# UNIT – 4 Operators and Expression

## Operators

C supports a rich set of operators. An operator is a symbol that tells the computer to perform mathematical or logical operations on operands. Operators are used in programs to manipulate data. C operators can be classified into a number of categories:

### Arithmetic operators:

Arithmetic operators are used for performing most common arithmetic (mathematical) operations.

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

```c
#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;
}
```

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

### Logical operators

Expressions which use logical operators are evaluated to either true or false.

| 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;
}
```

```
(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
```

## Assignment Operator:

An assignment operator assigns a value to a variable and is represented by "=" (equals to) symbol.

**Syntax:** Variable_name = expression.

The Compound assignment operators (+=, *=, -=, /=. %=) are also used.

| Operator | Name | Example |
|---|---|---|
| = | Assignment | C = A + B will assign value of A + B into C |
| += | Add AND assignment | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment | C %= A is equivalent to C = C % A |

```c
#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;
}
```

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

## Relational operators

These operators are used to form relational expressions, which evaluates to either true or false. (True - 1, false - 0). It is used for comparisons. C supports six relational operators.

| Operator | Meaning |
|---|---|
| < | is less than |
| <= | is less than or equal to |
| > | is greater than |
| >= | is greater than or equal to |
| == | is equal to |
| != | is not equal to |

```c
#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;
}
```

```
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
```

## Bitwise operators

These operators which are used for the manipulation of data at bit level are known as bitwise operators. These are used for testing the bits or shifting them right or left.

| Operator | Meaning |
|---|---|
| & | Binary (bitwise) AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | Bitwise left shift |
| >> | Bitwise right shift |
| ~ | Bitwise Ones Complement |

- **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)

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

```c
#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. In C Programming, bitwise OR operator is denoted by |

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

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

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

## Increment and Decrement Operators

The increment operator (++) increases its operand by 1 and the decrement (--) decreases its operand by 1.

They can be written as:

- x++ or ++x (post and pre increment)
- x- -or --x (post and pre decrement)

```c
#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;
}
```

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

Both increment and decrement operator are used on a single operand or variable, so it is called as a unary operator. Unary operators are having higher priority than the other operators it means unary operators are executed before other operators.

Increment and decrement operators can not apply on constant.

*x= 4++; // gives error, because 4 is constant*

```c
#include<stdio.h>
#include<conio.h>

void main()
{
int x,i;
i=10;
x=++i;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

Output

```
x: 11
i: 11
```

```c
#include<stdio.h>
#include<conio.h>

void main()
{
int x,i;
i=10;
x=i++;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

Output

```
x: 10
i: 11
```

```c
#include<stdio.h>
#include<conio.h>

void main()
{
int x,i;
i=10;
x=--i;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

Output

```
x: 9
i: 9
```

```c
#include<stdio.h>
#include<conio.h>

void main()
{
int x,i;
i=10;
x=i--;
printf("x: %d",x);
printf("i: %d",i);
getch();
}
```

Output

```
x: 10
i: 9
```

Example

```c
#include<stdio.h>
#include<conio.h>

void main()
{
int x,a,b,c;
a = 2;
b = 4;
c = 5;
x = a-- + b++ - ++c;
printf("x: %d",x);
getch();
}
```

Output

```
?
```

*Practice more exercises for these type of calculations*

### Conditional Operator

The ternary operator pair ? and : are used to construct conditional operator. An expression that makes use of conditional operator is called conditional expression.

Its general syntax is:

Condition? TRUE case : FALSE case;

**Example:** x=(a>b)?a:b;

```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;
}
```

```
If this year is leap year, enter 1. If not enter any integer: s
Number of days in February = 28
```

## Operator Precedence

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.

Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

| Operator | Priority | Associativity |
|---|---|---|
| {}, (), [] | 1 | Left to right |
| ++, --, ! | 2 | Right to left |
| *, /, % | 3 | Left to right |
| +, - | 4 | Left to right |
| <, <=, >, >=, ==, != | 5 | Left to right |
| && | 6 | Left to right |
| \|\| | 7 | Left to right |
| ?: | 8 | Right to left |
| =, +=, -=, *=, /=, %= | 9 | Right to left |

This represents the evaluation of expression starts from "what" operator.

### Associativity

It represents which operator should be evaluated first if an expression is containing more than one operator with same priority.

## Type casting

Typecasting is converting one data type into another one. It is also called as data conversion or type conversion. It is one of the important concepts introduced in 'C' programming.

'C' programming provides two types of type casting operations:

1. Implicit type casting
2. Explicit type casting

**Implicit type casting**

Implicit type casting means conversion of data types without losing its original meaning. This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.

Implicit type conversion happens automatically when a value is copied to its compatible data type. During conversion, strict rules for type conversion are applied. If the operands are of two different data types, then an operand having lower data type is automatically converted into a higher data type

```
#include<stdio.h>
int main(){
     short a=10; //initializing variable of short data type
     int b; //declaring int variable
     b=a; //implicit type casting
     printf("%d\n",a);
     printf("%d\n",b);
}
```

- **Explicit type conversion:**

  In explicit type conversion, the user has to enforce the compiler to convert one data type to another data type by using typecasting operator. This method of typecasting is done by prefixing the variable name with the data type enclosed within parenthesis.

  **Syntax:** (data type) variable/expression/value;

  **Example**:

  float sum;

  Sum = (int) (1.5 * 3.8);

  The typecast (int) tells the C compiler to interpret the result of (1.5 * 3.8) as the integer 5, instead of 5.7. Then, 5.0 will be stored in sum, because the variable sum is of type float.

**SPECIAL OPERATORS IN C:**

Below are some of the special operators that the C programming language offers.

| Operators | Description |
|---|---|
| & | This is used to get the address of the variable.<br>Example : &a will give address of a. |
| * | This is used as pointer to a variable.<br>Example : * a where, * is pointer to the variable a. |
| Sizeof () | This gives the size of the variable.<br>Example : size of (char) will give us 1. |

- example program for & and * operators in c:

```c
#include <stdio.h>
int main()
{
int *ptr, q;
q = 50;
/* address of q is assigned to ptr */
ptr = &q;
/* display q's value using ptr variable */
printf("%d", *ptr);
return 0;
}
```

- example program for sizeof() operator in c:

```c
#include <stdio.h>
#include <limits.h>

int main()
{
int a;
char b;
float c;
double d;
printf("Storage size for int data type:%d \n",sizeof(a));
printf("Storage size for char data type:%d \n",sizeof(b));
printf("Storage size for float data type:%d \n",sizeof(c));
printf("Storage size for double data type:%d\n",sizeof(d));
return 0;
}
```

```
Storage size for int data type:4
Storage size for char data type:1
Storage size for float data type:4
Storage size for double data type:8
```