11

OPERATORS AND EXPRESSIONS IN 'C'

11.1 INTRODUCTION

Operators form expressions by joining individual constants, variables, array elements as discussed in previous lesson. C includes a large number of operators which fall into different categories. In this lesson we will see how arithmetic operators, unary operators, relational and logical operators, assignment operators and the conditional operators are used to form expressions.

The data items on which operators act upon are called operands. Some operators require two operands while others require only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variable as operand.

11.2 OBJECTIVES

After going through this lesson you will be able to

- recognize arithmetic operators
- explain unary operators
- define relational, logical, assignment & conditional operators
- explain library functions

11.3 ARITHMETIC OPERATORS

There are five main arithmetic operators in 'C'. They are '+' for additions, '-' for subtraction, '*' for multiplication, '/' for division and '%' for remainder after integer division. This '%' operator is also known as modulus operator. For exponentiation there is no specific operator in 'C' instead there is one library function known as pow to carry out exponentiation.

Operands can be integer quantities, floating-point quantities or characters. The modulus operator requires that both operands be integers & the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero, though the operands need not be integers. Division of one integer quantity by another is referred to as integer division. With this division the decimal portion of the quotient will be dropped. If division operation is carried out with two floating- point numbers, or with one floating-point number. & one integer, the result will be a floating-point quotient.

If we take two variables say x and y and their values are 20 and 10 respectively, and apply operators like addition, subtraction, division, multiplication and modulus on them then their resulted values will be as follows:

x+y	30
x-y	10
x*y	200
x/y	2
x %y	0

If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra.

The interpretation of remainder operation is unclear when one of the operands is negative.

Operands that differ in type may undergo type conversion before the expression takes on its final value. The final result has highest precision possible, consistent with data types of the operands. The following rules apply when neither operand is unsigned.

- If one operand is a floating point type and the other is a char or an int, the char/int will be converted to the floating point type of the other operand and the result will be expressed as such. So, an operation between an int and a double will result in double.

- If both operands are floating-point types with different precisions, the lower-precision operand will be converted to the precision of the other operand and the result will be expressed in this higher precision.

  Float & double → double

  Float & long double → long double

  Double & long double → long double

- If neither operand is a floating point type or a long int, then both operands will be converted to int & the result will be int.
- If neither operand is a floating point type but one is a long int, the other will be converted to long int & the result will be long int.

The value of an expression can be converted to a different data type if desired. The 'C' programmer also has the choice of explicitly specifying how the values are to be converted in a mixed mode expression. This feature is known in 'C' as **coercion** and may be accomplished using what is called the **cast operator** . The name of data type to which the conversion is to be made is enclosed in parentheses and placed directly to the left of the value to be converted; the word "cast" never is actually used. The example of type casting is as follows:

```
int a=17;
float b;
b=(float)a+ 11.0;
```

The cast operator converts the value of int a to its equivalent float representation before the addition of 11.0 is carried out. The precedence of the cast operator is the same as that of the unary minus operator. The cast operator can be used on a constant or expression as well as on a variable e.g.

$$(int)\ 7.179$$

$$\frac{(double)\ (5*3/8)}{(float)(a+4)}$$

In the second example, the cast is performed only after the entire expression within the parentheses is evaluated.

Remember due to type casting, the data type associated with the expression itself is not changed, but it is the value of the expression that undergoes type conversion wherever the cast appears. This is particularly relevant when the expression consists of only a single variable.

The operator within C are grouped hierarchically according to their order of evaluation known as **precedence**. Obviously operations with a higher precedence are carried out before operations having a lower precedence. The natural order can be altered by making use of parentheses.

Arithmetic operators *,/ and % are under one precedence group and +,- are under another precedence group. The operators *, / and % have higher precedence than + and -. In other words, multiplication, division and remainder operations will be carried out before addition and subtraction.

Another important point to consider is the order in which consecutive operations within the same precedence group are carried out. This is known as **associativity**. Within each of the precedence groups described above, the associativity is left to right. In other sense, consecutive addition and subtraction operations are carried out from left to right, as are consecutive multiplication, division and remainder operations.

As stated earlier, the natural precedence of operations can be altered through the use of parentheses, thus allowing the arithmetic operations within an expression to be carried out in any desired order. In fact, parentheses can be nested, one pair within another. In such cases the innermost operations are carried out first, then the next innermost operations, and so on. It is perfectly acceptable to use redundant parentheses if they help to make the expression more meaningful. Whenever parentheses are used, however, be sure that there are as many left parentheses as right.

Let us understand this with the help of one example, say there are 3 variables a,b, and c having values 5,10 and 15 respectively. The different operations on these three variables and their result is as follows:

```
a+b/c=5
b*c-a=145
```

a*b/c= 3

(a+c)*b/a=40

In the first expression, division is done first because division has a higher precedence than addition.

In the second expression, the multiplication is done before the subtraction.

In the third expression, the calculation proceeds from left to right, but truncation takes place in the division operation.

Finally in the last case (a+c)*b/a, the contents of the parentheses are evaluated first, then it is multiplied & finally divided. In this case no truncation takes place.

Sometimes it is a good idea to use parentheses to clarify an expression, even though the parentheses may not be required, but use of overly complex expressions should be avoided. Such expressions are difficult to read, and they are often written incorrectly because of unbalanced parentheses.

INTEXT QUESTIONS

1. What is the effect of dividing an integer by a real quantity? Also, what is this effect called?
2. How would you use the cast operator to convert(7*9/11) to double?

11.4 UNARY OPERATORS

'C' includes a class of operators that act upon a single operand to produce a new value. Such operators are known as unary operators. Unary operators usually precedes their single operands, though some unary operators are written after their operands.

The most common unary operator is unary minus, where a minus sign precedes a numerical constant, a variable or an expression.

e.g.

-5,-10, -20(numbers)

x=-y(variable)

Of all the arithmetic operators, the unary minus has the highest precedence level. Thus in an expression such as

```
y=x+z* -b;
```

evaluation commences with the unary minus, which negates the value of b. Then z is multiplied by –b, and finally the addition takes place. The result is stored in the variable y. So, the use of parentheses to separate the two adjacent operators and avoid possible confusion.

```
Y=x+z*(-b);
```

In C, all numeric constants are positive. Thus a negative number is actually an expression, consisting of the unary minus operator, followed by a positive numeric constant.

It is to be noted here that the unary minus operation is distinctly different from the arithmetic operator which denotes subtraction(-). The subtraction operator requires two separate operands.

Two other commonly used unary operators are increment operator, ++, & the decrement operator - -. The increment operator causes its operand to increased by one, whereas the decrement operator causes its operand to be decreased by one. The operand used with each of these operators must be a single variable. For example, x is an integer variable that has been assigned a value of 10. The expression ++ x, which is equivalent to writing x= x+1, causes the value of x to be creased to 11. Similarly the expression --x, which is equivalent to x=x-1, causes the original value of x to be decreased to 9.

The increment and decrement operators can each be utilized in two different ways, depending on whether the operator is written before or after the operand. If the operator precedes the operand, then the value of operand will be altered before it is used for its intended purpose within the program. If, however the operator follows the operand then the value of the operand will be changed after it is used.

For example, if the value of x is initially 10, then if we increase it by saying ++x then the current value of x will be 11.

Similarly for decrement operator the current value of x will be 9 if we say -- x.

Another unary operator is the **sizeof** operator . This operator returns the size of its operand, in bytes. This operator always precedes its operand. The operand may be an expression, or it may be a cast.

e.g sizeof (x);
sizeof (y);

If x is of integer type variable and y is of floating point variable then the result in bytes is 2 for integer type and 4 for floating point variable.

Consider an array school[]= “National”

Then, sizeof (school) statement will give the result as 8.

A cast is also considered to be unary operators. In general terms, a reference to the cast operator is written as (type). Thus, the unary operators we have encountered so far are -,++,- -, sizeof & (type).

Unary operators have a higher precedence than arithmetic operators. Hence, if a unary minus operator acts upon an arithmetic expression that contains one or more arithmetic operators, the unary minus operation will be carried out first. The associativity of the unary operators is right to left.

INTEXT QUESTIONS

3. What are unary operators?
4. How many operands are associated with a unary operators?
5. Describe the difference between increment and decrement operators.
6. What is the associativity of unary operators?
7. What is sizeof operator?

11.5 RELATIONAL, LOGICAL, ASSIGNMENT, CONDITIONAL OPERATORS

Relational operators are symbols that are used to test the relationship between two variables, or between a variable and a constant. The test for equality, is made by means of two adjacent equal signs with no space separating them. ‘C’ has six relational operators as follows:

>	greater than
<	les than
!=	not equal to
>=	greater than or equal to
>=	less than or equal to

These operators all fall within the same precedence group, which is lower than the unary and arithmetic operators. The associativity of these operators is left-to-right.

The equality operators ==and != fall into a separate precedence group, beneath the relational operators. Their associativity is also from left-to-right.

These relational operator are used to form logical expression represeating condition thet are either true or false. The resulting expression will be of type integer, since true is represented by the integer value and false is represented by the value0.

Let us understand operations of these relational operaters with the help of an example: x=2, y=3, z=4.

x<y	true	1
(x+y) >=z	true	1
(y+z)>(x+7)	false	0
z!=4	false	0
y ==3	true	1

Here, true or false represents logical interpretation and they have values 1 and 0 respectively.

There are two logical operators in C language, they are **and or.** They are represented by && and !! respectively.

These operators are refered to as **logical and, logical or,** respectively. The result of a **logical and** operation will be true only if both operands are true, whereas the result of a **logical or** operation will be true if either operand is true or if both operands are true.

The logical operators act upon operands that are themselves logical expressions. Let us understand it, with the help of a following example:

Suppose x=7, y=5.5 , z= 'w'

Logical expressions using these variables are as follows:

Expression	Interpretation	Value
(x>=6) &&(z= =’w’)	true	1
(x>=6)¦¦ (y = =119)	true	1
(x<=6) && (z=='w')	false	0

Each of the logical operators falls into its own precedence group. Logical **and** has a higher precedence than logical **or**. Both precedence groups are lower than the group containing the equality operators. The associativity is left to right.

'C' also includes the unary operator !, that negates the value of a logical expression. This is known as logical negation or logical **NOT** operator. The associativity of negation operator is right to left.

Precedence of operators in decreasing order is as follows:

1. -, ++ - - Operators ! size of (type)
2. * / %
3. + -
4. < <= > > =
5. == !=
6. &&
7. ¦¦

If you have trouble remembering all these rules of precedence you can always resort to parentheses in order to express your order explicitly.

INTEXT QUESTIONS

8. What is the difference between = and == ?
9. What logical operator negates the truth value of the expression to its immediate right?
10. What integer value is equivalent to false?
11. What is the associativity of logical NOT?

There are several different assignment operators in C. All of them are used to form assignment expression, which assign the value of an expression to an identifier. The most commonly used assignment operator is =. The assignment expressions that make use of this operator are written in the form:

identifier=expression

where identifier generally represents a variable and expression represents a constant, a variable or a more complex expression.

Assignment operator = and the equality operator == are distinctly different. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two expressions have the same value. These two operators cannot be used in place of one another.

If the two operands in an assignment expression are of different data types, then the value of the expression on the right will automatically be converted to the type of the identifier on the left. The entire assignment expression will then be of this same data type. For example

A floating point value may be truncated if assigned to an integer identifier, a double-precision value may be rounded if assigned to a floating-point identifier, an integer quantity may be altered if assigned to a shorter integer identifier or to a character identifier.

Multiple assignments of the form

Identifier 1= identifier 2 = - - - -= expression are allowed in ‘C’.

In such situations, the assignments are carried out from right to left.

Let us understand this with the help of following example

x=y=10 (x and y are integer variables)

will cause the integer value 10 to be assigned to both x and y. Similarly, x=y=10.5

will cause the integer value 10 to be assigned to both x and y, truncation occurs when the floating point value 10.5 is assigned to the integer variable y.

‘C’ also contains the five additional assignment operators +=, -=, *=, /= & %=.

expression1+= expression2

is equal to

expression1= expression1 + expression2

Similarly expression1 - = expression2 is equivalent to expression1 = expression1 - expression 2

Assignment operators have a lower precedence than any of the other operators that have been discussed earlier. The decreasing order of precedence is given below, the associativity of assignment operators is right to left.

```
-   ++ - - ! sizeof (type)
* /  %
+ -
< <=  > >=
= =  !=
&&
| |
=  + =  - = * =  /=  %=
```

For example   x* = -3 *(y+z)/4

is equivalent to x= x*(-3 *(y+z)/4)

In this example first (y+z) will be evaluated, then multiplication will take place. Then division, and finally multiplication will take place.

INTEXT QUESTIONS

12. What are the 6 assignment operators discussed above ?
13. How can multiple assignments be written in C?

There is one conditional operator(?:) in ‘C’ language. An expression that makes use of the conditional operator is called a conditional expression.

A conditional expression is written in the form

Expression 1 ? expression 2 : expression 3

When evaluating a conditional expression, expression 1 is evaluated first. If expression 1 is true, then expression 2 is evaluated and this becomes the value of the conditional expression. If expression 1 is false, then expression 3 is evaluated and this becomes the value of the conditional expression

For example (i< 1) ? 0:200

i is integer variable here.

The expression (i<1) is evaluated first, if it is true the entire conditional expression takes on the value 0. Otherwise, the entire conditional expression takes on the value 200.

Conditional expression frequently appear on the righthand side of a simple assignment statement. The resulting value of the conditional expression is assigned to the identifier on the left.

The conditional operator has its own precedence, just above the assignment operator. The associativity is right-to-left.

Decreasing order of precedence is as follows:

1 - ++ - - ! sizeof (type)

2 * / %

3 + - ?:

4 < <= > >=

5 = = !=

6 &&

7 ! !
! !

8 ?:

9 = += -= *= /= %=

11.6 LIBRARY FUCNTIONS

Library functions carry out various commonly used operations or calculations. Some functions return a data item to their access point, others indicate whether a condition is true or false by returning 1 or 0 respectively, still others carry out specific operations on data items but do not return anything. Features which tend to be computer dependent are generally written as library functions.

Functionally similar library functions are usually grouped together as object programs in separate library files. These library files are supplied as a part of each C compiler.

A library function is accessed simply by writing the function name, followed by a list of arguments that represent information being passed to the function. The arguments must be enclosed in parentheses and separated by commas. The arguments can be constants, variable names or more complex expressions. The parentheses must be present, even if there are no arguments, A function that returns a data item can appear anywhere within an expression in place of a constant or an identifier. A function that carries out operations on data items but does not return anything can be accessed simply by writing the function name, since this type of function reference constitutes an expression statement. In order to use a library function it may be necessary to include certain specific information within the main portion of the program. This information is generally stored in special files supplied with the compiler. Thus, the required information can be obtained simply by accessing these special files. This is accomplished with the preprocessor statement.

```
 # include <filename>
```

The list of some commonly used library functions are:

abs(i)	to determine absolute value of i.
exp(i)	raise e to the power i.
log(d)	determine natural logarithm of d.
pow(d1,d2)	returns d1 raised to the d2 power
putchar(c)	send a character to the standard output device
sqrt(d)	return the square root of d.

You can see rest of the library functions available in C in Help topic of 'C'.

INTEXT QUESTIONS

14. Define the associativity of conditional operators.
15. Are the library functions actually a part of the 'C' language?

11.7 WHAT YOU HAVE LEARNT

In this lesson you have learnt about 'C' arithmatic, unary, relational, logical, assignment and conditional operators. The concept of library functions here also been discussed.

11.8 TERMINAL QUESTIONS

1. What is an operator? Describe different types of operators?
2. What is an operand ? What is the relationship between operators & operands ?
3. Summarize the rules that apply to expression whose operands are of different type?
4. When should parentheses be included within the expression?
5. How can the conditional operator be combined with the assignment operator to form an “if-else” type statement?

11.9 KEY TO INTEXT QUESTIONS

1. The integer is temporarily converted to its floating point equivalent before the calculation is performed. This automatic operation is known as implicit conversion.
2. (double) (7*9/11)
3. Operators that act upon a single operand to produce a new value are called unary operator.
4. Only one
5. Increment means increase the value, or add the specified value to the variable’s value. Decrement means decrease or less the value of the variable.
6. Right to left
7. It will display the total size of variable or data-type.
8. = is the assignment operator and == is the equality operator.
9. not or !

10. zero
11. right to left
12. =, +=, - =, * =, /= & %=
13. identifier 1 = identifier2 = - - - -= expressions.
14. Right to left
15. No