CHAPTER - 8

OPERATORS AND EXPRESSIONS IN C

OBJECTIVES OF THIS CHAPTER

8.1 INTRODUCTION

In the previous chapter, we have studied that variables and constants are used to store values in the program. To manipulate the data stored in these program elements, we have to use operators. In order to perform different types of operations on the data, C provides many types of operators. An operator shows that what type of operation will be performed on the data. C supports a rich set of built-in operators. We have already used some operators such as =, +, - etc. in the previous chapter. In this chapter, we are going to have a detailed discussion about various types of operators that are available in C. We shall also discuss the method of using these operators, their order of evaluation in expressions etc.

8.2 CONCEPT OF OPERATOR AND OPERAND

Operators are the symbols which are used to perform some specific type of operation on data. For example: + symbol is used to perform addition, * is used to perform multiplication, >= is used to perform comparison etc. Here +, * and >= are the operators to perform different types of operations. After performing the operations, all operators produce a value.

To perform any type of operation, we require Operands. **Operands** are the data items on which operators can perform operations. These operands can be either variables or constant values. Consider the following example:

```
a + 5 * 10
```

In this example, + and * are the operators which perform the operation on variable 'a' and constant values 5 and 10. Here the variable 'a' and constant values 5 and 10 are called the Operands.

8.3 EXPRESSION

An expression is like a formula in mathematics. An **expression** can be any valid combination of operators and operands. A valid combination is such a combination that confirms to the syntax rules of C language. A valid expression is also known as well-formed-expression. An expression after evaluation always returns a single value. This result can be used in the C programs. Expressions can be as simple as a single value and as complex as a large calculation. Consider the following examples:

```
x= 2.9;
```

It is a simple expressionwhere = operator is used with operands x and 2.9

```
x = 2.9 * y + 3.6 > z - ( 3.4 / z );
```

It is a somewhat complex expression which consists of many operators and operands. Here, =, *, +, >, - and / are the operators and x, 2.9, y, 3.6, 3.4 and z are the operands.

Now, consider the following combination of operators and operands which do not form a valid combination to be an expression:

```
x + y = z;
```

The above combination of operators and operands do not form a valid expression, although we are using valid operators and operands in the above example. But this combination of operators and operands do not follow the syntax rules of C language to be a valid expression. Left side of = operator must represent a valid memory location (identifier) to store value of z. Here, x+y cannot be a valid identifier, because a valid identifier cannot have special character other than underscore (as we learnt in the previous chapter).

All C expressions can be categorized into following two types:

8.3.1 Numerical Expressions

These expressions are used to perform numerical calculations. These expressions always return a numerical value after evaluating operators and operands. Consider the following examples:

```
4 + 3
3.2 - 7.8
```

After evaluation above numerical expression, a numerical value 7 and -4.6 will be produced.

8.3.2 Logical or Conditional Expressions

These expressions are used to perform logical or conditional operations. These expressions always return one of two possible values: either true (1) or false (0). Consider the following examples:

```
14 > 6
15 <= 6
```

After evaluating above conditional expressions, we receive true (1) for the first expression while false (0) for the second expression.

8.4 TYPES OF OPERATORS ACCORDING TO NUMBER OF OPERANDS

C providesa rich set of built-in operators. All these operators can be broadly divided into following three categories **according to number of operands** used by the operators:

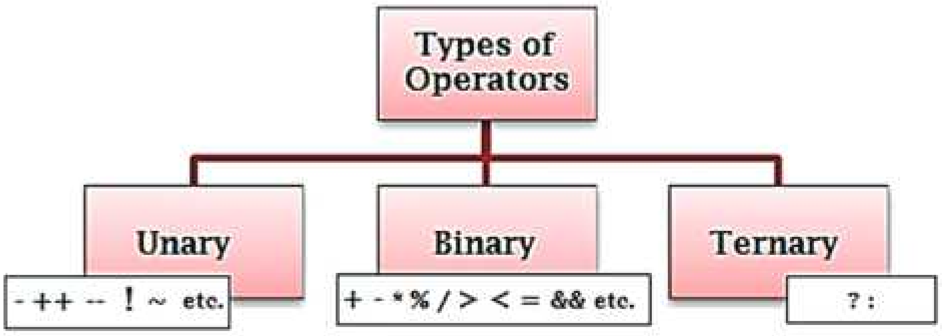


Fig. 8.1 Types of Operators according to number of operands

8.4.1 Unary operators

An operator which requires only one operand to perform its operation is called Unary operator. Common example of unary operator is unary minus. Any positive operand associated with unary minus gets its value changed. Consider the following example:

```
x= 10;
y =15;
z = x+(-y) ;
```

Here, z will be considered as z = 10+ (-15) which produce a result -5.

Since y is initially a positive integer variable, when operated by unary minus, gets its value changed. It will become negative. Some other examples of unary operators are: ++, --, ! and ~ operators.

8.4.2 Binary operators

An operator which requires two operands to perform its operation is calledBinary operator. Most of the operators in C language are of binary type. The syntax for using binary operators is given below:

Operand1 **Operator** *Operand2*

To use any binary operator, it must be put in between the operands. Consider the following examples:

```
a + b
a > b
a = b
```

In these examples, + > and = are the examples of binary operators which are placed in between two operands: a and b.

8.4.3 Ternary Operator

This operator is also known as Conditional Operator. An operator which requires three operands to perform its operation is calledternary operator. There is only one ternary operator in C. Ternary operator in C is represented using ? : symbols. The syntax for using this operator is given below:

exp1 ? exp2 : exp3;

Here, exp1 must be a conditional expression which produces a result either true (1) or false (0). If the value of exp1 is true then the exp2 will perform its function otherwise exp3 will perform its function. Consider the following example:

```
a=5;
b=10;
c = a > b  ?  a  :  b;
```

Here, the expression a>b will produce false (0) result (Operand1/exp1), therefore value of b (Operand3/exp3) will be stored in variable c. Variable a (Operand2/exp2) will not do any function because it will perform its function only if exp1 is true (1).

8.5 GENERAL CLASSIFICATION OF OPERATORS

All C operators can be generally categorized into following categories:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operators
6. Increment & Decrement Operators
7. Conditional Operators
8. Additional Operators

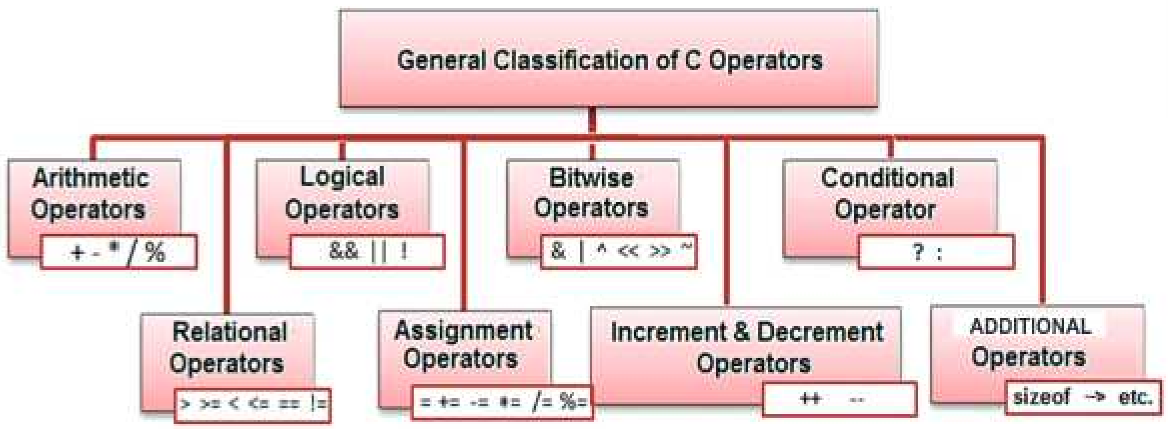


Fig. 8.2 General Classification of Operators in C

Bitwise operators are used for very low level operations i.e. for machine level programming or for performing bit level operations.

8.5.1 Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations such as: addition, subtraction, multiplication, division etc. There are five arithmetic operators in 'C'. All these operators are binary operators because all these operators require two operands to perform their operations. Following table shows the list and working of all these operators:

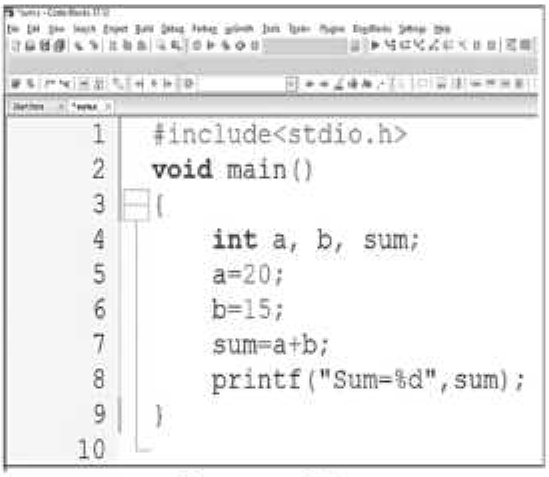
Name	Operator	Description	Examples	
Add	+	Used to perform Addition of numbers.	2+4 → 6	2.0+4.0 → 6.0
Subtract	-	Used to perform subtraction or used as any unary minus.	6-2 → 4	6.0-4.0 → 2.0
Multiply	*	Used to perform multiplication of numbers.	7*2 → 14	7.0*2.0 → 14.0
Divide	/	Used to perform division of numbers.	5/2 → 2 Integer division	5.0/2.0 → 2.5 Real Division
Modulus	%	Used to get remainder value after division of numbers.	7%4 → 3	5.0%2.0 Not allowed

Table: 8.1 Arithmetic Operators

For arithmetic operators, operands can be integer values, floating-point values or character values. 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 to be integers. Division of one integer 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 and one integer, the result will be a floating-point quotient.

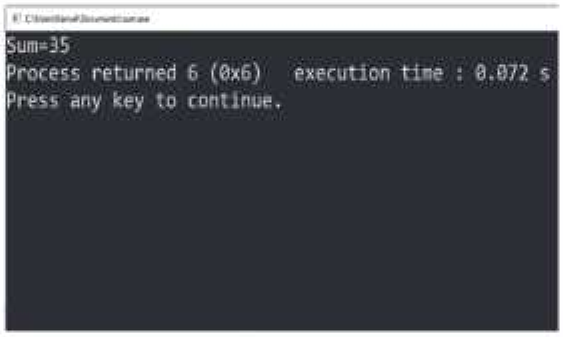
If one or both operands represent negative values, then the addition, subtraction, multiplication and division operations will result in values whose signs are determined by the usual rules of algebra. The interpretation of remainder operation is unclear when one of the operands is negative. Following programs show how to use arithmetic operators in C programming:

Program 8.1: Program in C to find sum of two numbers:



```c
#include<stdio.h>
void main()
{
    int a, b, sum;
    a=20;
    b=15;
    sum=a+b;
    printf("Sum=%d",sum);
}
```

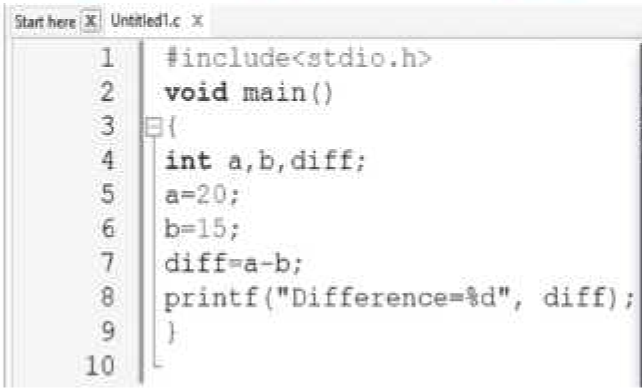
Program: 8.1



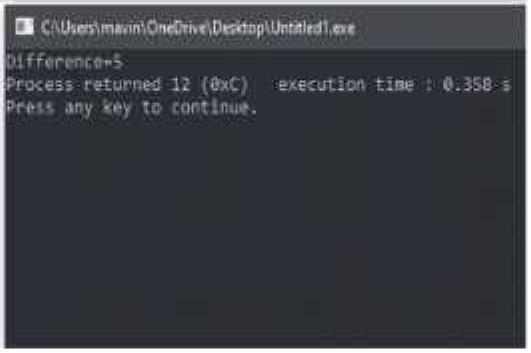
```
Sum=35
Process returned 6 (0x6)   execution time : 0.072 s
Press any key to continue.
```

Output of Program 8.1

Program 8.2: A program in C Language to find thedifference of two numbers



```c
#include<stdio.h>
void main()
{
int a,b,diff;
a=20;
b=15;
diff=a-b;
printf("Difference=%d", diff);
}
```

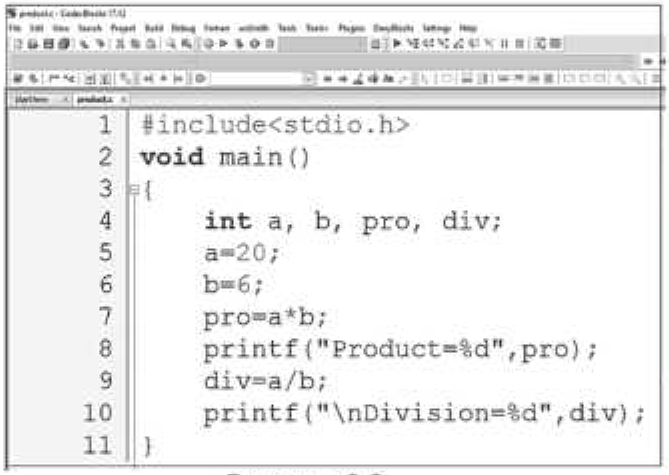


```
Difference=5
Process returned 12 (0xC)   execution time : 0.358 s
Press any key to continue.
```

Program: 8.2

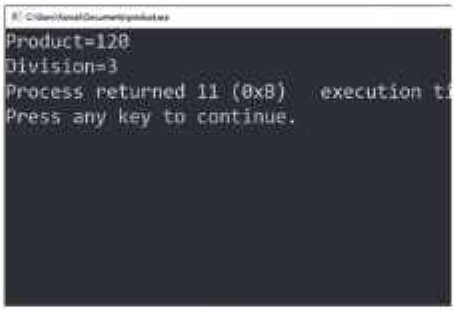
Output of Program 8.2

Program 8.3: A program in C Language to find product and division of two numbers:



```c
#include<stdio.h>
void main()
{
    int a, b, pro, div;
    a=20;
    b=6;
    pro=a*b;
    printf("Product=%d",pro);
    div=a/b;
    printf("\nDivision=%d",div);
}
```

Program 8.3



```
Product=120
Division=3
Process returned 11 (0xB)   execution ti
Press any key to continue.
```

Output of Program 8.3

8.5.2. Relational operators

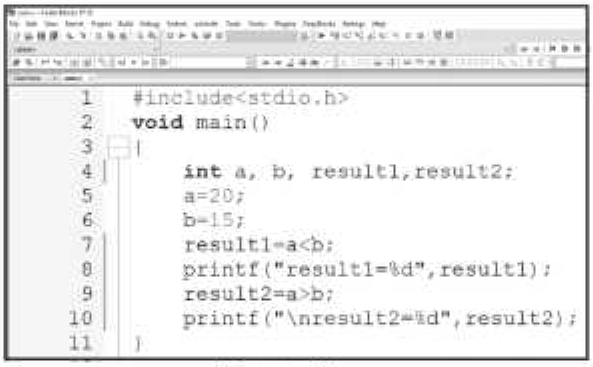
Relational operators are also called comparison operators. These operators are used to test the relationship between operands. In other words, these operators are used to compare values. After comparison, these operators return either true (1) or false (0) value. All the relational operators present in C language are of binary type. It means these operators require two operands to perform their operation. There are 6 relational operators in C which are given below in the table with examples:

Name	Operator	Description	Example	Result
Equals to	==	Used to check whether two values are equal	4==5 5==5	False True
Not Equal to	!=	Used to check whether two values are not equal	4! =5 4! =4	True False
Greater than	>	Used to check whether the first value is greater than second	4>5 5>4	False True
Less than	<	Used to check whether the first value is less than second	4<5 5<4	True False
Greater than or equal to	>=	Used to check whether first value is greater than or equal to second value	5>=5 6>=8 10>=5	True False True
Less than or equal to	<=	Used to check whether first value is lesser than or equal to second value	4<=5 4<=2 4<=4	True False True

Table 8.2 Relational Operators

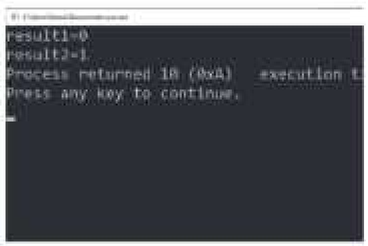
These relational operators are used to form logical expression representing condition. The resulting expression will be of type integer, since true is represented by the integer value 1 and false is represented by the value 0.

Program 8.4: A Program which shows the use of Relational Operators



```c
#include<stdio.h>
void main()
{
    int a, b, result1,result2;
    a=20;
    b=15;
    result1=a<b;
    printf("result1=%d",result1);
    result2=a>b;
    printf("\nresult2=%d",result2);
}
```

Program 8.4



```
result1=0
result2=1
Process returned 10 (0xA)   execution t
Press any key to continue.
```

Output of Program 8.4

In the program 8.4 in the line 7, the statement a<b returns 0 as the result is false because 20 < 15. Therefore, in line 8 it shows result1=0. Similarly, in line 9, the statement a>b returns 1 as the result is true because 20>15. Therefore, in line 10 it shows result2=1.

8.5.3 Logical operators

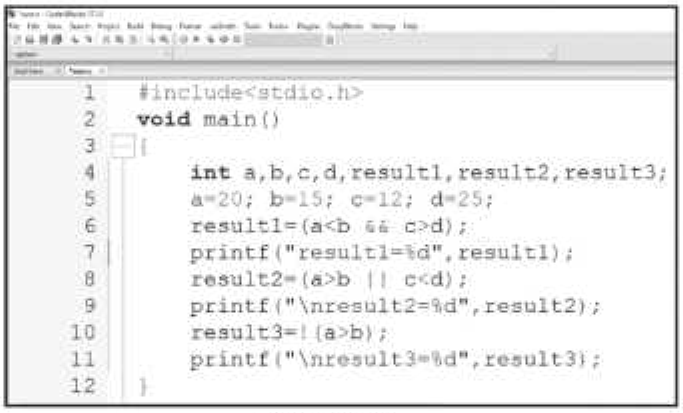
Logical operators are also called Boolean Operators. These operators are used to make compound relational expressions. In other words, we can say that these operators are used when we want to test more than one condition at a time. There are 3 Logical operators in C Language: 'Logical AND', 'Logical OR' and 'Logical NOT'. Here, 'Logical AND' and 'Logical OR' are the binary operators whereas 'Logical NOT' is unary operator. Two operands are required for 'Logical AND' and 'Logical OR' to perform their operations while one operand is required for 'Logical NOT' to perform its operation.

All Logical Operators also return either true or false value. The result of a logical AND operator will return true only if both operands are true, whereas the result of a logical OR operator will be true if either operand is true or if both operands are true. Logical NOT operator returns true only when its operand is false. Following table shows the list and working of all Logical Operators used in C language with suitable examples:

Name	Operator	Description	Example	Explanation and Result
AND	&&	Returns true only if both operands are true otherwise it returns false	3>5 && 4>5 3>5 && 4<5 3<5 && 4>5 3<5 && 4<5	False && False → False False && True → False True && False → False True && True → True
OR	||	Returns true if at least one of its operand is true otherwise it returns false	3>5 || 4>5 3>5 || 4<5 3<5 || 4>5 3<5 || 4<5	False || False → False False || True → True True || False → True True || True → True
NOT	!	Returns true only when its operand is false otherwise it returns false	!(3<5) !(3>5)	!(True) → False !(False) → True

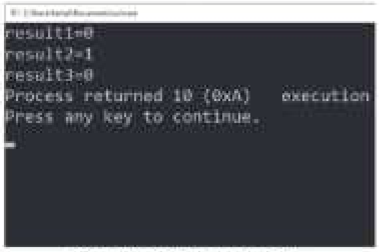
Table 8.3 Logical Operators

Program 8.5 A program in C Language to show the working of logical operators:



```
1   #include<stdio.h>
2   void main()
3   {
4       int a,b,c,d,result1,result2,result3;
5       a=20; b=15; c=12; d=25;
6       result1=(a<b && c>d);
7       printf("result1=%d",result1);
8       result2=(a>b || c<d);
9       printf("\nresult2=%d",result2);
10      result3=!(a>b);
11      printf("\nresult3=%d",result3);
12  }
```

Program 8.5



```
result1=0
result2=1
result3=0
Process returned 10 (0xA)   execution
Press any key to continue.
```

Output of Program 8.5

In the above program:

In line 6 → **result1 = false && false** and it will produce **result1= false (0)**

In line 8 → **result2 = true || true** and it will produce **result2=true (1)**

In line 10 → **result3 = !(true)** and it will produce **result3=false (0)**

8.5.4 Assignment operators

These Operators are used to assign or store values in variables etc. The symbol of assignment operator is =. Consider the following examples which show how to use assignment operator in C programs:

a = - 2;	// assigns -ve value (-2) to the variable.
b = 5;	// assigns value (5) to the variable.
c = a + b;	// assigns the result of expression to the variable.
a = a + 10;	// self-assignment of a variable.

Assignment operators can also be used as shorthand operators. Shorthand assignment operators are useful in self-assignment statements. Following table shows the examples of shorthand assignment operators used in C:

Let's assume   int a=5;

Shorthand Operator	Example for Shorthand Assignment	Equivalent Self-Assignment	Result
+=	a+ =2	a = a + 2	a=7
- =	a- =2	a = a - 2	a=3
*=	a* =2	a = a * 2	a=10
/=	a / =2	a = a / 2	a=2
%=	a%=2	a = a % 2	a=1

Table 8.4: Shorthand Assignment Operators

Assignment operator = and the equality (equal to) operator == both are different types of operators. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two operands 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. For example:

```
int a=3.5;
```

Here, float value 3.5 will automatically be converted to type integer, i.e. value of variable a will become 3

8.5.5 Bitwise Operators

Bitwise operators are used for very low level operations, i.e. for machine level programming or for performing bit level operations. C provides the following operators for handling bitwise operations:

1. << Bit shift left (a specified number or bit positions)
2. >> Bit shift right(a specified number of bit positions)
3. | Bitwise OR
4. ^ Bitwise XOR
5. & Bitwise AND
6. ~ Bitwise one's complement

8.5.6 Increment and Decrement Operators

These are the unary operators. The symbol ++ is used for increment operator while symbol - - used for decrement operator. The increment operator causes its operand to increase by one whereas the decrement operator causes its operand to decrease by one. The operand used with each of these operators must be a single variable. These operators cannot be applied directly on the constant values.

For example:

```
int x=10;     (x is an integer variable that has been assigned a value of 10)
```

Following expression causes the value of x to be increased to 11

```
++x;          (which is equivalent to x= x+1)
```

Similarly, following expressioncauses the original value of x to be decreased to 9:

```
--x,          (which is equivalent to x=x-1)
```

If we use 10++ or --10, it will be a wrong statement as we have already studied that these cannot be applied directly on the constant values.

The increment and decrement operators can each be utilized in two different ways. It depends on whether the operator is written before or after the operand:

- Prefix increment and decrement
- Postfix increment and decrement

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. This is called **pre increment/decrement.** If, however the operator follows the operand then the value of the operand will be changed after it is used. This is called post increment/ decrement.

For example:

If the value of x is initially 10, it can be increased by two methods:

```
y = ++x;            (pre increment)
```

Here,at first, the value of x will be incremented to 11 and then this incremented value of x will be assigned to variable y, i.e. y will also get value 11 (i.e. x=11 and y=11)

```
y = x++;            (post increment)
```

Here, at first, the value of x will be assigned to variable y (i.e. y will get value 10) then value of x will be incremented to 11 (i.e. y=10 and x=11)

Similarly, decrement operator can be used. For example:

```
y = --x;            (pre decrement)
```

Here,first of all, the value of x will be decremented to 9 and then this decremented value of x will be assigned to variable y, i.e. y will also get value 9 (i.e. x=9 and y=9)

```
y = x--;            (post decrement)
```

Here, first of all, the value of x will be assigned to variable y (i.e. y will get value 10) then value of x will be decremented to 9 (i.e. y=10 and x=9)

8.5.7 Conditional Operator (? :)

It is a ternary operator. There is one and only one ternary operator (? :) in 'C' language. An expression that makes use of the conditional operator is called a conditional expression. This operator has already been defined in the section 8.4.3 of this chapter. Please refer to the specified section for more details of this operator.

8.5.8 Additional Operators:

All the remaining operators that do not come under any above mentioned categories of operators, can be considered as additional operators. Examples of such operators are: sizeof( ) operator, pointer operators, member selection operators etc.

sizeof operator:

It is another unary 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 variable or data type. Consider the following examples:

```
sizeof (x);
sizeof (float);
```

In first example, if x is of char type variable then it returns the result 1, while in example second, we pass keyword for data type float which returns 4 as float data type occupies four bytes in memory.

8.6 TYPE CONVERSION

The value of an expression can be converted to a different data type in C, if desired.When value of one type is converted into some other type, it is called Type Conversion.Operands that differ in type may undergo type conversion before the expression takes on its final value. There are two ways of type conversions in C:

1. Implicit Conversion
2. Explicit Conversion

8.6.1 Implicit Conversion

This type of conversion is automatic. For this type of conversion, we use assignment (=) operator. This type of conversion is used when operand having lower data type is converted into higher data type. There is no loss of information in this type of conversion. Consider the following example for automatic conversion:

```
float n;
n = 5/2;
```

In this example, implicit conversion takes place, as integer type data, after integer division of 5/2 will produce result in the form of integer value, i.e. 2 (but not 2.5). This result will automatically be converted into float type value so that it can be stored in the float variable i.e. value of variable n will become 2.000000

8.6.2 Explicit Conversion

This is forceful conversion. For this type of conversion, we use cast operator. There may or may not be any loss of information in this type of conversion. This type of conversion is used when operand of higher data type is converted into lower data type.

The syntax for this type of casting is:

(Desired data type) Expression

The name of data type into which the conversion is to be made is enclosed in parentheses and placed directly to the left of the value to be converted. The example of type casting is as follows:

For example:

```
float n;
n=(float)5/2;
```

The cast operator converts the value of 5 to its equivalent float representation (i.e. 5.0) before the division by 2. Therefore, it will become float division and the result of division will be 2.500000 which will be stored in the float variable n and hence the value of n will become 2.500000. The cast operator can be used on a constant or expression as well as on a variable.

8.7 PRECEDENCE/ HIERARCHY OF OPERATORS

The operators within C are grouped hierarchically according to their order of evaluation, known as **precedence**. Operators with a higher precedence are carried out before operators having a lower precedence. In simple words, the sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators**. The natural order can be altered by making use of parentheses.

Precedence of commonly used operators in decreasing order is as follows:

1.	-	++	- - Operators	! ~	sizeof	(datatype)
2.	*	/	%			
3.	+	-				
4.	<	<=	>	>=		
5.	==	!=				
6.	&&					
7.	||					
8.	? :					
9.	=					

Consider the following example which illustrates how arithmetic expressions are evaluated using operators precedence:

```
a = 5 * 4 / 4 + 8 - 9 / 3;          (* is evaluated)
a = 20 / 4 + 8 - 9 / 3;             (/ is evaluated)
a = 5 + 8 - 9 / 3;                  (/ is evaluated)
a = 5 + 8 - 3;                      (+ is evaluated)
a = 13 - 3;                         (- is evaluated)
a = 10;                             result of expression
```

Points To Remember

1. **Operators** are the symbols which are used to perform some specific type of operation on data.
2. **Operands** are the data items on which operators can perform operations.
3. An **Expression** is like a formula in mathematics which can be any valid combination of operators and operands.
4. An operator which requires only one operand to perform its operation is calledUnary operator.
5. An operator which requires two operands to perform its operation is calledBinary operator.
6. Ternary operator is also known as **Conditional Operator** which requires three operands to perform its operation.

7. '%' operator is also known as **modulus** operator which works only on integer operands.

8. **Relational** operators are symbols that are used to test the relationship between two variables..

9. There are three **Logical** operators in C language, they are **and, or, not.** They are represented by &&, || and ! respectively

10. **Assignment operators** in C assign the value of an expression to an identifier and the most commonly used assignment operator is =.

11. The **increment** operator (++) causes its operand to be increased by one.

12. The **decrement** operator (- -) causes its operand to be decreased by one.

13. When value of one type is converted into some other type, it is called **Type Conversion.**

14. The sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators.**

EXERCISE

Part-A

1. Multiple Choice Questions

I. The symbols which are used to perform some specific type of operation on data are called?

a. Operands b. Operators
c. Expressions d. Formulas

II. Which operator acts only on one operand?

a. Unary b. Binary
c. Ternary d. Conditional

III. Which of the following is not a Logical Operator?

a. And (&&) b. OR (||)
c. Equality (==) d. NOT (!)

IV. Which symbol is used for Ternary Operator?

a. : ? b. ; ?
c. ? : d. ? ;

V. Which of the following cannot be considered as assignment operator?

a. = b. ==
c. += d. %=

2. Fill in the Blanks:

I. ___________are the data items on which operators can perform operations.

II. Unary operator acts on only _____________ operand.

III. __________ arithmetic operator performs only on integer operand.

IV. When value of one type is converted into some other type, it is called _________.

V. Ternary operator is also known as _______________.

3. **Very Short Answer Type Questions**

I. ++Operator causes its operand to be increased by one.

II. Arithmetic operators are used to test the relationship between two variables.

III. Number of Arithmetic Operators used in C programming are six.

IV. Size of () operator returns the size of its operand, in bytes.

V. Type conversion is of two types.

VI. There six relational operators are present in C Language.

Part-B

4. **Short Answer Type Questions. (Write the answers in 4-5 lines)**

I. Define Expression?

II. What is Operand?

III. What is Unary operator?

IV. Define Conditional operator?

V. What is Type Conversion?

VI. What is an operator? Write the name of different types of operators?

VII. Write about increment and decrement operators?

Part-C

5. **Long Answer Type Questions. (Write the answers in 10-15 lines)**

I. Explain the Arithmetic Operators? Write any program using Arithmetic Operators?

II. What are Relational operators? Write any program of Relational operator?

Lab Activity

- Write a C program to show the usage of Arithmetic operators
- Write a C program to show the usage of conditional (ternary) operator
- Write the C Programs to solve the following mathematical formulas:
  - z=10(5+1)%2
  - x=(5+7)÷(8+9*2)