ;
```

a → 0 0 0 0 0 0 0 0
~a → 1 1 1 1 1 1 1 1

By 1st bit, we get to the number is negative

2s complement to see the number.

0 0 0 0 0 0 0 0 → 1s complement
1
0 0 0 0 0 0 0 1 → 1

~a = -1.

Ques 1 What is the o/p of 2 & 3 ?

2 → 10
3 → 11

10 & 11 = 10 = 2 Ans

Ques 2 What is the o/p of 3 | 4 ?

3 → 011 011 | 100 = 111 = 7
4 → 100

Note → We can see that bit by bit operations are taking place.

Left and right shift operators

(i) Left shift operator
Symbol is <<

a = 5 → 000 --- 0101
In left shift move the bits towards left
↗ left shift
<< a a = a << 1 ; → Left shift 1 time

0000 --- 1010 → 10
In short we can say in left shift operator, the number is multiplied by 2.

Ex → 1 << 1 → 1 is left shifted by 1 times.
000 ... 001
Doing left shift, we get 0000 -- 010 = 2

(ii) Right shift operator
Symbol is >>

a = 8 000 -- 01000

a = a >> 1
O/p → 000 -- 00100 → 4

We are moving the bits towards right. In short, we can say that we are dividing the number by 2 in this case. But this is not always true. Try to find out why?

Suppose that we have a negative number & apply right shift on it, it might give a high positive value.

1 00 --- 0101 → -ve number
Now perform right shift on the above number

0 1 00 --- 0010 → high +ve number

Pre / Post Increment / Decrement operator
Many MCQs can be made on this topic

(i) ++a
This is pre-increment operator. This means first increment and then use a.

int a = 5;
++a; → Here a becomes 6

```
int b = 5;
cout << (++b);
```
→ 6 will be output

(ii) a++

This is the post increment operator. This means first use a & then increment the value of a.

```
int a = 5;
cout << (a++);
```
→ 5 will be output. But after this line value of a becomes 6.

Ex→
```
int a = 6
c = ++a + 1
cout << c << endl;
```
What will be the o/p of above code?

++ has higher priority than +

(++a) + 1
(++6) + 1 → 7 + 1 = 8 is the output.

Ex→
```
int a = 6;
int c = (a++) + 1;
cout << c;
```
What will be the o/p of above code?

First use & then increment

c = 6 + 1 = 7 is the output.

(iii) --a

This is the pre decrement operator. This means that first decrement and then use.

(iv) a-- This is the post decrement operator. This means that first use & then decrement.

Ex→

```
a = 3
b = 4
c = (++a) * (--b)
```

↓

4 * 3 = 12 is the output

```
c = (--a) * (b++)
```

↓

2 * 4 = 8

Ex→

```
int a = 5;
cout << (++a) * (++a);
```

This will have the o/p as 49 & we will discuss it when we study about the precedence of operators.

Break and continue keyword

(i) Break

```
for (int i=0 ; i<5 ; i++) {
        cout << "Bhavya";
        break;
}
```

As break; is encountered, the flow of the program will go outside the loop. After encountering break, the code below it inside the scope of loop will not be executed.

O/p → Bhavya

(ii) continue

```
for (int i=0; i<n; i++) {
        continue;
        cout << i;
}
```

O/p → } Nothing will be printed

As continue; is encountered, the code below it won't be executed but there will be no exit from the loop.

This is basically used if we want to skip any iteration.

Variable scoping

```
int a;      //declaration
int b = 5;  //initialization
b = 10;     // Updation
int b = 6;  Redefined & will give us error
```

```
main() {
    int b = 5;   //B1

    if ( ) {
    //B2 → int b = 7; → This won't give us error
    }                  because its scope will be
                       inside if only
}
```

B1 → within scope of main ()
B2 → within scope of if

Ex →
```
main ( ) {

    int b = 10;
    if (  ) {
        int b = 7;
        cout << b << endl;
    }
    cout << b << endl;
}
```

What is the output of the above code?
7 → if statement
10 → main() function

Global variable

```
int b=10; → Global variable
main ( ) {

}
```

Global variable can be accessed anywhere within the same file.

Note → Using global variable is considered as bad practice.
Redefinition of global variable can be done in other scope.

Operator precedence Priority of operators

Category	Operators	Associativity
Brackets	( ), [ ], →, ++, --	L to R
Unary	!, ~, ++, --, &	R to L
Multiplicative	*, /, %	L to R
Additive	+, -	L to R
Bitwise Shift	>>, <<	L to R
Relational	<, >, <=, >=	L to R
Equality	==, !=	L to R
Bitwise AND	&	L to R
Bitwise XOR	^	L to R

Note → We don't have to remember the above charts. We have to follow practice of using brackets.

If precedence is same, then we have to follow associativity i.e solve the expression from left to right or right to left.

Switch case
This is used when there are multiple conditions. However this can be done via if statements as well.

Syntax

```
switch (expr) {

  case 1 : ---  ; break;
  case 2 : ---  ; break;
      :
      :
  default : ---  ;

}
```

We have to follow a practice of using the break; after each case.

Note → Also if no case is matched with the expr, then default case will be executed.

What if break is not used?
If we don't use break, then whenever the case is matched, all the cases below that case will be executed.