The increment ++ and decrement - – operators increases and decreases the value of the variable by 1 respectively as seen in 'for' loop.

a++ means first use the value of a in the expression and then increment its value by 1, whereas the ++a means first increment the value of a and then use it in the expression.

If a = 10, then after –

y = ++a; // y = 11 and a = 11

y = a++; // y = 10 and a = 11

Now consider the following:

```
#include <stdio.h>
int main()
{
    int i, j = 6;
    i = j---2;
    printf("%d", i);
    return 0;
}
```

The above expression is valid. Compiler understands it as j-- -2 and not as j- --2 because – and ++operators works only on variables.

First j - - is evaluated giving 5, then 5-2 = 3. So **output is 3.**

Now consider the following expressions:

a) i = --3;
b) i, j = 2, k = 3; i = (j+k)++;

In a) expecting i = 2 or 3 (minus of minus 3) is invalid, because j -- or –-j are evaluated as j = j-1; so --3 cannot be evaluated as 3 = 3 – 1, which is absurd. So compiler will give an error.

Similarly in b) (j+k)++ won't be 5++ giving 6, because in expression (j+k) = (j+k) +1 there is an invalid lvalue. Expressions can't be on the left side of the assignment operator.

Finally consider the following program:

```
#include <stdio.h>
int main()
{
        int i = 10;
        printf("%d", ++(-i));
        return 0;
}
```

Options are:

a) 11 b) 10 c) -9 d) None

Ans. D) none.

Like in math, parenthesis are evaluated first. This –i gives a constant value -10 and we know ++ or -- cannot be operated on constants.

We may then say remove the parenthesis. Then ++-i will still be an error because there is nothing like ++-. And if we imagine that this should evaluate as –i = -i + 1, then again we are using an expression as lvalue which is not allowed.