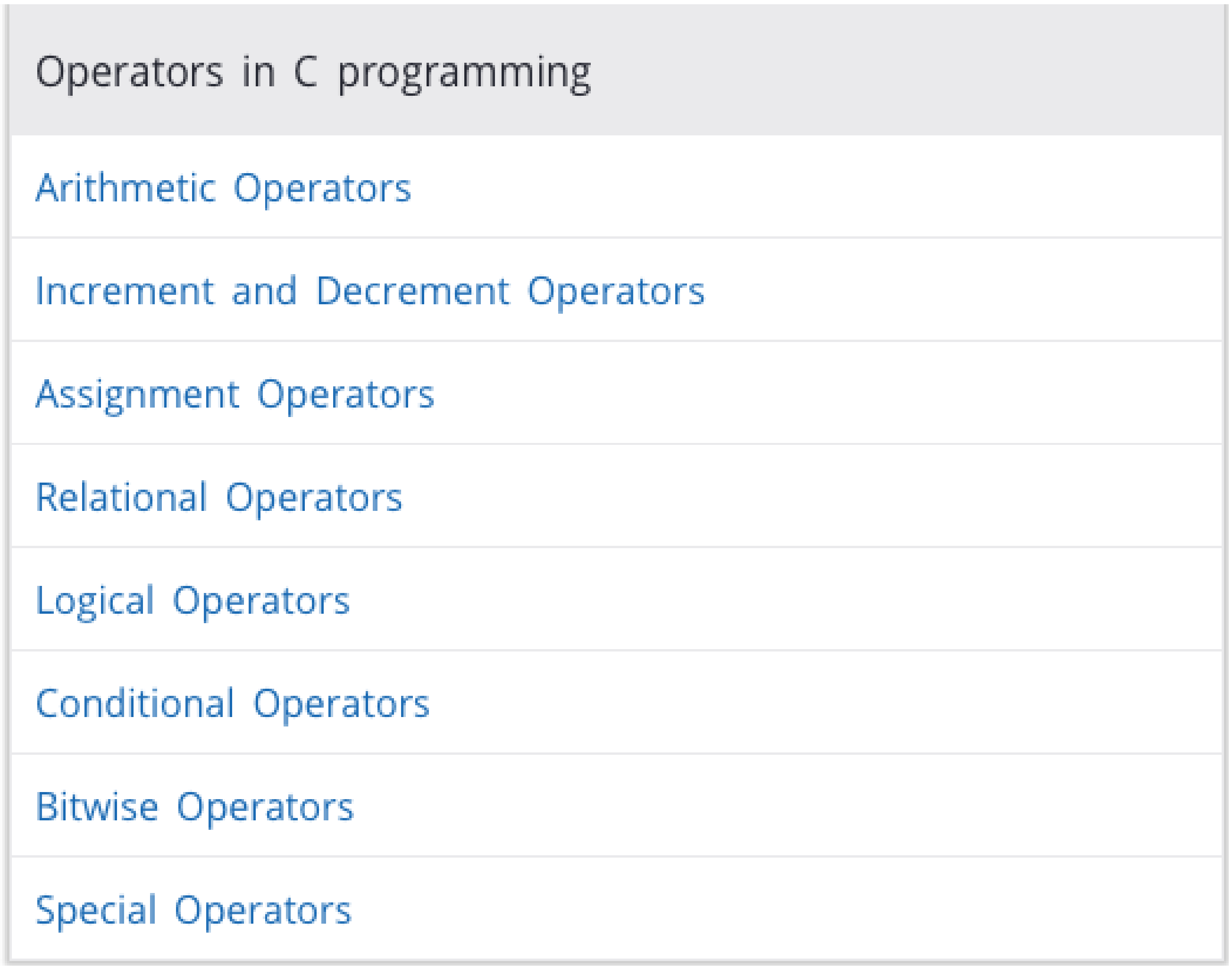
C Programming

Kasun De Zoysa

Operators

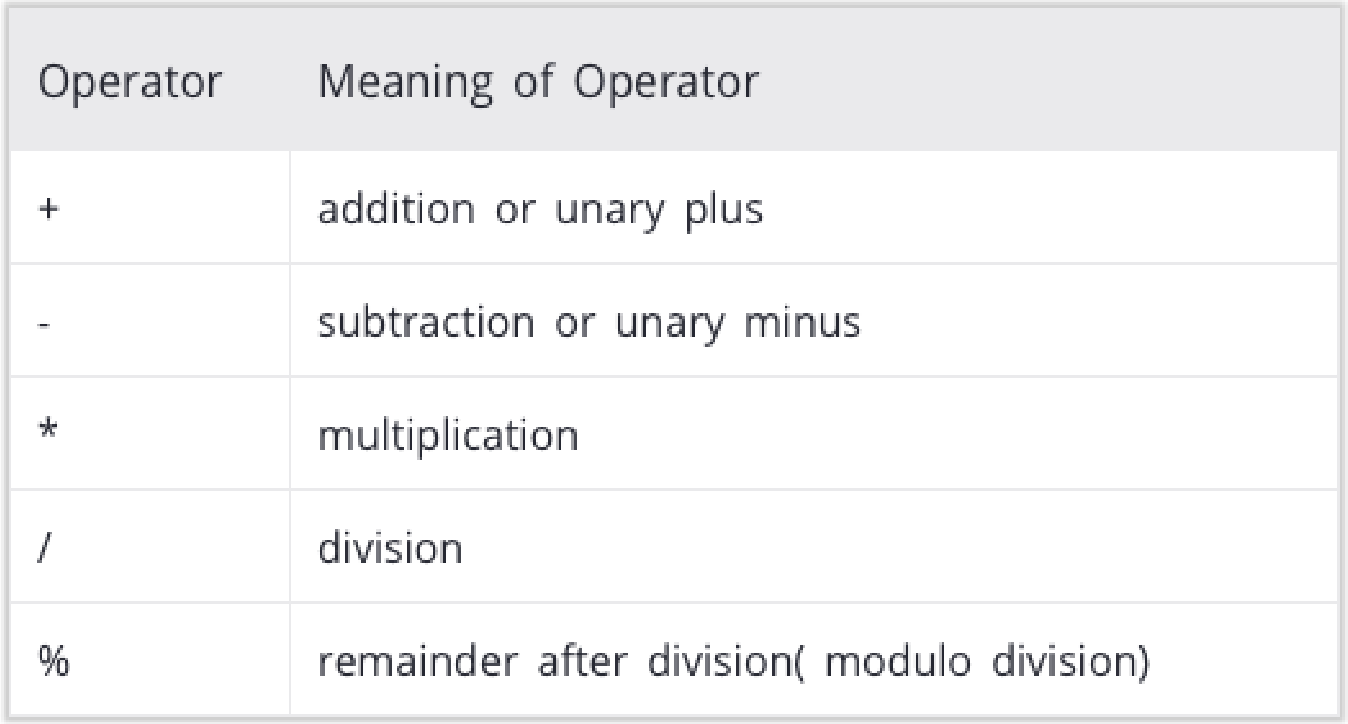
C programming has various operators to perform tasks including arithmetic, conditional and bitwise operations.



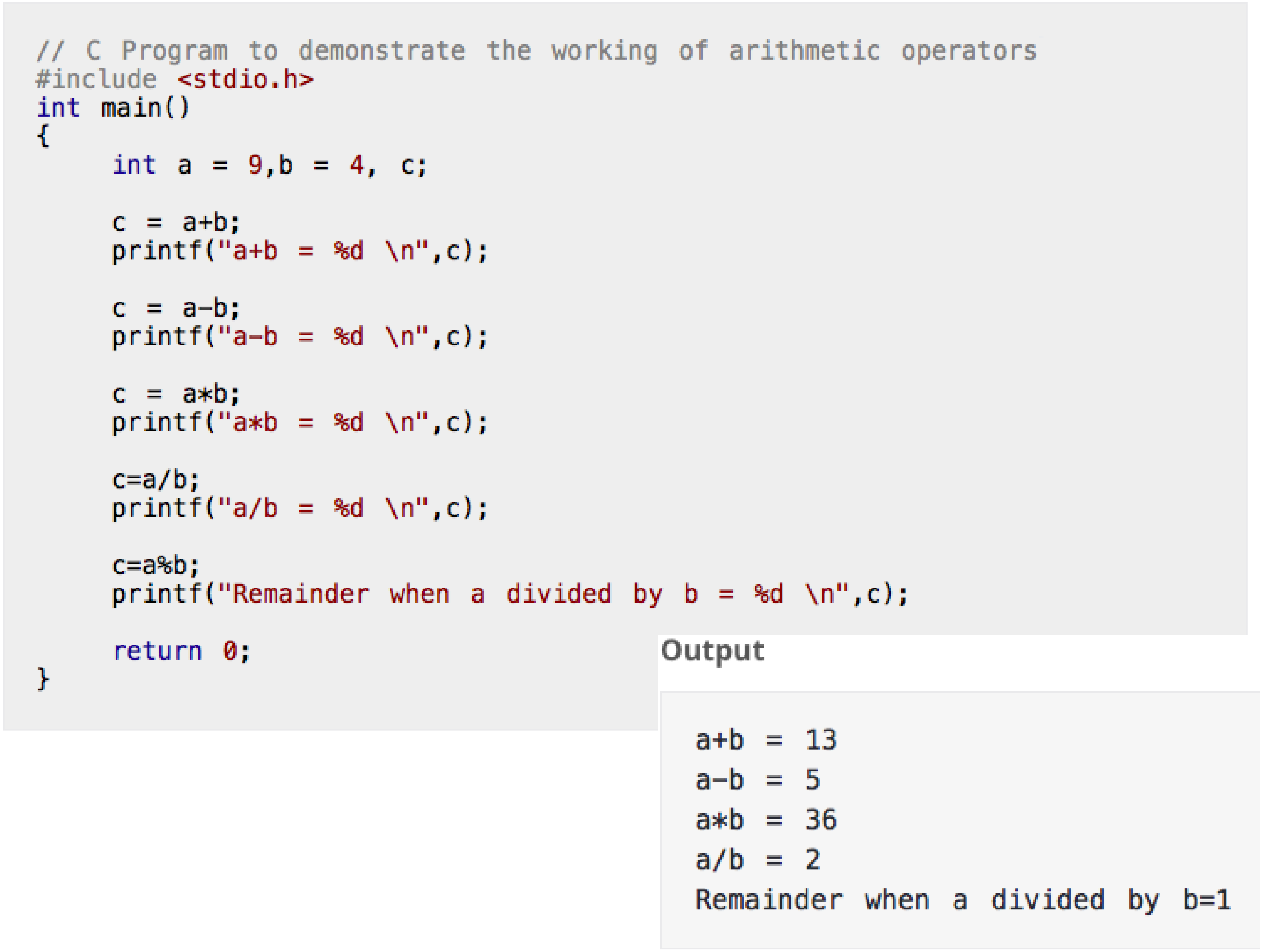
Operators in C programming
Arithmetic Operators
Increment and Decrement Operators
Assignment Operators
Relational Operators
Logical Operators
Conditional Operators
Bitwise Operators
Special Operators

An arithmetic operator

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).



Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)



```c
// C Program to demonstrate the working of arithmetic operators
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Output

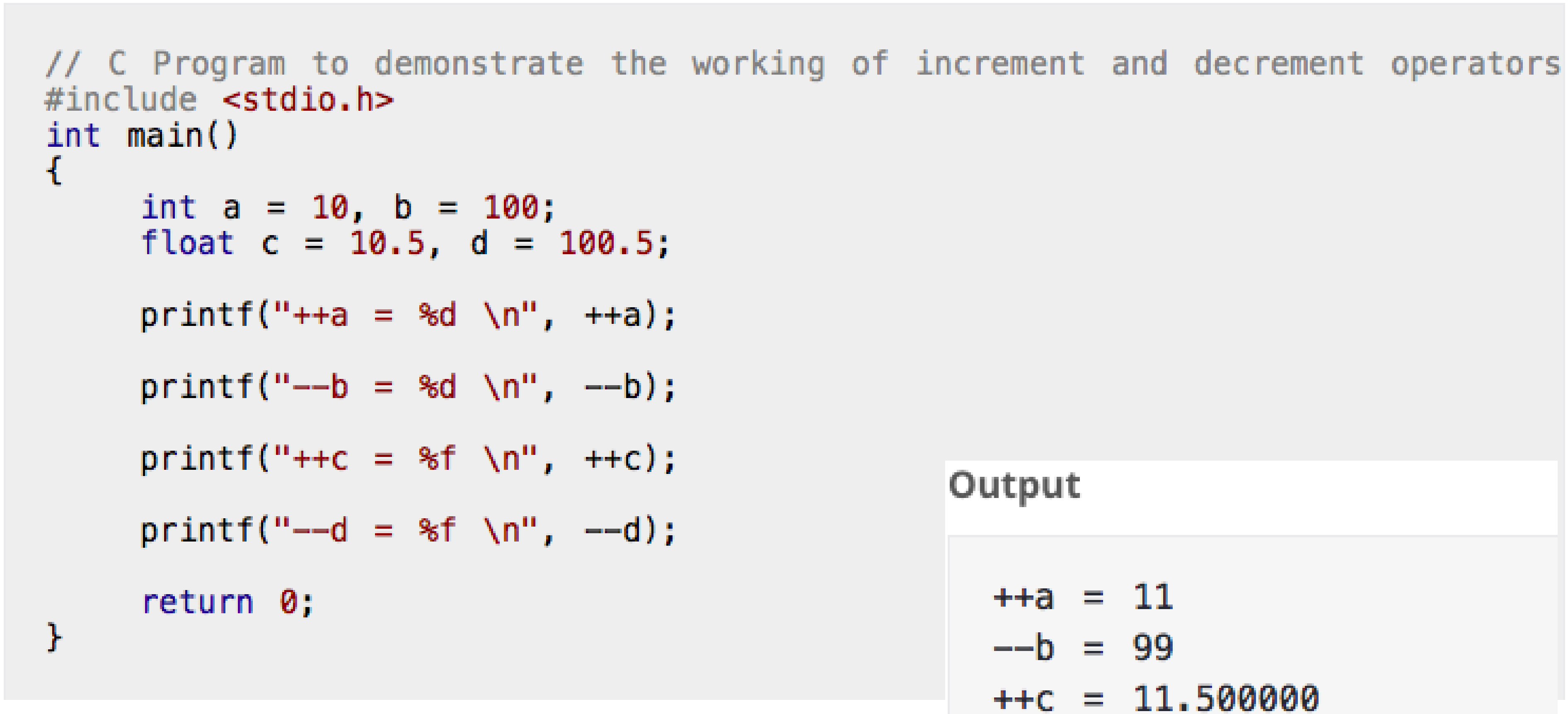
```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Increment and decrement operators

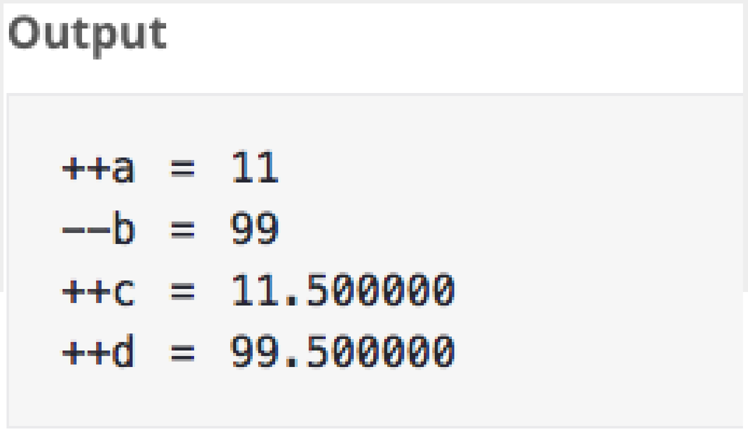
C programming has two operators **increment** ++ and **decrement** -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1.

These two operators are unary operators, meaning they only operate on a single operand.



```c
// C Program to demonstrate the working of increment and decrement operators
#include <stdio.h>
int main()
{
     int a = 10, b = 100;
     float c = 10.5, d = 100.5;

     printf("++a = %d \n", ++a);

     printf("--b = %d \n", --b);

     printf("++c = %f \n", ++c);

     printf("--d = %f \n", --d);

     return 0;
}
```



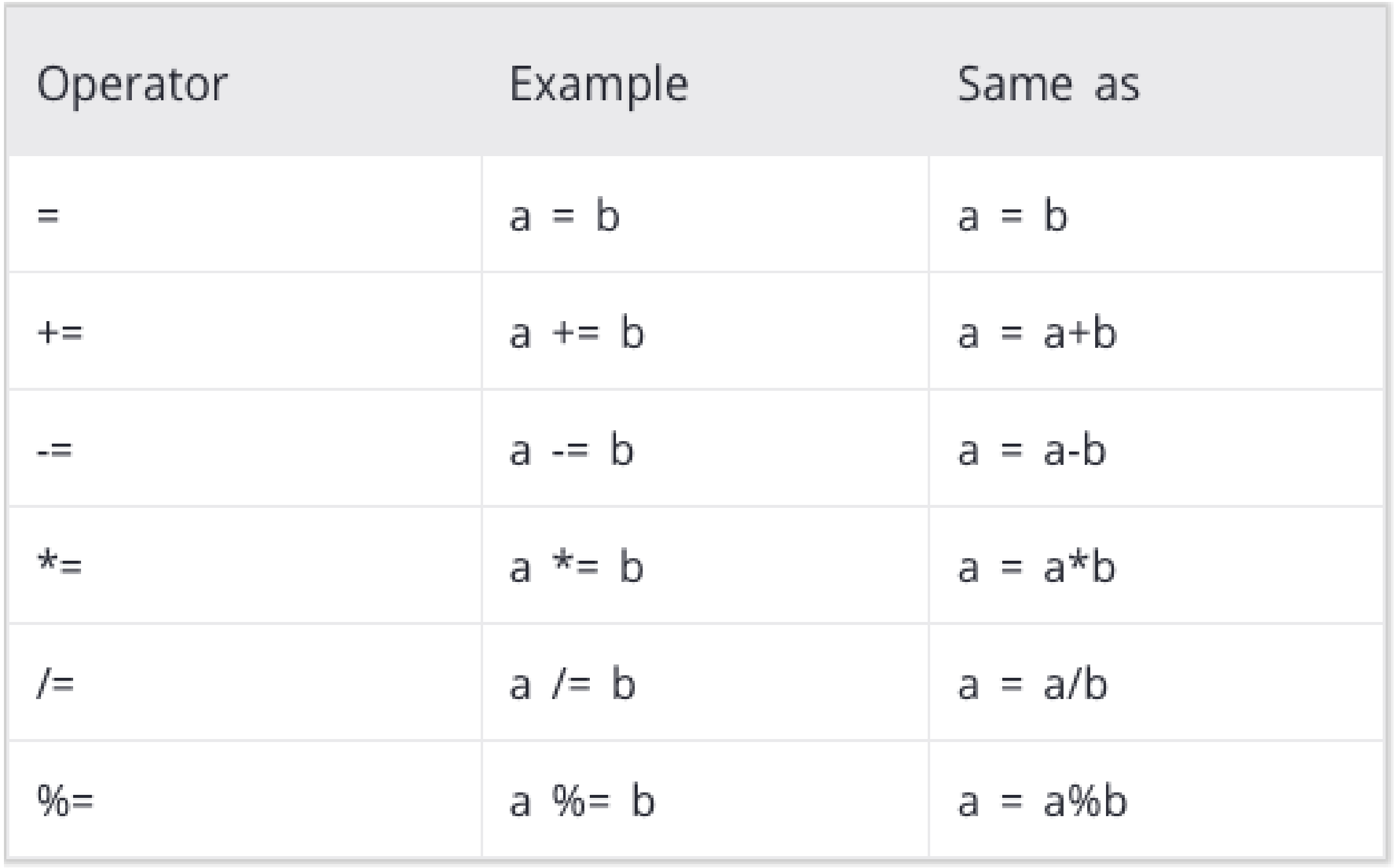
Output

```
++a = 11
--b = 99
++c = 11.500000
++d = 99.500000
```

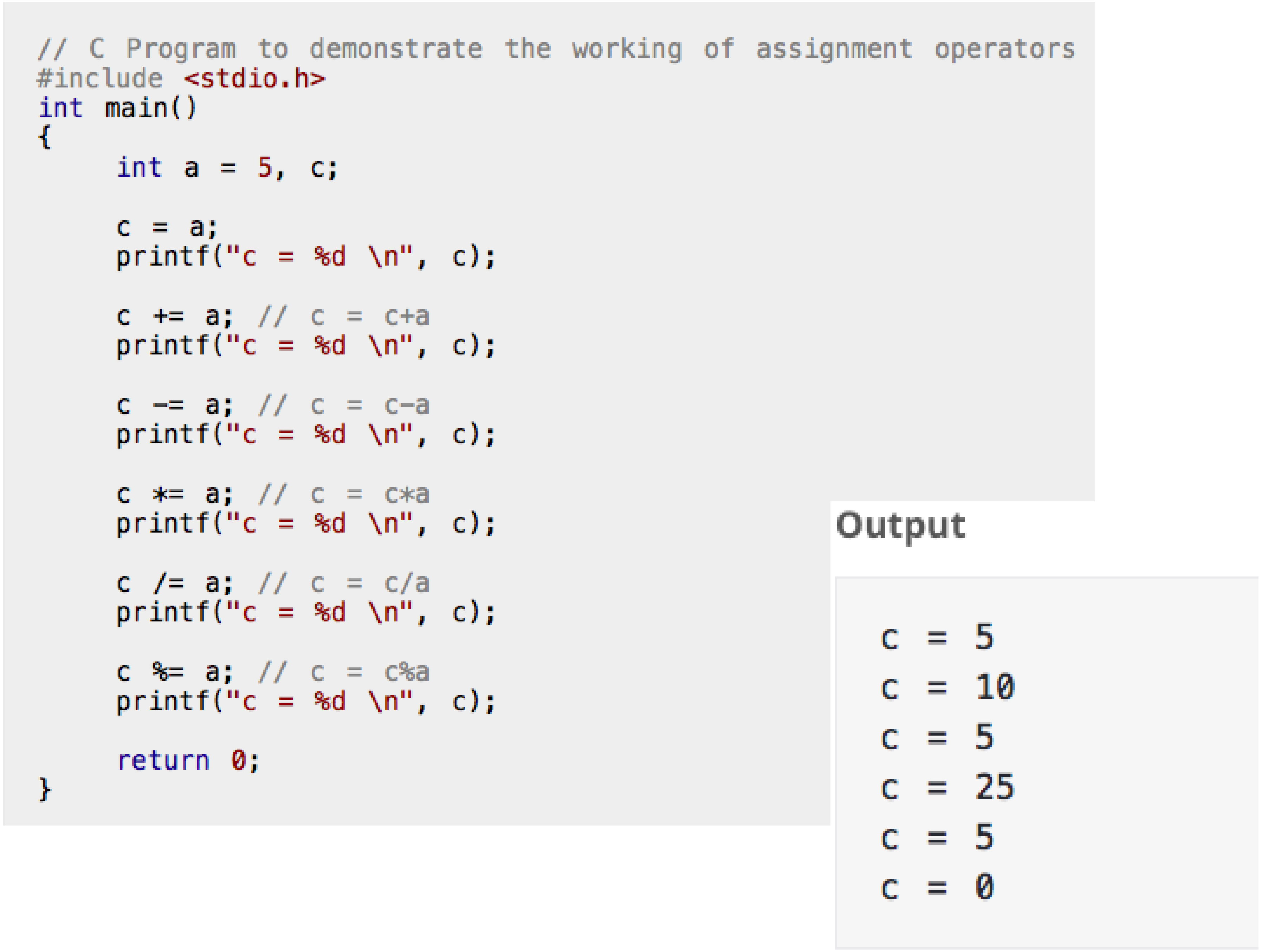
The operators ++ and -- are used as prefix. These two operators can also be used as postfix like a++ and a--.

Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =



Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b



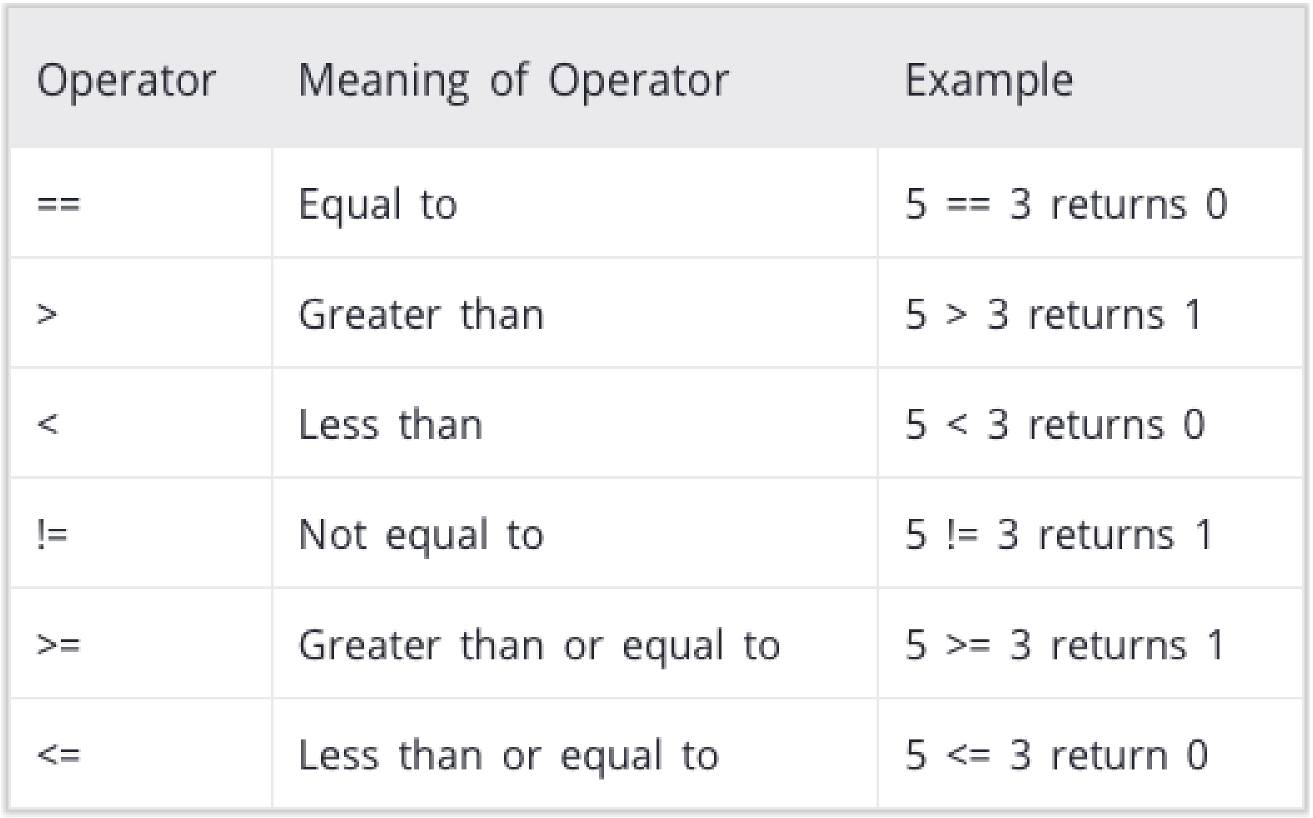
```c
// C Program to demonstrate the working of assignment operators
#include <stdio.h>
int main()
{
     int a = 5, c;

     c = a;
     printf("c = %d \n", c);

     c += a; // c = c+a
     printf("c = %d \n", c);

     c -= a; // c = c-a
     printf("c = %d \n", c);

     c *= a; // c = c*a
     printf("c = %d \n", c);

     c /= a; // c = c/a
     printf("c = %d \n", c);

     c %= a; // c = c%a
     printf("c = %d \n", c);

     return 0;
}
```

Output

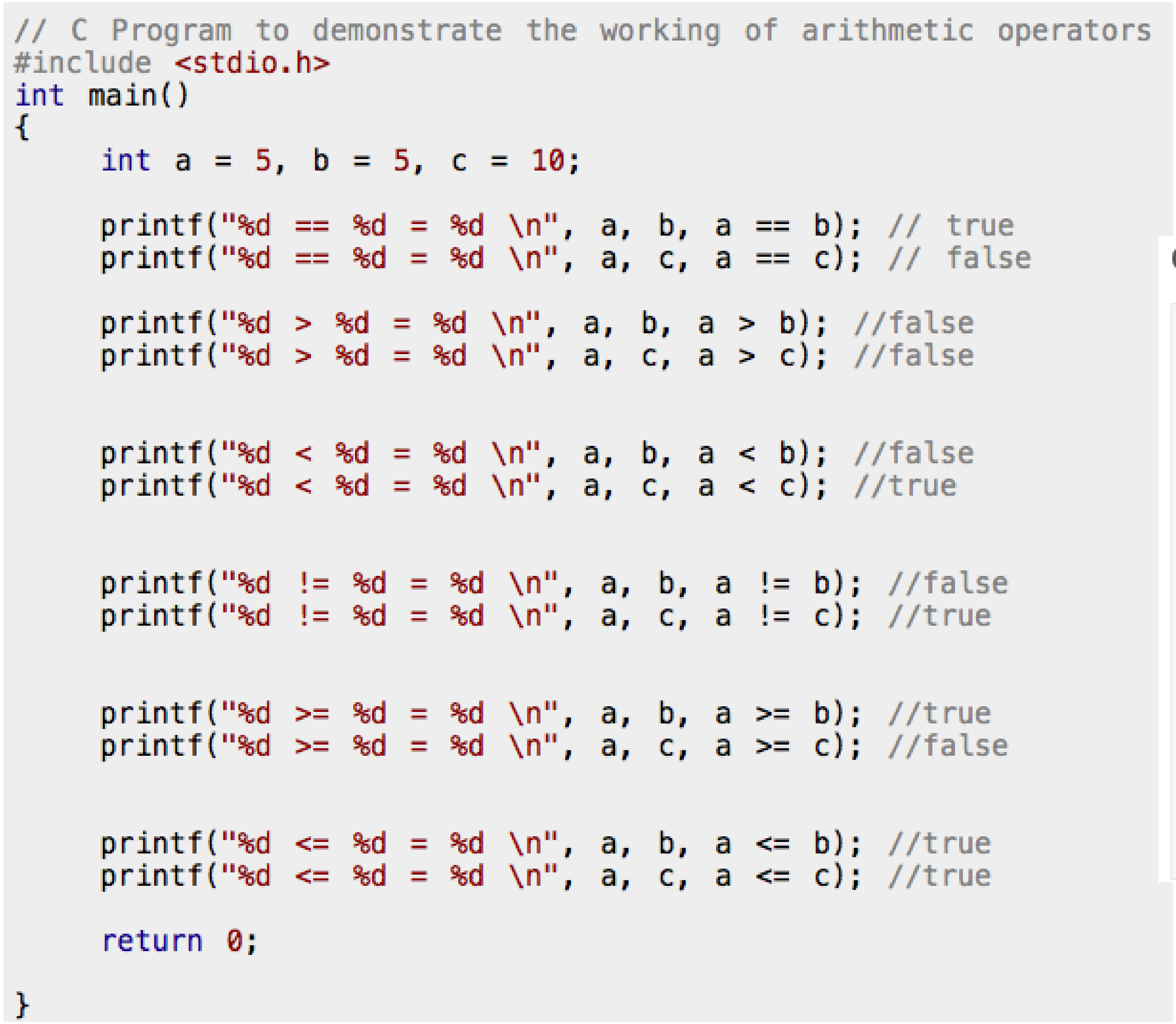
```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

Relational Operators

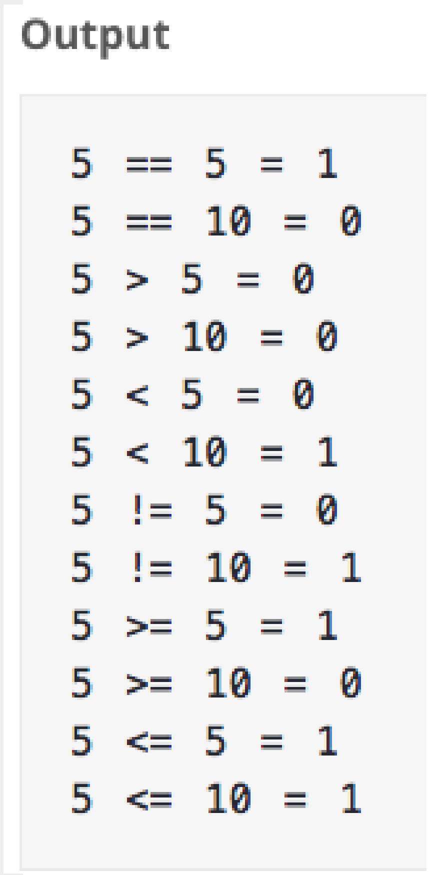
A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0. Relational operators are used in decision making and loops.



Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

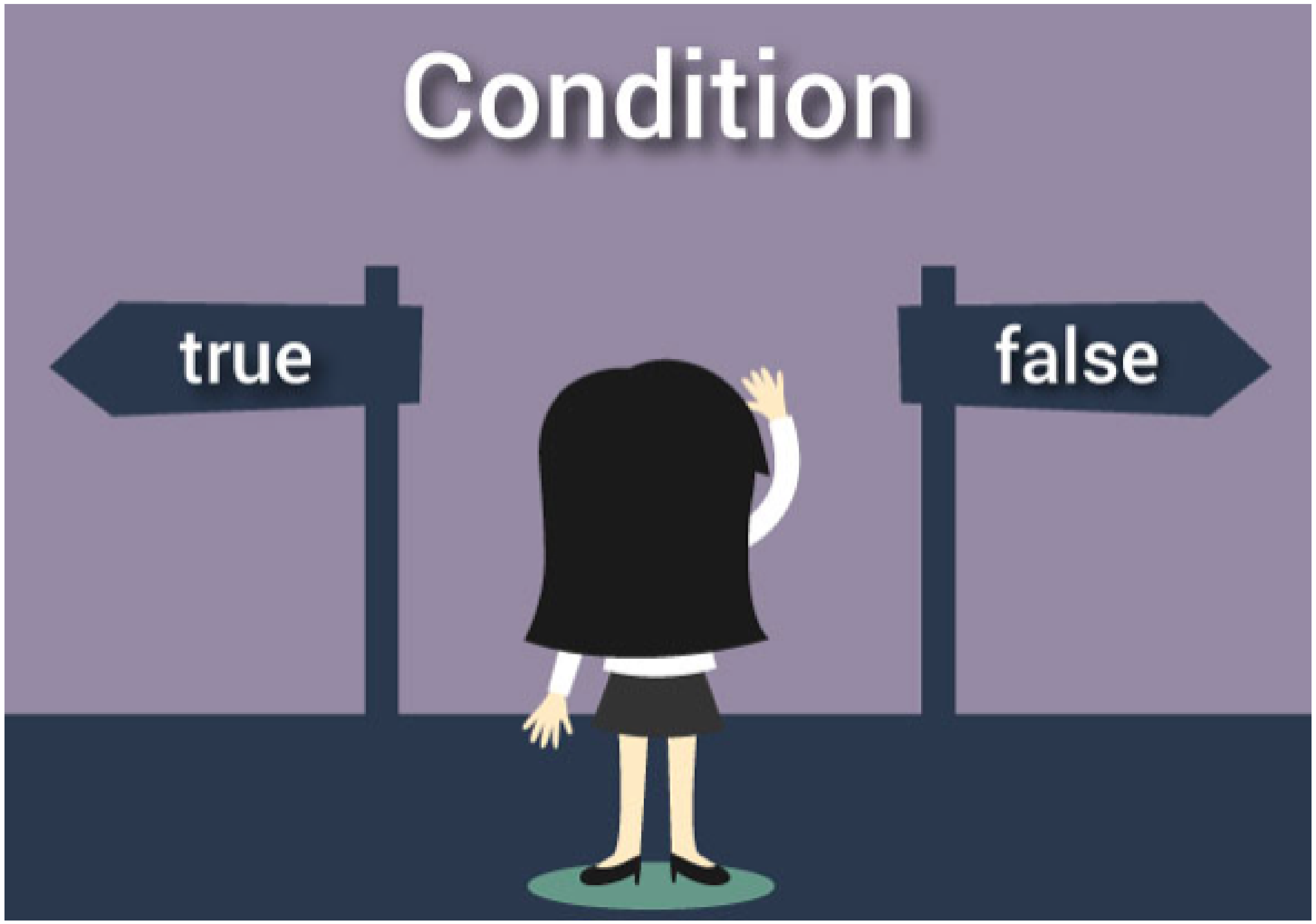


```c
// C Program to demonstrate the working of arithmetic operators
#include <stdio.h>
int main()
{
     int a = 5, b = 5, c = 10;

     printf("%d == %d = %d \n", a, b, a == b); // true
     printf("%d == %d = %d \n", a, c, a == c); // false

     printf("%d > %d = %d \n", a, b, a > b); //false
     printf("%d > %d = %d \n", a, c, a > c); //false


     printf("%d < %d = %d \n", a, b, a < b); //false
     printf("%d < %d = %d \n", a, c, a < c); //true


     printf("%d != %d = %d \n", a, b, a != b); //false
     printf("%d != %d = %d \n", a, c, a != c); //true


     printf("%d >= %d = %d \n", a, b, a >= b); //true
     printf("%d >= %d = %d \n", a, c, a >= c); //false


     printf("%d <= %d = %d \n", a, b, a <= b); //true
     printf("%d <= %d = %d \n", a, c, a <= c); //true

     return 0;

}
```



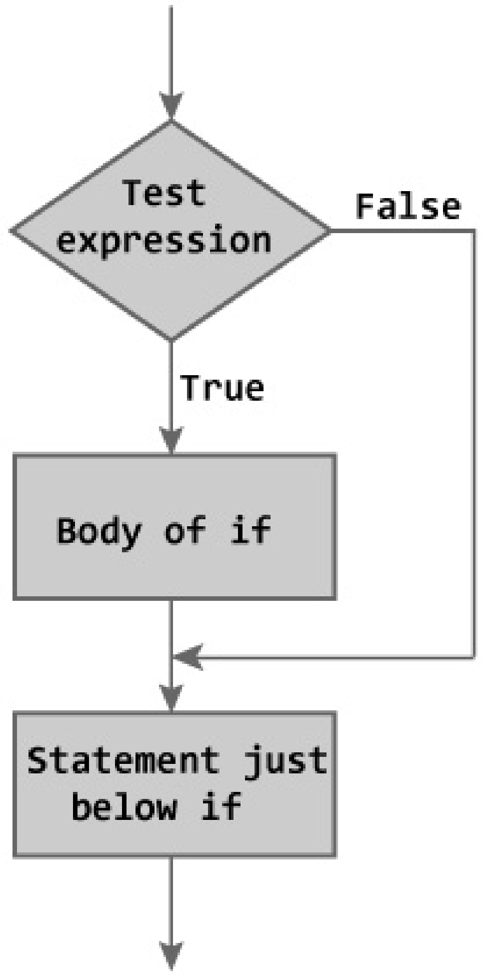
Output

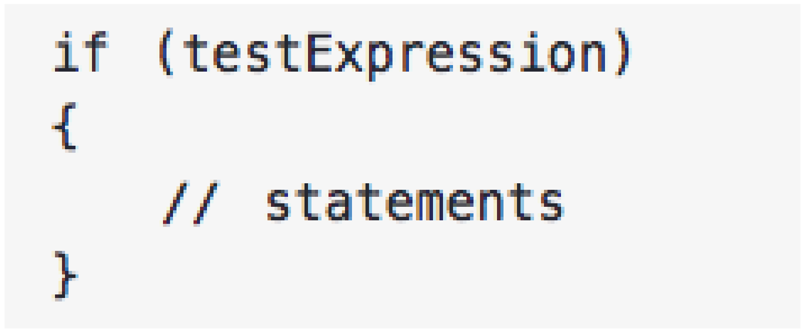
```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```



In programming, decision making is used to specify the order in which statements are executed.

C if statement





```
if (testExpression)
{
    // statements
}
```

The if statement evaluates the test expression inside the parenthesis.

If the test expression is evaluated to true (nonzero), statements inside the body of if is executed.

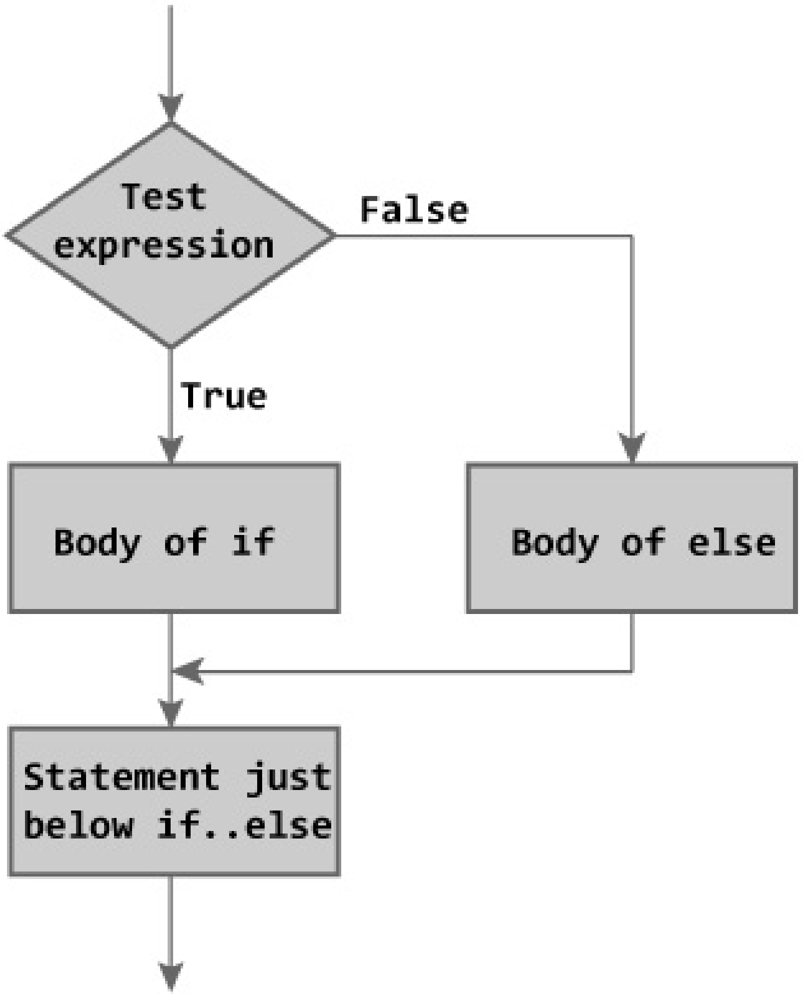
If the test expression is evaluated to false (0), statements inside the body of if is skipped from execution.

C if...else statement

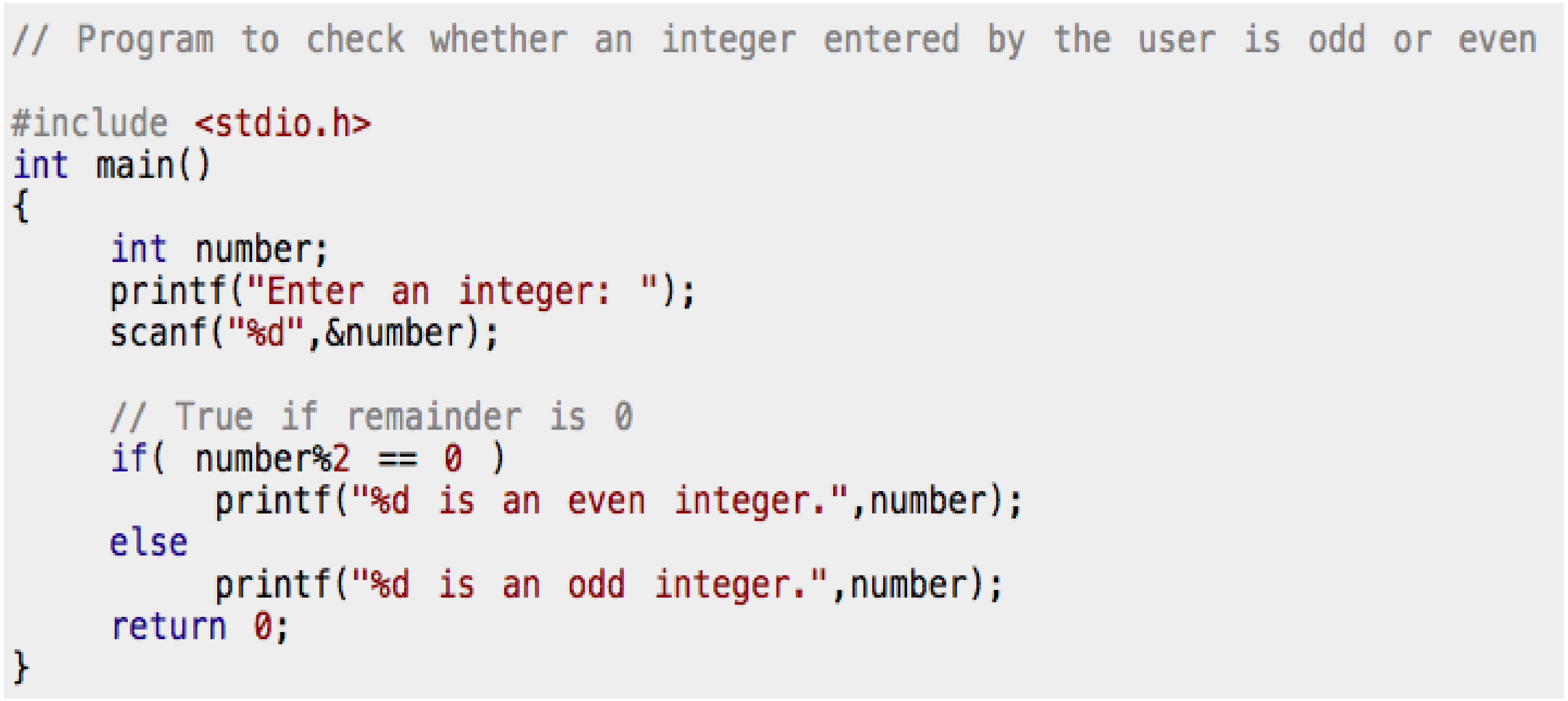
```
if (testExpression) {
     // codes inside the body of if
}
else {
     // codes inside the body of else
}
```

If test expression is true, codes inside the body of if statement is executed and, codes inside the body of else statement is skipped.

If test expression is false, codes inside the body of else statement is executed and, codes inside the body of if statement is skipped.



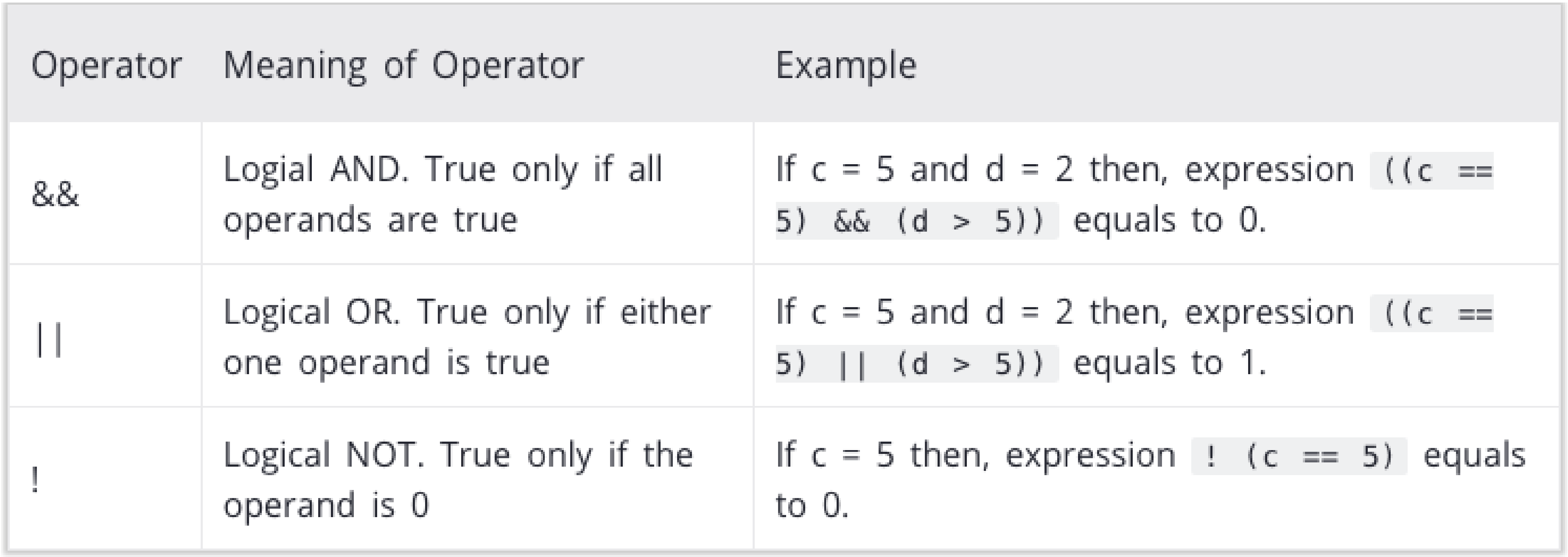
Odd and Even Numbers



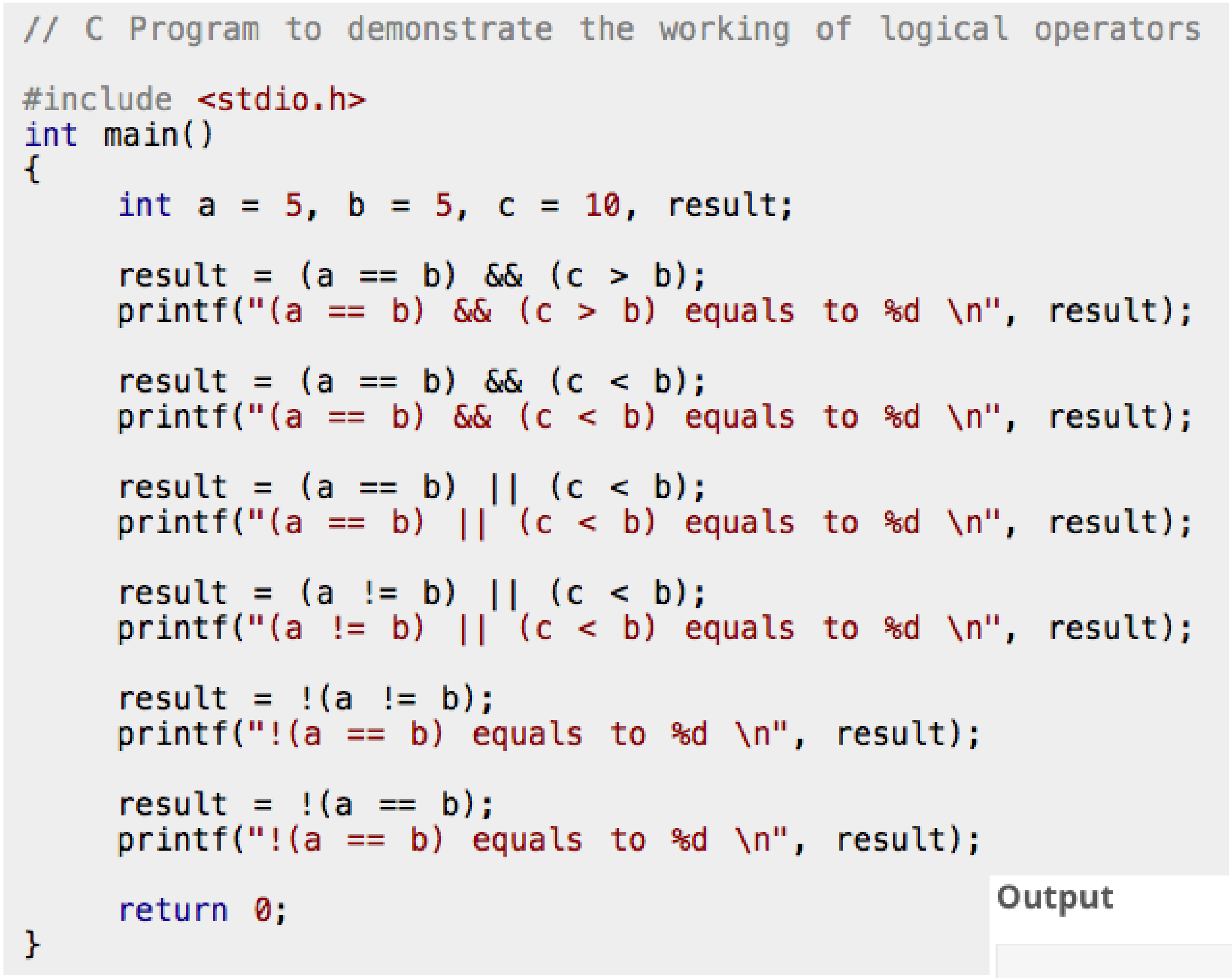
```c
// Program to check whether an integer entered by the user is odd or even

#include <stdio.h>
int main()
{
     int number;
     printf("Enter an integer: ");
     scanf("%d",&number);

     // True if remainder is 0
     if( number%2 == 0 )
          printf("%d is an even integer.",number);
     else
          printf("%d is an odd integer.",number);
     return 0;
}
```

Logical Operators

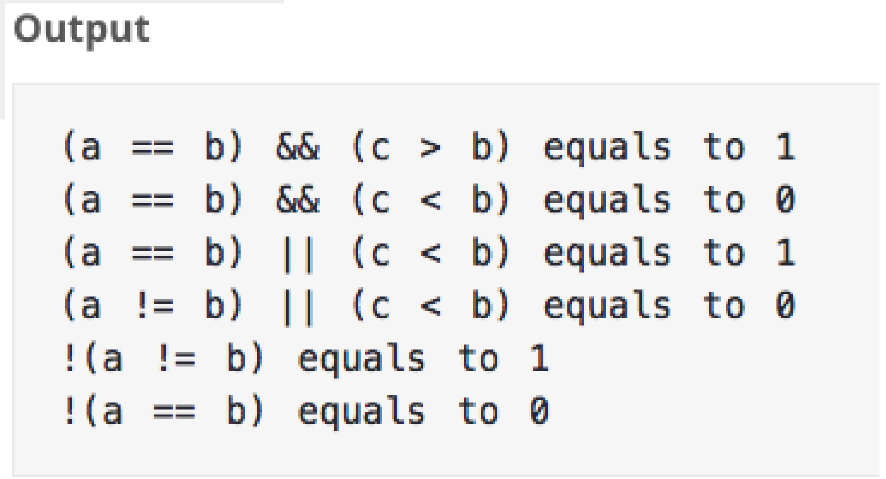
An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.



Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression `((c == 5) && (d > 5))` equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression `((c == 5) || (d > 5))` equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression `! (c == 5)` equals to 0.



```c
// C Program to demonstrate the working of logical operators

#include <stdio.h>
int main()
{
     int a = 5, b = 5, c = 10, result;

     result = (a == b) && (c > b);
     printf("(a == b) && (c > b) equals to %d \n", result);

     result = (a == b) && (c < b);
     printf("(a == b) && (c < b) equals to %d \n", result);

     result = (a == b) || (c < b);
     printf("(a == b) || (c < b) equals to %d \n", result);

     result = (a != b) || (c < b);
     printf("(a != b) || (c < b) equals to %d \n", result);

     result = !(a != b);
     printf("!(a == b) equals to %d \n", result);

     result = !(a == b);
     printf("!(a == b) equals to %d \n", result);

     return 0;
}
```

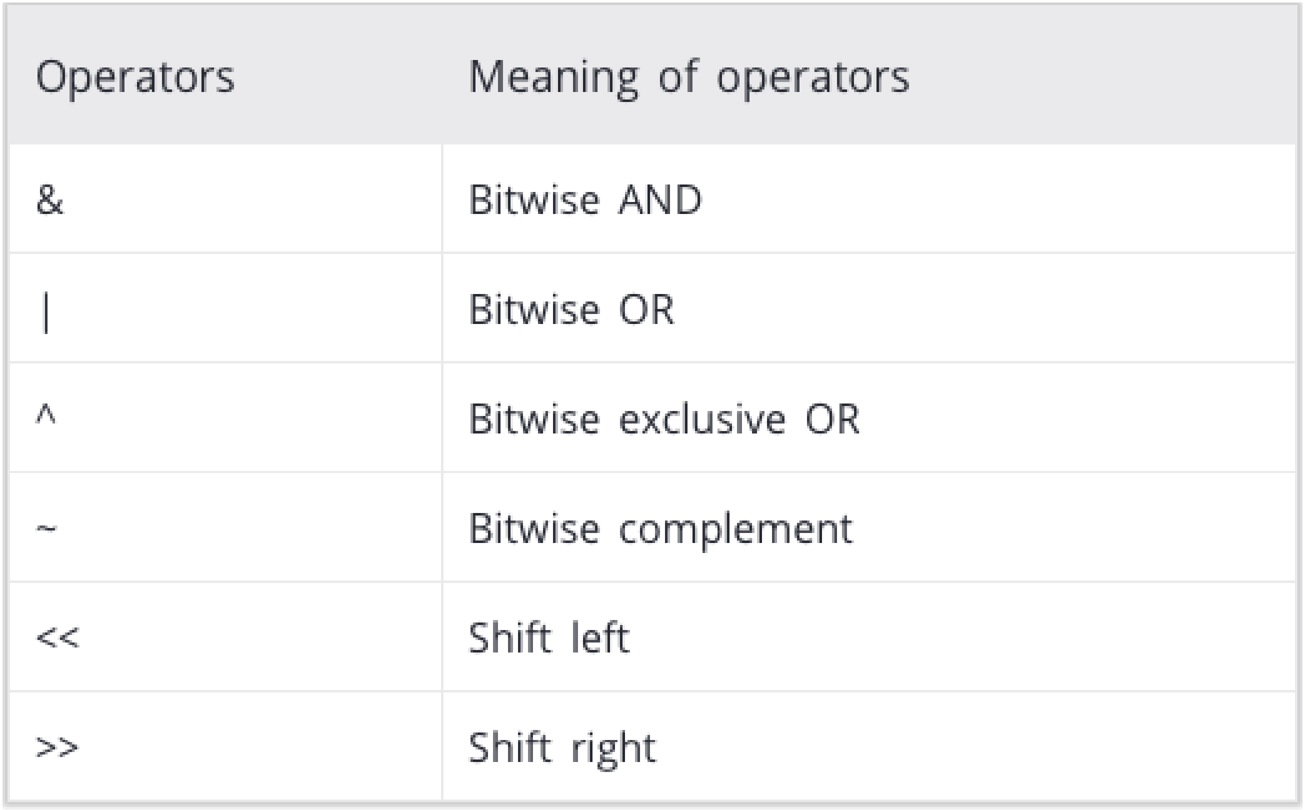


Output

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

Bitwise Operators

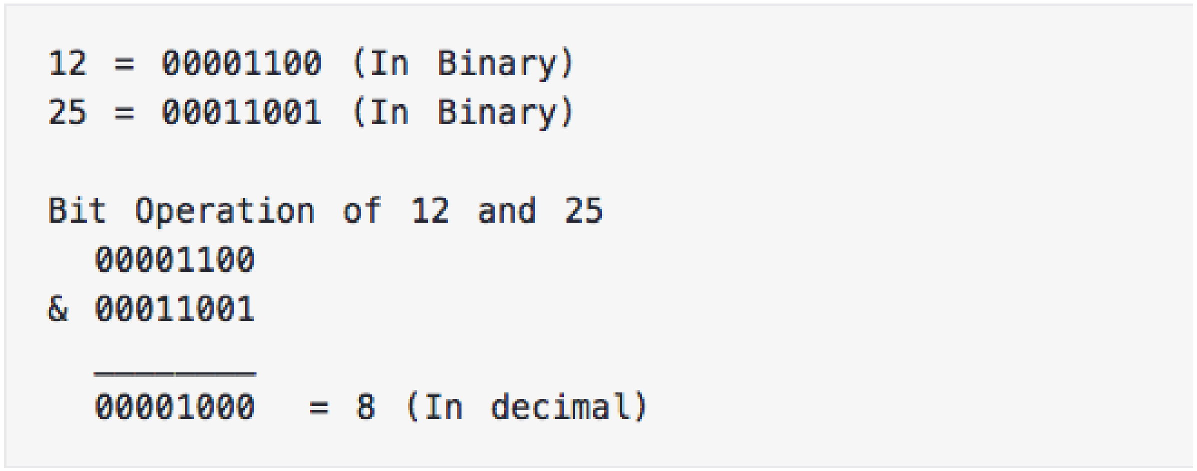
During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power. Bitwise operators are used in C programming to perform bit-level operations.



Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

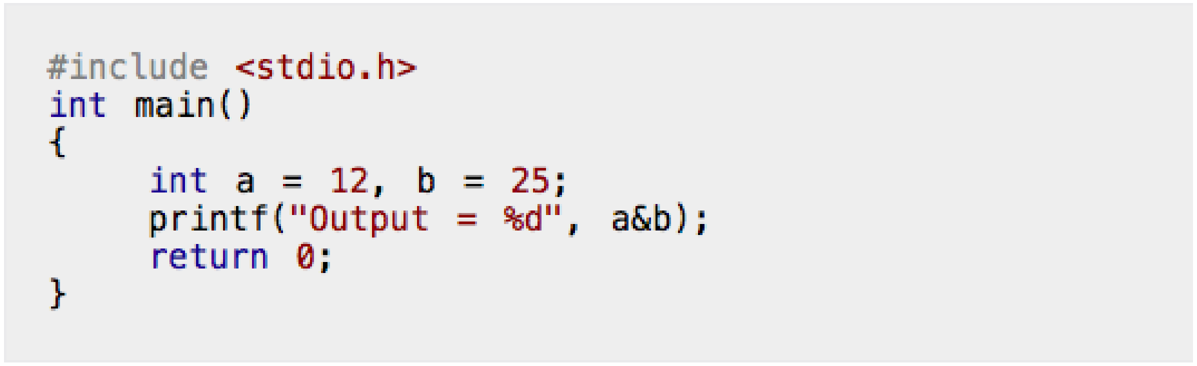
Bitwise AND operator &

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.



```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

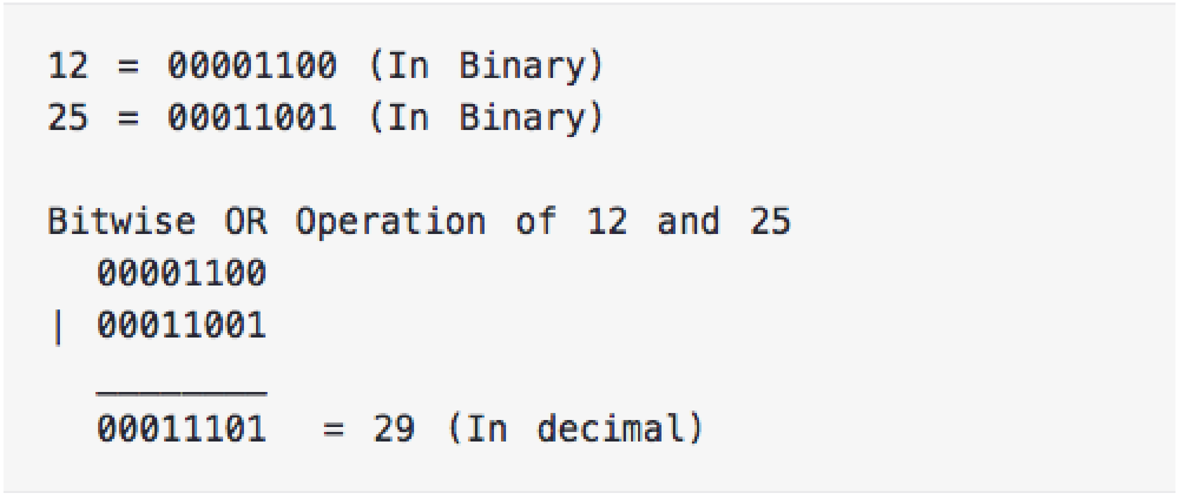
Example #1: Bitwise AND



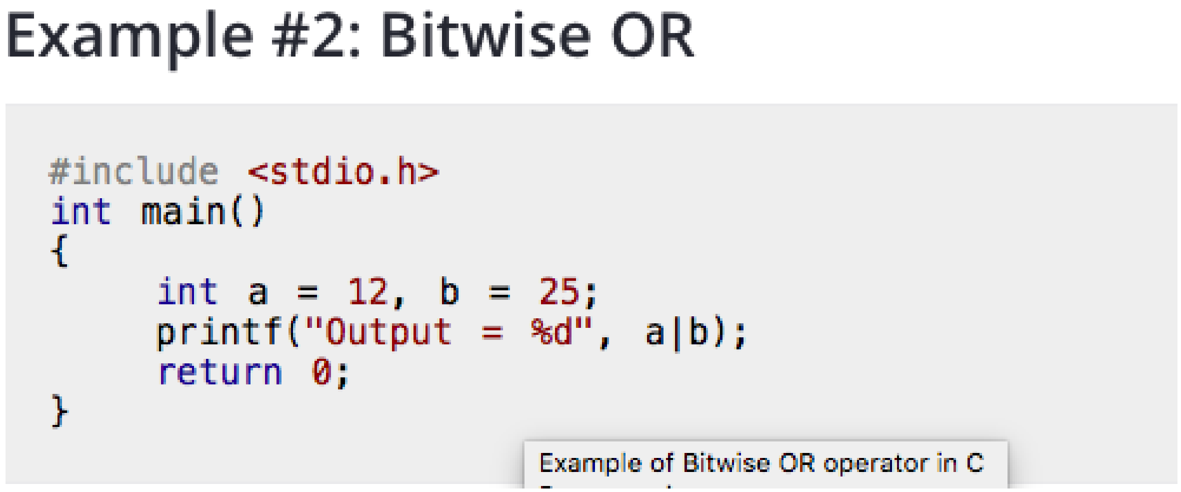
```
#include <stdio.h>
int main()
{
     int a = 12, b = 25;
     printf("Output = %d", a&b);
     return 0;
}
```

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1



```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```



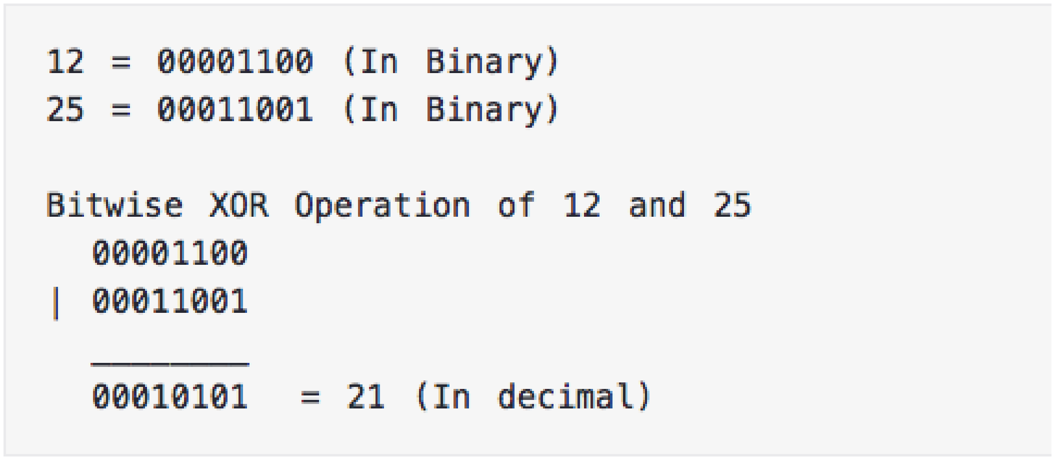
Example #2: Bitwise OR

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

Example of Bitwise OR operator in C

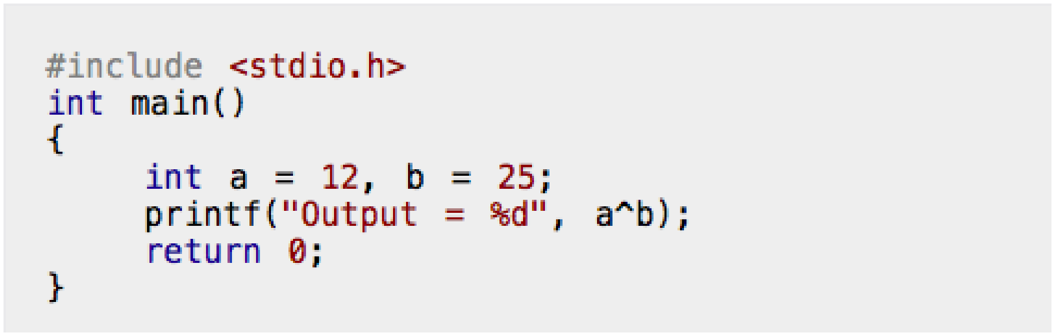
Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.



```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
| 00011001
  ________
  00010101  = 21 (In decimal)
```

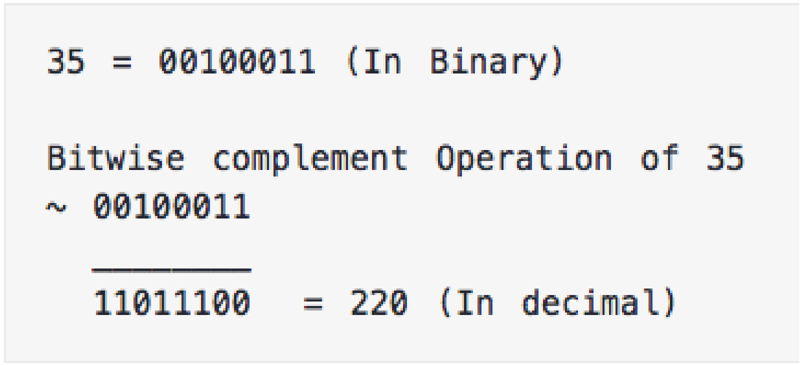
Example #3: Bitwise XOR



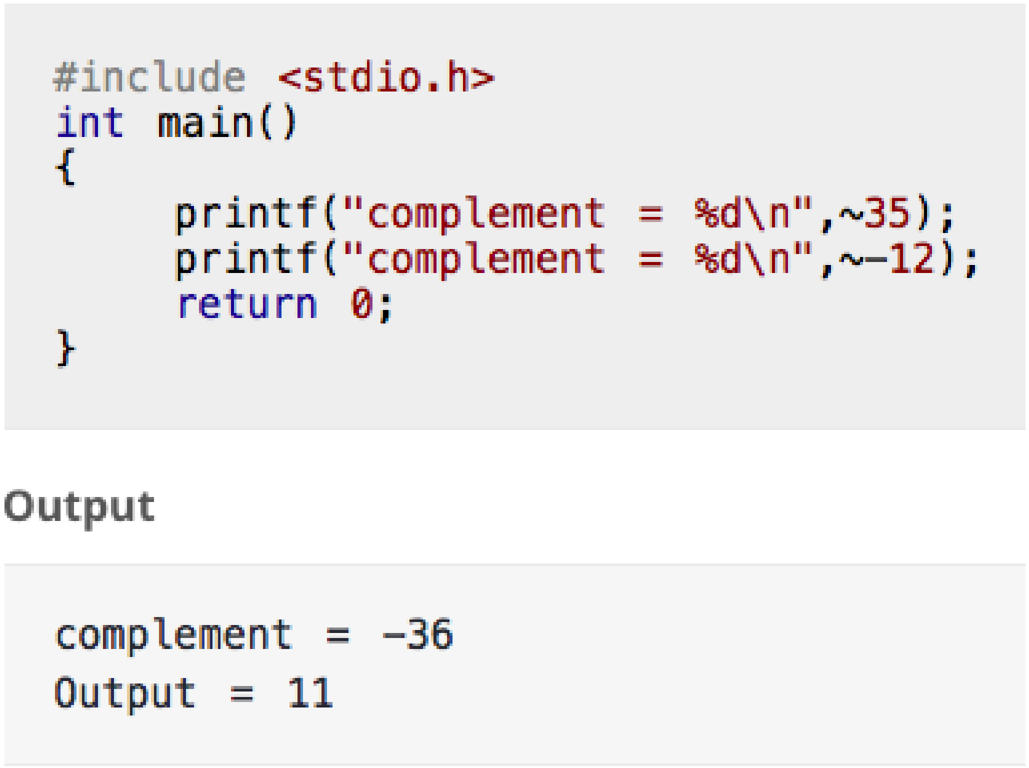
```c
#include <stdio.h>
int main()
{
     int a = 12, b = 25;
     printf("Output = %d", a^b);
     return 0;
}
```

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.



```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
  ________
  11011100  = 220 (In decimal)
```



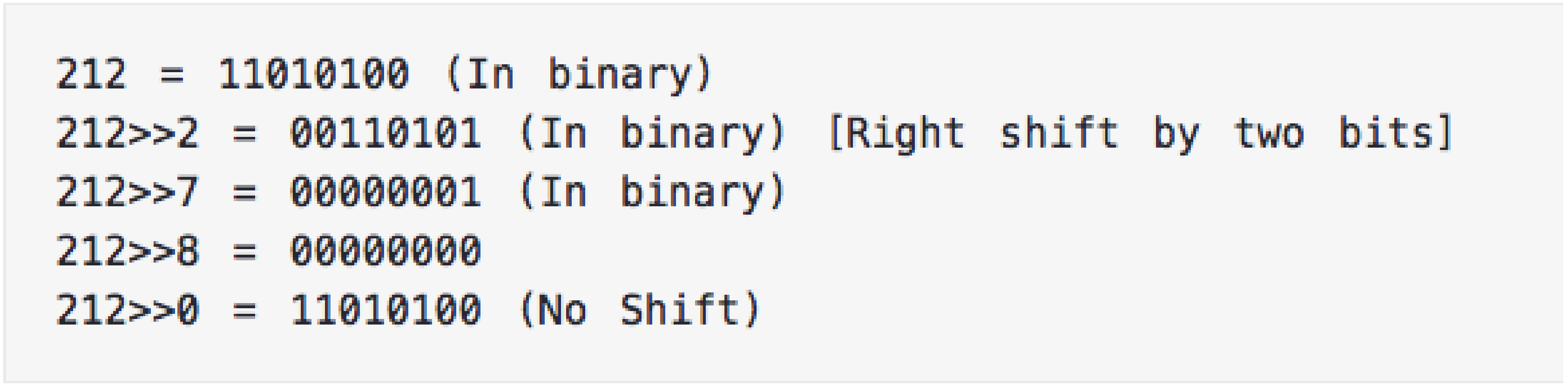
```
#include <stdio.h>
int main()
{
    printf("complement = %d\n",~35);
    printf("complement = %d\n",~-12);
    return 0;
}
```

Output

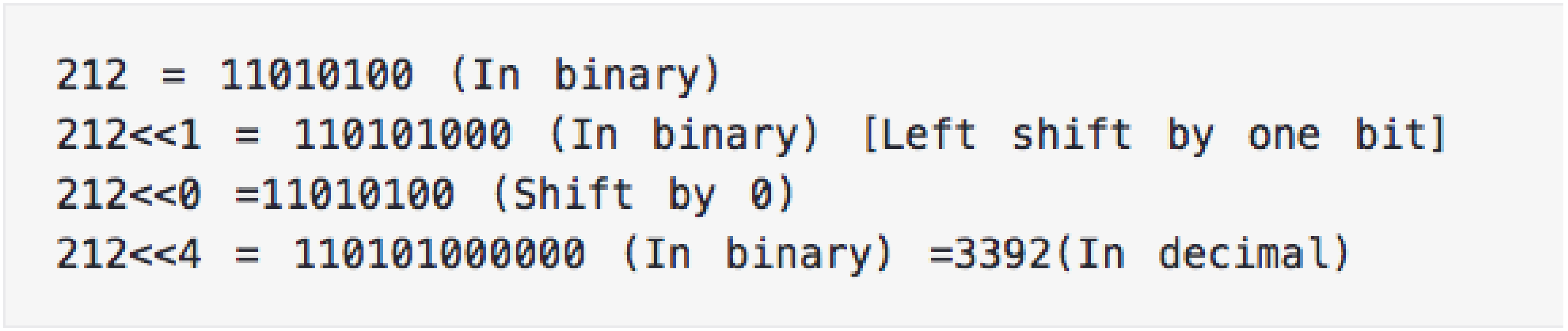
```
complement = -36
Output = 11
```

Shift Operators

There are two shift operators in C programming:
1. Right shift operator
2. Left shift operator.



```
212 = 11010100 (In binary)
212>>2 = 00110101 (In binary) [Right shift by two bits]
212>>7 = 00000001 (In binary)
212>>8 = 00000000
212>>0 = 11010100 (No Shift)
```



```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 =11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

```c
#include <stdio.h>
int main()
{
     int num=212, i;
     for (i=0; i<=2; ++i)
          printf("Right shift by %d: %d\n", i, num>>i);

      printf("\n");

      for (i=0; i<=2; ++i)
          printf("Left shift by %d: %d\n", i, num<<i);

      return 0;
}
```

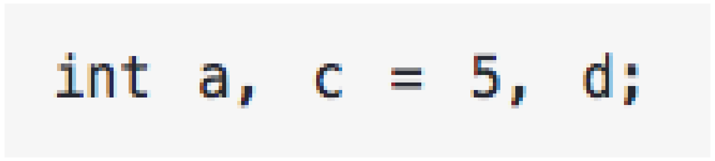
```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

Other Operators

Comma Operator

Comma operators are used to link related expressions together.

For example:



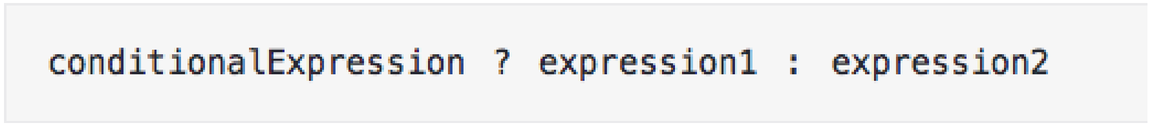
```
int a, c = 5, d;
```

The sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

C Ternary Operator (?:)

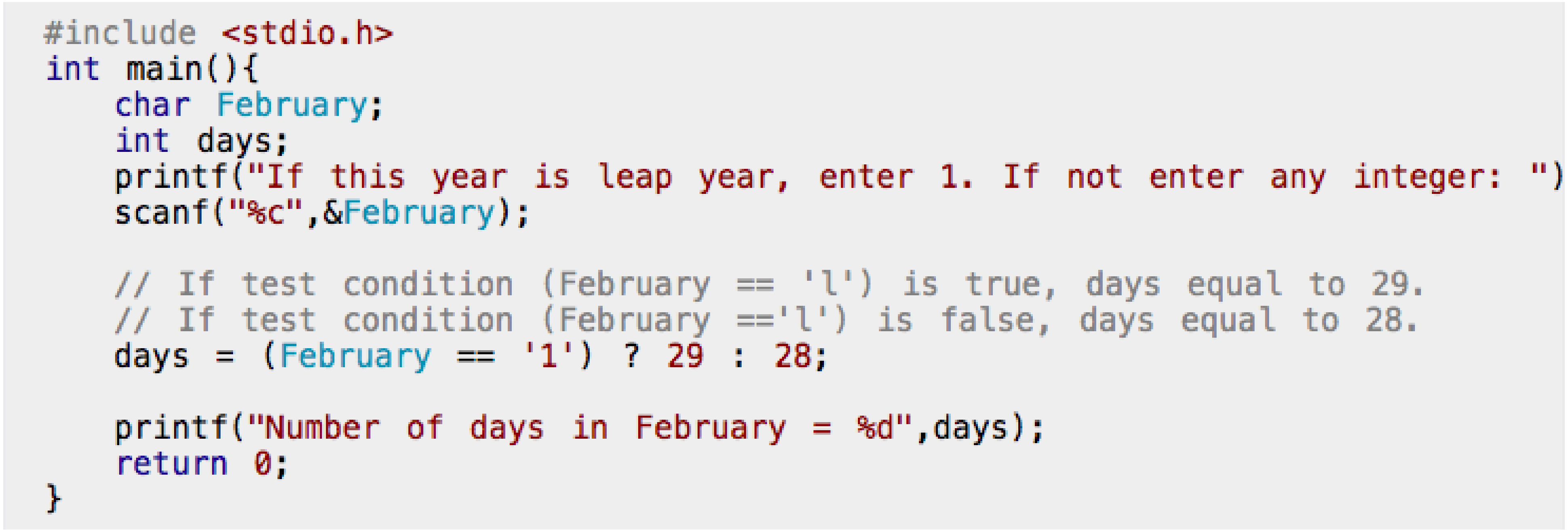
A conditional operator is a ternary operator, that is, it works on 3 operands.



```
conditionalExpression ? expression1 : expression2
```

C Ternary Operator (?:)

The conditional operator works as follows:
1. The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
2. If conditionalExpression is true, expression1 is evaluated.
3. If conditionalExpression is false, expression2 is evaluated.



```c
#include <stdio.h>
int main(){
    char February;
    int days;
    printf("If this year is leap year, enter 1. If not enter any integer: ")
    scanf("%c",&February);

    // If test condition (February == 'l') is true, days equal to 29.
    // If test condition (February =='l') is false, days equal to 28.
    days = (February == '1') ? 29 : 28;

    printf("Number of days in February = %d",days);
    return 0;
}
```

Problems

1) The temperature is 35C; Write a program convert this temperature into Fahrenheit.

ºF =ºC * 1.8000 + 32.00

2) Write a C program that takes a number from the user and checks whether that number is either positive or negative or zero.

3) Write a C program to check a given character is Vowel or Consonant.