KATHMANDU UNIVERSITY SCHOOL OF MANAGEMENT

BBIS

COM 102 : 3 Credit Hours

4. Operators in C

09/01/2022

Outline

4.1 Arithmetic Operators

4.2 Assignment Operators

4.3 Logical Operators

4.4 Bitwise Operators

4.5 Unary Operator

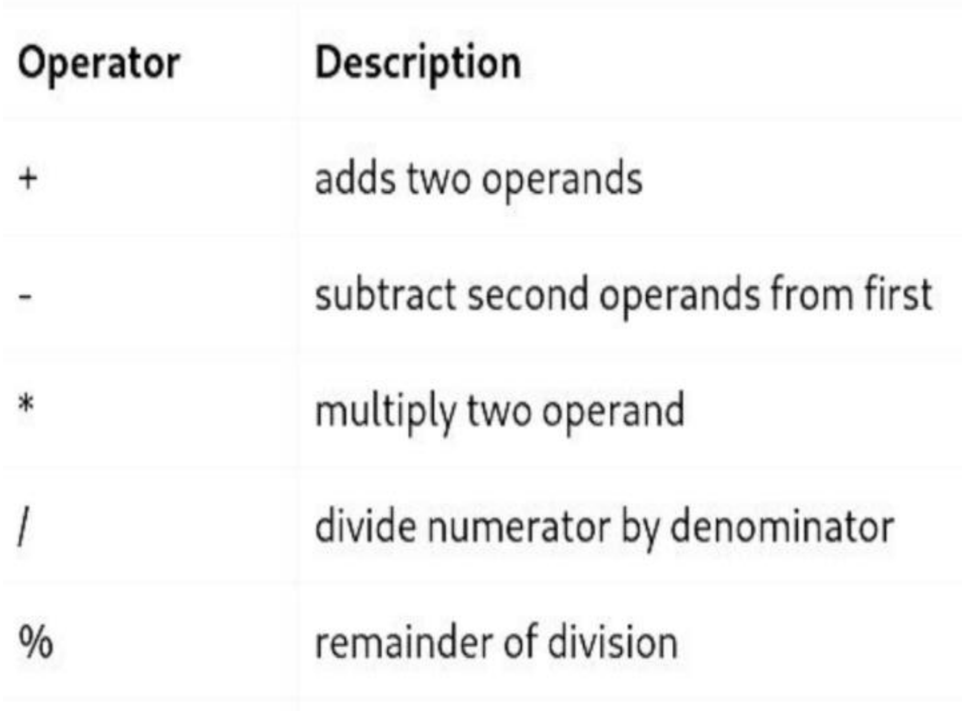
4.6 Conditional Operator

Operators

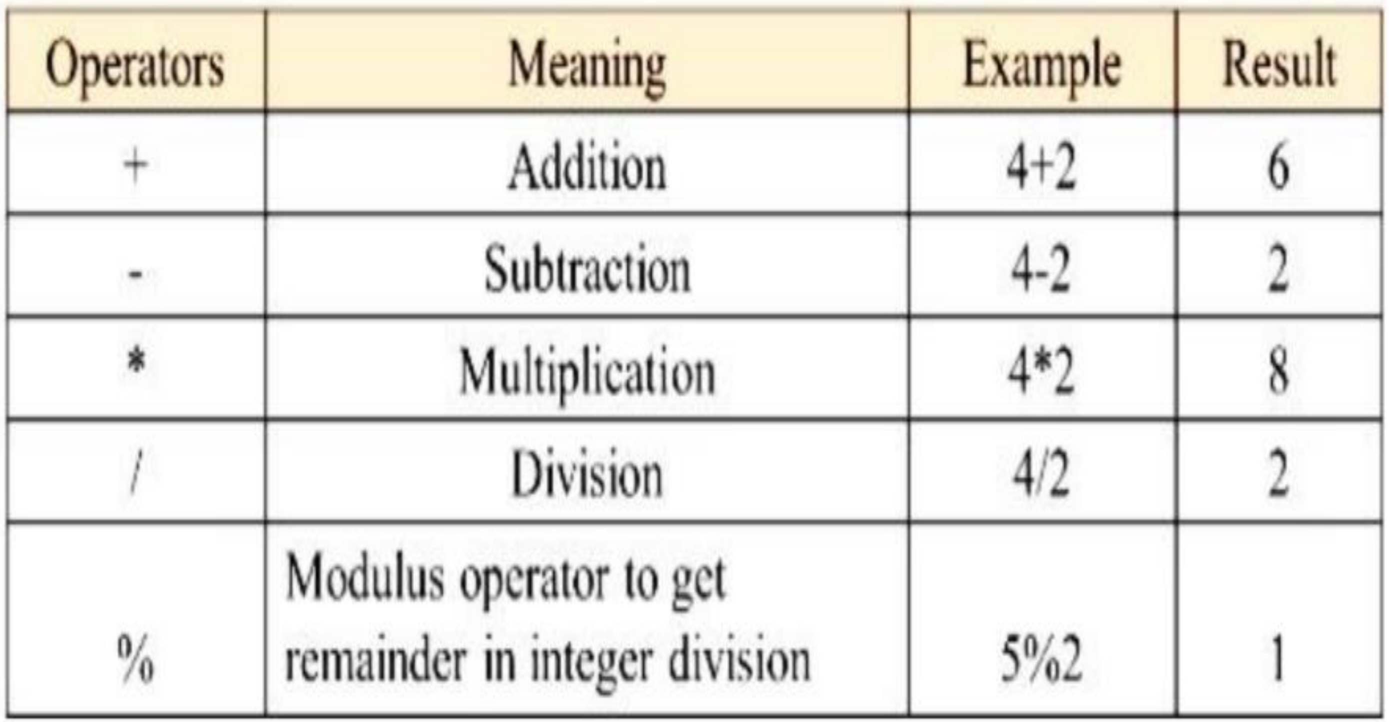
- An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.
- These C operators join individual constants and variables to form expressions.
- C language is rich in built-in operators and provides the following types of operators:
  - Arithmetic Operators
  - Increment and Decrement Operators
  - Assignment Operators
  - Logical Operators
  - Relational Operators
  - Conditional Operator
  - Bitwise Operators
  - Special Operators

Arithmetic Operators

- Arithmetic Operators are used to performing mathematical calculations like addition (+), subtraction (-), multiplication (*), division (/) and modulus (%).



Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division



Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1

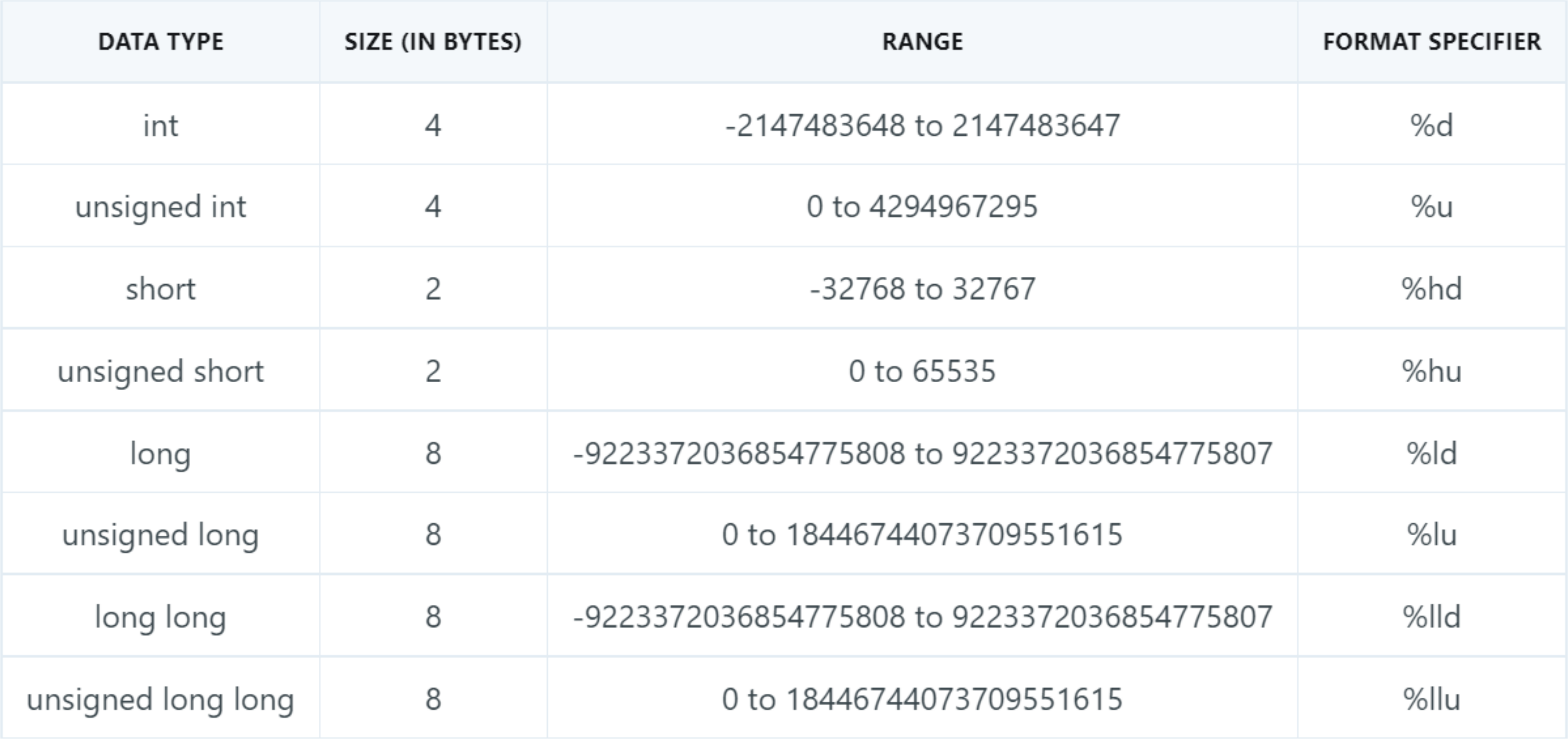
Arithmetic Operators: Example

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

int a = 40, b = 20;
int add,sub,mul,div,mod;
add = a+b;
sub = a-b;
mul = a*b;
div = a/b;
mod = a%b;
printf("Addition of a, b is : %d\n", add);
printf("Subtraction of a, b is : %d\n", sub);
printf("Multiplication of a, b is : %d\n", mul);
printf("Division of a, b is : %d\n", div);
printf("Modulus of a, b is : %d\n", mod);
return 0;
}
```

Format specifier

- The format specifiers are used in C for input and output purposes.
- Using this concept the compiler can understand that what type of data is in a variable during taking input using the scanf() function and printing using printf() function.
- Here is a list of format specifiers.

Integer overflows occur when the result of an arithmetic operation is a value, that is too large to fit in the available storage space.



DATA TYPE	SIZE (IN BYTES)	RANGE	FORMAT SPECIFIER
int	4	-2147483648 to 2147483647	%d
unsigned int	4	0 to 4294967295	%u
short	2	-32768 to 32767	%hd
unsigned short	2	0 to 65535	%hu
long	8	-9223372036854775808 to 9223372036854775807	%ld
unsigned long	8	0 to 18446744073709551615	%lu
long long	8	-9223372036854775808 to 9223372036854775807	%lld
unsigned long long	8	0 to 18446744073709551615	%llu

```
#include <stdio.h>
main() {
   char ch = 'B';
   printf("%c\n", ch); //printing character data
   //print decimal or integer data with d and I

   int x = 45, y = 90;
   printf("%d\n", x);
   printf("%i\n", y);

   float f = 12.67;
   printf("%f\n", f); //print float value
   printf("%e\n", f); //print in scientific notation

   int a = 67;
   printf("%o\n", a); //print in octal format
   printf("%x\n", a); //print in hex format

   char str[] = "Hello World";
   printf("%s\n", str);
   printf("%20s\n", str); //shift to the right 20 characters including the string
   printf("%-20s\n", str); //left align
   printf("%20.5s\n", str); //shift to the right 20 characters including the string, and print string up to 5 character
   printf("%-20.5s\n", str); //left align and print string up to 5 character
}
```

Increment and Decrement Operators

- Increment and Decrement Operators are useful operators generally used to minimize the calculation.
- Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1.
- These two operators are unary operators, meaning they only operate on a single operand.
  - ○ ++x is same as x = x + 1 or x += 1
  - ○ --x is same as x = x - 1 or x -= 1
- Increment and decrement operators can be used only with variables. They can't be used with constants or expressions.

  int x = 1, y = 1;
    - ++x; // valid
    - ++5; // invalid - increment operator operating on a constant value
    - ++(x+y); // invalid - increment operating on an expression

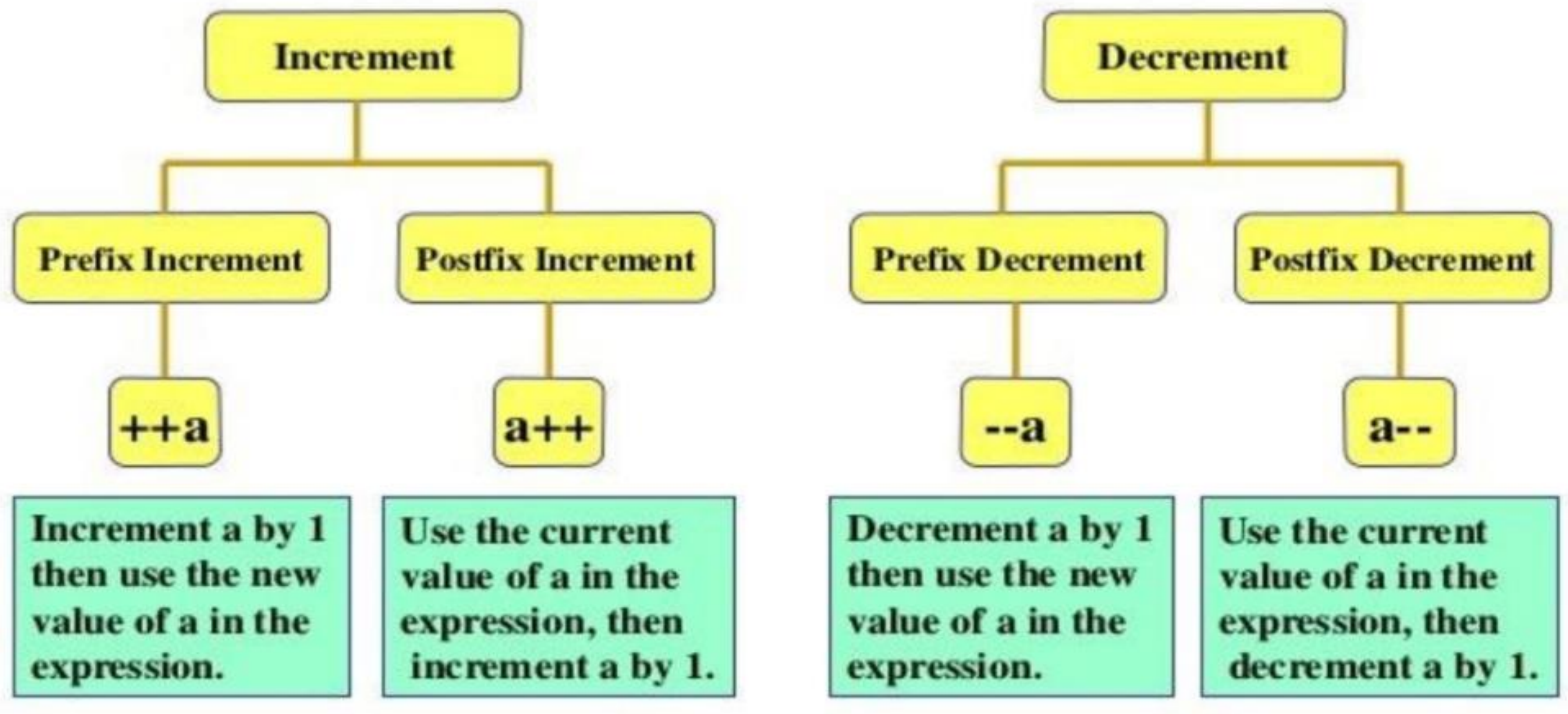
Prefix and Postfix Increment and Decrement

Increment/Decrement operators are of two types:

1. Prefix increment/decrement operator.
2. Postfix increment/decrement operator.

```
#include<stdio.h>
int main() {
int var1 = 5, var2 = 5;
// 5 is displayed
// Then, var1 is increased to 6.
printf("%d\n", var1++);
// var2 is increased to 6
// Then, it is displayed.
printf("%d\n", ++var2);
return 0;
}
```

…



Increment

Prefix Increment

++a

Increment a by 1 then use the new value of a in the expression.

Postfix Increment

a++

Use the current value of a in the expression, then increment a by 1.

Decrement

Prefix Decrement

--a

Decrement a by 1 then use the new value of a in the expression.

Postfix Decrement

a--

Use the current value of a in the expression, then decrement a by 1.

Prefix increment/decrement

The prefix increment/decrement operator immediately increases or decreases the current value of the variable. This value is then used in the expression. Let's take an example:

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

Here first, the current value of x is incremented by 1. The new value of x is then assigned to y.

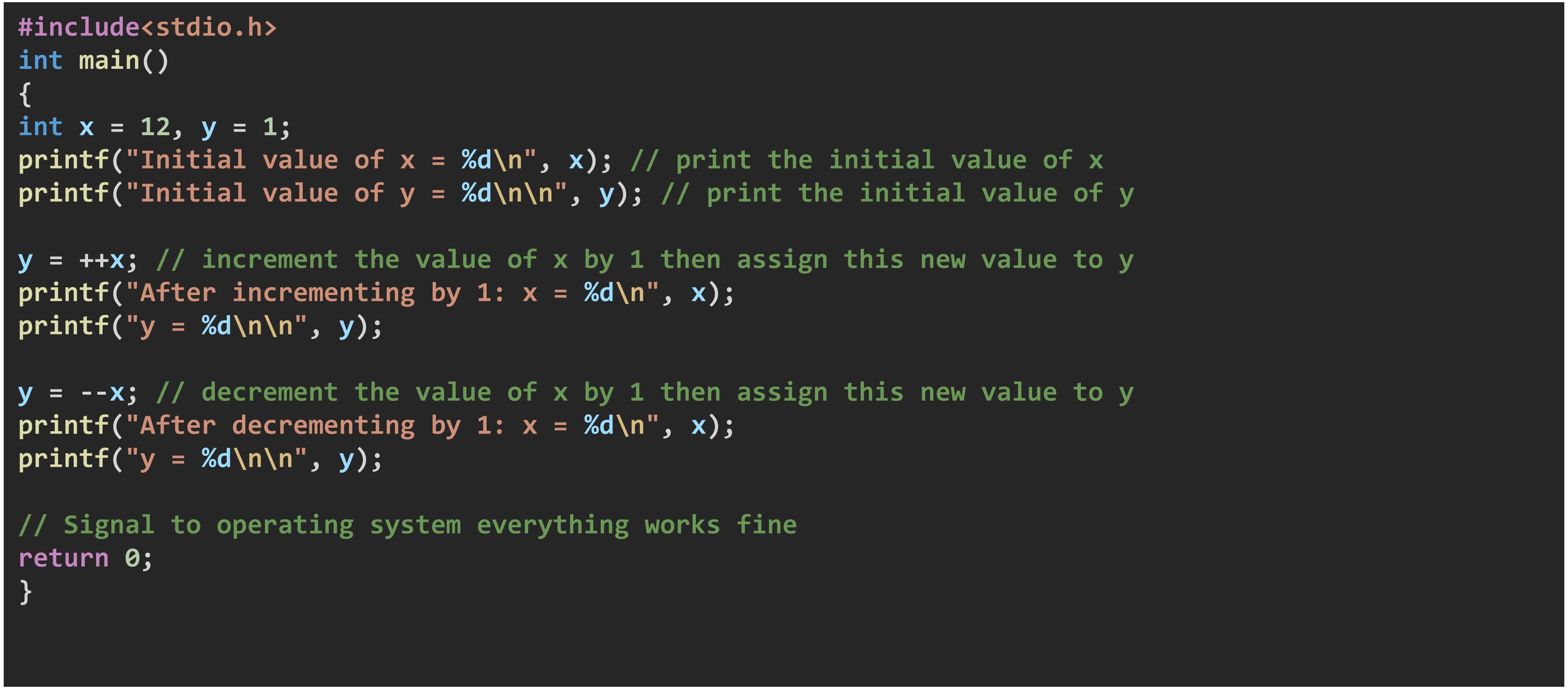
Similarly, in the statement:

```
y = --x;
```

the current value of x is decremented by 1. The new value of x is then assigned to y.

The following program demonstrates prefix increment/decrement operator in action:

…



```c
#include<stdio.h>
int main()
{
int x = 12, y = 1;
printf("Initial value of x = %d\n", x); // print the initial value of x
printf("Initial value of y = %d\n\n", y); // print the initial value of y

y = ++x; // increment the value of x by 1 then assign this new value to y
printf("After incrementing by 1: x = %d\n", x);
printf("y = %d\n\n", y);

y = --x; // decrement the value of x by 1 then assign this new value to y
printf("After decrementing by 1: x = %d\n", x);
printf("y = %d\n\n", y);

// Signal to operating system everything works fine
return 0;
}
```

Postfix Increment/Decrement operator

The postfix increment/decrement operator causes the current value of the variable to be used in the expression, then the value is incremented or decremented. For example:

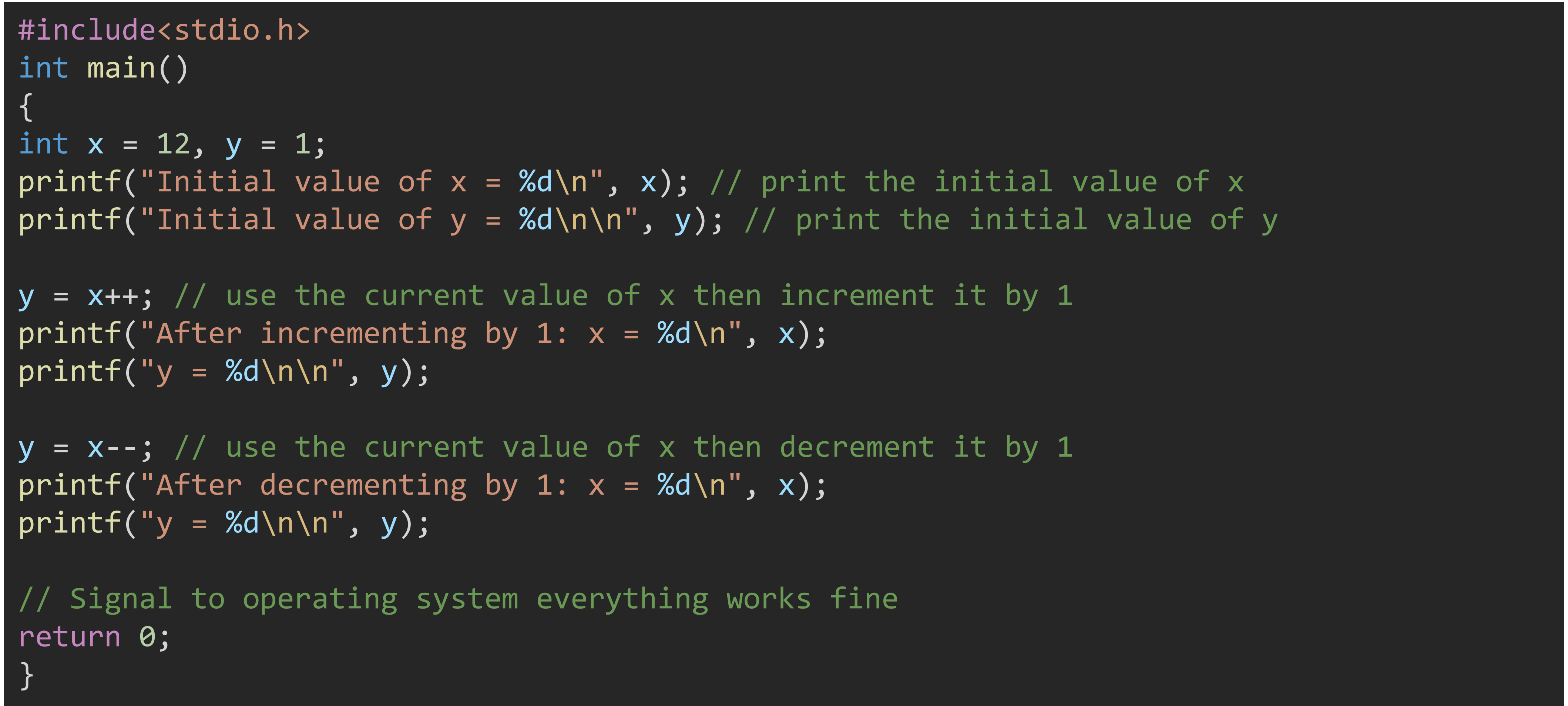
y = x++;

Here first, the current value of x is assigned to y then x is incremented.

Similarly, in the statement:

y = x--;

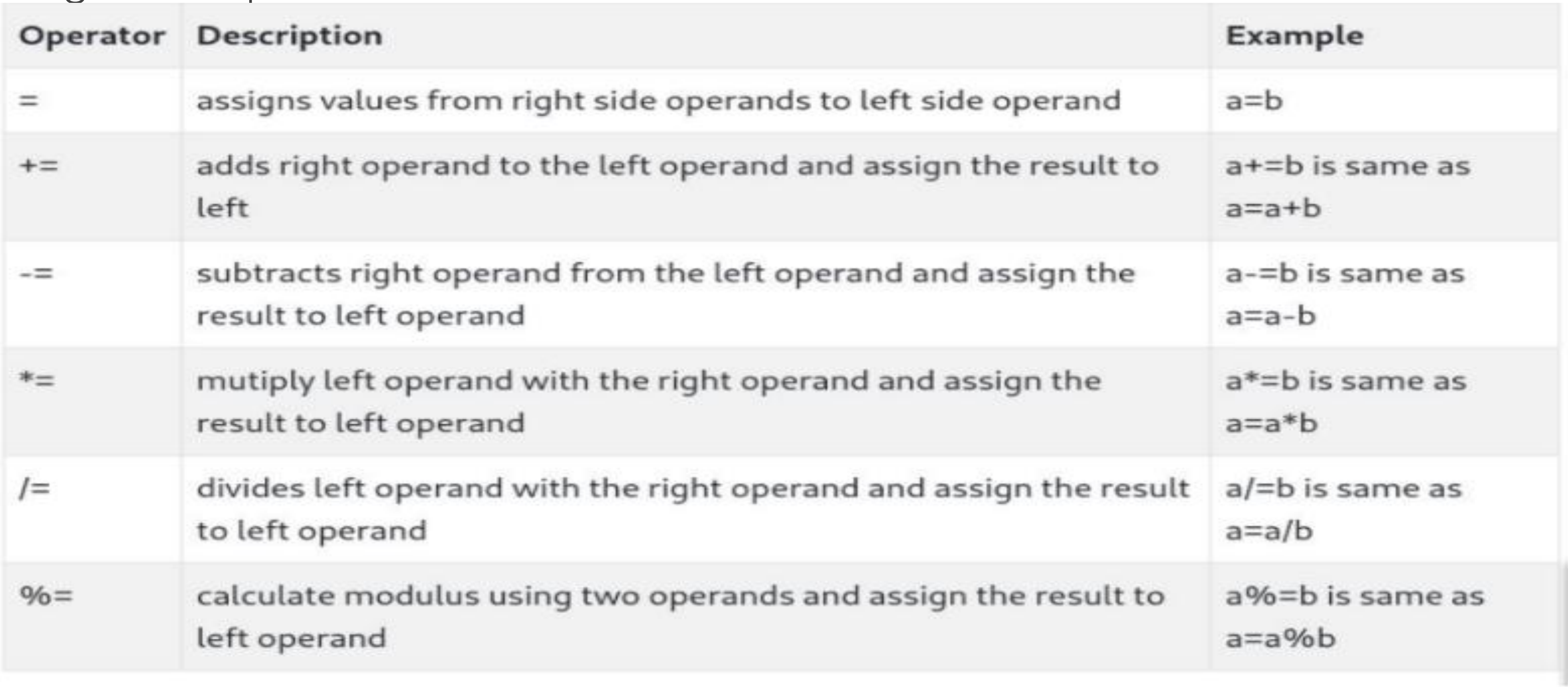
the current value of x is assigned to y then x is decremented.



```c
#include<stdio.h>
int main()
{
int x = 12, y = 1;
printf("Initial value of x = %d\n", x); // print the initial value of x
printf("Initial value of y = %d\n\n", y); // print the initial value of y

y = x++; // use the current value of x then increment it by 1
printf("After incrementing by 1: x = %d\n", x);
printf("y = %d\n\n", y);

y = x--; // use the current value of x then decrement it by 1
printf("After decrementing by 1: x = %d\n", x);
printf("y = %d\n\n", y);

// Signal to operating system everything works fine
return 0;
}
```

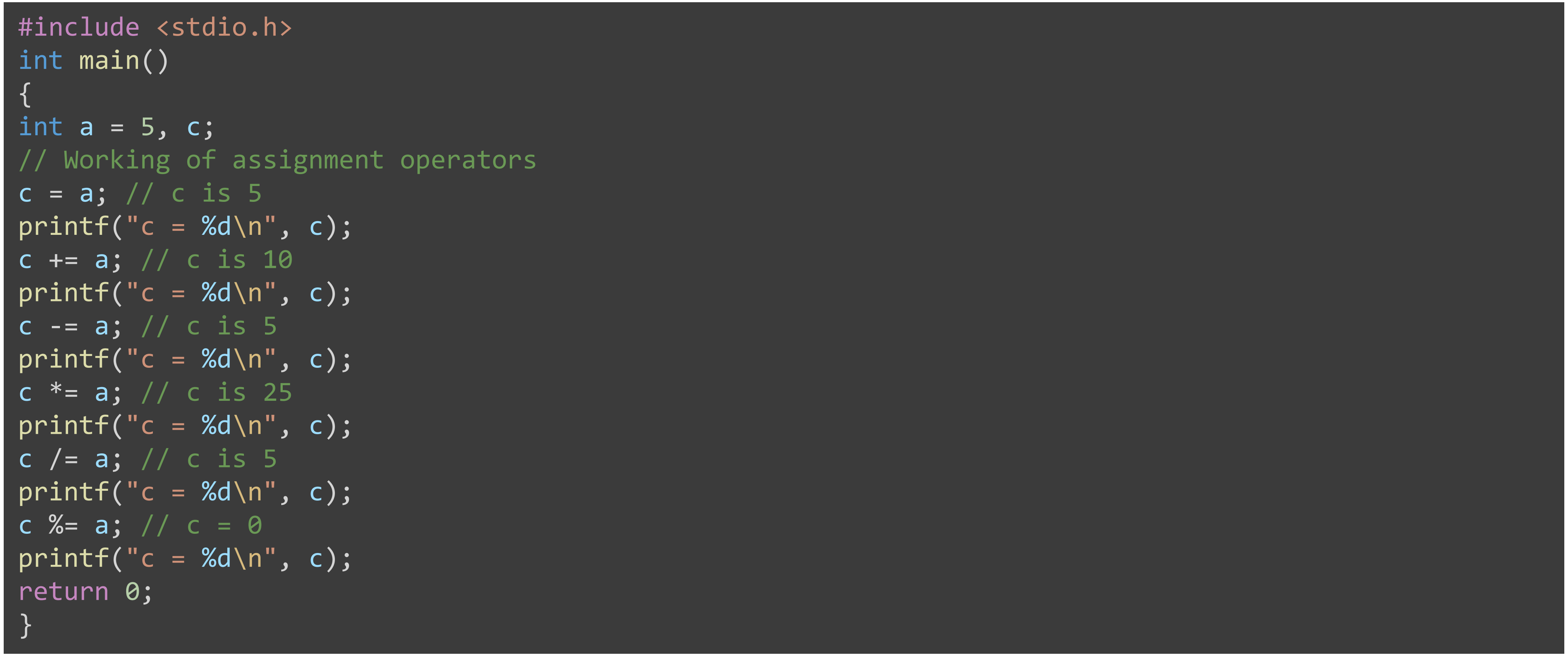
Assignment Operators

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is =



Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

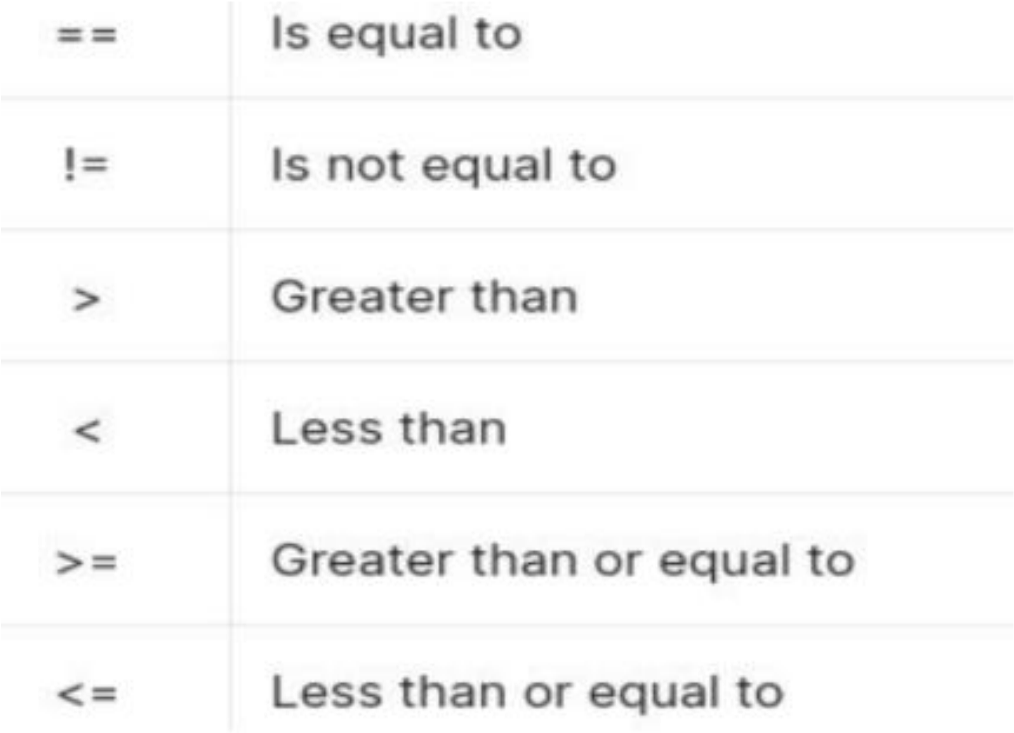
Assignment operators: Example



```c
#include <stdio.h>
int main()
{
int a = 5, c;
// Working of assignment operators
c = a; // c is 5
printf("c = %d\n", c);
c += a; // c is 10
printf("c = %d\n", c);
c -= a; // c is 5
printf("c = %d\n", c);
c *= a; // c is 25
printf("c = %d\n", c);
c /= a; // c is 5
printf("c = %d\n", c);
c %= a; // c = 0
printf("c = %d\n", c);
return 0;
}
```

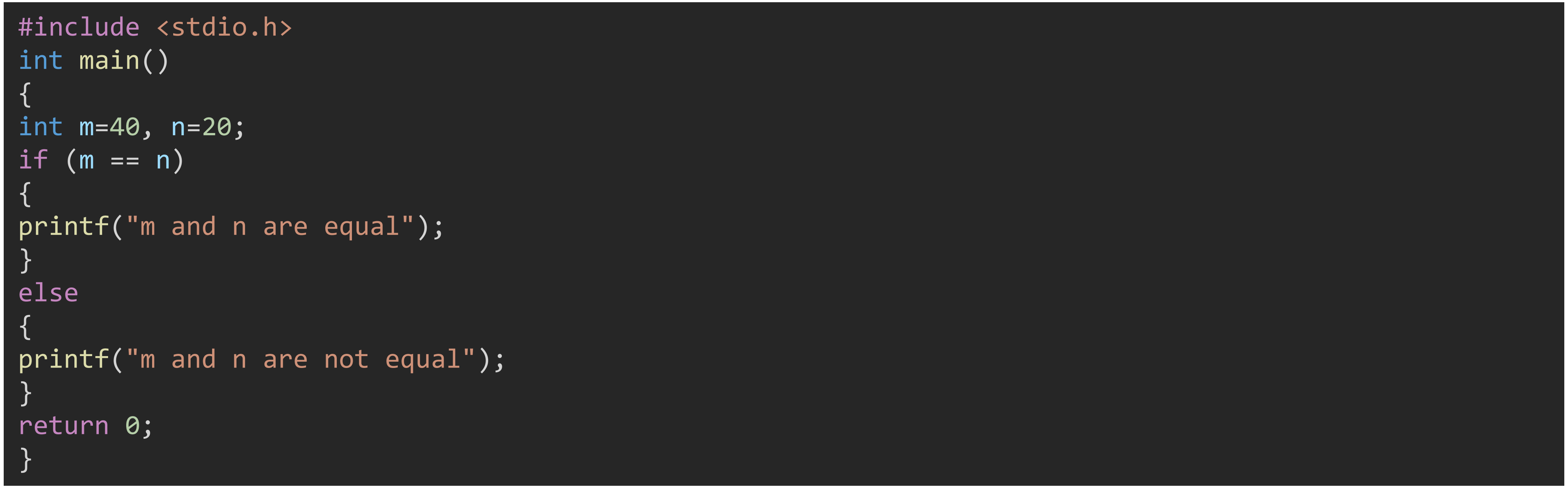
Relational Operators

- A relational operator checks the relationship between two operands.
- If the relation is true, it returns1; if the relation is false, it returns value 0.
- Relational operators are used in decision making and loops.
- A = 5 , B = 6;
- A==B;
- A!=B;



==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Relational Operators: Example



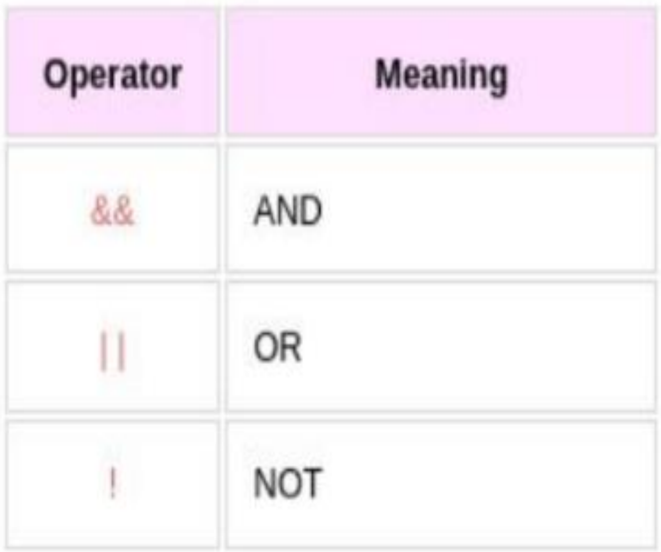
```c
#include <stdio.h>
int main()
{
int m=40, n=20;
if (m == n)
{
printf("m and n are equal");
}
else
{
printf("m and n are not equal");
}
return 0;
}
```

Classroom Assignment

- WAP to find a larger number among two numbers input by the user (use relational operator for the comparison).
- Hints:
  - A =6
  - B=7
  - If (A>B) {
    - Pri…. A is greater.
  - }
  - Else { b is greater

Logical Operators

- C provides three logical operators when we test more than one condition to make decisions.
- These are: && (meaning logical AND), | | (meaning logical OR) and ! (meaning logical NOT).



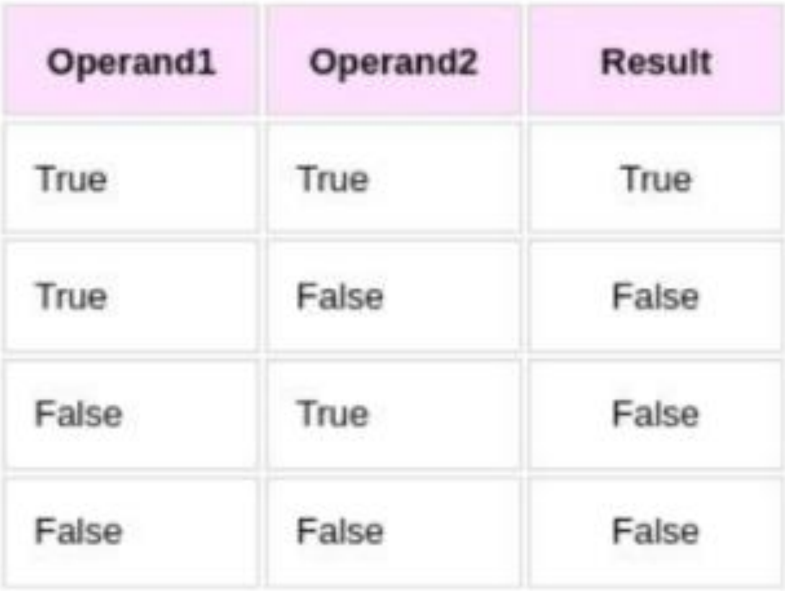
Operator	Meaning
&&	AND
||	OR
!	NOT

&& and | | are binary operators while ! is a unary operator.

Binary operators act upon a two operands to produce a new value.

AND (&&) operator

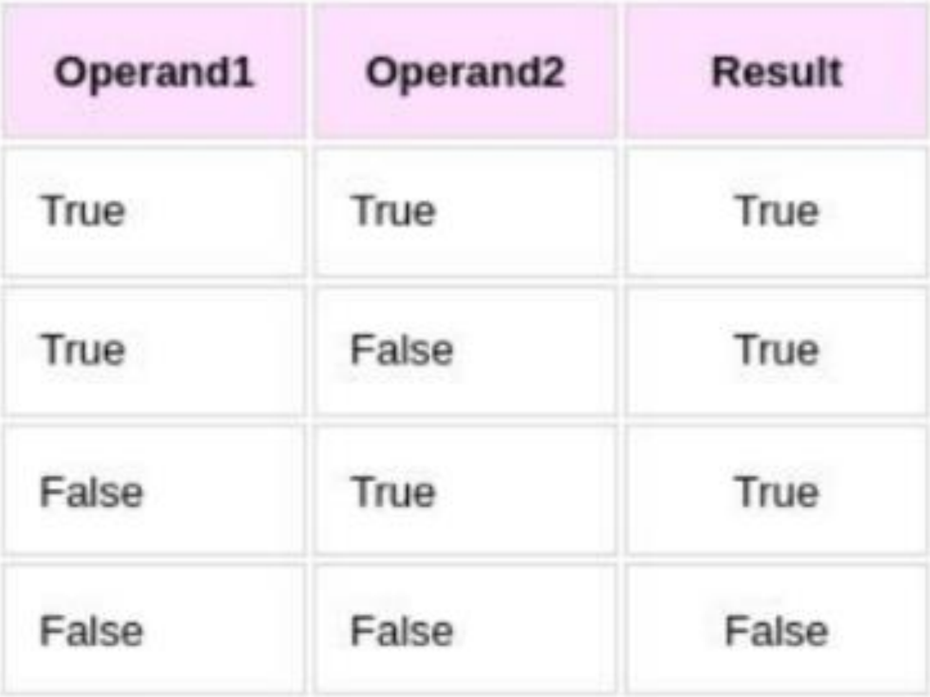
- The logical AND operator (&&) returns the boolean value true if both operands are true and returns false otherwise.
- Syntax: operand1 && operand2
- Truth table of AND operator is:



Operand1	Operand2	Result
True	True	True
True	False	False
False	True	False
False	False	False

OR (II) operator

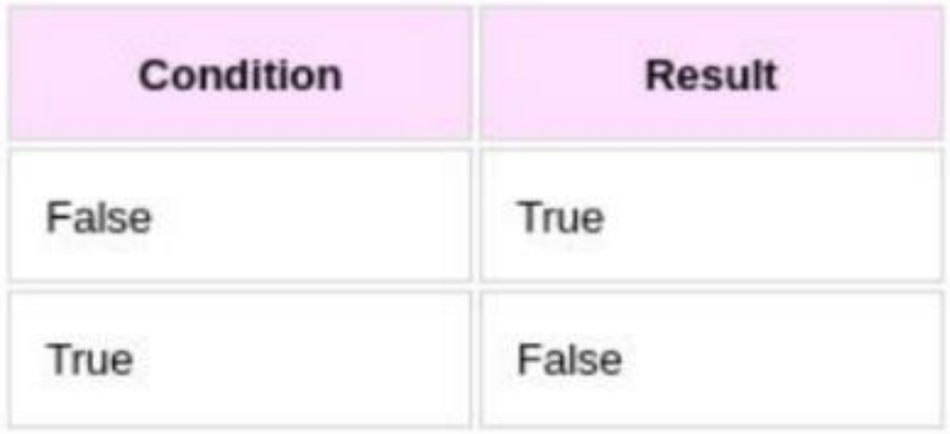
- The logical OR operator (| |) returns the boolean value true if either or both operands is true and returns false otherwise.
- Syntax: operand1 | | operand2
- Truth Table of OR operator is:



Operand1	Operand2	Result
True	True	True
True	False	True
False	True	True
False	False	False

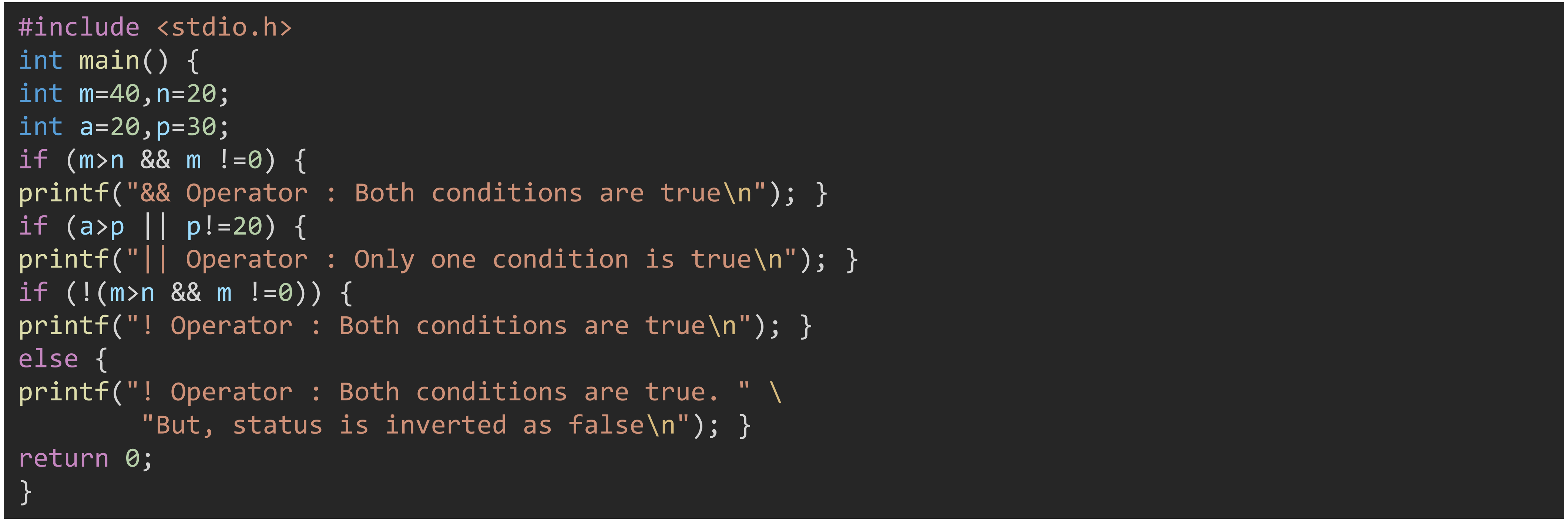
NOT (!) operator

- The logical NOT operator(!) negates the value of the condition.
- If the value of the condition is false then it gives the result true. If the value of the condition is true then it gives the result false.
- Syntax: !operand
- The truth table of logical NOT operator is:



Condition	Result
False	True
True	False

Logical Operator: Example



```
#include <stdio.h>
int main() {
int m=40,n=20;
int a=20,p=30;
if (m>n && m !=0) {
printf("&& Operator : Both conditions are true\n"); }
if (a>p || p!=20) {
printf("|| Operator : Only one condition is true\n"); }
if (!(m>n && m !=0)) {
printf("! Operator : Both conditions are true\n"); }
else {
printf("! Operator : Both conditions are true. " \
        "But, status is inverted as false\n"); }
return 0;
}
```

Conditional Operator

The Conditional Operator in C, also called a Ternary operator, is one of the Operators, which used in the decision-making process.

```c
#include <stdio.h>
int main()
{
int age; // variable declaration
printf("Enter your age");
scanf("%d",&age); // taking user input for age variable
(age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting")); // conditional operator
return 0;
}


Syntax:
(Text Expression)? statement1 : statement2;
```

Bitwise Operators

- The bitwise operators are the operators used to perform the operations on the data at the bit-level.
- When we perform the bitwise operations, then it is also known as bit-level programming.
- It consists of two digits, either 0 or 1.
- It is mainly used in numerical computations to make the calculations faster.
  - 1
  - 0000 0000 0000 0001
  - 256 128 64 32 16 8 4 2 1

Bitwise Operators in C

Operator	Meaning of Operator
&	Bitwise AND Operator
|	Bitwise OR Operator
^	Bitwise exclusive OR Operator
~	Bitwise NOT Operator (Unary Operator)
<<	Left Shift Operator
>>	Right Shift Operator

Bitwise AND Operator

- Bitwise AND operator is denoted by the single ampersand sign (&). Two integer operands are written on both sides of the (&) operator.
- If the corresponding bits of both the operands are 1, then the output of the bitwise AND operation is 1; otherwise, the output would be 0.

x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise AND Operator

We have two variables a and b.

```
   Int a =6;
   Int b=4;
```

The binary representation of the above two variables are given below:

```
………….8 4 2 1
   a = 0110
   b = 0100
   --------------------
      01 00
```

When we apply the bitwise AND operation in the above two variables, i.e., a&b, the output would be:

```
   Result = 0100
```

Example:

Checking for Odd and Even Numbers using Bitwise AND (&)

```
#include <stdio.h>
#include <stdlib.h>

int main(){
  int x=3;
  if(!(x&1)){
    printf("x is even");
  } else {
    printf("x is odd!");
  }
}
```

Checking if a number is a power of 2

```
#include <stdio.h>
#include <stdlib.h>

int main(){
  int a=32;
  if(a > 0 && (a & (a - 1)) == 0){
    printf("%d is a power of 2", a);
  }
  return EXIT_SUCCESS;
}
```

Bitwise OR operator

- The bitwise OR operator is represented by a single vertical sign (|).
- Two integer operands are written on both sides of the (|) symbol. If the bit value of any of the operand is 1, then the output would be 1, otherwise 0.

x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise OR Operator

We consider two variables,

a = 23;

b = 10;

The binary representation of the above two variables would be:

a = 0001 0111

b = 0000 1010

When we apply the bitwise OR operator in the above two variables, i.e., a|b , then the

output would be:

Result = 0001 1111

Bitwise exclusive OR operator

- The ^ operator is bitwise XOR. The usual bitwise OR operator is inclusive OR.
- XOR is true only if exactly one of the two bits is true.
- Two operands are written on both sides of the exclusive OR operator. If the corresponding bit of any of the operand is 1 then the output would be 1, otherwise 0.

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Exclusive OR

We consider two variables a and b,

a = 12;

b = 10;

The binary representation of the above two variables would be:

a = 0000 1100

b = 0000 1010

When we apply the bitwise exclusive OR operator in the above two variables (a^b), then

the result would be:

Result = 0000 0110

Example:

1. WAP to swap two numbers without using another variable.

```
// C code to swap using XOR
#include <stdio.h>
int main()
{
    int x = 10, y = 5;

    // Code to swap 'x' (1010) and 'y' (0101)
    x = x ^ y; // x now becomes 15 (1111)
    y = x ^ y; // y becomes 10 (1010)
    x = x ^ y; // x becomes 5 (0101)

    printf("After Swapping: x = %d, y = %d", x, y);

    return 0;
}
```

Complement(~), Left Shift (<<), Right Shift (>>)

1. The **<< (left shift)** in C or C++ takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

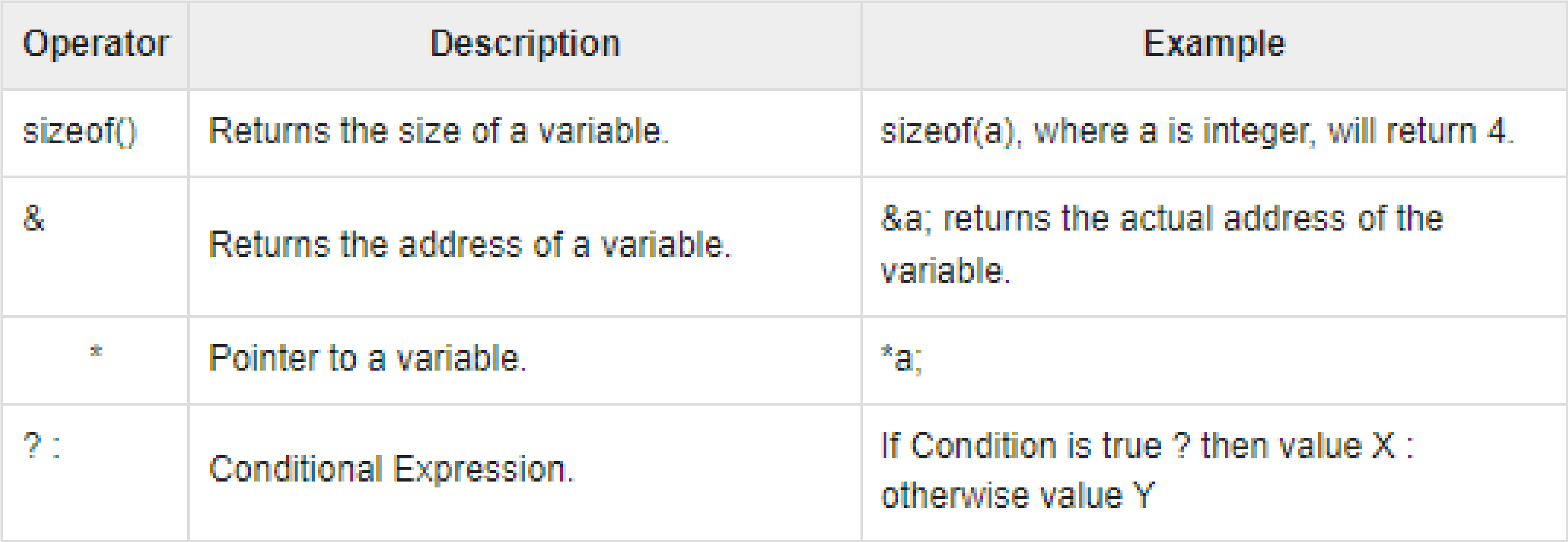
2. The **>> (right shift)** in C or C++ takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

3. The **~ (bitwise NOT)** in C or C++ takes one number and inverts all bits of it.

Complement(~), Left Shift (<<), Right Shift (>>)

```c
// C Program to demonstrate use of
bitwise operators
#include <stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;

    // The result is 00000001
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a & b);

    // The result is 00001101
    printf("a|b = %d\n", a | b);

   // The result is 00001100
    printf("a^b = %d\n", a ^ b);

    // The result is 11111010
    printf("~a = %d\n", a = ~a);

    // The result is 00010010
    printf("b<<1 = %d\n", b << 1);

    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);

    return 0;

}
```

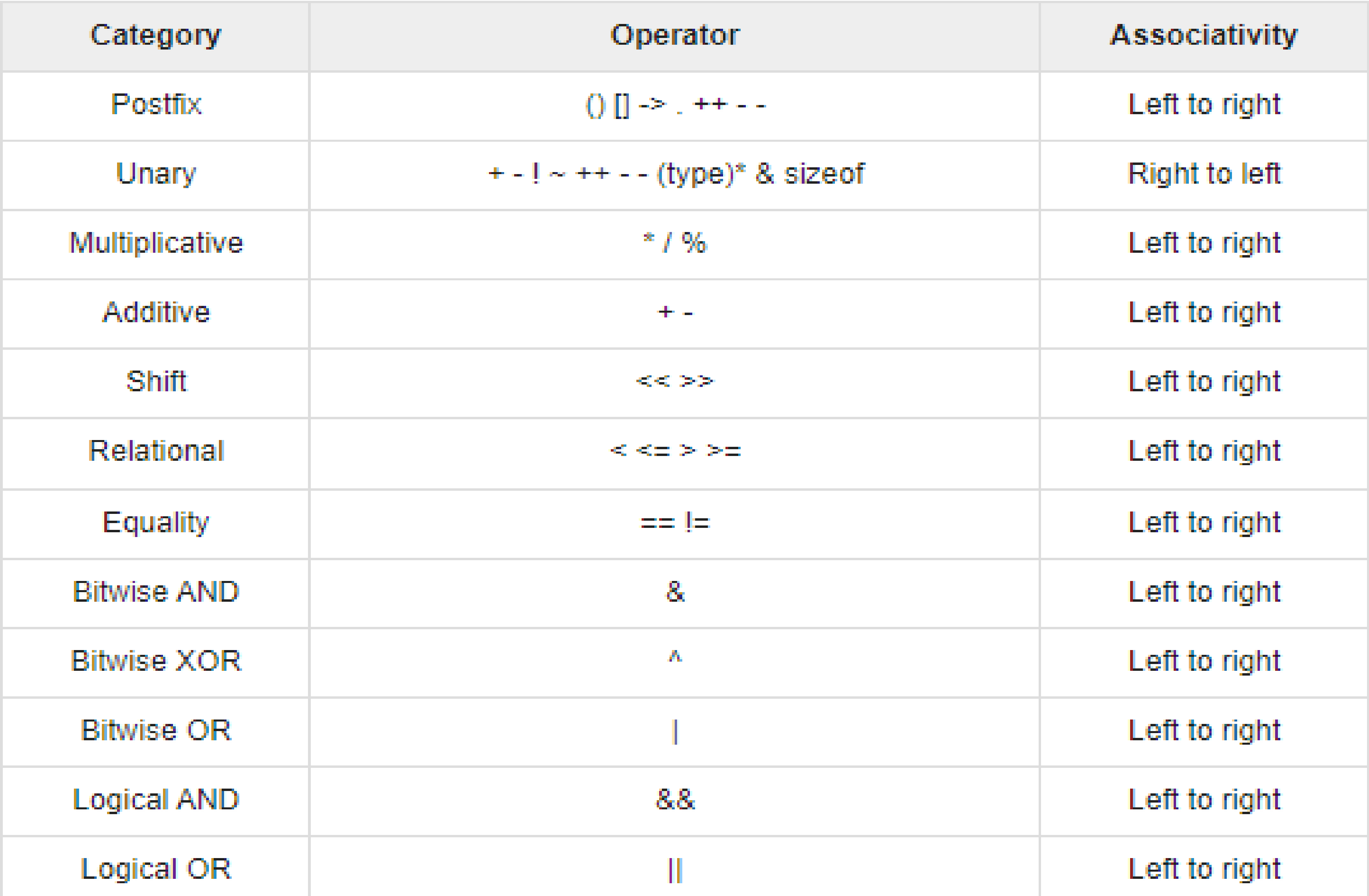
Special Operators



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 Precedence in C

- 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.



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

Any queries???