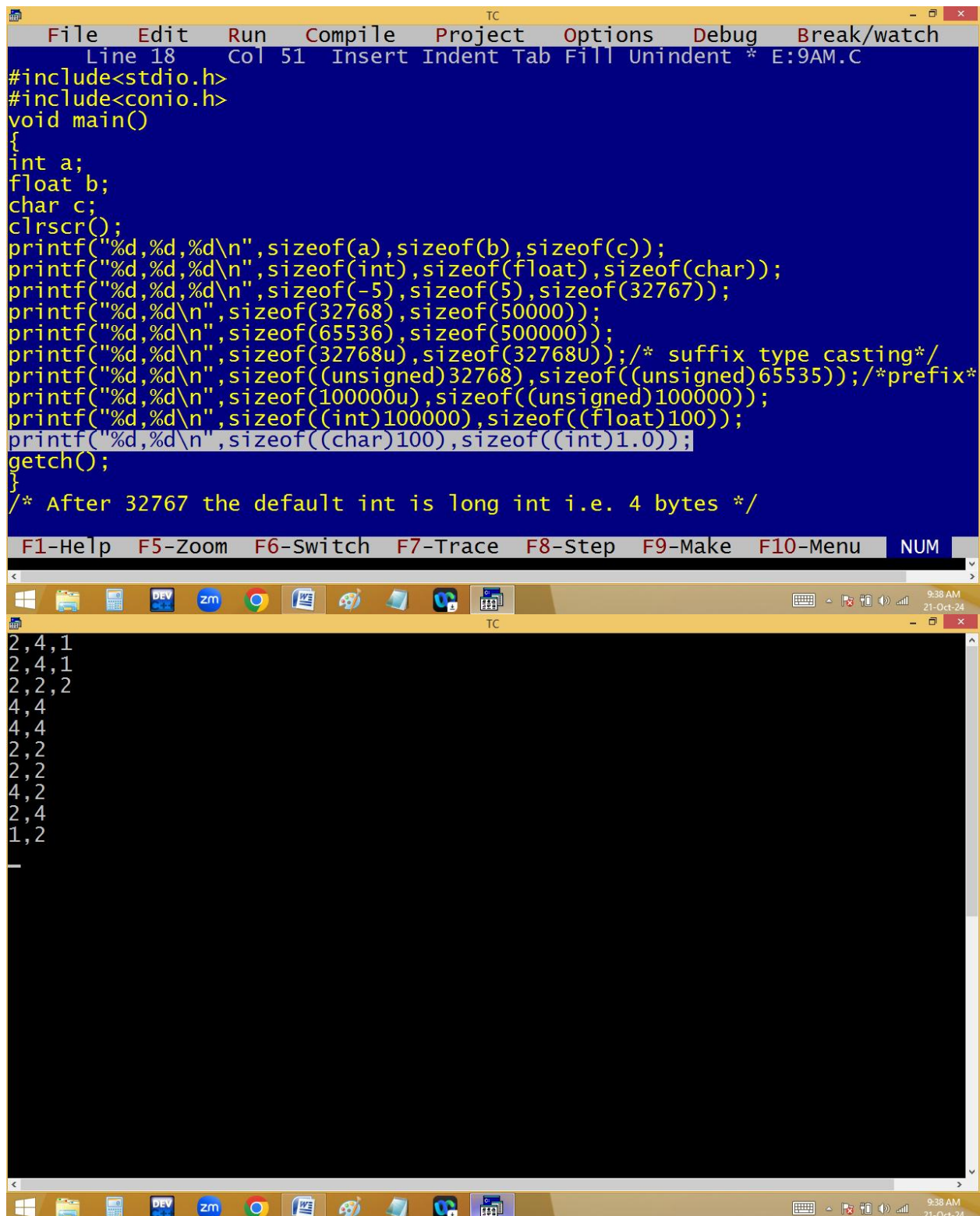


Sizeof() operator: It return the no of bytes required for a variable / value / data type.



```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 18 Col 51 Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a;
float b;
char c;
clrscr();
printf("%d,%d,%d\n",sizeof(a),sizeof(b),sizeof(c));
printf("%d,%d,%d\n",sizeof(int),sizeof(float),sizeof(char));
printf("%d,%d,%d\n",sizeof(-5),sizeof(5),sizeof(32767));
printf("%d,%d\n",sizeof(32768),sizeof(50000));
printf("%d,%d\n",sizeof(65536),sizeof(500000));
printf("%d,%d\n",sizeof(32768u),sizeof(32768U));/* suffix type casting*/
printf("%d,%d\n",sizeof((unsigned)32768),sizeof((unsigned)65535));/*prefix*/
printf("%d,%d\n",sizeof(100000u),sizeof((unsigned)100000));
printf("%d,%d\n",sizeof((int)100000),sizeof((float)100));
printf("%d,%d\n",sizeof((char)100),sizeof((int)1.0));
getch();
}
/* After 32767 the default int is long int i.e. 4 bytes */

F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu NUM
2,4,1
2,4,1
2,2,2
4,4
4,4
4,4
2,2
2,2
4,2
2,4
1,2
```

```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 14 Col 1 Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
void main()
{
float a=1.1;
double b=1.1;
char c;
clrscr();
printf("%d,%d\n",sizeof(a),sizeof(b));
printf("%d,%d\n",sizeof(1.1),sizeof(long double));
printf("%d,%d\n",sizeof(1.1f),sizeof((float)1.1));
printf("%d,%d\n",sizeof(1.1l),sizeof(1.1L));
printf("%d,%d\n",sizeof((int)1.1),sizeof((unsigned)1.1));
getch();
}
/* In c the default float is double i.e. 8 bytes */

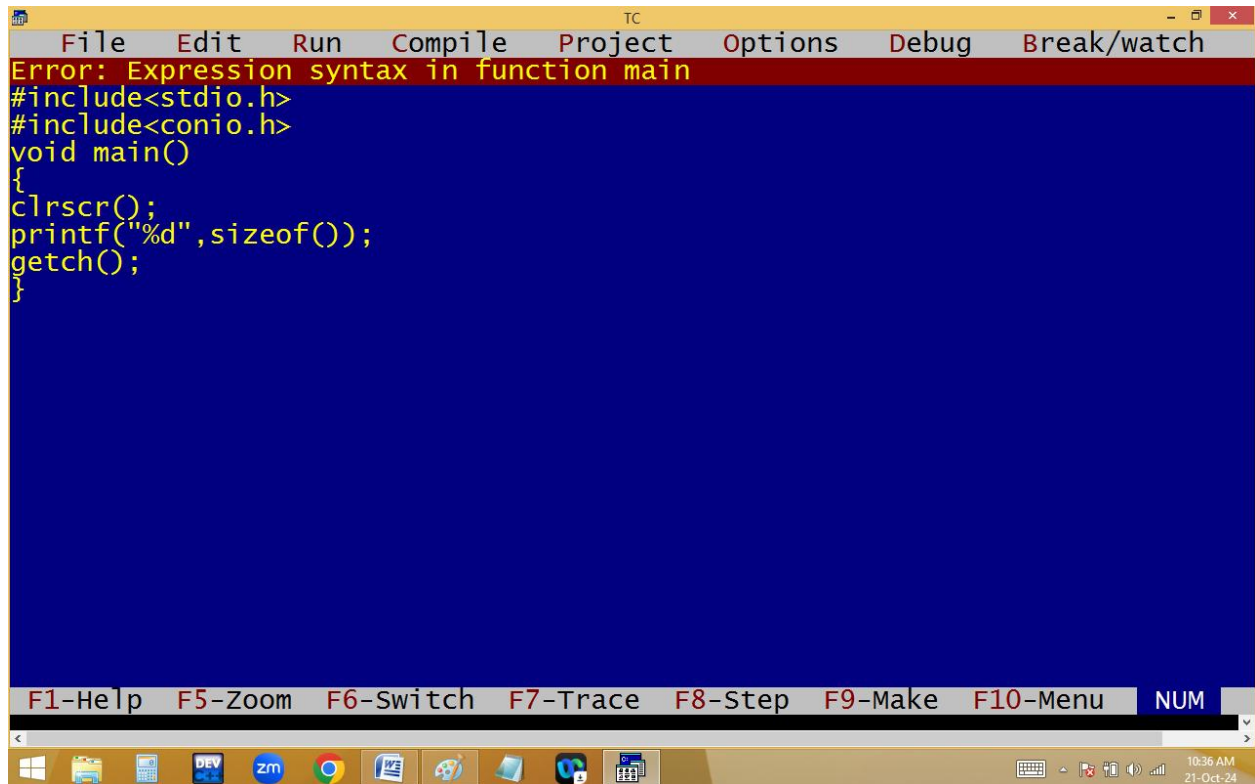
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu NUM
4,8
8,10
4,4
10,10
2,2
-
```

```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 1 Col 1 Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
void main()
{
char a[10]="kishore", b[]="kishore";
clrscr();
printf("%d,%d\n",sizeof(a),sizeof(b));
printf("%d,%d\n",sizeof("kishore"),sizeof("kishore\0"));
printf("%d,%d\n",printf("kishore"),printf("kishore\0"));
printf("%d,%d\n",sizeof(printf("kishore")),sizeof(printf("kishore+1")));
printf("%d,%d\n",sizeof(sizeof("kishore")),sizeof("kishore+1"));
printf("kishore address is %u\n","kishore");
printf("%d,%d\n",sizeof("kishore")+1,sizeof("kishore"+1));
printf("%d\n",sizeof("kishore")>printf("kishore"));
printf("%d, %d\n",sizeof(1,1.2),sizeof(1.2,1));
printf("%d, %d\n",sizeof(1,1.2),sizeof(1.2,1));
printf("%d, %d\n",sizeof(1+1.2),sizeof(1.2+1));
getch();
}
/* Inside sizeof expressions not considered */

F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu NUM
10,8
8,9
kishorekishore7,7
2,2
2,10
kishore address is 515
9,2
kishore1
8, 2
8, 2
8, 8
```

```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 12 Col 33 Insert Indent Tab Fill Unindent * E:9AM.C
#include<stdio.h>
#include<conio.h>
void main()
{
int a=999;
clrscr();
printf("%d, %d\n",sizeof("1.23"),sizeof(sizeof(1.2)));
printf("%d\n",sizeof(++a));
printf("%d\n",a);
printf("%d\n",sizeof(a=555));
printf("%d\n",a);
printf("%d",sizeof("sizeof()"));_
getch();
}
/* Inside sizeof expressions not considered */

5, 2
2
999
2
999
9
_
```



The screenshot shows the Turbo C++ IDE with a menu bar (File, Edit, Run, Compile, Project, Options, Debug, Break/watch) and a toolbar with function key shortcuts (F1-Help, F5-Zoom, F6-Switch, F7-Trace, F8-Step, F9-Make, F10-Menu, NUM). The main window has a blue background and displays the following C code:

```
Error: Expression syntax in function main
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("%d",sizeof());
getch();
}
```

The error message "Error: Expression syntax in function main" is displayed at the top of the code area. The Windows taskbar at the bottom shows the time as 10:36 AM on 21-Oct-24.

BITWISE OPERATORS

Bitwise operator's works on bits.

Turbo-c is a 16 bit compiler. Due to this bitwise operations are limited to 16 bits only [2^0 to 2^{15}].

Bitwise operators operate **integer** type values only.

We have to calculate only the **on** bits [**1**].

When the first bit[**Sign bit**] is **1** then the number is **Negative** and it is **0** then the number is **positive**.

They are very much used in system software development.

Note: Bitwise operator is low level feature.

C-Language supports following bitwise operators.

& -Bitwise and

| - Bitwise or

^ - XOR ==> Exclusive OR

~ - Compliment operator

<< - Left shift operator

>> - Right shift operator

& - Bitwise and: In this both bits are 1's then result bit is 1. Otherwise result bit is 0.

Eg: **25 & 15 = 9**

25 = 0000 0000 0001 1001
15 = 0000 0000 0000 1111

$$\begin{array}{r} 2 \overline{) 25} \\ 2 \overline{) 12} - 1 \\ 2 \overline{) 6} - 0 \\ 2 \overline{) 3} - 0 \\ 2 \overline{) 1} - 1 \end{array}$$

$$\begin{array}{r} 2 \overline{) 15} \\ 2 \overline{) 7-1} \\ 2 \overline{) 3-1} \\ \quad 1-1 \end{array}$$

$25 \& 15 = 9$

25 = 0000 0000 0001 1001

15 = 0000 0000 0000 1111

&

0000 0000 0000 1001

$\downarrow \quad \downarrow$

$2^3 + 2^0$

$\downarrow \quad \downarrow$

8 + 1 = 9

| - Bitwise or: In this both bits are 0's then result bit is 0. Otherwise result bit is 1.

Eg: 25 | 15 = 31

$$25 \mid 15 = 31$$

$$25 = 0000 \ 0000 \ 0001 \ 1001$$

$$15 = 0000 \ 0000 \ 0000 \ 1111$$

$$0000 \ 0000 \ 0001 \ 1111$$

$2^4 + 2^3 + 2^2 + 2^1 + 2^0$
 $16 + 8 + 4 + 2 + 1 = 31$

^ - XOR [Exclusive OR]: In this both bits are same then result bit is 0. Otherwise result bit is 1.

Eg: $25 \wedge 15 = 22$

$$25 \wedge 15 = 22$$

$$25 = 0000 \ 0000 \ 0001 \ 1001$$

$$15 = 0000 \ 0000 \ 0000 \ 1111$$

\wedge

$$0000 \ 0000 \ 0001 \ 0110$$

$$2^4 + 2^2 + 2^1$$

$$16 + 4 + 2 = 22$$

~ - Compliment operator: In compliment operation the bits are complimented. i.e.

1's become 0's and 0's become 1's. Due to this +Ve no becomes -Ve and -Ve no becomes +Ve.

eg: $\sim 25 \rightarrow -26$

$$\begin{array}{r}
 25 = \begin{array}{|c|c|c|c|} \hline 0000 & 0000 & 0001 & 1001 \\ \hline 1111 & 1111 & 1110 & 0110 \\ \hline \end{array} \\
 \qquad \qquad \qquad \begin{array}{c} \diagup \quad \diagdown \quad \diagup \quad \diagdown \\ -128 + 64 + 32 + 4 + 2 = -26 \\ -128 + 102 = -26 \end{array}
 \end{array}$$

$$25 = 0000\ 0000\ 0001\ 1001$$

$$\sim = 1111\ 1111\ 1110\ 0110$$

$$\begin{array}{c}
 \diagdown \quad \diagup \quad \diagdown \\
 5 \quad 2 \quad 1
 \end{array}$$

$$2+4+32+64+128+256+512+1024+2048+4096+8192+16384-32768=-26$$

$$\sim -25 = 0000\ 0000\ 0001\ 1001$$

$$1's\ \sim = 1111\ 1111\ 1110\ 0110$$

$$2's\ \sim = \underline{0000\ 0000\ 0000\ 0001}$$

$$1111\ 1111\ 1110\ 0111$$

$$\begin{array}{c}
 \diagup \quad \diagdown \\
 2^4 \quad 2^3 \\
 16 + 8 = 24
 \end{array}$$

$$\begin{array}{cccc}
 1 & 0 & 0 & 01 \\
 0 & 1 & 0 & 1 \\
 \hline 1 & 1 & 0 & 10
 \end{array}$$

Note: When starting bit is 1 given no is -Ve.

Eg: $\sim -25 \rightarrow +24$

$\sim -25 = +24$

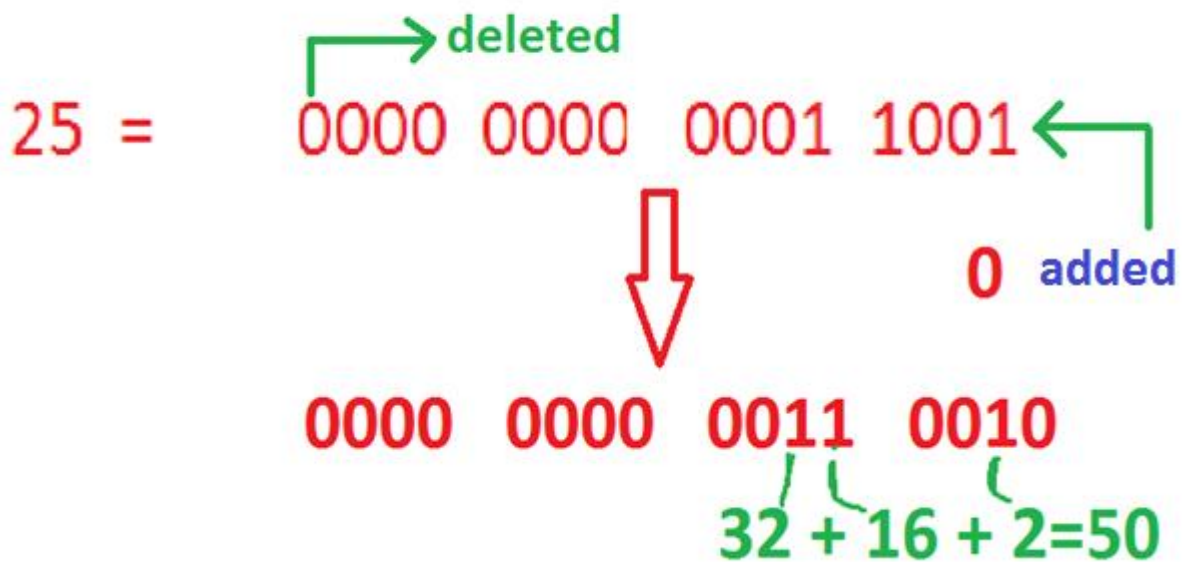
25 =	0000 0000	0001 1001	
	1111 1111	1110 0110	<== 1's compliment
		+1	<== 2's Compliment
<hr/>			
	1111 1111	1110 0111	
	0000 0000	0001 1000	
		↓ ↓	
		16+8=24	

<< - left shift operator:

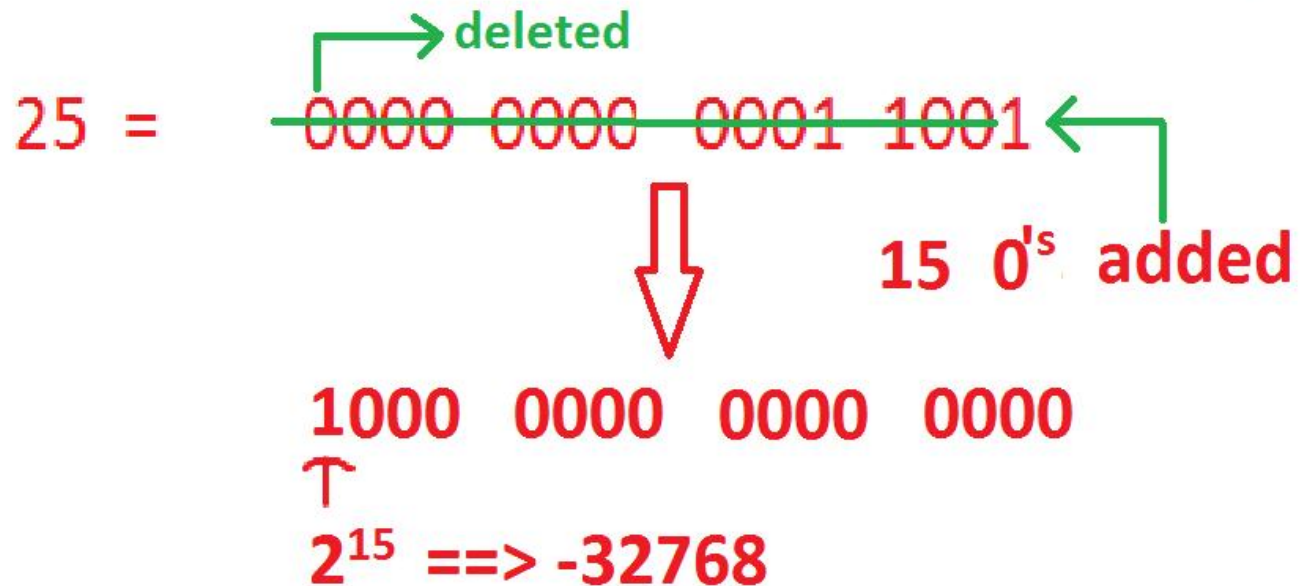
In left shift operation, the specified no of bits are deleted from left side and the same no of **zeros** added on right side. In left shift operation, most probably the value is multiplied with 2 that no of times.

Eg: $25 \ll 1 = 50$, $25 \ll 2 = 100$, $25 \ll 15 = -32768$,
 $25 \ll 16 = 0$

eg: $25 \ll 1 = 50$



eg: $25 \ll 15 = -32768$



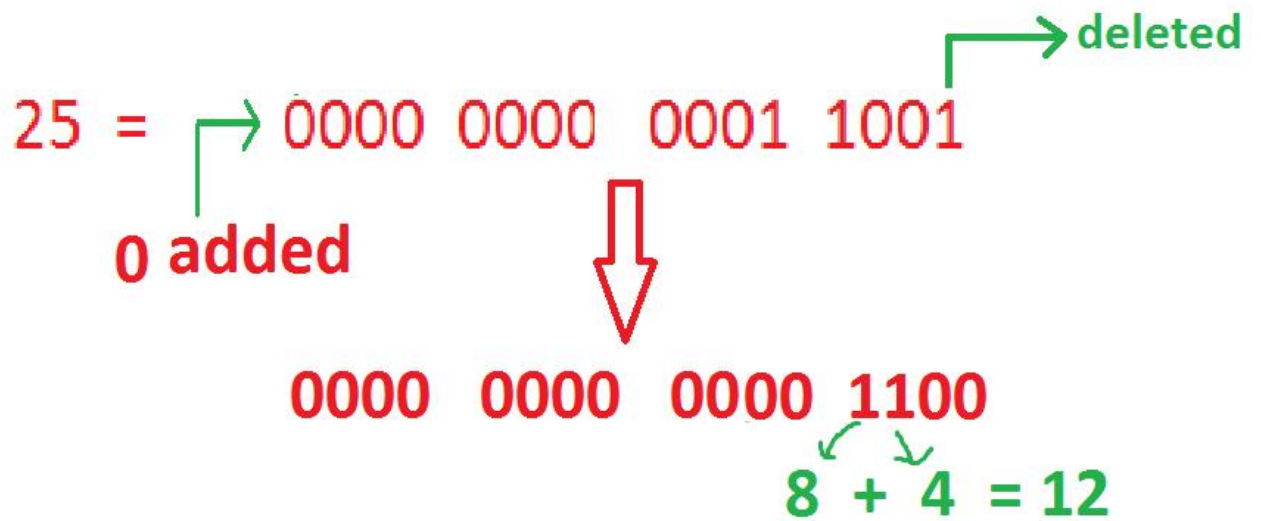
Note: When starting bit 1 no is negative.

>> - Right shift operator:

In right shift operation, the bits are moved to right side i.e. the specified no.of bits are deleted from right side and same no.of **zero's** are added left side. Due to this always the number is divided with 2 that no of times.

Eg: $25 \gg 1 = 12$, $25 \gg 2 = 6$, $25 \gg 3 = 3$, $25 \gg 4 = 1$, $25 \gg 5 = 0$

eg: $25 \gg 1 = 12$



eg: $25 \gg 5 = 0$

