Module 2: Operators and its Precedence

CONTENTS

Learning Objectives

1. To know the Operators in C language
2. To Study about different types of Operators
3. To know Precedence & Associatively of C operators
4. To solve examples with various operators

1. Introduction

Expressions in C language are little complicated because of the number of different data types and operators that can be mixed together. Any expression in programming language like C is constructed using combinations of **operators** and **operands**. C language is rich in built-in operators. These operators are either ***binary operators***, which takes two operands or ***unary operators***, which take only one operand. Operators are classified into different categories depending on their nature of operation like arithmetic operators, logical and relational operators, assignment operators, bit operators etc.

As C language has a large number of operators, the precedence of the operators is of greater importance to the programmer. The evaluation order of the operators It will be discussed after studying different types of operators. Let us now begin with the basics of the operators.

2. Operators and expressions

Operator is a symbol that tells the compiler to perform mathematical or logical functions. It is possible to use operators as tools to update or alter the values of variables or data. The data items on which an operator acts is called its ***operands***. The operators can be unary, binary or ternary depending upon whether it operates on one, two or three operands. An operator along with its operands constitutes a simple expression. A compound expression can be formed by using simple expressions as operands of the different types of operators.

3. Types of operators

Like any other programming language, in C language there are number of operators we can use to write an expression. The types of operators are

1. Arithmetic operators
2. Logical operators
3. Relational operators

4. Assignment operators
5. Conditional operators
6. Increment/Decrement operators
7. Bit wise operators
8. Special operators (Comma, Size of , Arrow , & and *)

3.1 Arithmetic operators

There are **five** arithmetic operators which are normally used for arithmetic operations. These operators are binary in nature i.e. they operate on two operands. Table -1 indicates the arithmetic operators in C language.

Table-1: Arithmetic operators

Operator	Meaning	Example
+	Addition of two operands	10+2 =12
-	Subtraction of two operands	10-2 =8
*	Multiplication of two operands	10*2 =20
/	Division of numerator and denominator. Truncation of quotient after integer division.	10/2 =5
%	Modulus operator to provide remainder after integer division.	10%3=1

The operators for addition, subtraction, multiplication and division perform the four basic arithmetic operations. When operator / (for division) is applied to integer the quotient will be truncated. For example, 7/2 = 3.

The modulus operator % returns the remainder after integer division. The % operator cannot be used for float or double data types.

The operator + and – are binary operators and have the same precedence during the calculations but they have lower precedence than *, / and % operator. To change the

order of precedence the brackets ( ) are used. The operator – can be used for both binary and unary operations.

3.2 Logical operators

There are **three** logical operators which are normally used for logical operations. The result of the operation can be True(1) or false(0). Table -2 indicates the arithmetic operators in C language. These operands can assume 0 or 1 values.

Table-2: Logical Operators

Operators	Meaning	Example
&&	Logical AND operation	a && b
||	Logical OR operation	a || b
!	NOT operation i.e. complementation	! a

The operators && and || are binary in nature i.e. they operate on two operands and the result of the operation can be true or false. The not operator (!) is a unary operator and operates only on single operand.

The expression connected by these operators is evaluated left to right and evaluation stops as soon as the True or False condition is known.

3.3 Relational operators

The relational operators are used to check the relationship between two operands. If the relationship is true then it returns value 1 and if the relationship is false, it returns value 0. There are 6 relational operators in C language. Table-3 gives the meaning and utility of these operators.

Table – 3: Relational operators

Operators	Meaning	Example
==	**Equal to**: Checks whether both operands are equal or not	(a ==b)
!=	**Not equal to**: Checks whether both operands are equal or not	(a !=b)
>	**Greater than**: Checks whether left operand is greater than the value of right operand	(a > b)
<	**Less than**: Checks whether left operand is less than the value of right operand	(a < b)
>=	**Greater than or equal to**: Checks whether left operand is greater than or equal to the value of right operand	(a >= b)
<=	**Less than or equal to**: Checks whether left operand is less than or equal to the values of right operand	(a <= b)

The relational operators <, >, <= and >= have the same precedence. The equality operator (= =) and inequality operator (!=) have lower precedence than the other remaining operators. All the relational operators have lesser precedence than the arithmetic operators. Relational and logical operators are normally used in decision making statements in the C program.

3.4 Assignment operator

Assignment operators are normally used while assigning value to a variable and to assign the result of an expression to a variable. The most common assignment operator is =. The assignment operator can be used in the following format.

variable = constant;

or

variable=variable;

or

variable = expression;

There are six different assignment operators. The basic assignment operator (=) can be combined with arithmetic operators to generate shorthand for other assignment operator. Table-4 indicates variety of assignment operators.

Table-4: Variety of assignment operators

Operators	Meaning	Example	Equivalent to
=	Assign	a=b	a=b
+=	Addition and assign	a+=b	a=a+b
-=	Subtraction and assign	a-=b	a=a-b
*=	Multiply and assign	a*=b	a=a*b
/=	Divide and assign	a/=b	a=a/b
%=	Modulus and assign	a%=b	a=a%b

The advantages of shorthand assignment operators are:

1. The statements become more compact.
2. The statement is more efficient.
3. The statement is easy to write and read.

3.5 Conditional operators

Conditional operators are normally used for decision making in C language. The conditional operators are used with expression, unlike other operators in C language. It is a ternary operator and takes three arguments. The format of operators is

expression1? expression 2 : expression 3;

Conditional operator takes three operands and co nsists of two symbols ? and : symbols. A conditional expression is evaluated by first evaluating expression1. If the resultant value is TRUE then expression2 is evaluated and the value of expresion2 becomes the result. Otherwise expression3 is evaluated and its value becomes the result. For example –

y = (a>0) ? 5: -5;

If a is greater than 0 then value of y will be 5 but if a is less than 0 then value of y will be -5. For example, for finding larger of 2 numbers, we can write,

large= (a>b) ? a : b;

This assigns large the value of a if a is greater than b; otherwise it assigns the value of b to large.

Precedence of the conditional operator is lower than all other operators except assignment and comma operators.

3.6 Increment and Decrement operators

In C language, ++ and - - are called increment and decrement operators respectively. Both increment and decrement operators are unary operators i.e. used on single operand. ++ adds 1 to operand and - - subtracts 1 from the operand. These operators can be used both as prefix where the operator is present before operand and postfix, where the operator appears after the operand as shown in table-5.

Table -5: Increment and decrement operator

Operator	Meaning	Example	Result after operation if a=2	
++variable	pre increment	y=++a;	y = 3	a=3
variable++	post increment	y=a++;	y = 2	a=3
- -variable	pre decrement	y=--a;	y = 1	a=1
variable- -	post decrement	y=a--;	y = 2	a=1

The precedence of increment and decrement is highest as compared to all arithmetic operators.

3.7 Bitwise operators

C language provides six bitwise operators. These operators work on the bits and perform bit-by-bit operations. The bitwise operations need two operands, both of which must be integer types. Table -6 indicates different bitwise operators.

Table-6: Bitwise operators

Operators	Meaning	a	b	Y (output)
&	AND	00001010 (10)	00011001 (25)	00001000 (08)
|	Inclusive OR	00001010 (10)	00011001 (25)	00011011 (27)
^	Exclusive OR	00001010 (10)	00011001 (25)	00010011 (19)
~	Complement	00001010	~a =	11110101
<<	Shift Left	00001010	a<<2	00101000
>>	Shift Right	00001010	a>>2	00000010

Bitwise AND operator (&) – It is binary operator works on 2 operands. The output of bitwise AND is 1 if both the corresponding bits of operand is 1. If either of bit is 0 or both are 0 then the output will be 0.

Bitwise inclusive OR operator (|)– It is binary operator works on 2 operands. The output of bitwise OR is 1 if either of the bits is 1 or both bits are 1. If both bits are 0 then the output will be 0.

Bitwise exclusive OR operator (^) – It is binary operator works on 2 operands. The output of bitwise XOR operation is 1 if the corresponding bits of operands are opposite. If both bits are 0 or 1 then the output will be 0.

Bitwise complement operator(~) – It is a unary operator work on only one operand. It changes the corresponding bit of the operand to opposite bit i.e. 0 to1 and 1 to 0.

There are two shift operators in C language. The right shift (>>) and left shift (<<) operator.

Right shift operators (>>)- It moves all the bits towards right by certain number of bits which can be specified after the operator e.g. a>>2 right shift the contents by two bits and a>>5 right shifts the contents by 5 bits.

Left Shift Operator (<<)- It moves all the bits towards the left by certain number of bits which can be specified after the operator e.g. a<<2 left shifts the contents by two bits and a<<5 left shifts the contents by 5 bits.

3.8 Special operators (Comma, Size of , Arrow , & and *)

In addition to specific function operators there are few miscellaneous operators also known as special operators. Table-7 gives the details of these special operators.

Table-7: Special opearators

Operators	Meaning	Example
sizeof()	Size of objects & data types	sizeof (a)
&	Address Operator (Unary operator)	&a;
*****	Pointer (Unary operator)	*a;
,	Series operator	int a=5,b,sum=0;
> And .	Arrow and . operator (to access member of structure)	(&a)->y var.a

Size of operator – The sizeof operator is used to know the size of a variable. For example, sizeof(a), where a is a integer and returns 4 i.e. 4 bytes.

& operator - It returns the address of a variable. For example &a will return the actual address of the variable.

*** operator** – It provides pointer to a variable. For example *a.

Arrow and dot operator - both operators provide access to members of structure of union. They differ in that arrow (->) is used when the variable is a pointer to a structure or union. The dot(.) operator is used when the variable is itself the structure or union.

4. Precedence & associativity of operators:

A table-8 is given below which summarize the precedence and associativity of C operators. The order of precedence is listed from highest to lowest.

Table-8: Precedence and Associativity of C Operators

Symbol	Type of operation	Associativity
[ ] ( ) . –> postfix ++ and postfix – –	Expression	Left to right
Prefix ++ and prefix — size of& * + –~ !	Unary	Right to left
Typecasts	Unary	Right to left
*** / %**	Multiplicative	Left to right
+ –	Additive	Left to right
<<>>	Bitwise shift	Left to right
<><= >=	Relational	Left to right
== !=	Equality	Left to right

&	Bitwise-AND	Left to right
^	Bitwise-exclusive-OR	Left to right
|	Bitwise-inclusive-OR	Left to right
&&	Logical-AND	Left to right
||	Logical-OR	Left to right
? :	Conditional-expression	Right to left
= *= /= %= += –= <<= >>= &= ^= |=	Simple and compound assignment	Right to left
,	Sequential evaluation	Left to right

Example Programs of C operators

a) Increment and Decrement.

1. Prefix notation will increment the variable **BEFORE** the expression is evaluated.

Here is an example:

```
main()
 {
int a=1;
printf(" a is %d", ++a);
}
```

2. Postfix notation will increment AFTER the expression evaluation.

```
main()
 {
int a=1;
printf(" a is %d", a++);
}
```

In both examples, the final value of a will be 2. BUT the first example will print 2 and the second will print 1.

b) Assignment operators

C has the following assignment operators:

=, +=, -=, *=, /=, %=, is the obvious one. It takes the result of the expression on the right and assigns it to the variable on the left (lvalue).

```
main ()
{
int a, b=4, c=10;
a = b + c;
}
```

b is assigned the value 4 ,c is assigned the value 10 , a is assigned the result of b plus C (14). All the other operators act in a similar way. If you find yourself coding the example onthe left, it could be changed to the example on the right.

```
main()
{
int a=4;
a = a * 4;
}
```

```
main()
{
int a=4;
a *= 4;
}
```

```
main()
{
int a=10, b=2;
a /= b * 5;
}
```

```
main()
{
int a=10, b=2;
a = a / (b*5);
}
```

Again both produce the same answer (1).

Conditional Expressions.

We have a shorthand construct for some **if ... else ...** constructs.

Consider the following two examples

Example 1	Example 2

if (x == 1)
y = 10;
else
y = 20;
	y = (x == 1) ? 10 : 20;

These examples both perform the same function. If x is 1 then y becomes 10 else y becomes 20. The example on the right evaluates the first expression '(x ==1)' and if **true** evaluates the second '10'. If **false** the third is evaluated.

5. Summary:

The type of operations that can be performed on the data objects are specified by the operators. The data items on which an operator acts are called operands. Combine different types of arithmetic operators forms arithmetic expressions. The operators can be unary, binary and ternary depending upon whether it operates on one, two or three operands. The evaluation order of the operators in an expression will be determined by the operator precedence rules decided by the language.

Development Team

Principal Investigator:	Dr. Arvind D Shaligram, Savitribai Phule Pune University, Pune
Paper Coordinator:	Dr. Vijay P Labade, Fergusson College, Pune
Content Writer:	Dr. Vijay P Labade, Fergusson College, Pune
Content Reviewer:	Dr. Ajay Kumar (Director), Jayawant Institute of Business Studies Tathwade, Pune