UNIT 4
OPERATORS AND EXPRESSION

LH – 4HRS

PRESENTED BY:

ER. SHARAT MAHARJAN

C PROGRAMMING

CONTENTS (LH – 4HRS)

Operator:

- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

For example, consider the below statement:

c=a+b;

- Here, ‘+’ is the operator known as the *addition operator* and ‘a’ and ‘b’ are operands. The addition operator tells the compiler to add both of the operands ‘a’ and ‘b’.

Operator Classification:

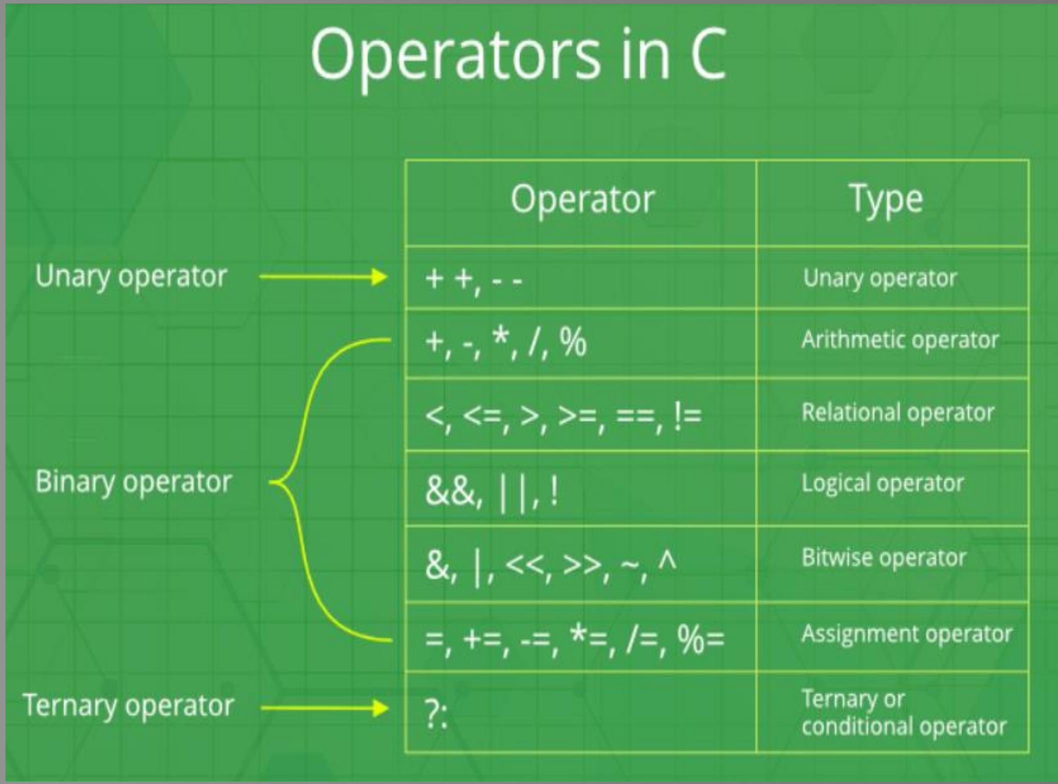
a. According to Number of Operands

- **Unary Operators**: The operators which require only one operand to operate are called unary operators. E.g. ++(increment operator) and --(decrement operator) are unary operators.

- **Binary Operators**: The operators which require two operands to operate are called binary operators. E.g. : +(plus), -(minus), *(multiply), /(division), <(less than), >(greater than), etc are binary operators.
- **Ternary Operators**: The operators which require three operands to operate are called ternary operators. E.g. the operator pair “?:” is a ternary operator in C.

b. **<u>According to Utility and Action:</u>**

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Increment and Decrement Operators
6. Conditional Operators (Ternary Operato
7. Bitwise Operators
8. Special Operators (Comma Operator and size of Operator)

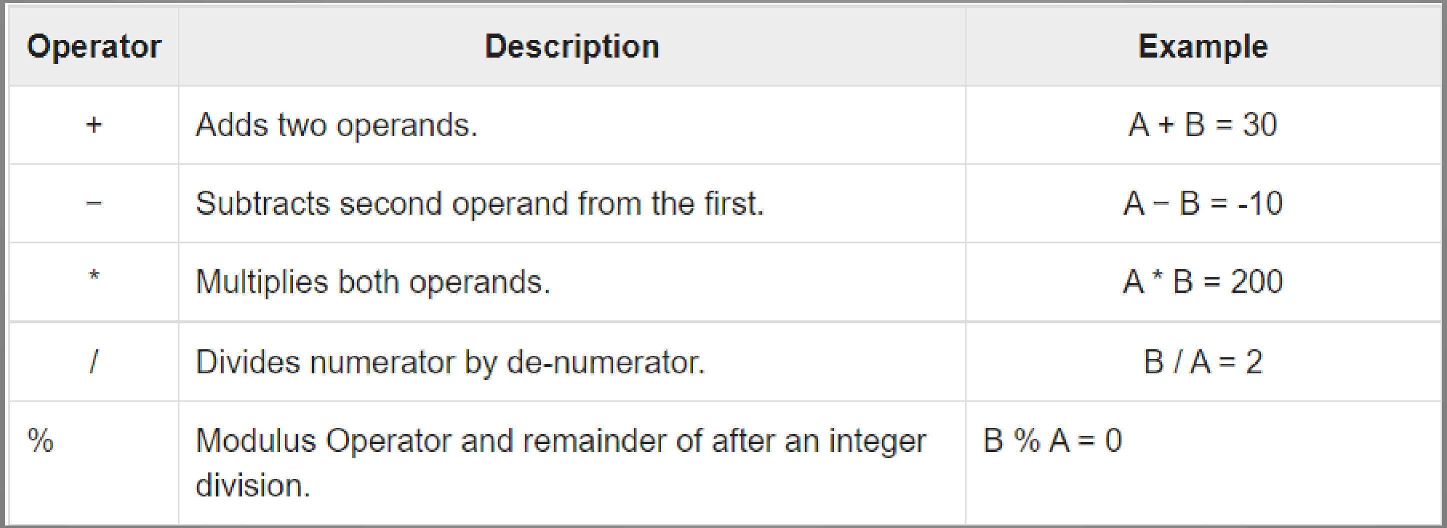


Operators in C

	Operator	Type
Unary operator	++, --	Unary operator
Binary operator	+, -, *, /, %	Arithmetic operator
	<, <=, >, >=, ==, !=	Relational operator
	&&, ||, !	Logical operator
	&, |, <<, >>, ~, ^	Bitwise operator
	=, +=, -=, *=, /=, %=	Assignment operator
Ternary operator	?:	Ternary or conditional operator

4.1 Arithmetic operator

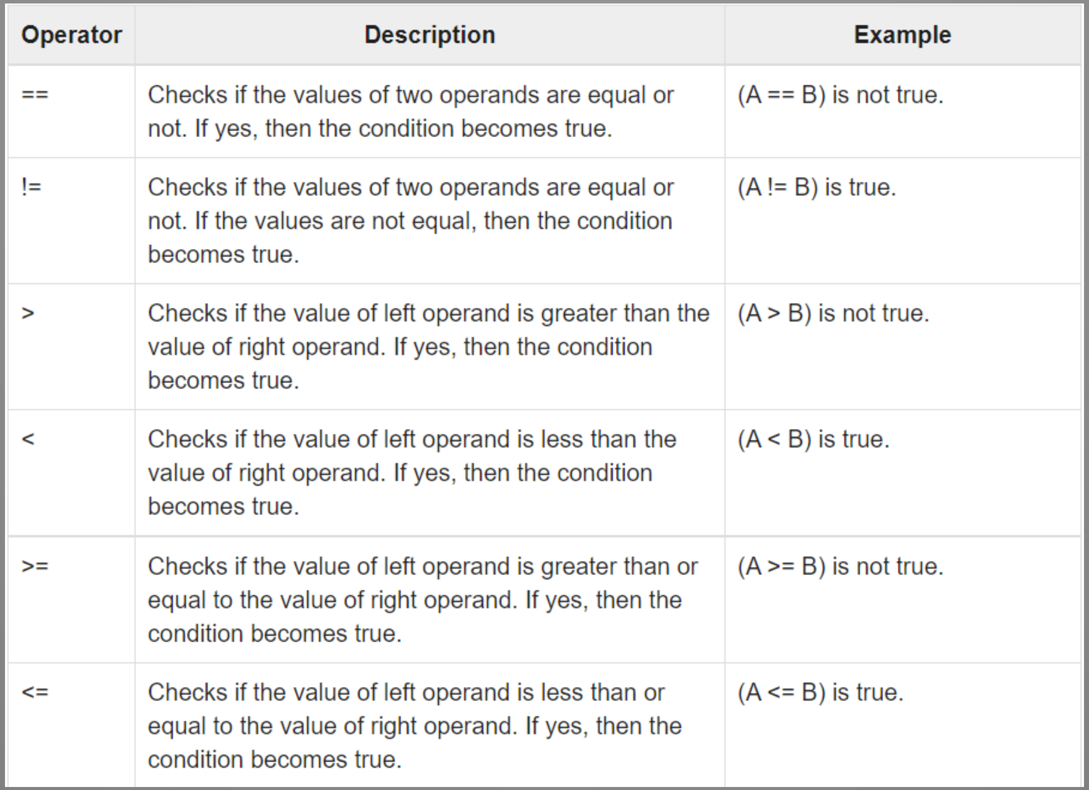
- These are the operators used to perform arithmetic/mathematical operations on operands.
- The following table shows all the arithmetic operators supported by the C language. Assuming variable **A** holds 10 and variable **B** holds 20 then –



Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0

4.2 Relational operator

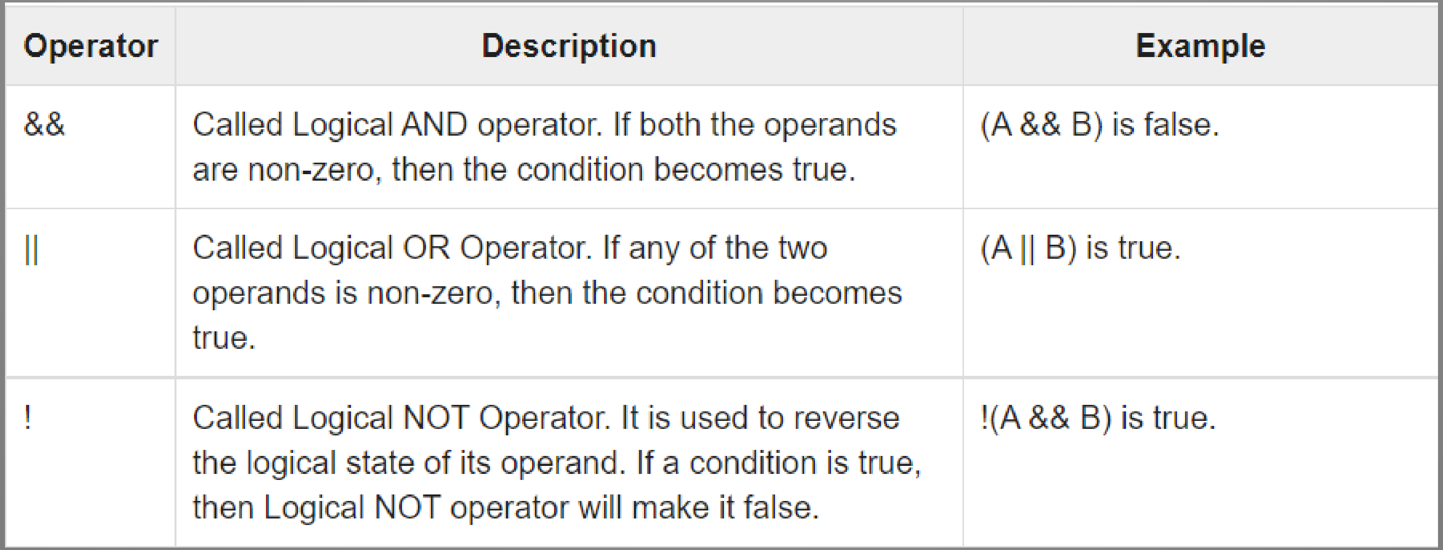
- These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, an operand is greater than the other operand or not, etc.
- The following table shows all the relational operators supported by C. Assuming variable **A** holds 10 and variable **B** holds 20 then −



Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

4.3 Logical operator

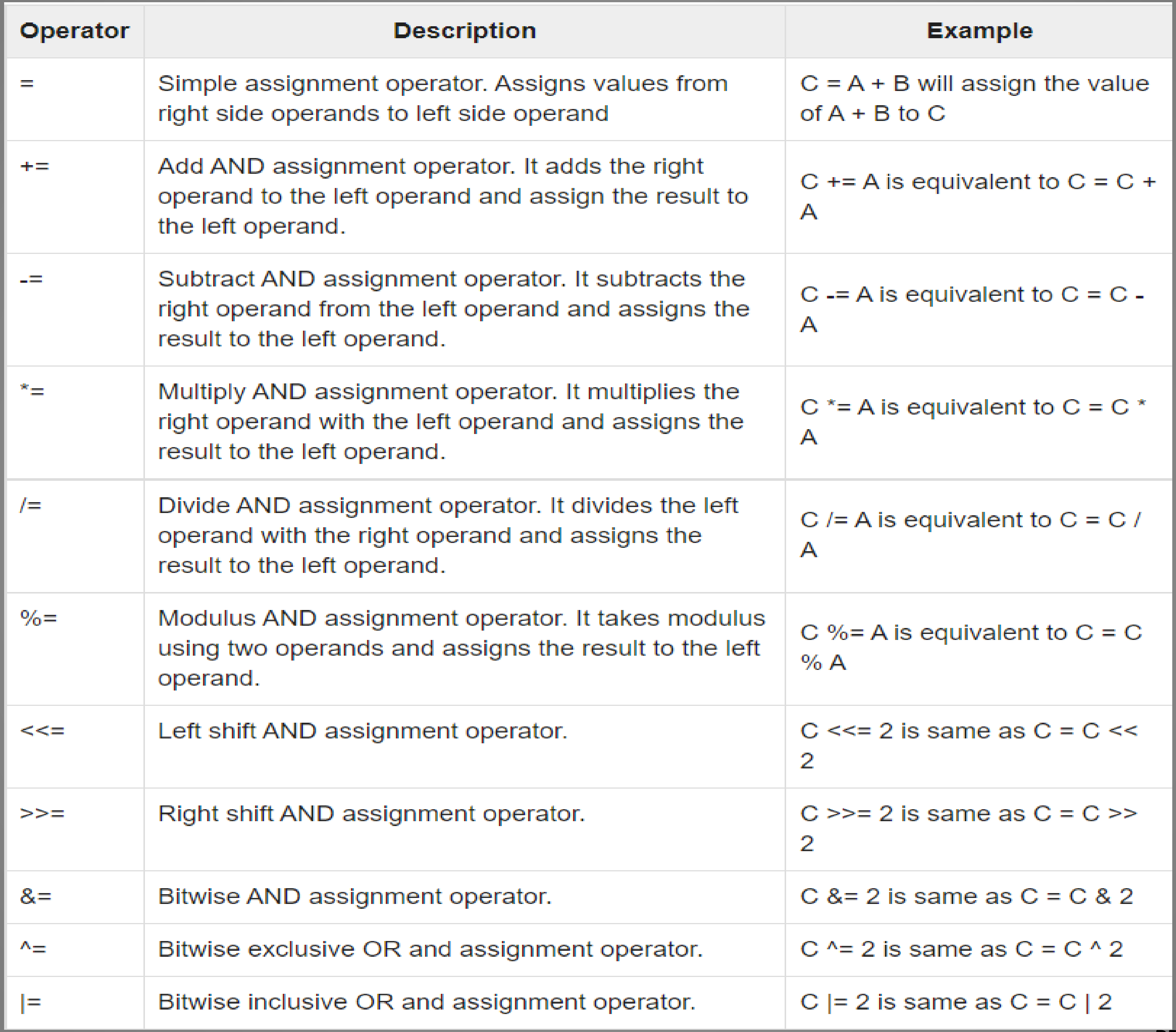
- Logical Operators are used to combine two or more conditions or to complement the evaluation of the original condition in consideration.
- For example, the **logical AND** represented as **‘&&’ operator in C or C++** returns true when both the conditions under consideration are satisfied. Otherwise, it returns false.
- Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −



Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

4.4 Assignment operator

- Assignment operators are used to **assign value to a variable**.
- The **left side** operand of the assignment operator is a **variable** and the **right side** operand of the assignment operator is a **value**.
- The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.
- The following table lists the assignment operators supported by the C language −



Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

4.5 Operator increment/decrement

- C has increment and decrement operators: ++ and --.
- The operator **++ adds 1** to the operand while **– subtracts 1**. Both are unary operator and takes the form: **prefix (++x)** or **postfix(x--).** The **x++ is same as x=x+1** (or x+=1).
- While ++x and x++ means the same things when they form statement independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.
- Consider an example: **x=10; y=++x; in this case the values of x and y will be 11. Suppose if we write the statement as x=10; y=x++; in this case the values of x is 11 and y is 10.**

4.6 Conditional operator

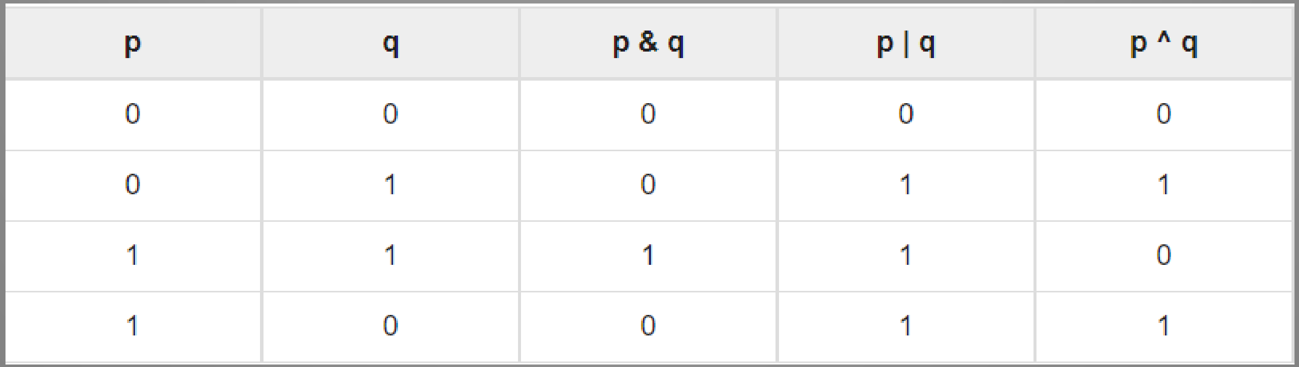
- It is also known as ternary operator. A ternary operator pair “?:” is available in C to construct conditional expressions of the form: **exp1?exp2:exp3**; where exp1, exp2 and exp3 are expressions.
- First evaluate exp1, if exp1 is true then execute exp2. If exp1 is false then execute exp3. Note that only one of the expressions (either exp2 or exp3) is executed.

Consider the following statement

- **int x=5,y=6,z;**
- **z=(x>y)? x : y; in this example, z will be assigned the value of y because the value of y is larger than value of x. This operator is same as the function of if else statement.**

4.7 Bitwise operator

- The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands.
- Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –



p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

A = 0011 1100

B = 0000 1101

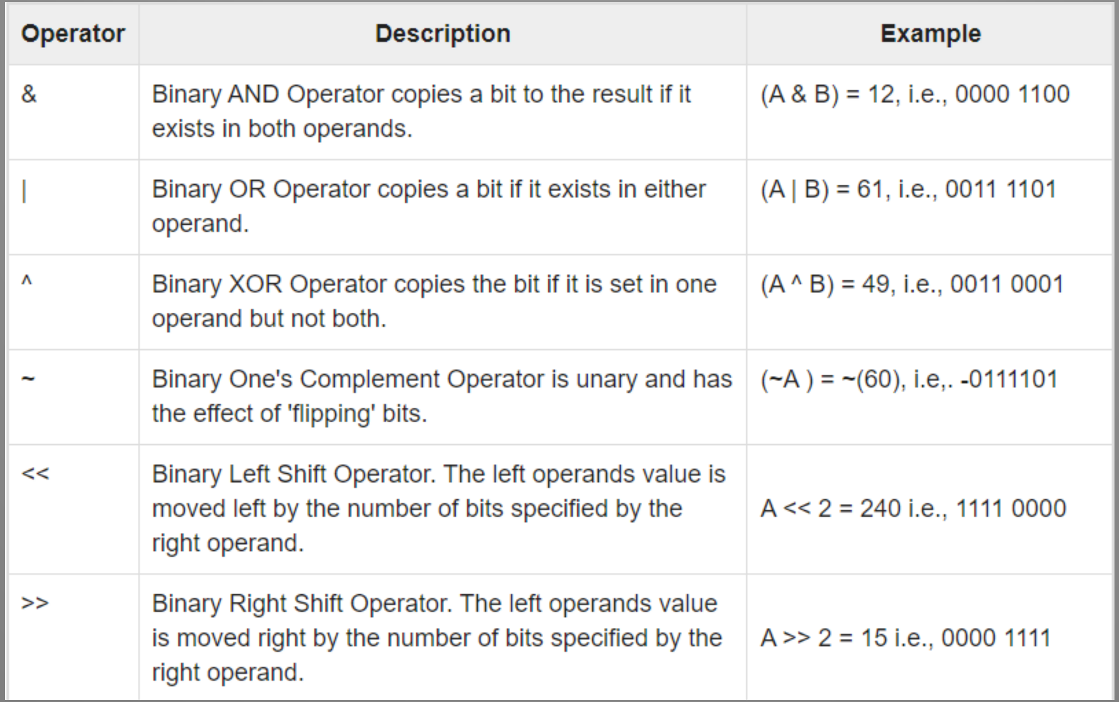
A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. **Assume variable 'A' holds 60 and variable 'B' holds 13, then –**



Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

4.8 Comma operator

- It is also special type of operator used in C.
- It is used to separate the identifiers/expressions, variable declaration etc.

For example:

```
int p,q,r;
int a=2,b=5;
```

4.9 Sizeof Operator

- This is unary operator which finds out number of bytes that any object occupies in computer's memory. It looks like a function but really an operator.
- The syntax of sizeof operator is

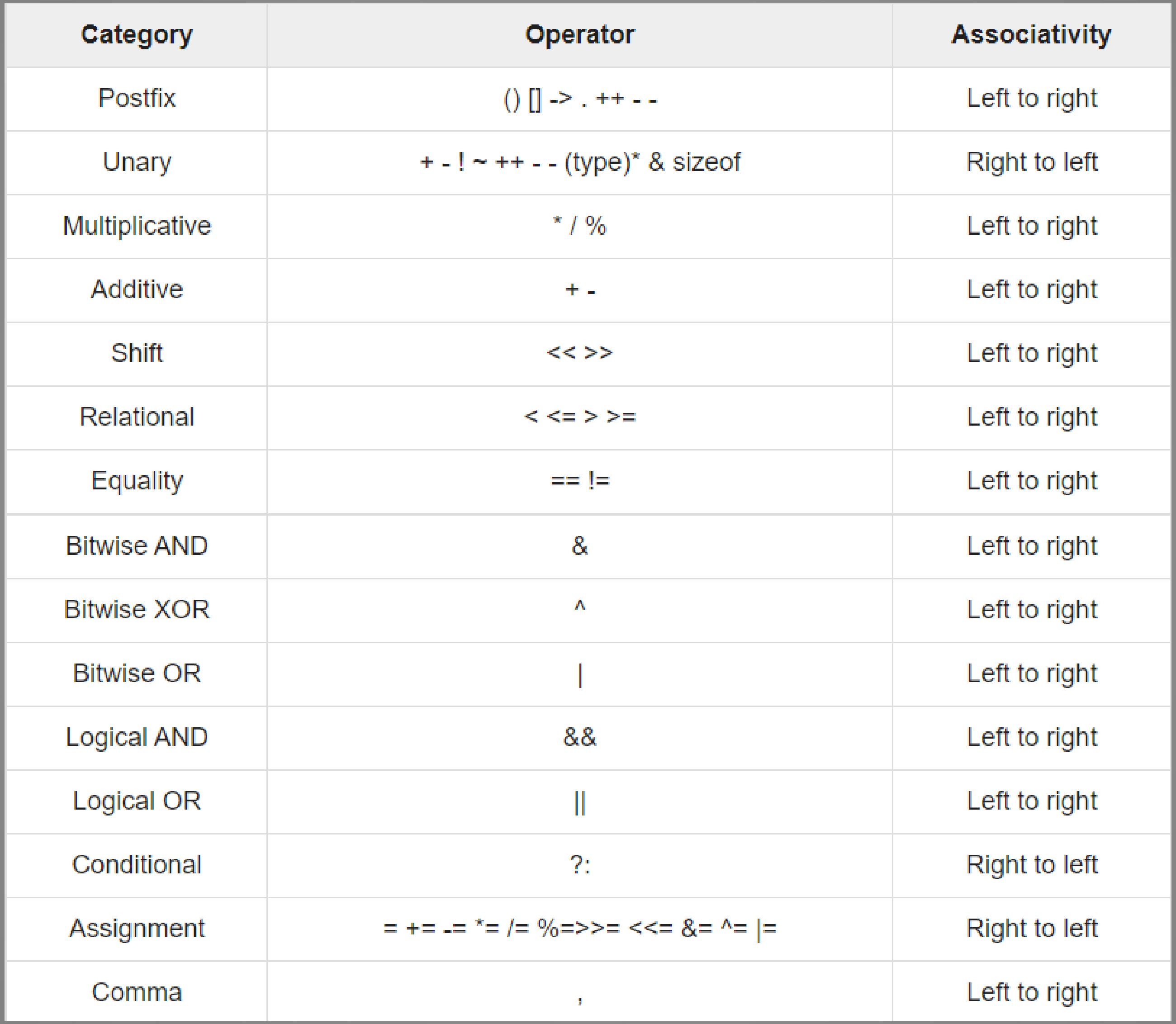
  sizeof(object)
- For example:

  int x;

  then sizeof(x) will return 4.

4.10 Operator Precedence and Associativity

- 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.
- Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

4.11 Expressions and its Evaluation

- Let us consider of evaluating expression z=(x/(y-z)+p)*q.
- First evaluate (y-z) then this difference will divide to x and result is added to p. Finally the total result is multiplied by q.

Example: Evaluate the following expression

i=2*3/4+4/4+8-2+5/8

Solution:

Stepwise evaluation of above expression

i=2*3/4+4/4+8-2+5/8	
i=6/4+4/4+8-2+5/8	operator * is evaluated
i=1+4/4+8-2+5/8	operator / is evaluated
i=1+1+8-2+5/8	operator / is evaluated
i=1+1+8-2+0	operator / is evaluated
i=2+8-2+0	operator + is evaluated
i=10-2+0	operator + is evaluated
i=8+0	operator - is evaluated
i=8	operator + is evaluated

4.12 Type Casting in Expression, Program Statement

- Typecasting is converting one data type into another one. It is also called as data conversion or type conversion in C language. It is one of the important concepts introduced in ‘C’ programming.
- ‘C’ programming provides two types of type casting operations:

1. **Implicit type casting (Automatic type conversion)**
2. **Explicit type casting (Type cast)**

1. Implicit type conversion

- In this, the lower 'type' is automatically converted to the 'higher' type before the operation proceeds and the result is of the 'higher' type.
- This type of conversion is automatically done by compiler at the time of compilation.
- The following rule will be used to convert data types automatically.

char->short->int->long->float->double->long double

Example:

```
#include <stdio.h>
int main() {
  int  num = 13;
  char c = 'k'; /* ASCII value is 107 */       OUTPUT:  sum = 120.000000
  float sum;
  sum = num + c;
  printf("sum = %f\n", sum );}
```

2. Explicit type conversion

- Sometimes, a programmer needs to convert a value from one type to another in a situation where the compiler will not do it automatically. It is done by programmer as per need to convert one type to another is called explicit type conversion.

Syntax: **(data type) expression;**

- For example: p is of int type and it is needed to be converted into float type then

```
int p;
(float)p; //p will be converted into float type
```

- The value of p will remain unchanged but type will be changed. **For example if value of p=10 then after cast its value will be 10.0.**

THANK YOU FOR YOUR ATTENTION