Bitwise operation in c

-> In C, bit wise operators are used to perform operations directly on the binar representations of numbers.

-> These operators work by manipulating individual bits (0s and 1s) in a number.

-> The following 6 operators are bit wise operators (also known as bit operators as they work at the bit-level).

-> They are used to perform bitwise operations in c.

```
# Include<stdio.h>
int main()
{
unsigned int a = 60;
unsigned int b = 13;
int result = 0;
result = a & b;
Printf ("a & b = %d\n", result);
result = a|b;
Printf ("a | b = %d\n", result);
result = a ^ b;
Printf ("a ^ b = %d\n", result);
printf ("~a = %u (as unsigned int)\n", ~a);
result = a << 2;
Printf ("a << 2 = %d\n", result);
result = a >> 2;
Printf ("a >> 2 = %d\n", result);
return 0;
}
```

→ Bitwise XOR operator (^) in C performs a binary exclusive OR operation on corresponding bits of its operands.

→ The result is 1 if the bits are different, and 0 if they are the same.

→ Here is an example demonstrating the bitwise XOR operator in C:

Code:

```
#include <stdio.h>
int main()
{
int a = 10
int b = 6;
int result = a ^ b;
printf ("a = %d (Binary: %d)\n", a, 1010);
printf ("b = %d (Binary: %d)\n", b, 0110);
printf ("Result of a ^ b = %d\n", result);
return 0;
}
```

AND

Bitwise AND operator

→ The output of bitwise AND is, 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

→ In C programming, the bitwise AND operator is denoted by $\boxed{\&}$.

→ Let us suppose the bitwise AND operation of two integers 12 and 25

Code:

```
# include <stdio.h>
 int main ()
 {
  int a = 12, b = 25;
  printf(" output = %d", a & b);
  return 0;
}
```

→ Bitwise OR operation

→ The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C programming, bitwise OR operation is denoted by $\boxed{|}$.

Ex: OR

```
#include <stdio.h>
int main ()
{
    int a=12,  b=25;
   Printf ("output = %d", a|b);
   return 0;
}
```

output = 29

→ NOT

→ The bitwise NOT operator in C, denoted by the tilde symbol ~, Performs a one's complement operation on a number

→ This means it inverts each bit of its operand: 0s become 1s, and 1s become 0s.

→ The result is often a negative number due to two's complement representation used for signed integers

Code: NOT

```
# include <stdio.h>
int main ()
{
```

```
int num = 5;
int result = ~num;
Printf("original number: %d\n", num);
Printf("Bitwise NOT result: %d\n", result);
return 0;

}
```

Q Relation operators in c language

→ Relational operators in c are used to compare two values and determine the relationship between them.

→ They return a boolean result: 1 (true) if the condition is met, and 0 (false) otherwise.

→ These operators are fundamental for decision-making and controlling program flow in C.

* 4<2

→ The expression 4<2 uses the "less than" relational operator (<)

→ 4

code 4 < 2

```
# include <stdio.h>
int main ()
{
    int result;
   result = (4 < 2);
printf ("The result of (4< 2) is: %d\n",
        result);
  if (4 < 2)
  {
     Printf (" This message will not be
              Printed because 4 is not less
              than 2.\n");
  } else {
     Printf (" This message will be Printed
              be cause 4 is notless than 2\n
  }
  return 0;
}
```

* 4 > 2

→ The expression 4 > 2 uses the "greater than relational operator (>)

→ The expression evaluates to a truth value (true or false)

code

```
# include<stdio.h>
int main ()
{
  int result = (4 > 2);
  printf (" The result of (4 > 2) is: %d\n",
          result);
  if (4 > 2)
  {
   printf (" 4 is indeed greater than 2\n");
  } else {
   printf ("4is not greater than 2.\n");
  }
  return 0;
}
```

* 4 <= 2

-> In C, The expression 4<= 2 uses the less than or equal to (<=) relational operator to compare the two numbers.

code 4< = 2

```
# include <stdio.h>

 int main ()

 {

   int result = (4 <= 2);

   printf ("%. d\n", result);

   return 0;

 }
```

* 4 >= 2

-> The C. code 4>= 2 uses the greater than or equal to relational operator to compare the two values.

-> The evaluates to 1(true) because the number 4 is greater than or equal to 2.

Code 4>=2

```
# include <stdio.h>
int main ()
{
int a = 4;
int b = 2;
if (a >= b)
{
  Printf("%d is greater than or equal to
        %d\n", a, b);
} else {
  Printf("%d is less than %d\n", a, b);
}
 return 0;
}
```

* 4!= 2

→ compare two values to check if they are not equal

→ when to use this operator when you need to determine if two values are different.

code 4!=2

```
# include <stdio.h>
int main()
{
 int a = 4, b = 2;
 if (a != b)
 {
   Printf ("a is not equal to b\n");
 }
 else {
   Printf ("a is equal to b\n");
 }
   return 0;
}
```

* 4 == 2

→ compares two values to check if they are equal

→ This shows the use of the == operator to compare if two values are the same.

code 4==2

```
# include <stdio.h>
int main ()
{
  int a=4, b= 2;
 if (a == b)
 {
    Printf ("a is equal to b\n");
 }
  else {
      Printf ("a is not equal to b\n");
 }
   return 0;
 }
```