Unit IV- Part 02
(Operators in C)

Lecture Slide

AS2023

In pursuit of preparing tomorrow's technologists

Objectives

By the end of this session, students will be able to:

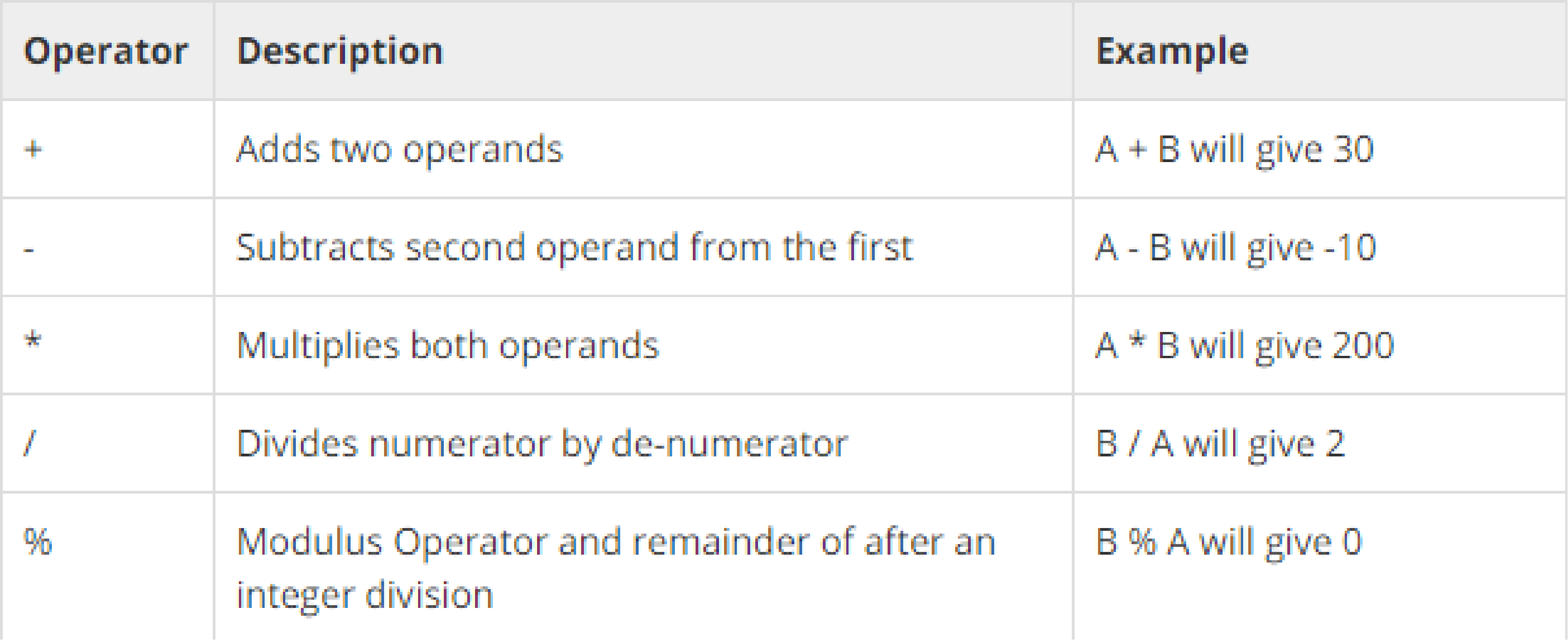
- Explain the different operators
- Appropriately choose the operators for problem solving
- Demonstrate conditional operator
- Understand the precedence of those operators

Operators

- Operators are the symbols that tells the compiler to perform specific mathematical or logical manipulations.
- Used in program to manipulate the data and variables
- Usually a part of the mathematical or logical expressions
- C operators can be classified into following types
  - Arithmetic Operators
  - Relational operators
  - Logical operators
  - Assignment operators
  - Increment operators
  - Conditional operators
  - Bitwise operators
  - Special operators

Arithmetic Operators

- Assume variable A & B having values 10 & 20 respectively



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

- **Note:** Here A & B are operands

Arithmetic Operators Cont..

- Integer division truncates any fractional part & modulo (%) operation produces the remainder of the integer division
- **Integer arithmetic** always yields integer value

  Ex. 5+5=10
- **Real arithmetic** yields real number

  Ex. 1.0/3.0=0.333333
- % operator cannot be used with real number
- real number is produced by **mixed-mode Arithmetic**

  Ex. 15/10.0=1.5

Arithmetic Expression

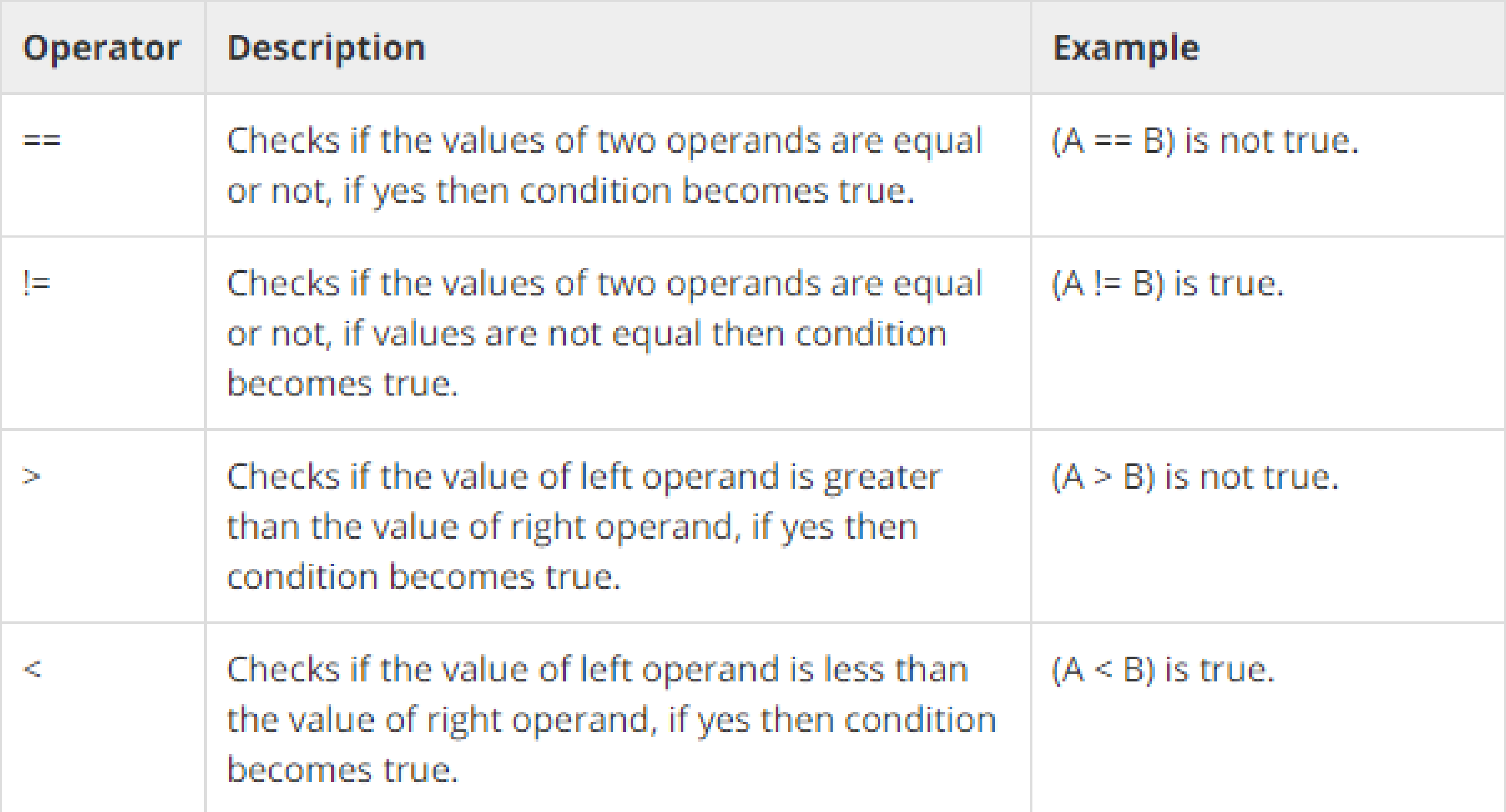
- Combination of variables, constants and operators arranged as per the syntax of the language
- Every algebraic expression has to converted to C expression

  For ex. (xy/z) can be expressed as $x*y/z$
- Expression are evaluated using an assignment statement of the form

  Variable = expression;
- An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence of operators as shown below
  - High priority → * / %
  - Lowest priority→ + -
- Example: x=a-b/3+c*2-1=10 where a=9, b=12 and c=3

Relational Operators

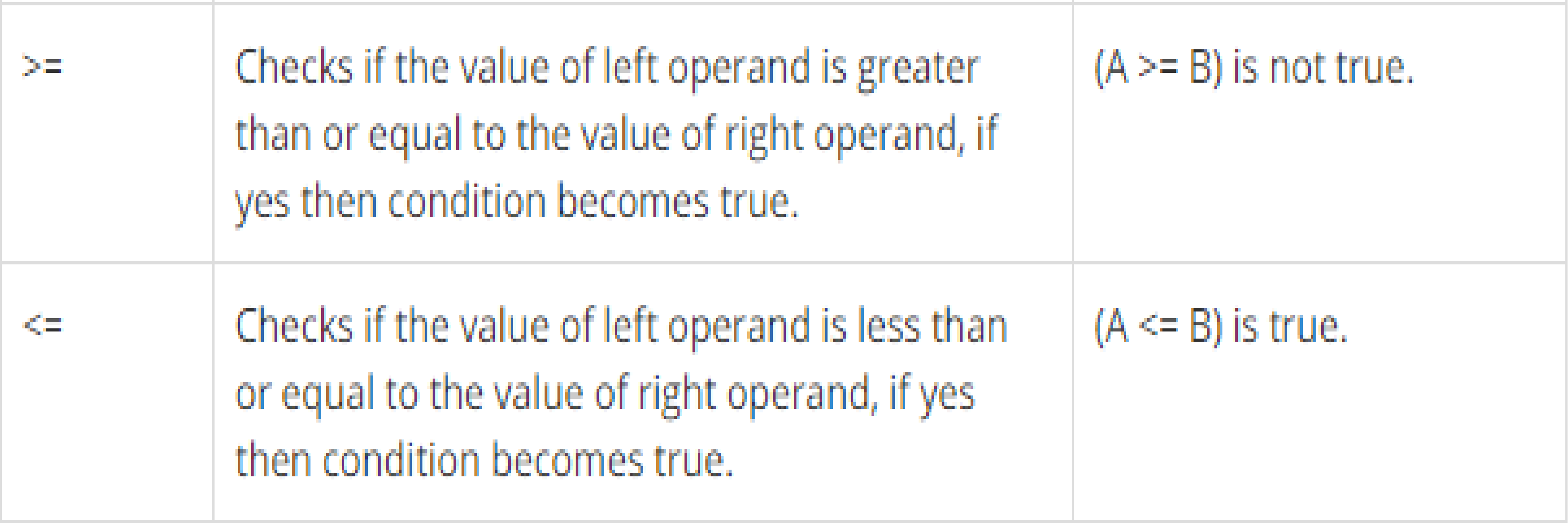
- Assume A holds 10 & B holds 20



Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then 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 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 condition becomes true.	(A < B) is true.

Relational Operators

- Assume A holds 10 & B holds 20



>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then 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 condition becomes true.	(A <= B) is true.

Relational Operators

- Expression containing relational operator is called relational expression
- The value of relation operation is either **0** or **1**
- operators are only supported in the form shown in previous slide

  Ex. =< & => is an invalid operator
- The operands of a relational operators must evaluate to a number
- Characters are valid operand
- It should not be used for comparing strings

  Ex. “hello” & “bye”
- It used in decision statement like **if & while** statement

Relative Precedence

- Relative precedence of relational & logical operators is as follows

Highest	!
	> >= < <=
	== !=
	&&
lowest	||

- It is important to remember this when we use these operators in compound expressions

Assignment Operators

- It is used to assigned the result of an expression to a variable
- “ = ” is an assignment operator

  Ex. A=10; c=a+b;
- C has a set of shorthand assignment operator which takes the following form

  V OP = exp;

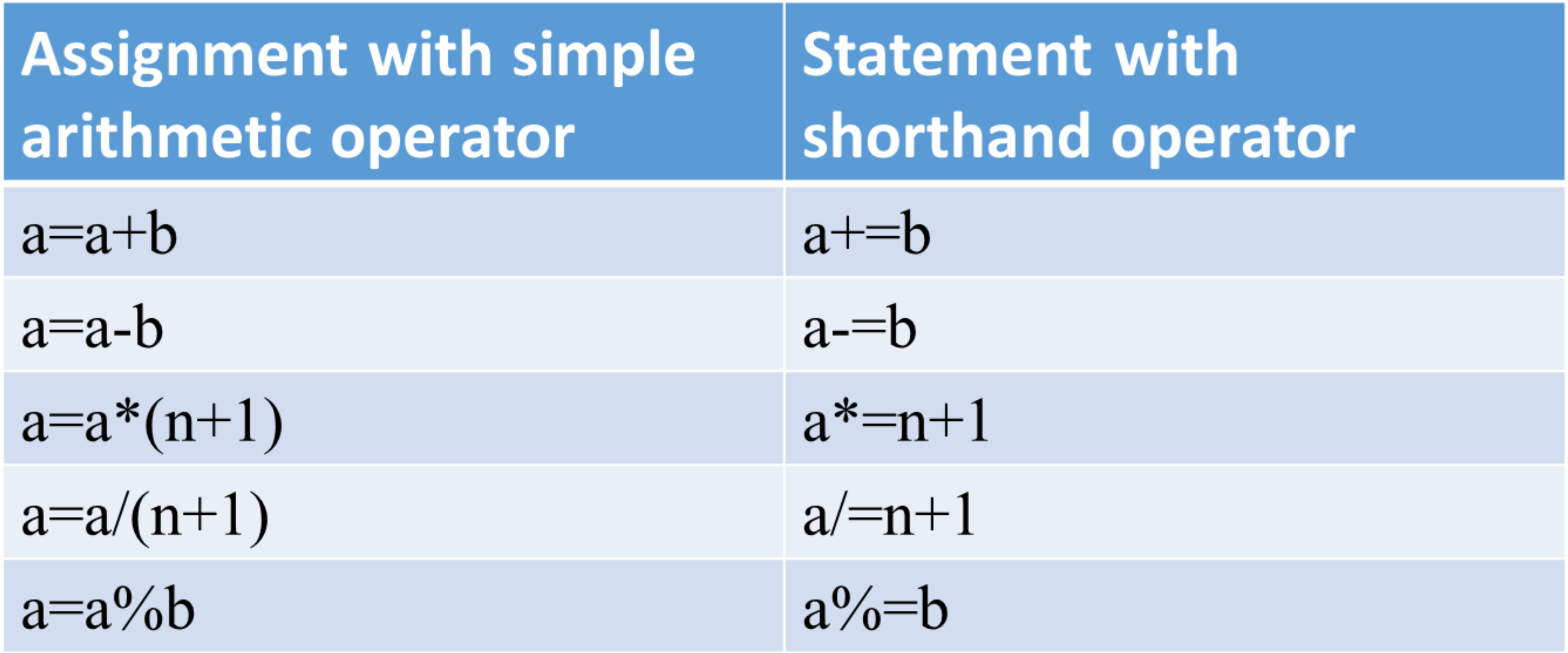
  Where V is a variable, exp is an expression & OP is c binary arithmetic operator

- the above shorthand assignment operator is equivalent to **v=v+ (exp);**

Example: x+=y+1; which means x=x+(y+1);

Assignment Operators

- Commonly used shorthand operator



Assignment with simple arithmetic operator	Statement with shorthand operator
a=a+b	a+=b
a=a-b	a-=b
a=a*(n+1)	a*=n+1
a=a/(n+1)	a/=n+1
a=a%b	a%=b

Increment & Decrement Operators

- C allows two useful operators not generally found in other languages
- These are increment (++) and decrement (--) operators
- The ++ adds 1 to the operand while -- subtracts 1
- Both are unary operator and takes the following form

  ++m; or m++;

  --m; or m--
- Used in looping statements like for and while
- pre increment or decrement and post-increment or decrement means the same thing when statement are formed, but they behave differently when they are used in the expression

Increment & Decrement Operators

- ❑When postfix ++ or -- is used with a variable in an expression , the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one

  Ex. M = 5;

  Y = M++;

  Y=5, M=6

- ❑When prefix ++ or – is used in an expression, the variable is increment (decremented) first and then expression is evaluated using the new value of the variable

  Ex. M = 5;

  Y = ++M;

  M=6, Y=6

Conditional Operator

- A ternary pair “?:” is available in C to construct conditional expression of the form

  exp1 ? exp2 : exp3; *where exp1, exp2 & exp3 are expressions*

- Example

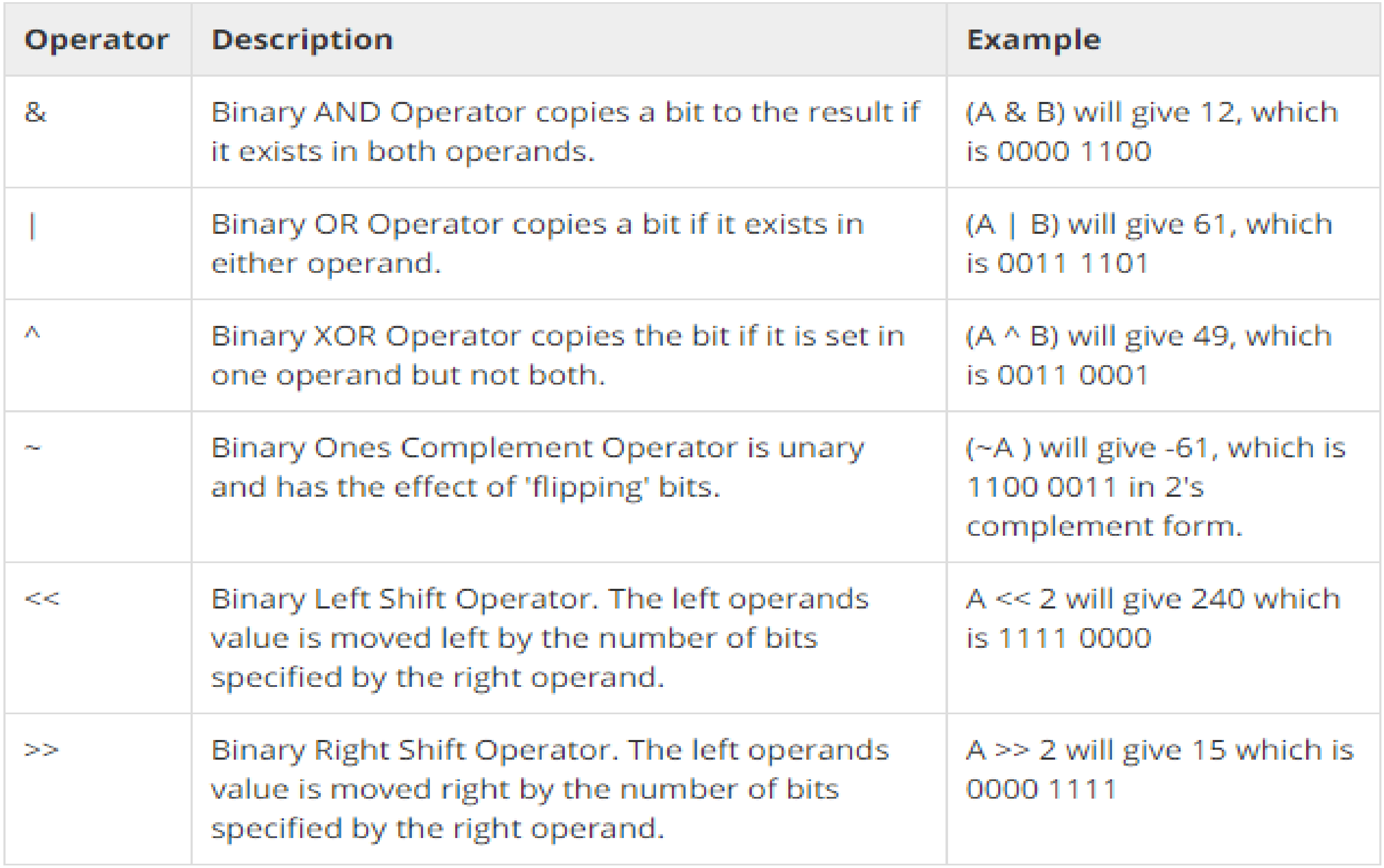
```
A=10;
B=15;
X= (A > B)? A:B;
```

Bitwise Operators

- Bitwise operator is used to manipulate the data at bit level
- Used for testing the bits or shifting them right or left
- It may not be applied to float or double

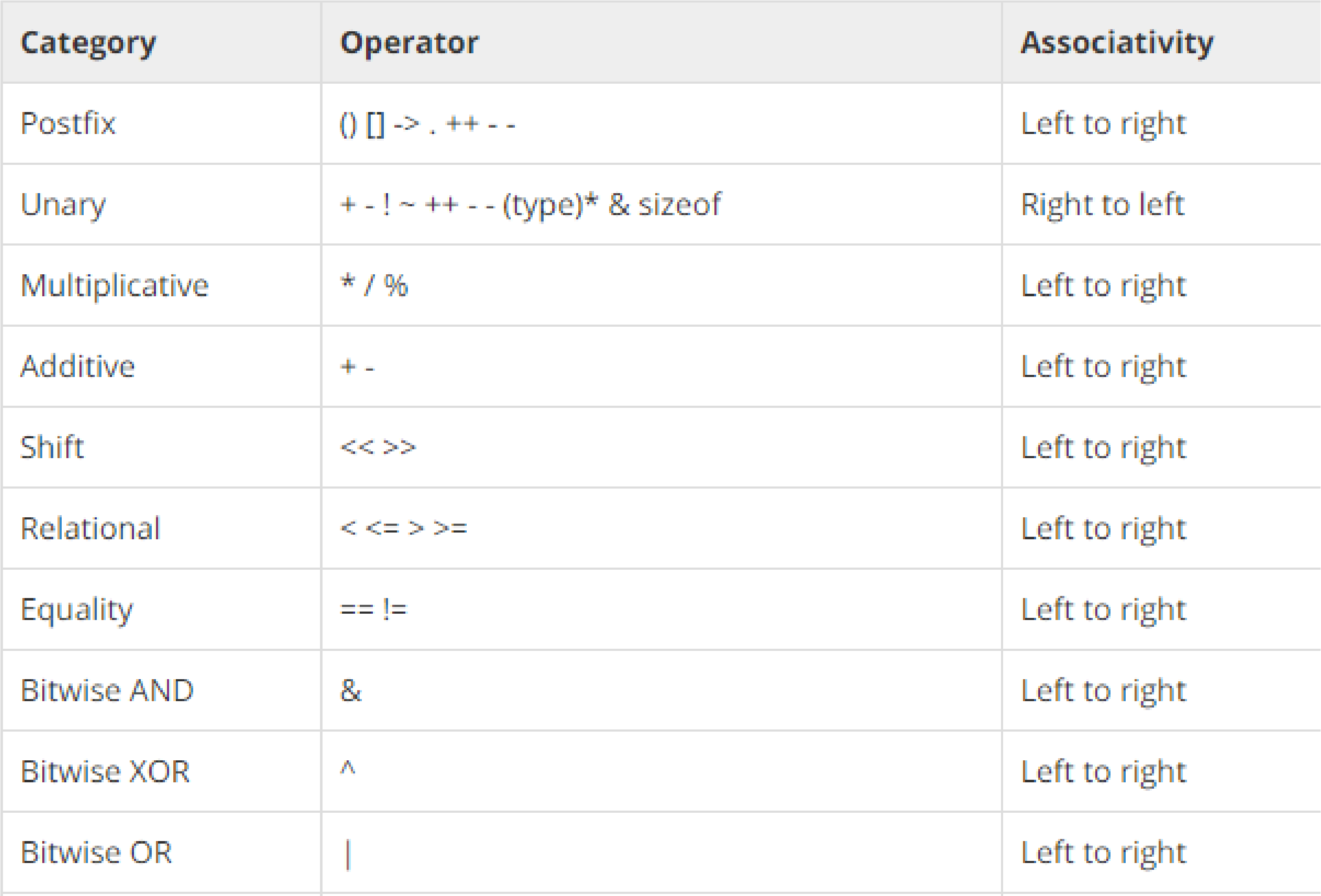
Bitwise Operators

Assume A holds 60 & B holds 13



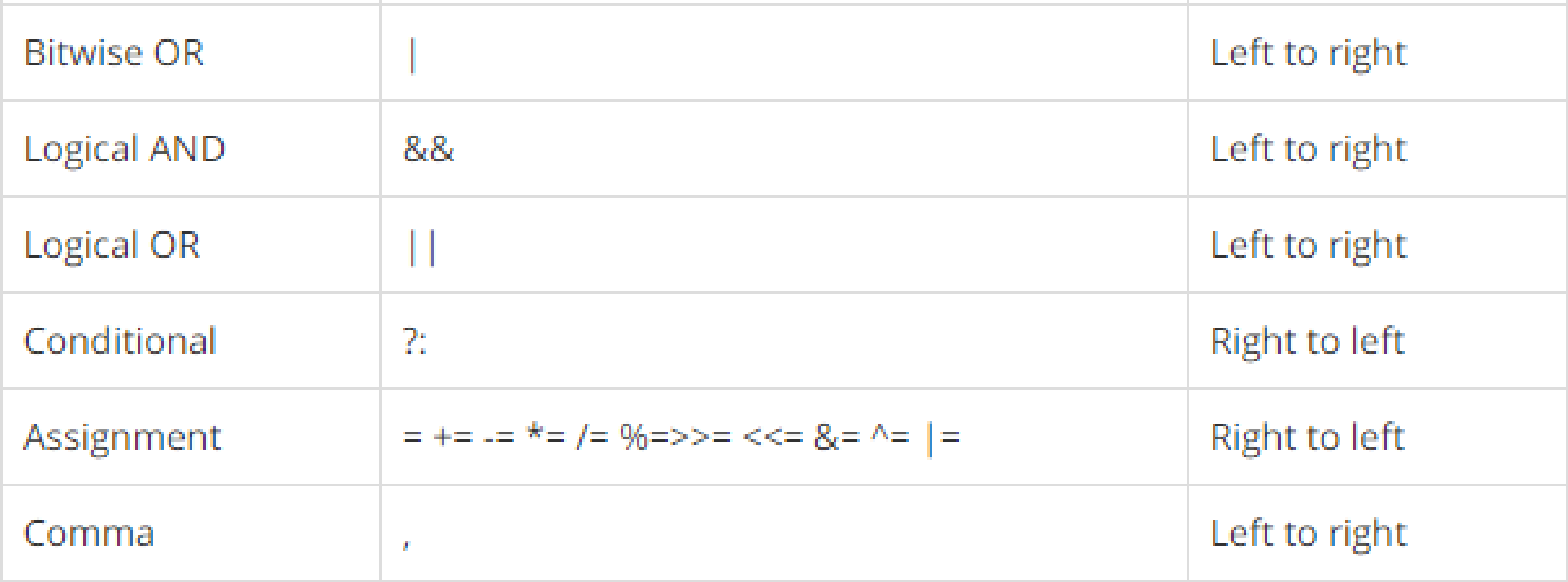
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is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

Precedence in C



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

Precedence in C



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

Thank you