Programming in C

Dr. Rupali P. Patil

Department of Electronics and Telecommunications

Topics for today

Module 2:
Operators and Expressions in C

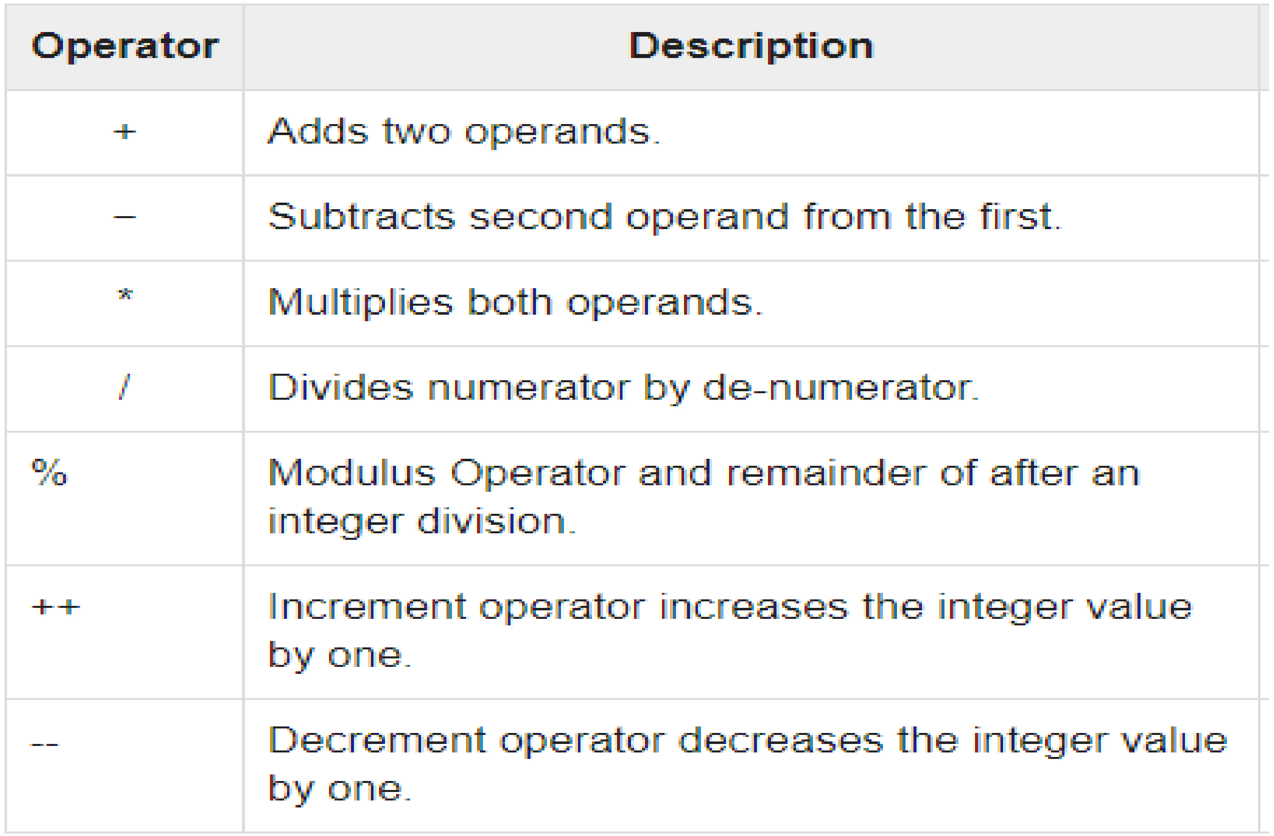
Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Operators in C: Arithmetic Operators

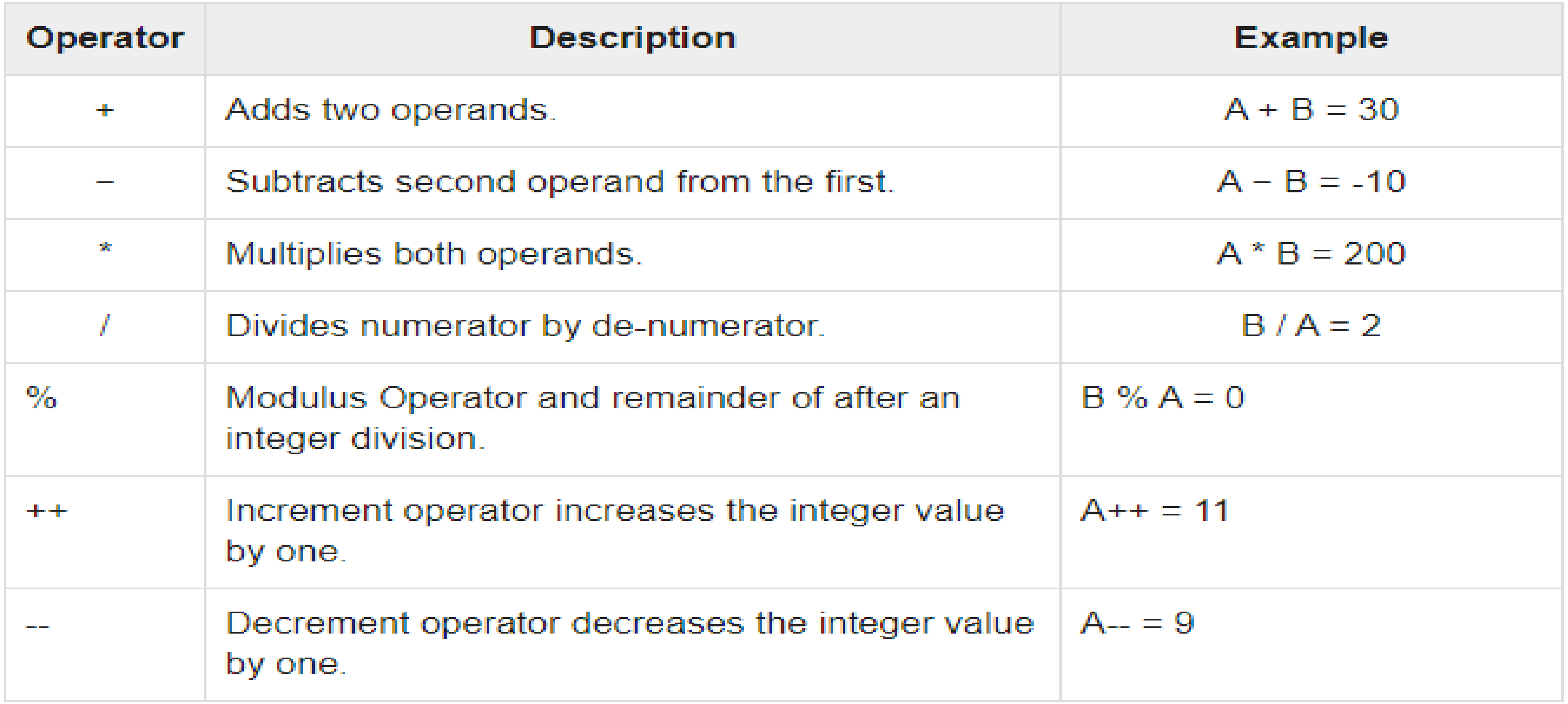
The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then −



Operator	Description
+	Adds two operands.
−	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and remainder of after an integer division.
++	Increment operator increases the integer value by one.
--	Decrement operator decreases the integer value by one.

Operators in C: Arithmetic Operators

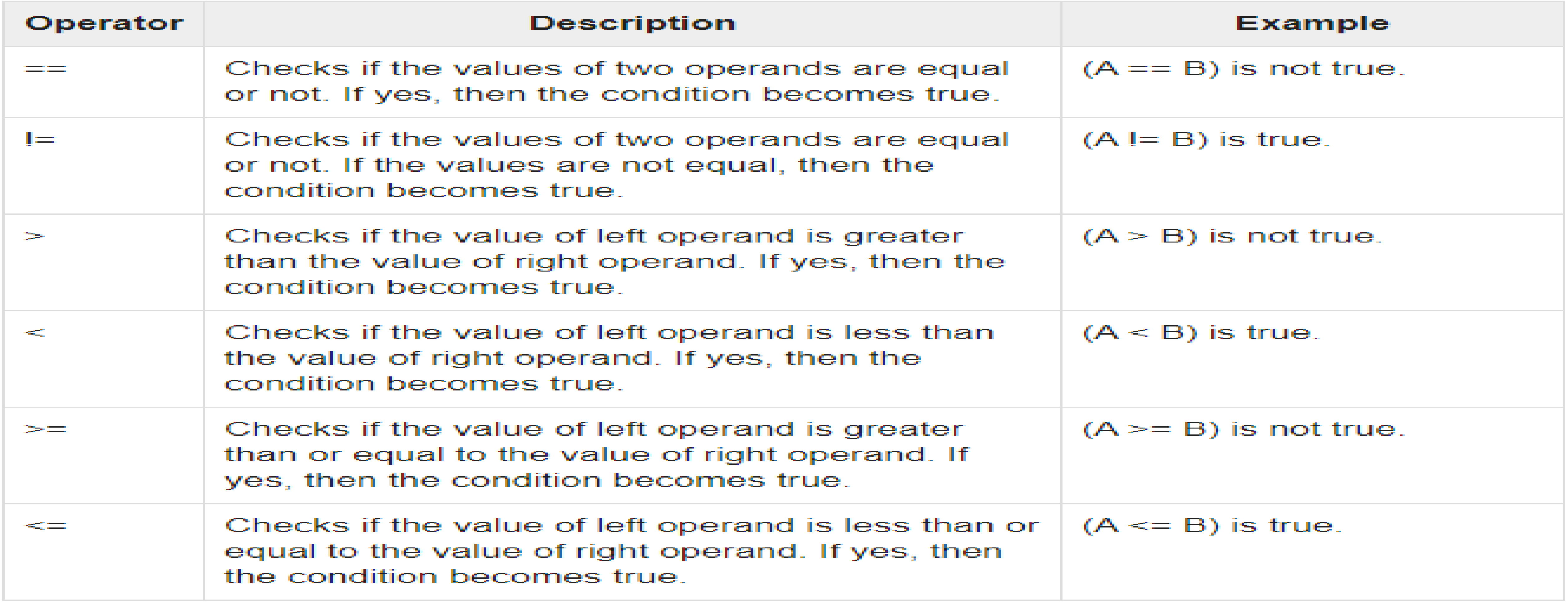
The following table shows all the arithmetic operators supported by the C language. Assume variable A holds 10 and variable B holds 20 then −



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
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Operators in C: Relational Operators

The following table shows all the relational operators supported by C. Assume 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.

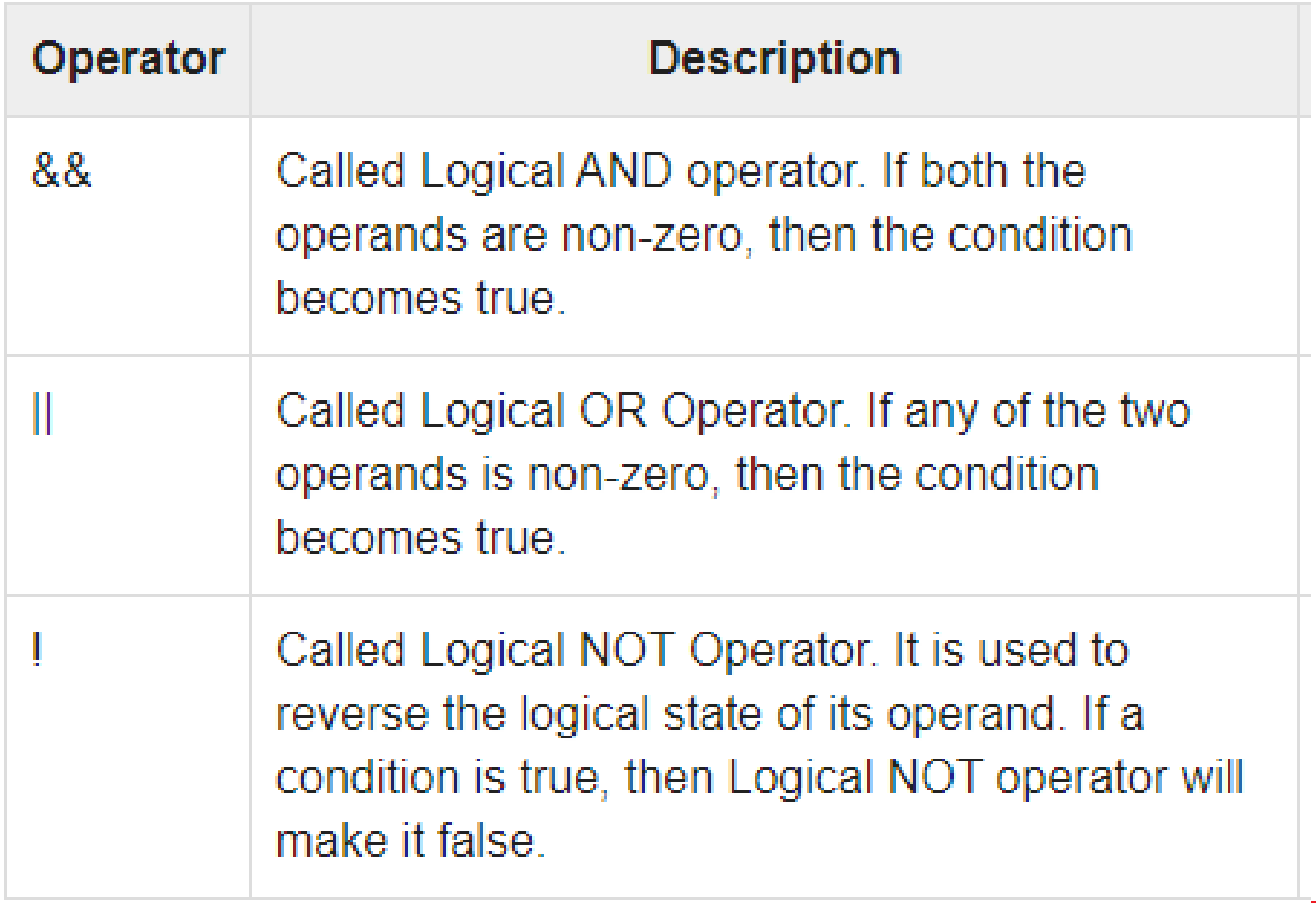
Operators in C: Relational Operators

The following table shows all the relational operators supported by C. Assume variable A holds 10 and variable B holds 20 then −

```
#include <stdio.h>
int main() {

   int a = 21;
   int b = 10;
   int c ;
   if( a == b ) {
      printf("a is equal to b\n" );
   } else {
      printf(" a is not equal to b\n" );
   }

   if ( a < b ) {
      printf("a is less than b\n" );
   } else {
      printf("a is not less than b\n" );
   }
}
```

Operators in C: Logical Operators

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −



Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes 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.

Operators in C: Logical Operators

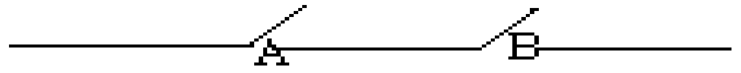
Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −

Truth table for AND,

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

Z=A.B

Switch realisation of AND:



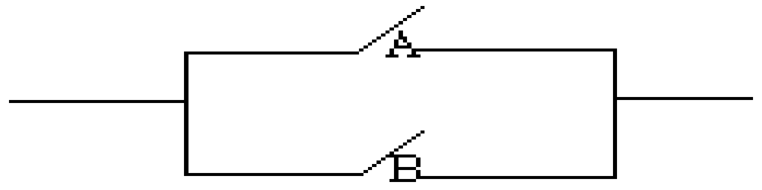
Continuity occurs only when both A AND B are closed.

Truth table for OR,

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

Z=A+B

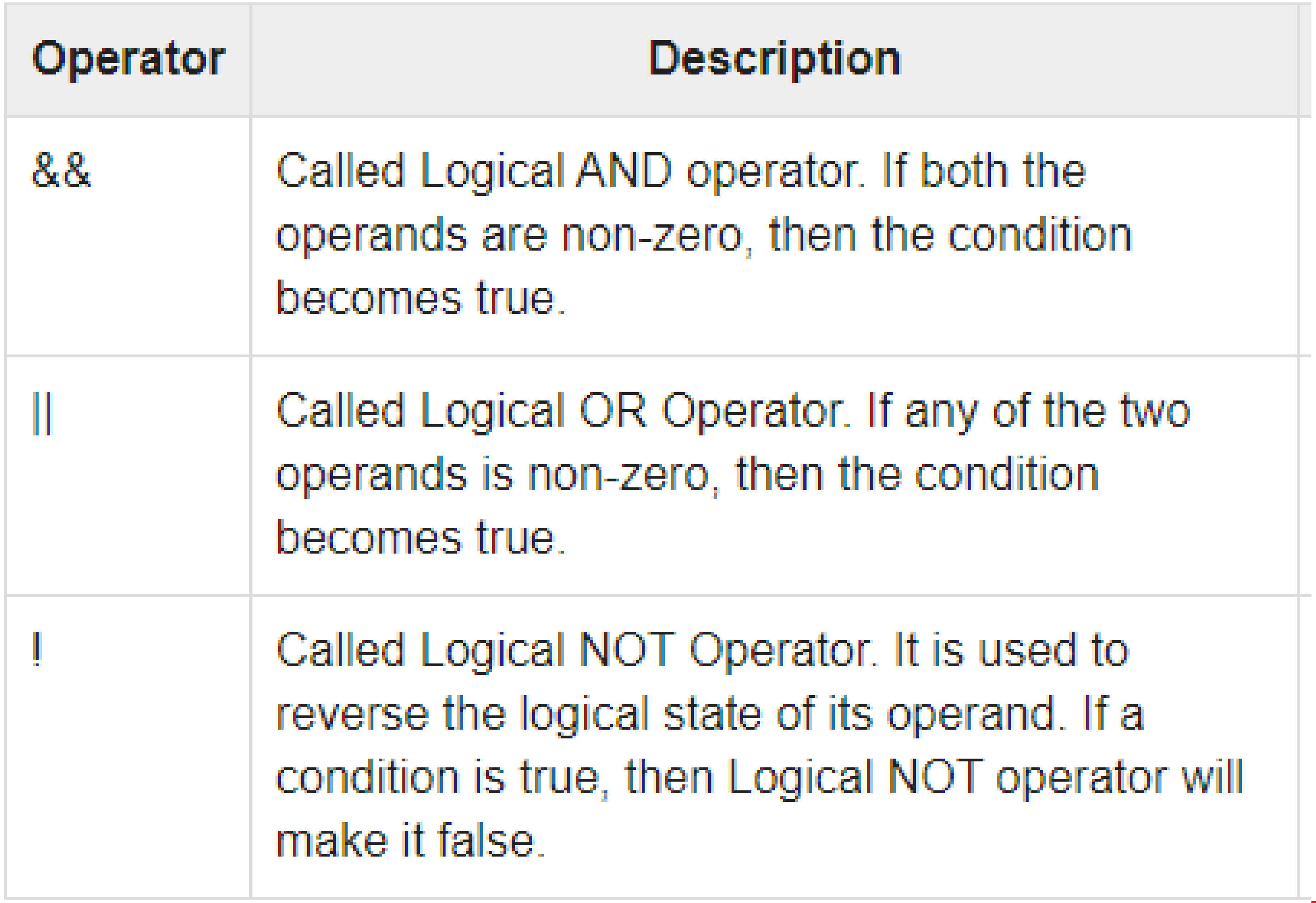
Switch realisation of OR:



Continuity if A or B or both are closed.

Operators in C: Logical Operators

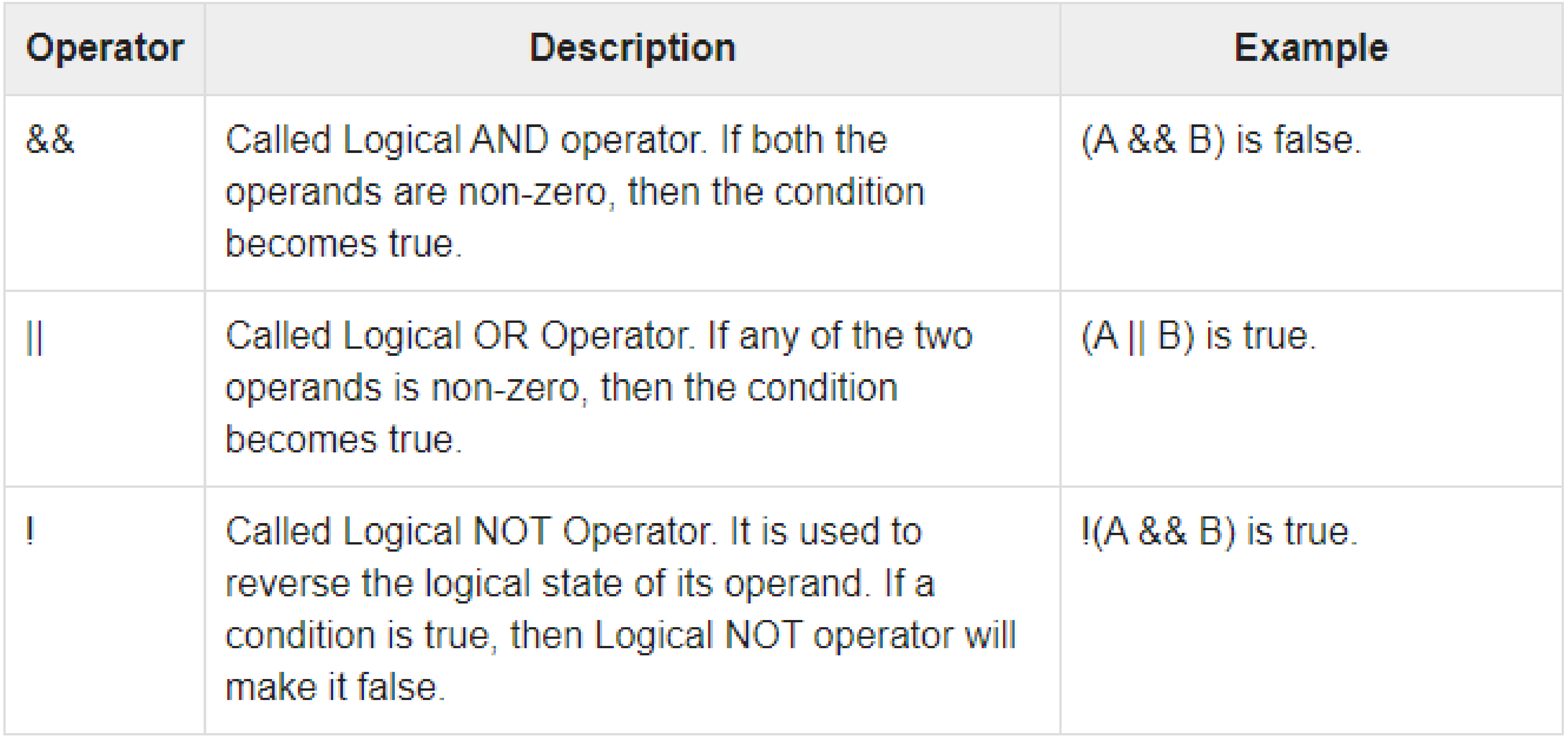
Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −



Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes 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.

Operators in C: Logical Operators

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.

Operators in C: Logical Operators

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −

Write a program:

Take any two integer numbers and print whether the condition is true or not true

Operators in C: Logical Operators

Following table shows all the logical operators supported by C language. Assume variable A holds 1 and variable B holds 0, then −

```
#include <stdio.h>
int main() {
   int a = 5;
   int b = 20;
   int c ;

   if ( a && b ) {
      printf("Condition is true\n" );
   }

   if ( a || b ) {
      printf("Line 2 - Condition is true\n" );
   }
}
```

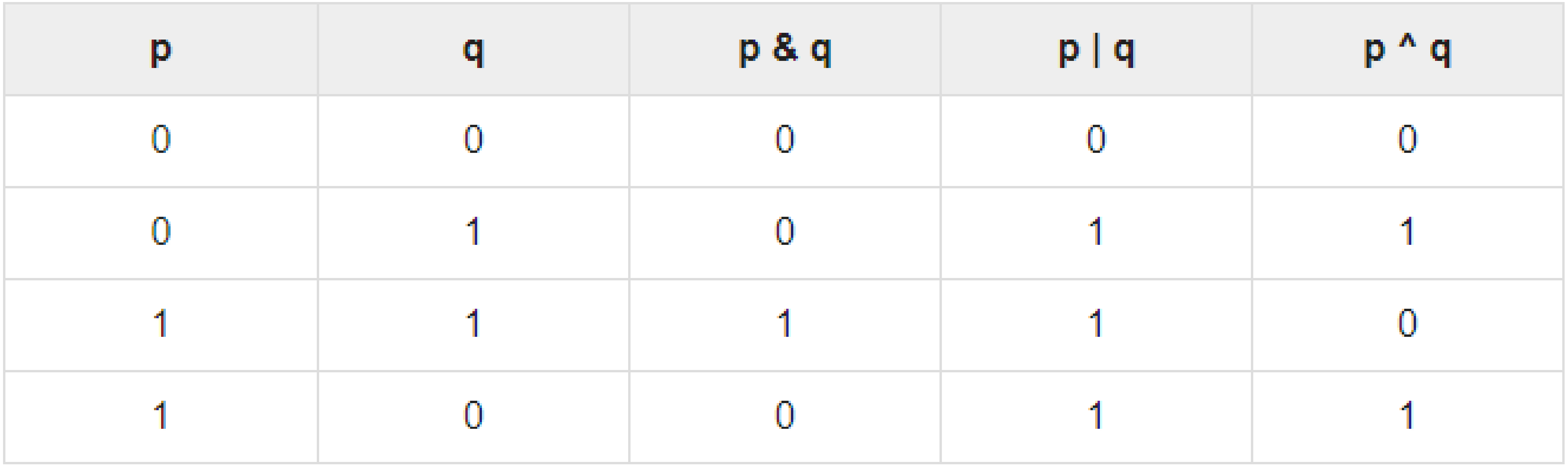
Operators in C: Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

AND

OR

Exclusive OR
EXOR



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

Operators in C: Bitwise Operators

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

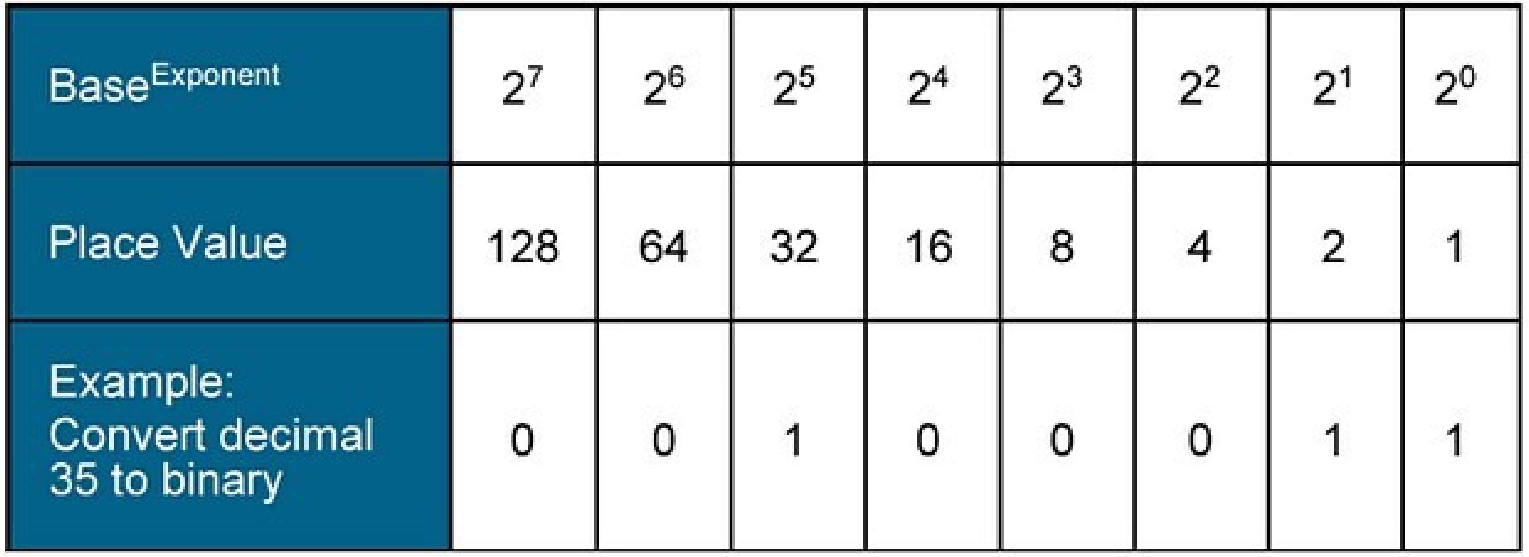
B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

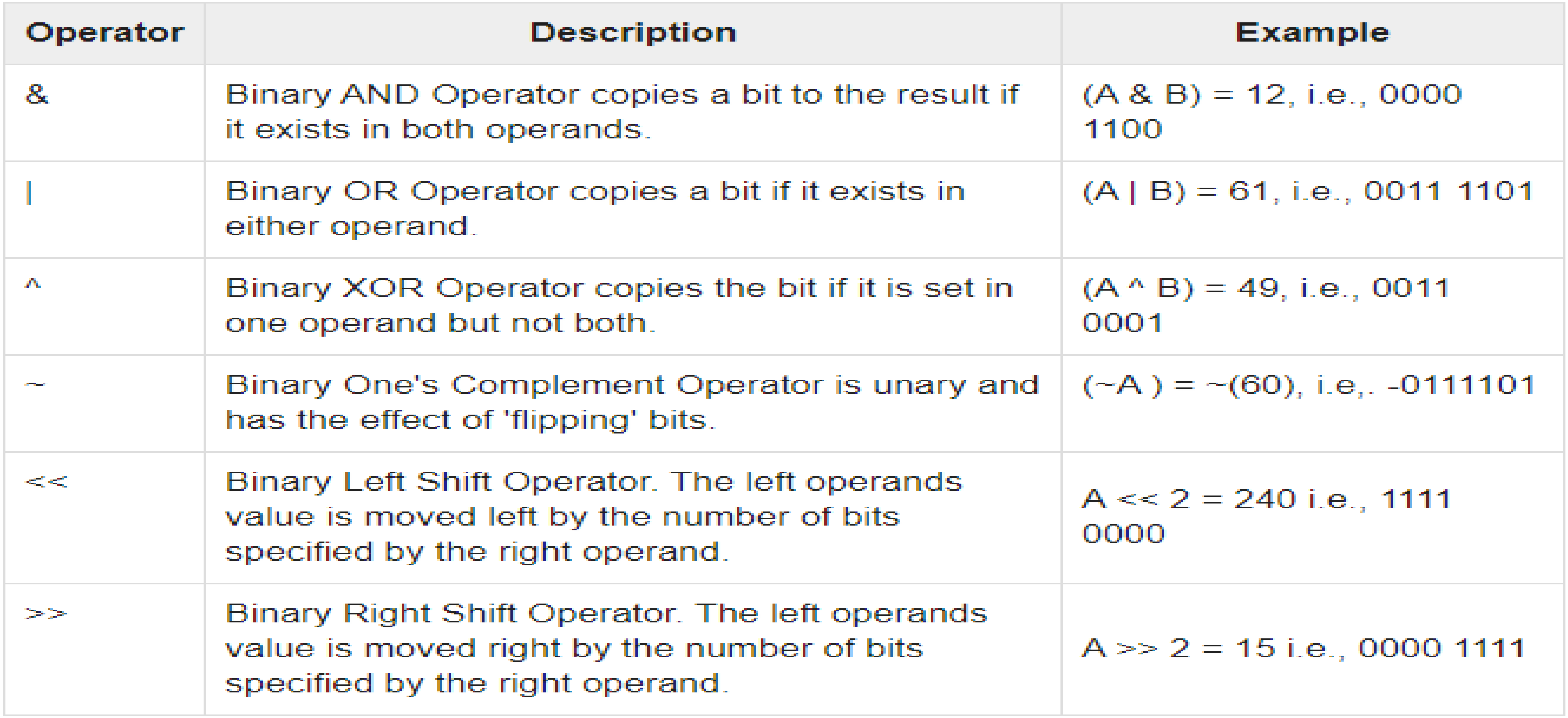
A^B = 0011 0001

~A = 1100 0011



Base^{Exponent}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Place Value	128	64	32	16	8	4	2	1
Example: Convert decimal 35 to binary	0	0	1	0	0	0	1	1

Operators in C: Bitwise Operators



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

Operators in C: Bitwise Operators

```
#include <stdio.h>

main() {

   int a = 60;              /* 60 = 0011 1100 */
   int b = 13;              /* 13 = 0000 1101 */
   int c = 0;

   c = a & b;       /* 12 = 0000 1100 */
   printf("Line 1 - Value of c is %d\n", c );

   c = a | b;       /* 61 = 0011 1101 */
   printf("Line 2 - Value of c is %d\n", c );

   c = a ^ b;       /* 49 = 0011 0001 */
   printf("Line 3 - Value of c is %d\n", c );
}
```

Operators in C: Bitwise Operators

```
#include <stdio.h>

main() {

   int a = 60;              /* 60 = 0011 1100 */
   int b = 13;              /* 13 = 0000 1101 */
   int c = 0;

   c = ~a;         /*-61 = 1100 0011 */
   printf("Line 4 - Value of c is %d\n", c );

   c = a << 2;     /* 240 = 1111 0000 */
   printf("Line 5 - Value of c is %d\n", c );

   c = a >> 2;     /* 15 = 0000 1111 */
   printf("Line 6 - Value of c is %d\n", c );
}
```

Operators in C: Print decimal numbers in binary format: showbits function

```
#include<stdio.h>
main()
{
int i,a,b,c;
printf("Enter a number(Only Integer): ");
scanf("%d",&a);
for(i=3;i>=0;i--)
{
b= 1<<i;
c= a&b;
c==0?printf("0"):printf("1");
}
printf("\n")
}
```

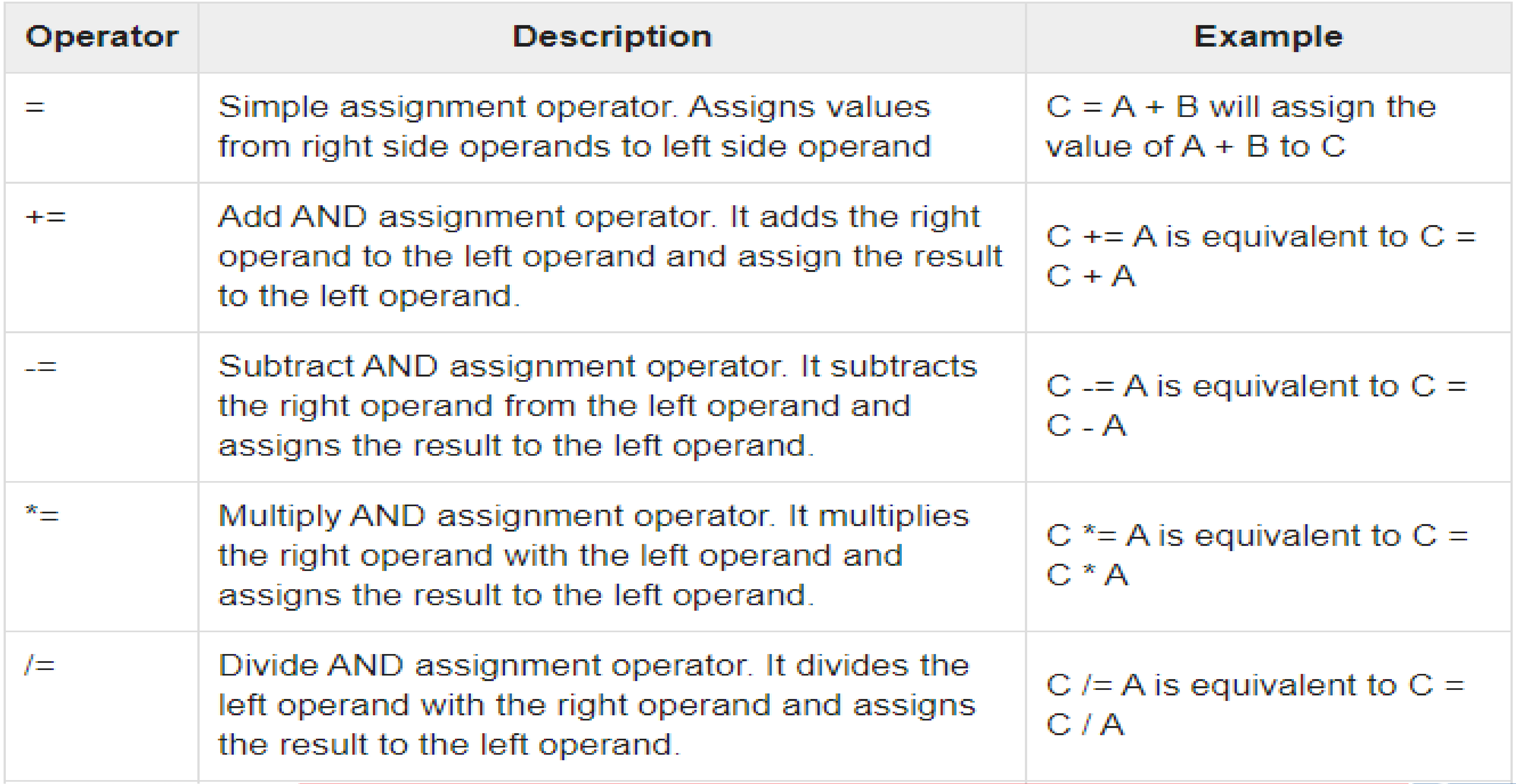
a= 2 = 0 0 1 0

i	b	C=a & b	C==0	Yes/no	Print
3	1000	0000		yes	0
2	0100	0000		yes	0
1	0010	0010		No	1
0	0001	0000		Yes	0

Operators in C: Print decimal numbers in binary format: showbits function

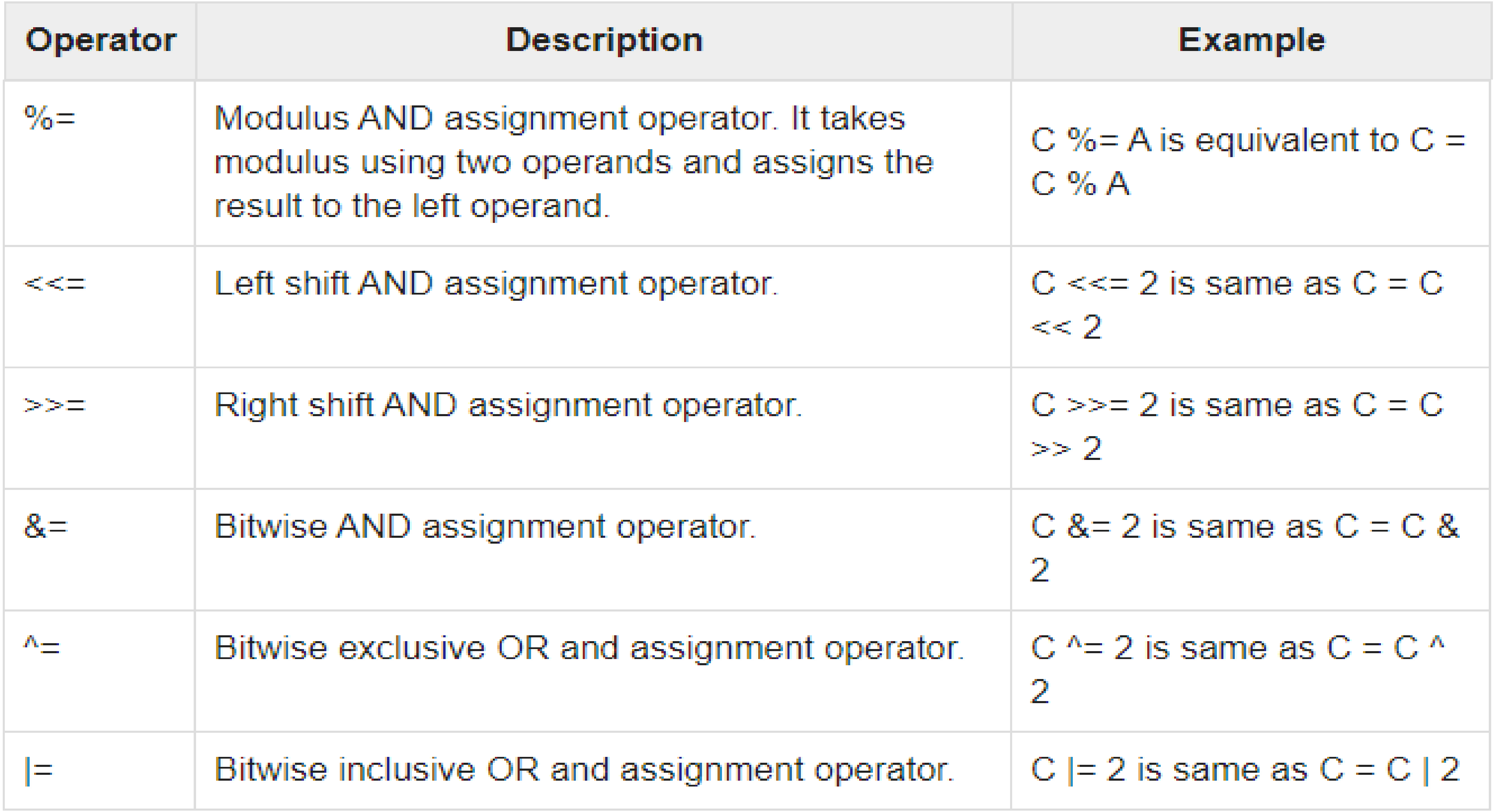
Print decimal numbers in binary: 1835 , 260 and 183

Operators in C: Assignment Operators



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

Operators in C: Assignment Operators

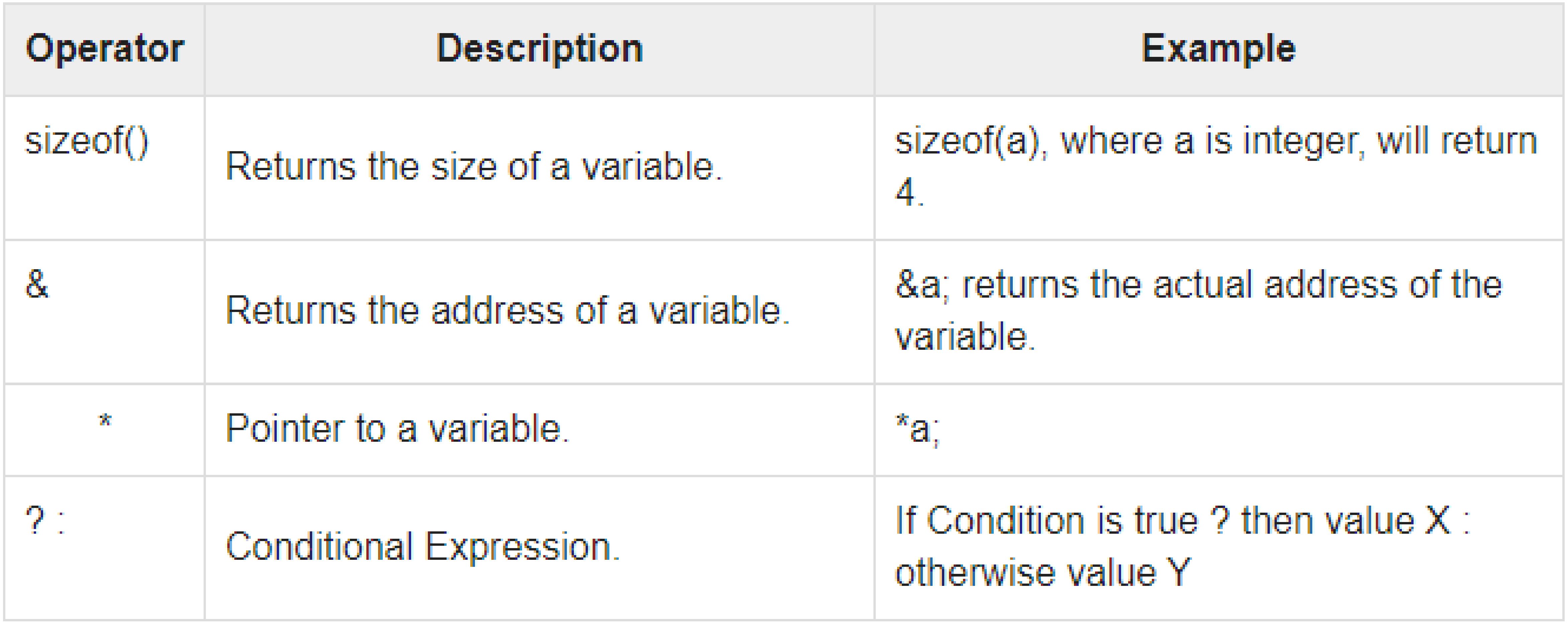


Operator	Description	Example
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is 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

Operators in C: Assignment Operators

```
#include <stdio.h>

int  main() {

   int a = 21;
   int c ;

   c =  a;
   printf("Line 1 - =  Operator Example, Value of c = %d\n", c );

   c +=  a; = 42
   printf("Line 2 - += Operator Example, Value of c = %d\n", c );

   c -=  a; = 21   ; c/=   a =  1
   printf("Line 3 - -= Operator Example, Value of c = %d\n", c );
```

Operators in C: Other Operators



Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Operators in C: Other Operators: Sizeof

```
#include <stdio.h>

main() {

   int a = 4;
   short b;
   double c;

   /* example of sizeof operator */
   printf("Line 1 - Size of variable a = %d\n", sizeof(a) );
   printf("Line 2 - Size of variable b = %d\n", sizeof(b) );
   printf("Line 3 - Size of variable c= %d\n", sizeof(c) );
```

Operators in C: Other Operators: & and *

```
#include <stdio.h>

main() {

   int a = 4;
   short b;
   double c;
   int* ptr;

   /* example of & and * operators */
   ptr = &a;       /* 'ptr' now contains the address of 'a'*/
   printf("value of address of a is  %d\n", ptr);
   printf("value of a is  %d\n", a);
   printf("*ptr is %d\n", *ptr);
```

Operators in C: Other Operators: ?

```
#include <stdio.h>

main() {

   int a = 4;
   short b;
   double c;

   /* example of ternary operator */
   a = 10;
   b = (a == 1) ? 20: 30;
   printf( "Value of b is %d\n", b );

   b = (a == 10) ? 20: 30;
   printf( "Value of b is %d\n", b );
```

Operators in C: Menu driven program for simple calculator

```
#include <stdio.h>
int main() {
    char operator;
    double first, second;
    printf("Enter an operator (+, -, *,): ");
    scanf("%c", &operator);
    printf("Enter two operands: ");
    scanf("%lf %lf", &first, &second);

    switch (operator) {
    case '+':
        printf("%.1lf + %.1lf = %.1lf", first, second, first + second);
        break;
    case '-':
        printf("%.1lf - %.1lf = %.1lf", first, second, first - second);
        break;
    default:
        printf("Error! operator is not correct");
    }
    return 0;
}
```

Operators in C: Menu driven program for operators in C

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

Type Conversions: Typecasting

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

Type Conversions: Typecasting---**Implicit type casting**

- Implicit type casting means conversion of data types without losing its original meaning.
- This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.

Type Conversions: Typecasting---**Implicit type casting**

- Implicit type casting means conversion of data types without losing its original meaning.
- This type of typecasting is essential when you want to change data types **without** changing the significance of the values stored inside the variable.

```
#include<stdio.h>
int main()
{
        short a=10; //initializing variable of short data type
        int b; //declaring int variable
        b=a; //implicit type casting
        printf("%d\n",a);
        printf("%d\n",b);
}
```

Type Conversions: Typecasting---Implicit type casting

```
#include <stdio.h>
int main()
{
   int  number = 1;
   char character = 'k'; //ASCII value of k =107
   int sum;
   sum = number + character;
   printf("Value of sum : %d\n", sum );
}
```

Type Conversions: Typecasting---Implicit type casting

```
#include <stdio.h>
int main()
{
   int  number = 1;
   char character = 'k';
   int sum;
   sum = number + character;
   printf("Value of sum : %d\n", sum );
}
```

Here, compiler has done an integer promotion by converting the value of 'k' to ASCII before performing the actual addition operation.

Type Conversions: Typecasting---**Implicit type casting**

Important Points about Implicit Conversions

- Implicit type of type conversion is also called as standard type conversion.
- We do not require any keyword or special statements in implicit type casting.
- Converting from smaller data type into larger data type is also called as type promotion. In the above example, we can also say that the value of "k" is promoted to type integer.
- The implicit type conversion always happens with the compatible data types.

We cannot perform implicit type casting on the data types which are not compatible with each other such as:

1. Converting float to an int will truncate the fraction part hence losing the meaning of the value.
2. Converting double to float will round up the digits.
3. Converting long int to int will cause dropping of excess high order bits.

In all the above cases, when we convert the data types, the value will lose its meaning. Generally, the loss of meaning of the value is warned by the compiler.

Type Conversions: Typecasting---Implicit type casting

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

```
Sum = 120.000000
```

First of all, the c variable gets converted to integer, but the compiler converts num and c into "float' and adds them to produce a 'float' result.

Type Conversions: Typecasting---Implicit type casting

```
int main(void)
{
   // Local Declarations
   bool b = true;
   char c = 'X';
   float d = 1234.5;
   int i = 123;
   short s = 98;

   // Statements
   printf("bool + char is char:     %c\n", b + c);
   printf("int * short is int:      %d\n", i * s);
   printf("float * char is float:   %f\n", d * c);

   // bool promoted to char
   c = c + b;

   // char promoted to float
   d = d + c;

   b = false;

   // float demoted to bool
   b = -d;

   printf("\nAfter execution \n");
   printf("char + true:     %c\n", c);
   printf("float + char:    %f\n", d);
   printf("bool = -float:   %d\n", b);

   return 0;
}
```

```
Output:
bool + char is char:     Y
int * short is int:      12054
float * char is float:   108636.000000

After execution
char + true:     Y
float + char:    1323.500000
bool = -float:   1
```

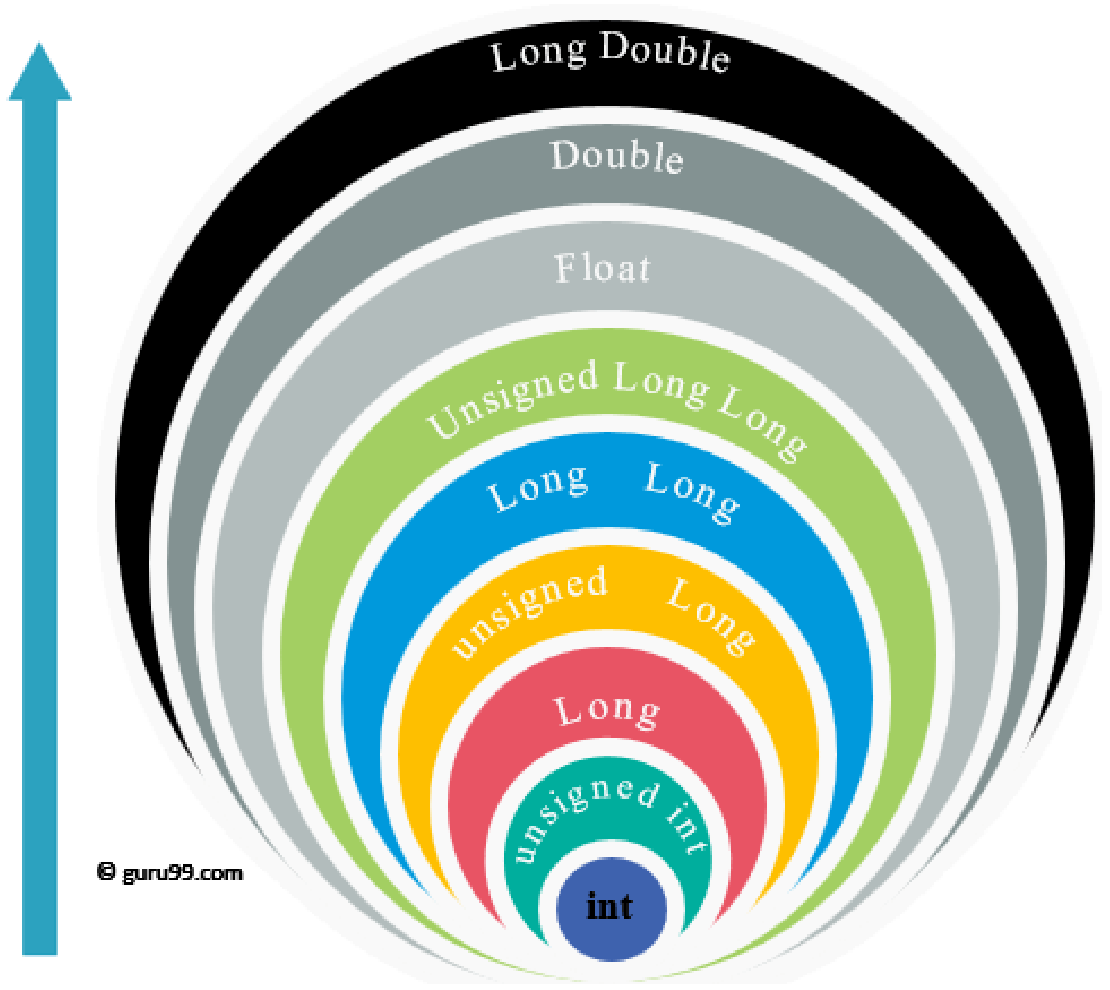
Type Conversions: Typecasting---Implicit type casting

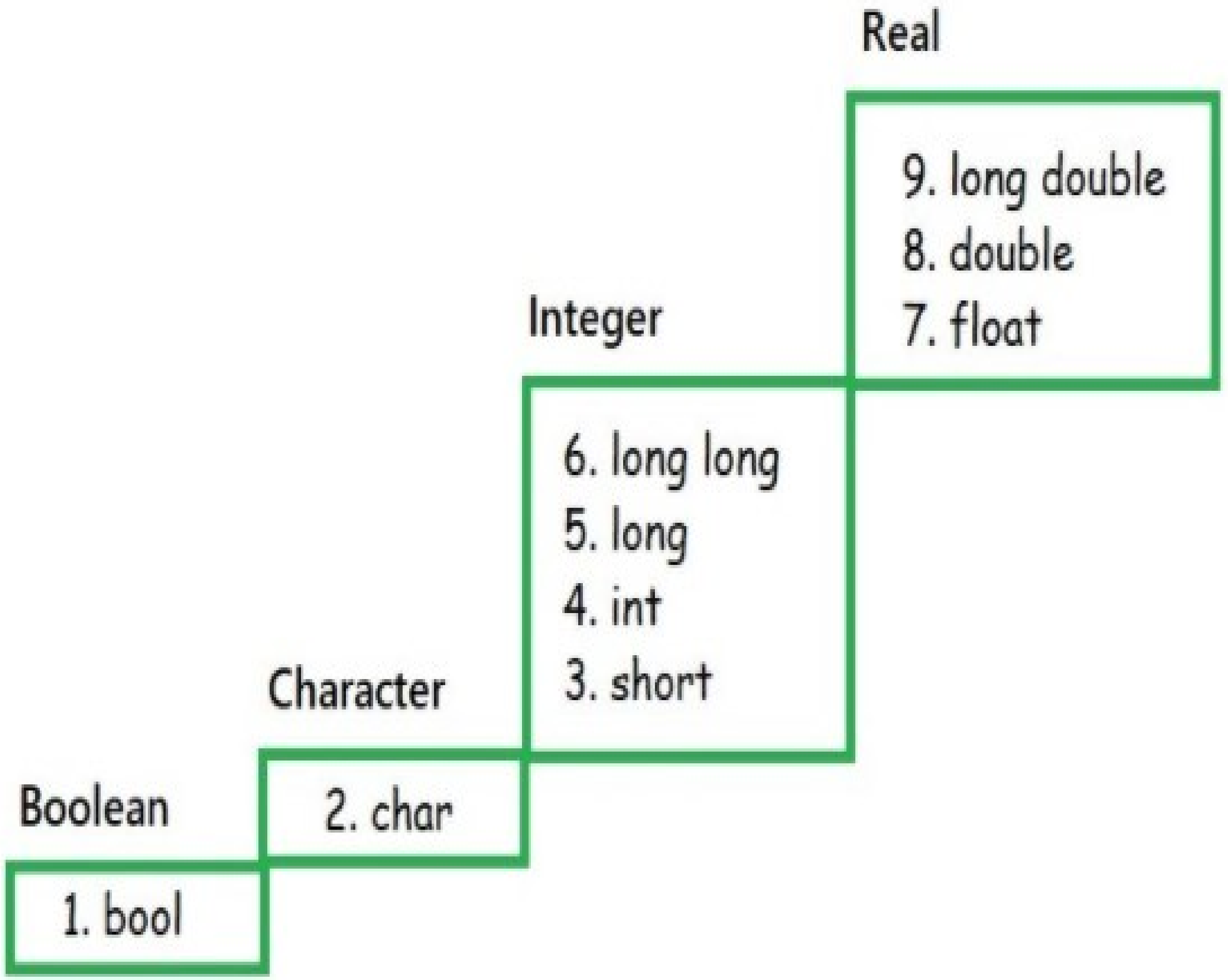
```
#include<stdio.h>

int main()
{
   float f_val1 = 97.12, f_val2;
   int i_val1, i_val2;
   char ch_val1, ch_val2;

   // float is demoted to int, only 97 is assigned to i_val1
   i_val1 = f_val1;

   // int is demoted to char,
   ch_val1 = i_val1;

   // float is demoted to int, only 12 is assigned to i_val2
   i_val2 = 12.45f;

   // char is promoted to int, now
   // i_val1 contains ASCII value of character 'e' i.e 101
   i_val2 = 'e';

   /* double is demoted to float, since by default floating point constants are of type double */

   f_val2 = 12.34;

   // Print the value of i
   printf("Value of i_val1 = %d\n", i_val1);

   // Print the character corresponding to ASCII value 97
   printf("Value of ch_val1 = %c\n", ch_val1);

   // Print the ASCII value of character 'e'
   printf("Value of i_val2 = %d\n", i_val2);

   // Print f_val2 with 2 digits of precision
   printf("Value of f_val2 = %.2f\n", f_val2);

return 0;
```

Expected Output:

```
Value of i_val1 = 97
Value of ch_val1 = a
Value of i_val2 = 101
Value of f_val2 = 12.34
```

Type Conversions: Typecasting---Arithmetic Conversion Hierarchy





Type Conversions: Typecasting---**Explicit type casting**

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion.

```
int result, var1=10, var2=3;
result=var1/var2;
```

Type Conversions: Typecasting---Explicit type casting

In implicit type conversion, the data type is converted automatically. There are some scenarios in which we may have to force type conversion.

```
int result, var1=10, var2=3;
result=var1/var2;
```

To force the type conversion in such situations, we use explicit type casting.

It requires a type casting operator. The general syntax for type casting operations is as follows:

(type-name) expression

Here,

The type name is the standard 'C' language data type.

An expression can be a constant, a variable or an actual expression.

Type Conversions: Typecasting---Explicit type casting

```
#include<stdio.h>
int main()
{
        float a = 1.2;
        //int b  = a; //Compiler will throw an error for this
        int b = (int)a + 1;
        printf("Value of a is %f\n", a);
        printf("Value of b is %d\n",b);
        return 0;
}
```

Output?

```
a= 1.200000
b=2
```

Type Conversions: Typecasting---**Explicit type casting**

Summary

Typecasting is also called as type conversion

It means converting one data type into another.

Converting smaller data type into a larger one is also called as type promotion.

'C' provides an implicit and explicit way of type conversion.

Implicit type conversion operates automatically when the compatible data type is found.

Explicit type conversion requires a type casting operator.

Keep in mind the following rules for programming practice when dealing with different data type to prevent from data loss :

Integers types should be converted to float.

Float types should be converted to double.

Character types should be converted to integer.

Type Conversions: Typecasting---Explicit type casting

```
bool        x = true;
char        y = ‘X’;
int       i = 123;
short       s = 98;
long double d = 1234.5678;

a = x + y; Printf(“variable a = %d”, a) = 89
Printf(“variable a = %c”, a) = Y
z= i * s;
d * y;
```

Type Conversions: Typecasting---Explicit type casting

```
float f;
int a = 25, b = 13;
f = a/b
```

Correct the result

Print both results with n without typecasting

Type Conversions: Typecasting---Explicit type casting

```
#include<stdio.h>

int main()
{
    int a = 25, b = 13;
    float result;

    result = a/b;

    // display only 2 digits after decimal point
    printf("(Without typecasting) 25/13 = %.2f\n", result );

    result = (float)a/b;

    // display only 2 digits after decimal point
    printf("(With typecasting) 25/13 = %.2f\n", result );

return 0;
}
```

Expected Output:

```
(Without typecasting) 25/13 = 1.00
(With typecasting) 25/13 = 1.92
```

Type Conversions: Hierarchy of Operations

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Example 1.1: Determine the hierarchy of operations and evaluate the following expression:

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

Stepwise evaluation of this expression is shown below:

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

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8 operation: *

i = 1 + 4 / 4 + 8 - 2 + 5 / 8 operation: /

i = 1 + 1+ 8 - 2 + 5 / 8 operation: /

i = 1 + 1 + 8 - 2 + 0 operation: /

i = 2 + 8 - 2 + 0 operation: +

i = 10 - 2 + 0 operation: +

i = 8 + 0 operation : -

i = 8 operation: +

Type Conversions: Hierarchy of Operations

Algebric Expression
$a \times b - c \times d$
$(m + n)\ (a + b)$
$3x2 + 2x + 5$
$\frac{a+b+c}{d+e}$
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$