SIEC: BASIC C PROGRAMMING

L #07: C OPERATORS

Seung Beop Lee

School of International Engineering and Science

CHONBUK NATIONAL UNIVERSITY

Outline

- C Operators

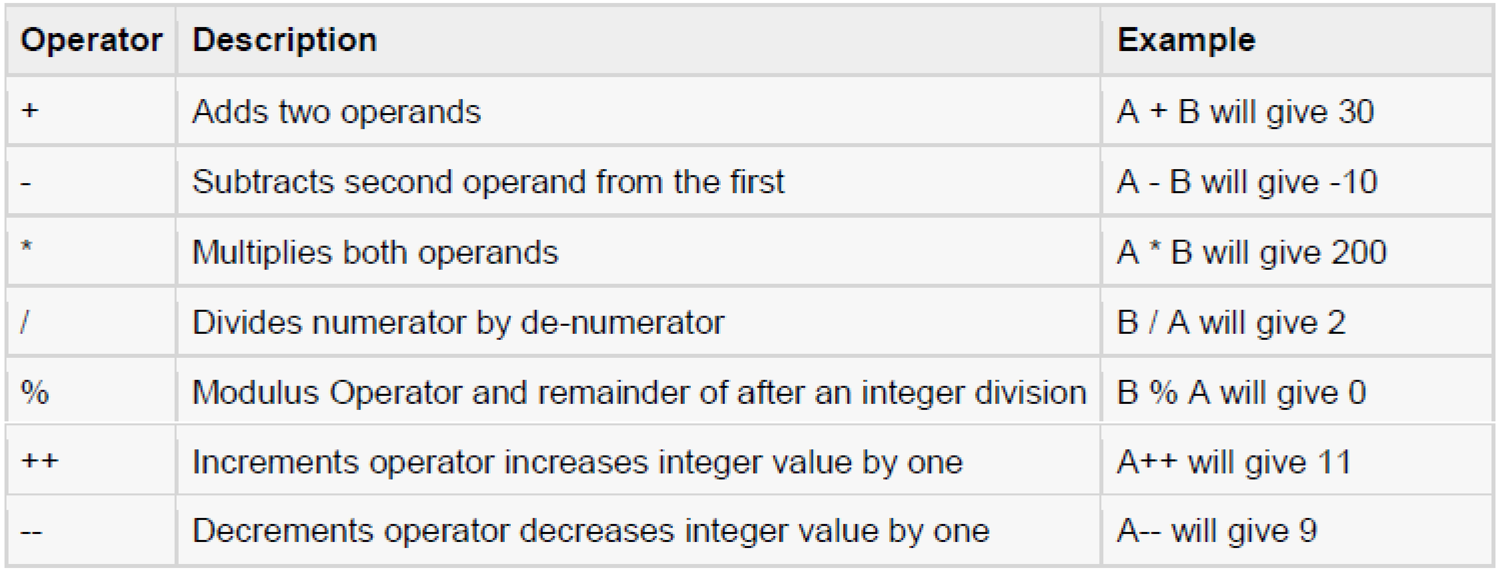
C Operators

C Operators

- Operators
  - **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 6 types of operators:
    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
    - Bitwise Operators
    - Assignment Operators
    - Misc Operators

C Operators

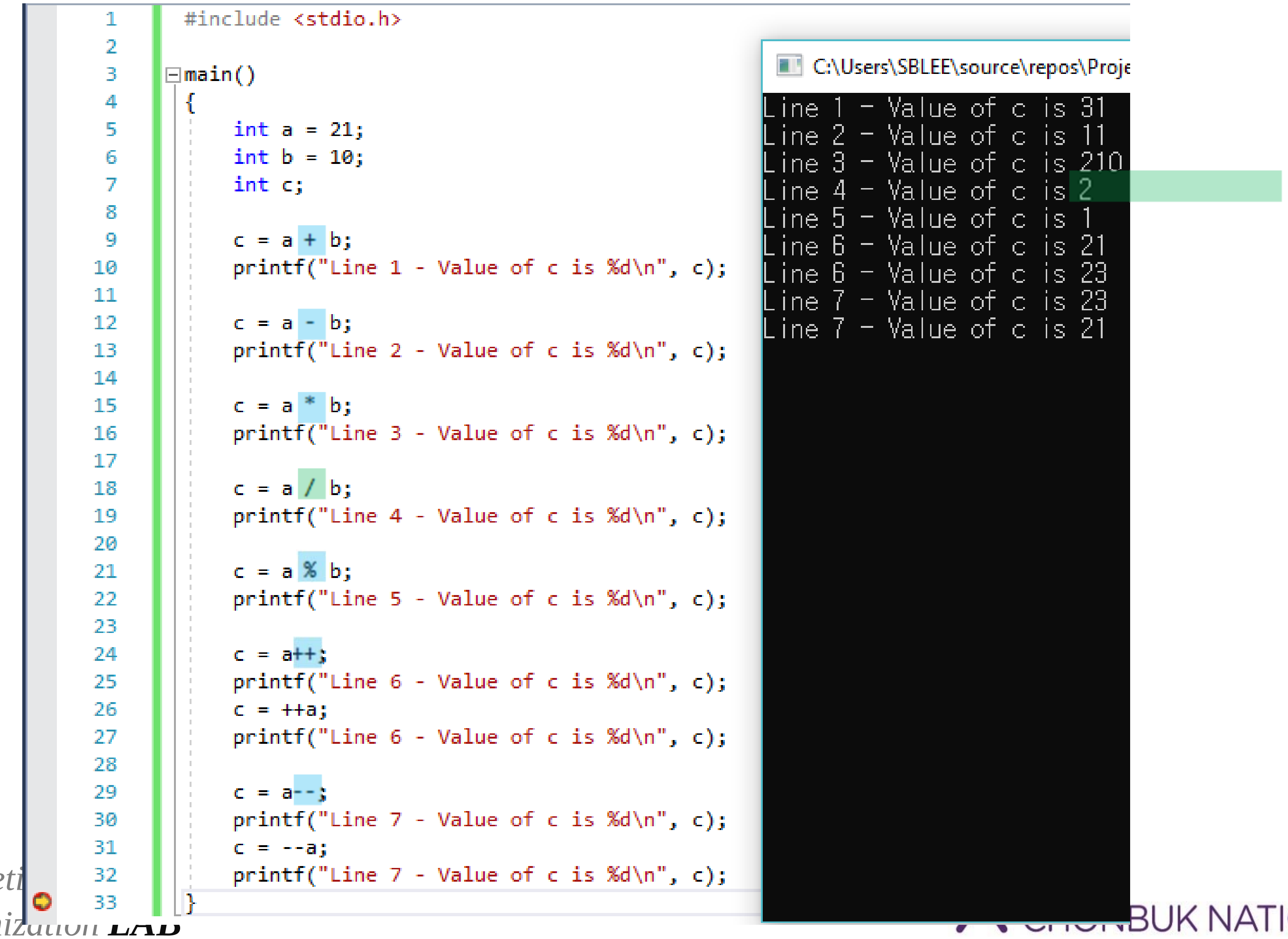
- Arithmetic Operators
  - The following table shows all the arithmetic operators supported by C language.
  - Assume variable A holds 10 and variable B holds 20.



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
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

C Operators

- Arithmetic Operators
  - In order to understand all the arithmetic operators available in C programming language, try the following example in the self-coding class.



```
#include <stdio.h>

main()
{
    int a = 21;
    int b = 10;
    int c;

    c = a + b;
    printf("Line 1 - Value of c is %d\n", c);

    c = a - b;
    printf("Line 2 - Value of c is %d\n", c);

    c = a * b;
    printf("Line 3 - Value of c is %d\n", c);

    c = a / b;
    printf("Line 4 - Value of c is %d\n", c);

    c = a % b;
    printf("Line 5 - Value of c is %d\n", c);

    c = a++;
    printf("Line 6 - Value of c is %d\n", c);
    c = ++a;
    printf("Line 6 - Value of c is %d\n", c);

    c = a--;
    printf("Line 7 - Value of c is %d\n", c);
    c = --a;
    printf("Line 7 - Value of c is %d\n", c);
}
```

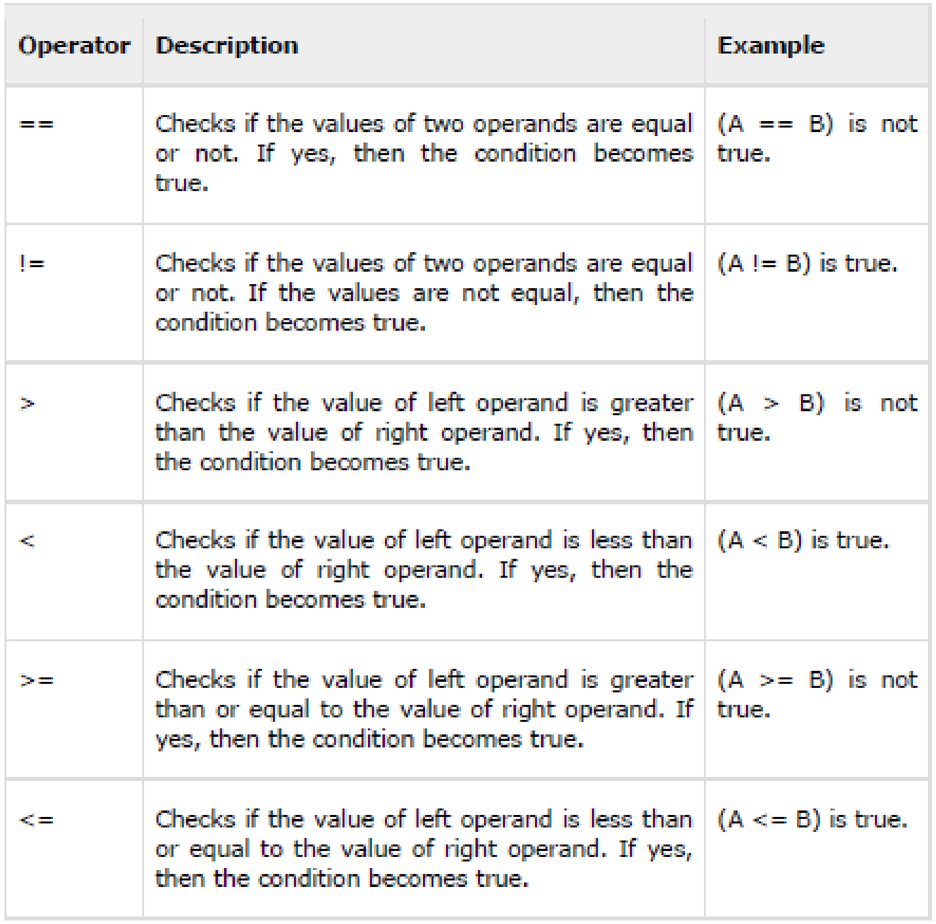
C:\Users\SBLEE\source\repos\Proje

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 6 - Value of c is 23
Line 7 - Value of c is 23
Line 7 - Value of c is 21
```

C Operators

Relational Operators

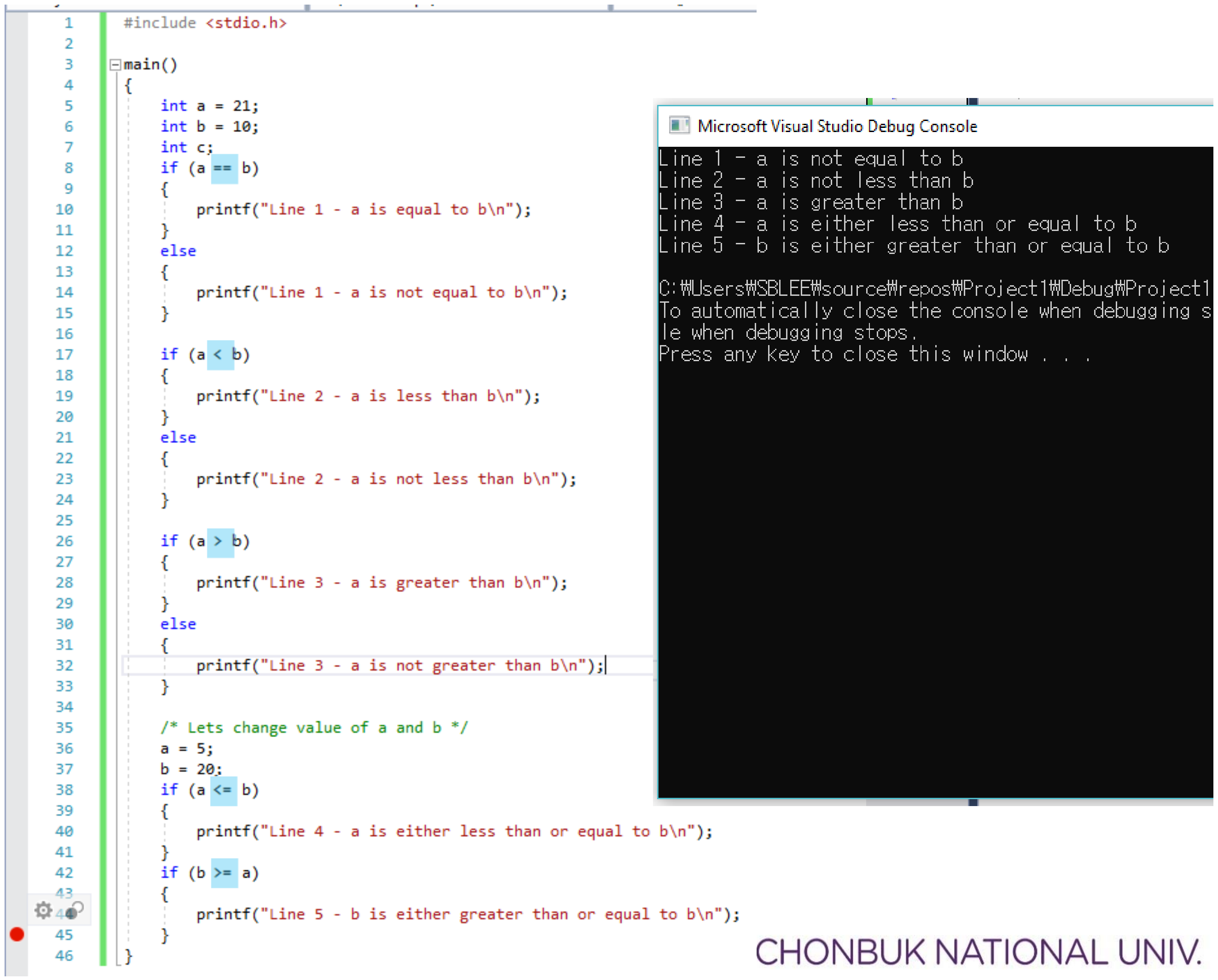
- The following table shows all the relational operators supported by C.
- The relational operators is used to make a decision in C programming.
- Assume variable A holds 10 and variable B holds 20, then:



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.

C Operators

- Relational Operators
  - The following example is in order to understand all the relational operators available in C:



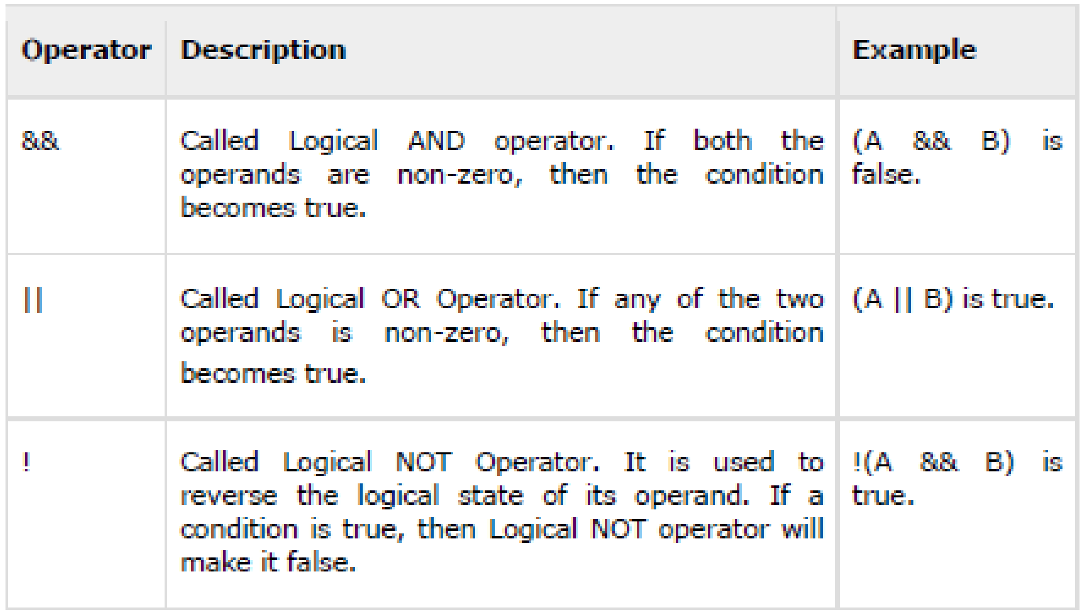
```c
#include <stdio.h>

main()
{
    int a = 21;
    int b = 10;
    int c;
    if (a == b)
    {
        printf("Line 1 - a is equal to b\n");
    }
    else
    {
        printf("Line 1 - a is not equal to b\n");
    }

    if (a < b)
    {
        printf("Line 2 - a is less than b\n");
    }
    else
    {
        printf("Line 2 - a is not less than b\n");
    }

    if (a > b)
    {
        printf("Line 3 - a is greater than b\n");
    }
    else
    {
        printf("Line 3 - a is not greater than b\n");
    }

    /* Lets change value of a and b */
    a = 5;
    b = 20;
    if (a <= b)
    {
        printf("Line 4 - a is either less than or equal to b\n");
    }
    if (b >= a)
    {
        printf("Line 5 - b is either greater than or equal to b\n");
    }
}
```

Microsoft Visual Studio Debug Console

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to b
Line 5 - b is either greater than or equal to b

C:₩Users₩SBLEE₩source₩repos₩Project1₩Debug₩Project1
To automatically close the console when debugging s
le when debugging stops.
Press any key to close this window . . .
```

C Operators

- Logical Operators
  - The following table shows all the logical operators supported by C.
  - The logical operator operators is also used to make a decision in C programming.
  - When making a decision in C, 1 means true and 0 means false in the logical operators.
  - Assume variable A holds 1 and variable B holds 0, then:

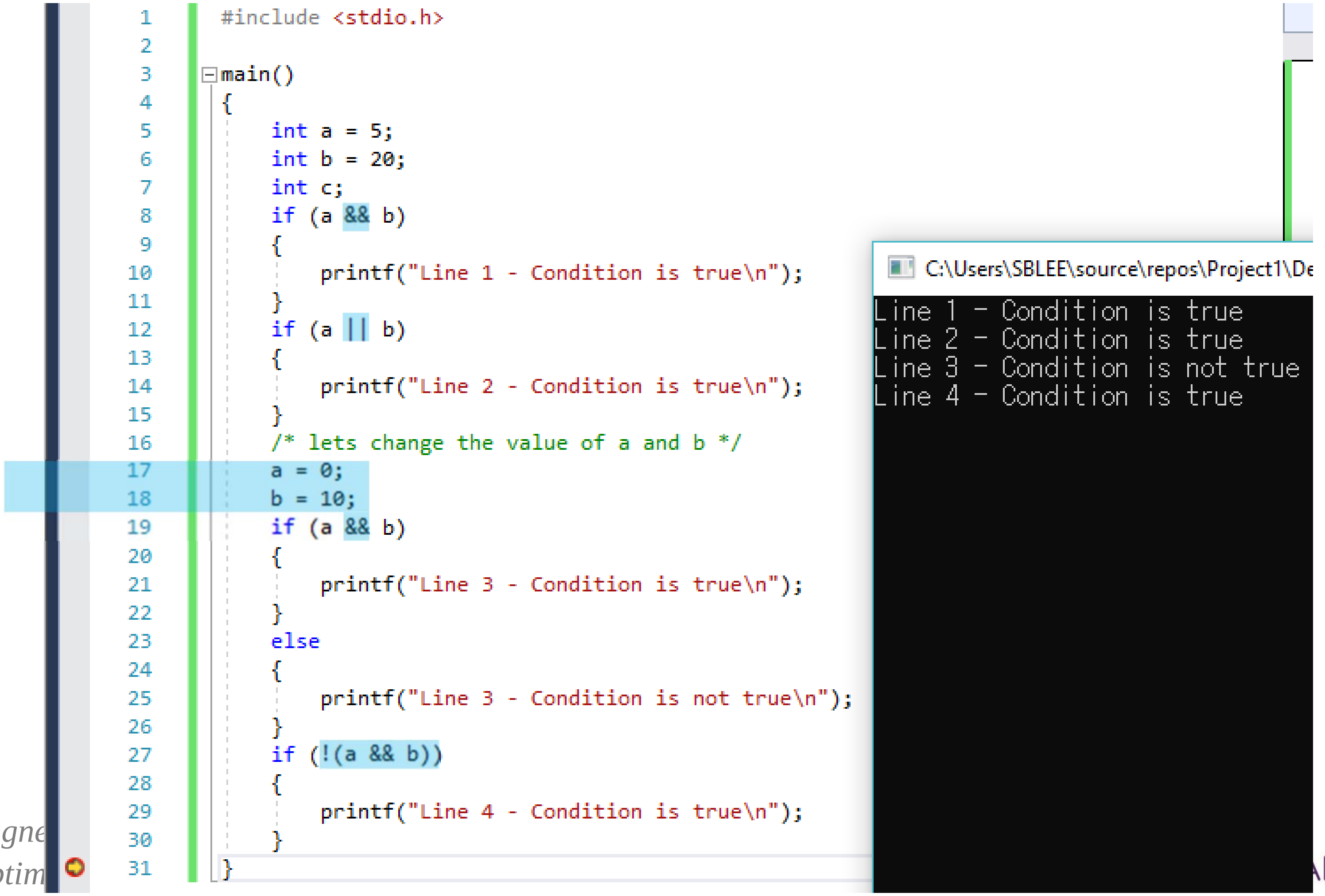


Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. 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.

- ➢ A && B == 0. Therefore, A && B is false.
- ➢ A || B == 0. Therefore, A || B is true.
- ➢ !(A && B) means !(false), i.e., !(0). Therefore, !(A && B) == !(0) == 1. So, !(A && B) is true.

C Operators

- Logical Operators
  - The following example is in order to understand all the logical operators available in C:

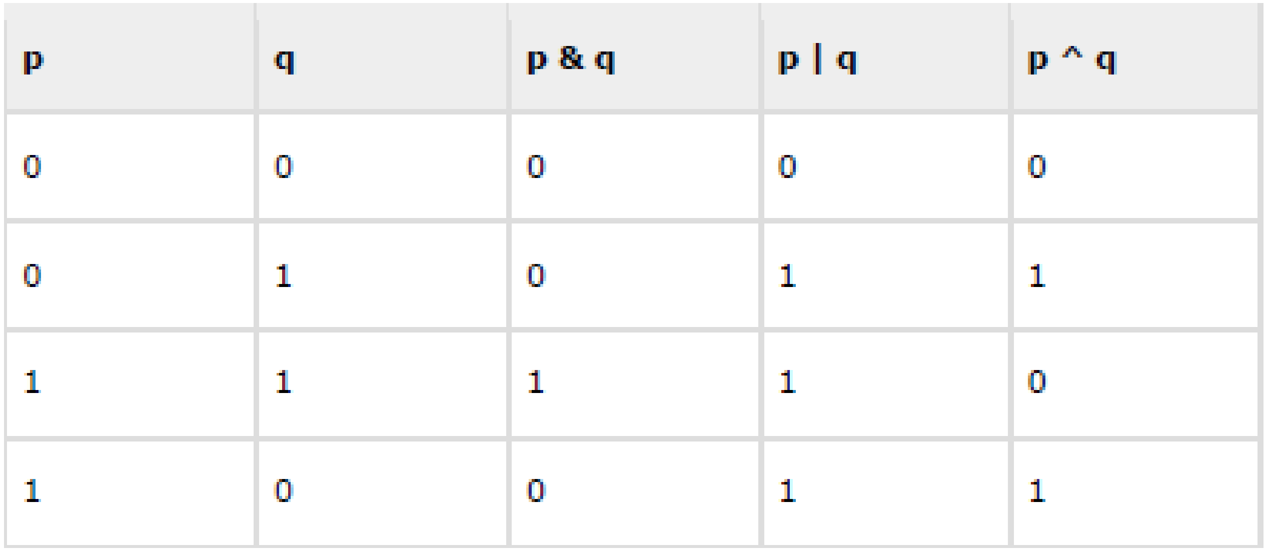


```c
#include <stdio.h>

main()
{
    int a = 5;
    int b = 20;
    int c;
    if (a && b)
    {
        printf("Line 1 - Condition is true\n");
    }
    if (a || b)
    {
        printf("Line 2 - Condition is true\n");
    }
    /* lets change the value of a and b */
    a = 0;
    b = 10;
    if (a && b)
    {
        printf("Line 3 - Condition is true\n");
    }
    else
    {
        printf("Line 3 - Condition is not true\n");
    }
    if (!(a && b))
    {
        printf("Line 4 - Condition is true\n");
    }
}
```

```
Line 1 - Condition is true
Line 2 - Condition is true
Line 3 - Condition is not true
Line 4 - Condition is true
```

C Operators

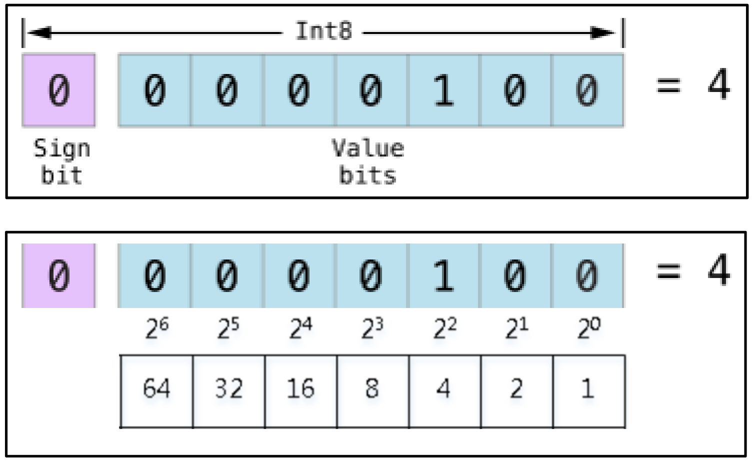
- Bitwise Operators
  - Bitwise operators work on bits and perform bit-by-bit operation.
  - The Bitwise operation is only a rule.
  - The truth table for &, |, and ^ is as follows:

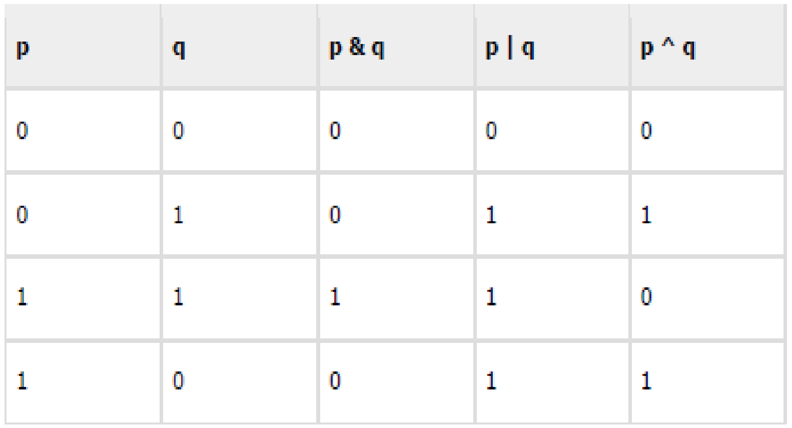


p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

C Operators

- Bitwise Operators
  - Assume A = 60 and B = 13; in binary format, they will be 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 = 32+16+8+4 = 60

B = 0000 1101 = 8+4+1 = 13

A&B = 0000 1100

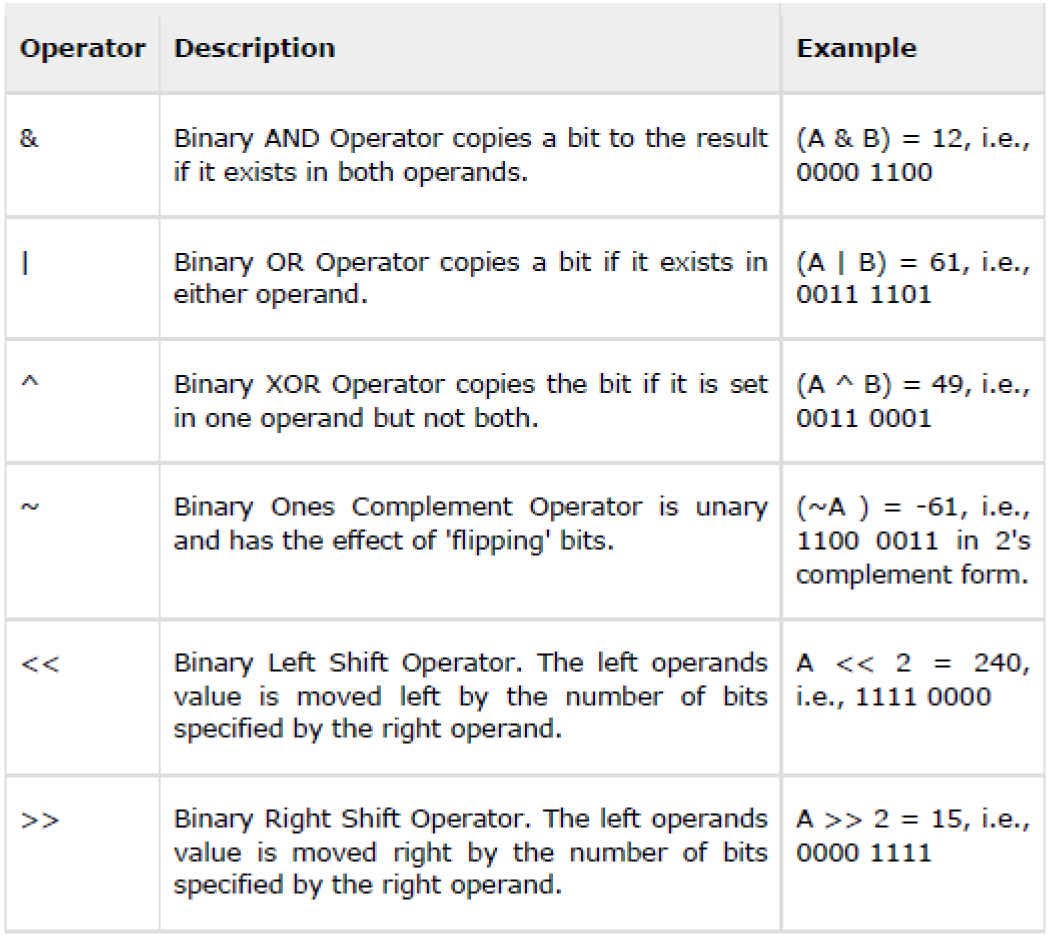
A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

C Operators

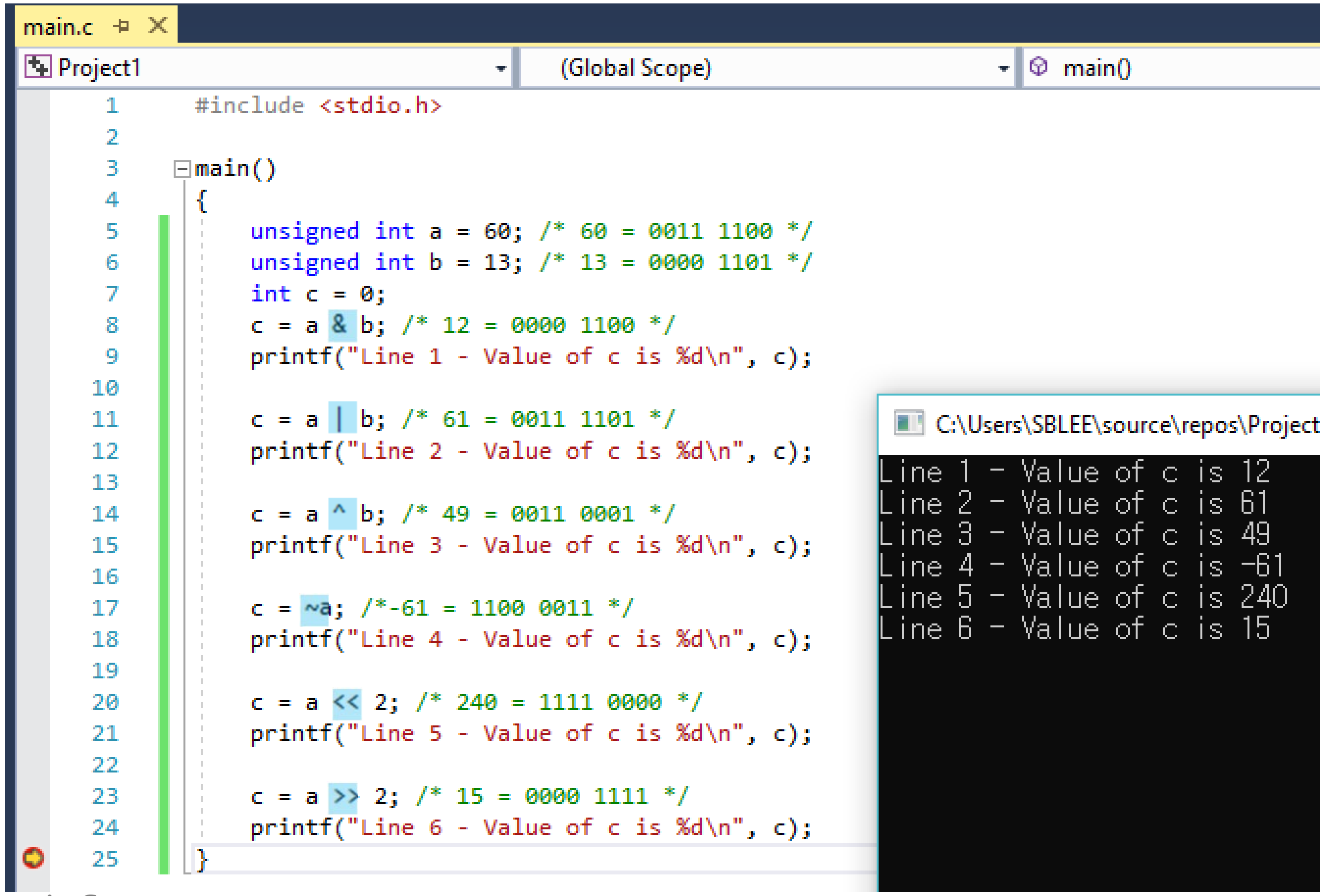
- Bitwise Operators
  - The following table shows all the bitwise operators supported by C.
  - Assume variable A holds 60 and variable B holds 13, then:



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 Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -61, i.e., 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 = 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

C Operators

- Bitwise Operators
  - The following example is in order to understand all the bitwise operators available in C:

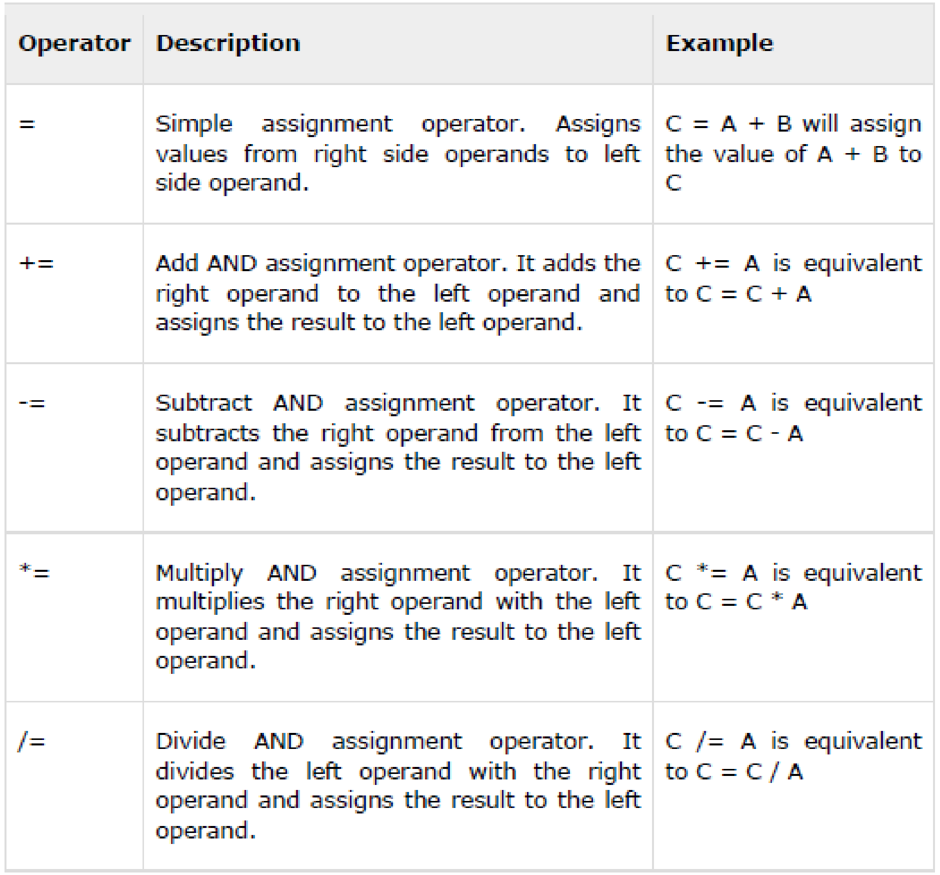


```c
#include <stdio.h>

main()
{
    unsigned int a = 60; /* 60 = 0011 1100 */
    unsigned 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);

    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);
}
```

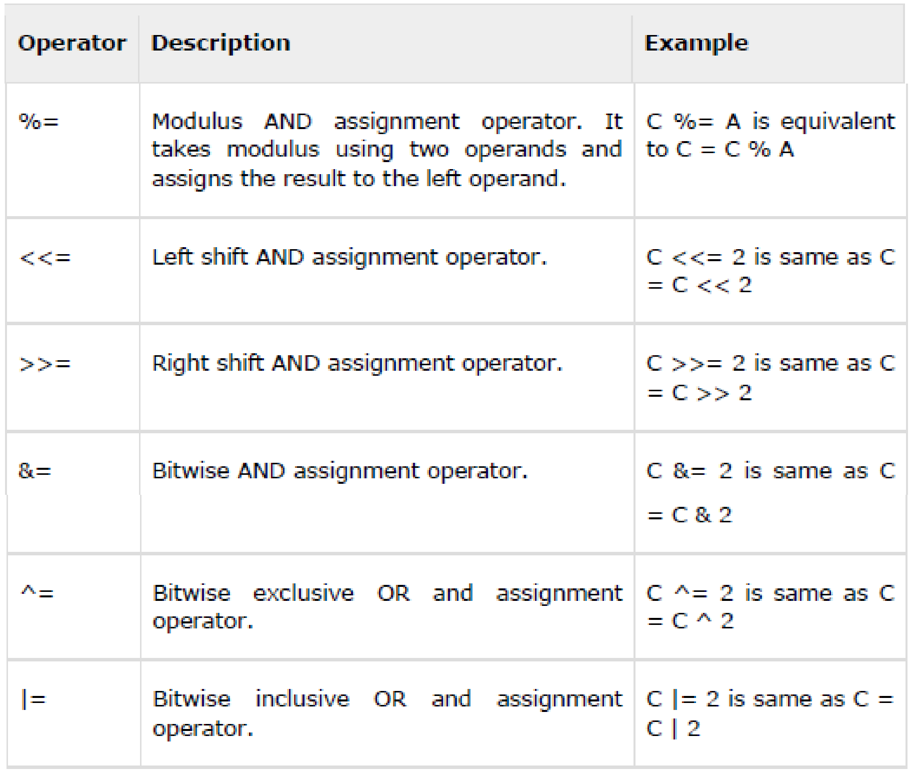
```
Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15
```

C Operators

- Assignment Operators
  - The following tables lists the assignment operators supported by the C language.



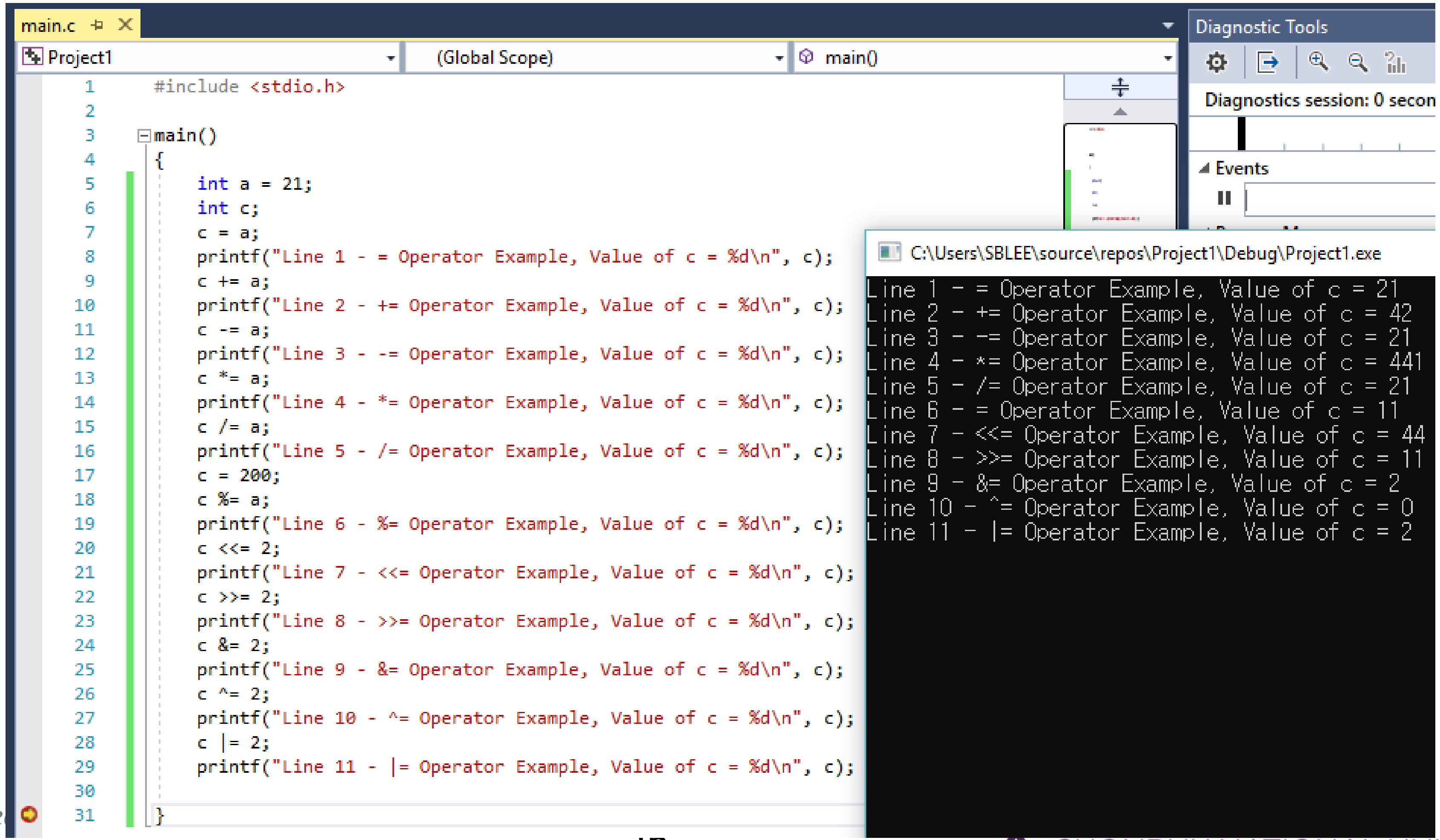
Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent 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



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

C Operators

- Assignment Operators
  - The following example is in order to understand all the assignment operators available in C:

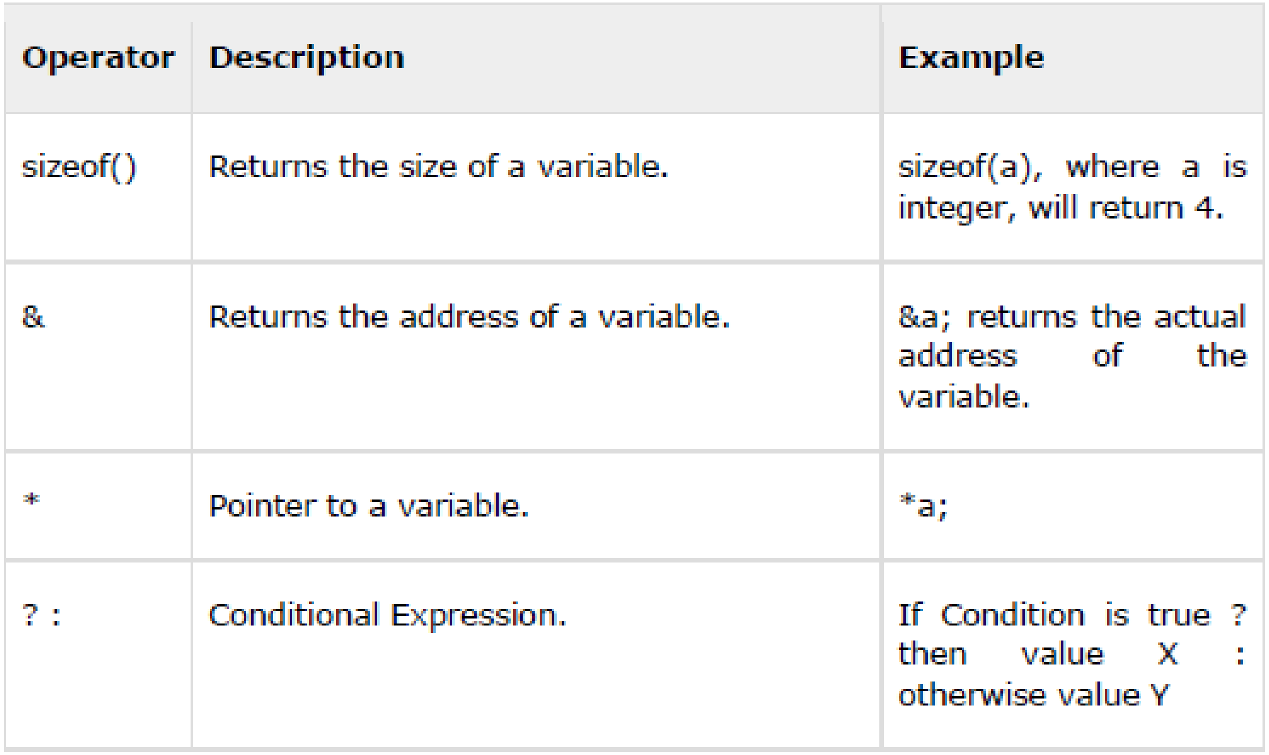

main.c
Project1
(Global Scope)
main()
Diagnostic Tools
Diagnostics session: 0 secon
Events
C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```c
#include <stdio.h>

main()
{
    int a = 21;
    int c;
    c = a;
    printf("Line 1 - = Operator Example, Value of c = %d\n", c);
    c += a;
    printf("Line 2 - += Operator Example, Value of c = %d\n", c);
    c -= a;
    printf("Line 3 - -= Operator Example, Value of c = %d\n", c);
    c *= a;
    printf("Line 4 - *= Operator Example, Value of c = %d\n", c);
    c /= a;
    printf("Line 5 - /= Operator Example, Value of c = %d\n", c);
    c = 200;
    c %= a;
    printf("Line 6 - %= Operator Example, Value of c = %d\n", c);
    c <<= 2;
    printf("Line 7 - <<= Operator Example, Value of c = %d\n", c);
    c >>= 2;
    printf("Line 8 - >>= Operator Example, Value of c = %d\n", c);
    c &= 2;
    printf("Line 9 - &= Operator Example, Value of c = %d\n", c);
    c ^= 2;
    printf("Line 10 - ^= Operator Example, Value of c = %d\n", c);
    c |= 2;
    printf("Line 11 - |= Operator Example, Value of c = %d\n", c);

}
```

```
Line 1 - = Operator Example, Value of c = 21
Line 2 - += Operator Example, Value of c = 42
Line 3 - -= Operator Example, Value of c = 21
Line 4 - *= Operator Example, Value of c = 441
Line 5 - /= Operator Example, Value of c = 21
Line 6 - = Operator Example, Value of c = 11
Line 7 - <<= Operator Example, Value of c = 44
Line 8 - >>= Operator Example, Value of c = 11
Line 9 - &= Operator Example, Value of c = 2
Line 10 - ^= Operator Example, Value of c = 0
Line 11 - |= Operator Example, Value of c = 2
```

C Operators

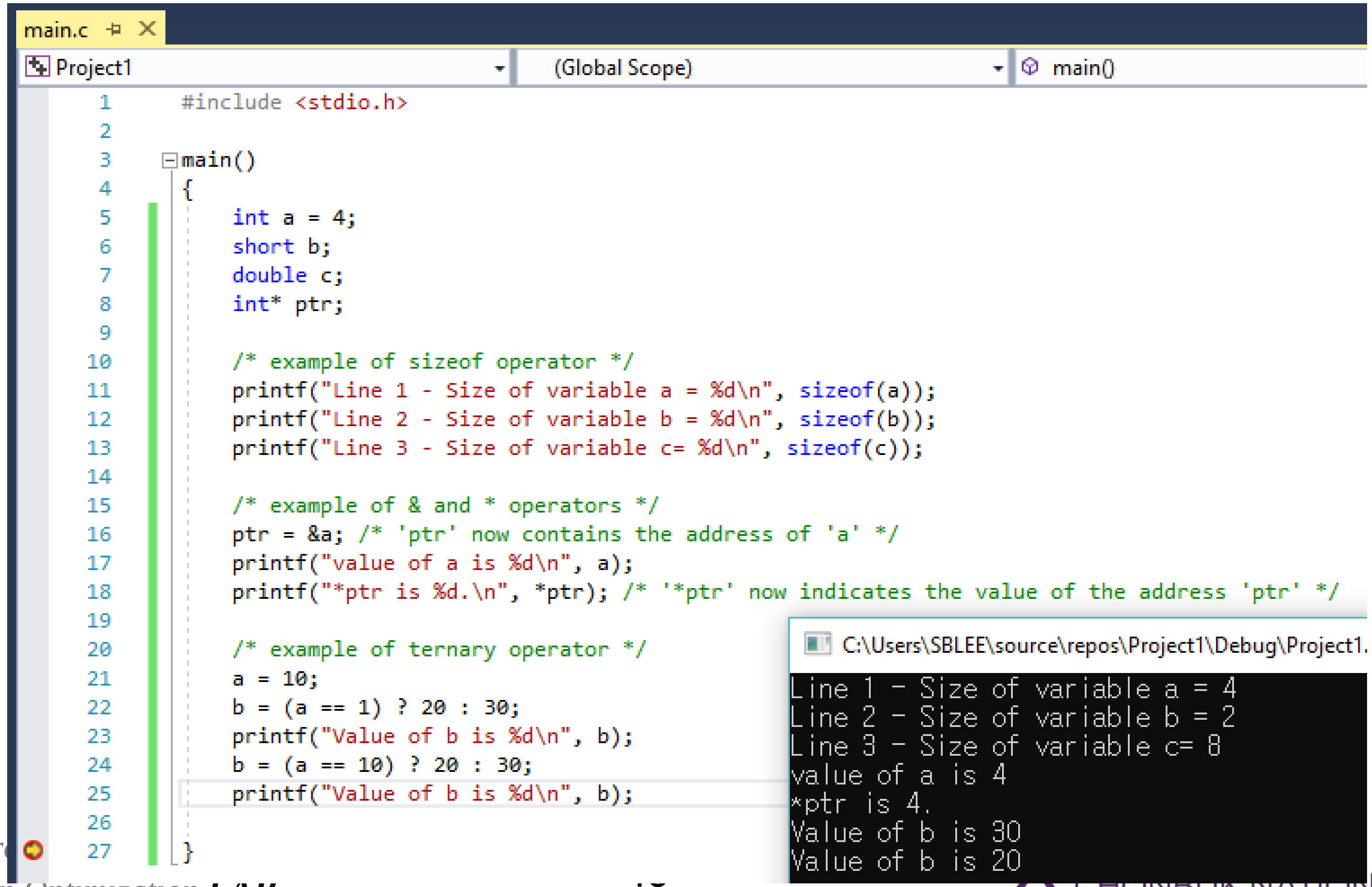
- Misc (miscellaneous) Operators
  - Besides the operators discussed above, there are a few other important operators including sizeof() and ? : supported by the C Language.



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

C Operators

- Misc (miscellaneous) Operators
  - The following example is in order to understand all the miscellaneous operators available in C:



main.c
Project1 (Global Scope) main()

```c
#include <stdio.h>

main()
{
    int a = 4;
    short b;
    double c;
    int* ptr;

    /* 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));

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

    /* 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);

}
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.

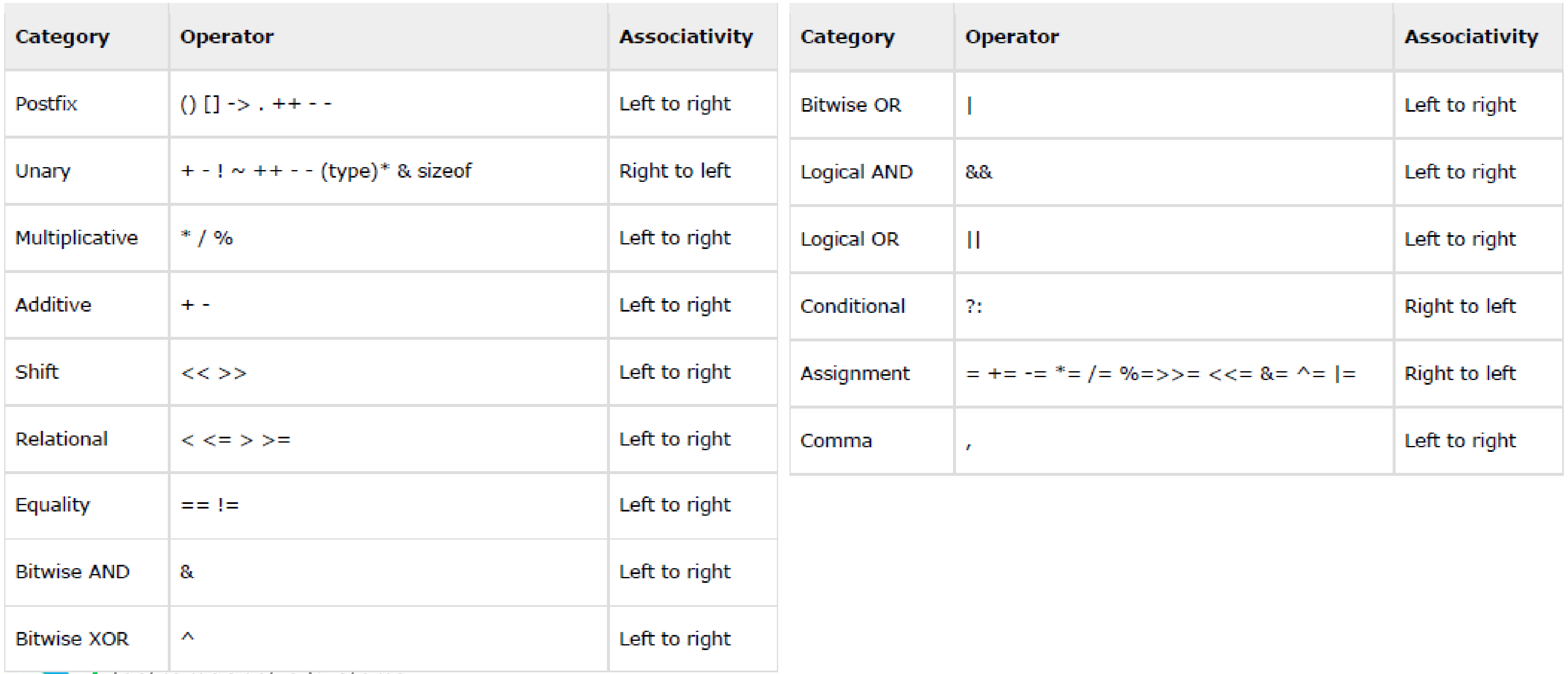
```
Line 1 - Size of variable a = 4
Line 2 - Size of variable b = 2
Line 3 - Size of variable c= 8
value of a is 4
*ptr is 4.
Value of b is 30
Value of b is 20
```

C Operators

- 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.
  - 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.

C Operators

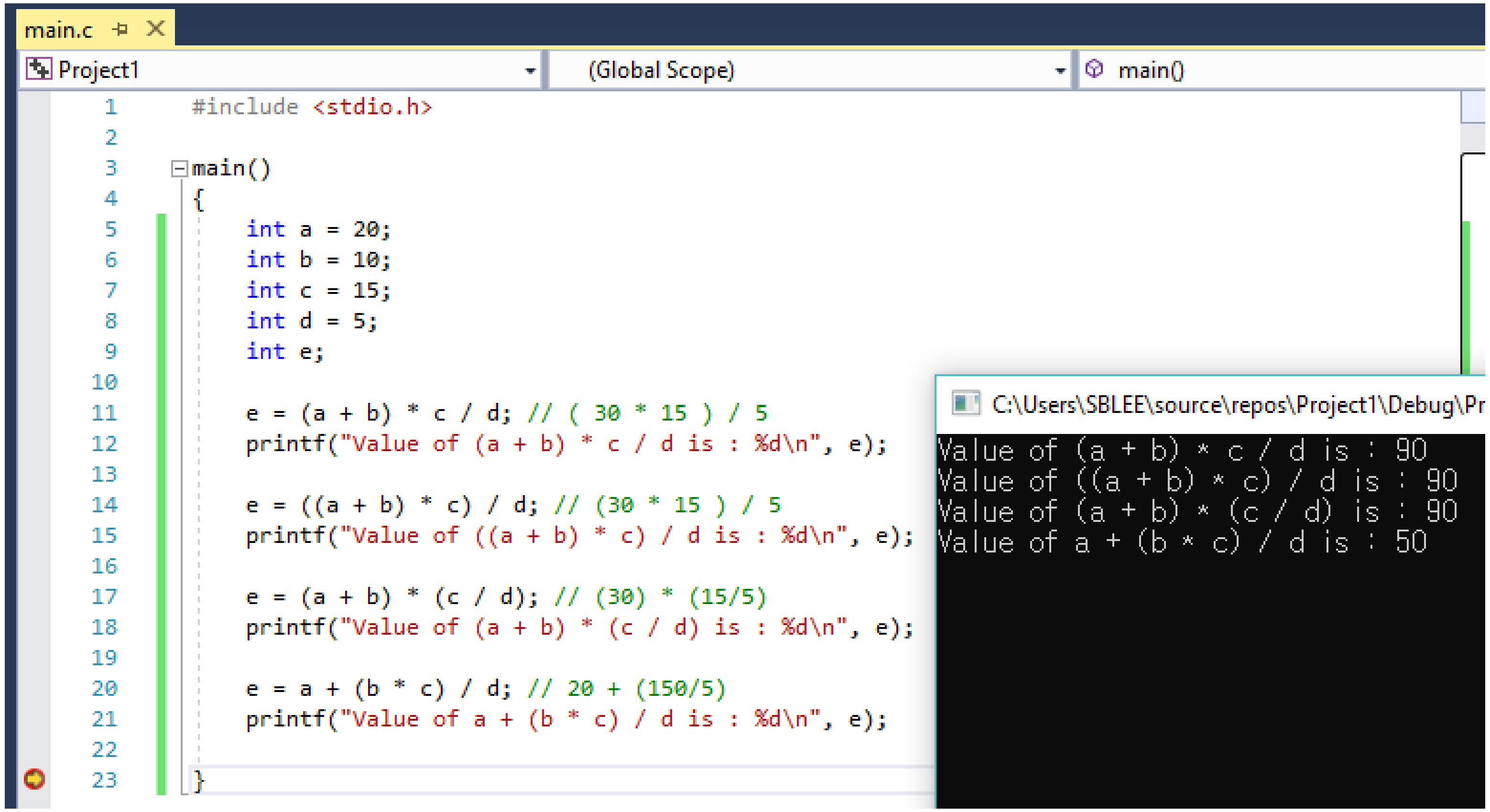
- Operators Precedence in C
  - Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.
  - Within an expression, higher precedence operators will be evaluated first.



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

C Operators

- Operators Precedence in C
  - The following example is in order to understand operator precedence in C:



```c
#include <stdio.h>

main()
{
    int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;

    e = (a + b) * c / d; // ( 30 * 15 ) / 5
    printf("Value of (a + b) * c / d is : %d\n", e);

    e = ((a + b) * c) / d; // (30 * 15 ) / 5
    printf("Value of ((a + b) * c) / d is : %d\n", e);

    e = (a + b) * (c / d); // (30) * (15/5)
    printf("Value of (a + b) * (c / d) is : %d\n", e);

    e = a + (b * c) / d; // 20 + (150/5)
    printf("Value of a + (b * c) / d is : %d\n", e);

}
```

C:\Users\SBLEE\source\repos\Project1\Debug\Pr

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50
```

Thank You

(seungbeop.lee@gmail.com)

Self-coding class

Self-coding class for the lecture 5 and 6

- Self-coding class
  - After the self-coding class, please submit your codes for all the examples we covered in the **lecture 5 and 6** by e-mail (seungbeop.lee@gmail.com).
  - If you don't submit your codes for all the examples of the **lecture 3 and 4**, please submit them by e-mail (seungbeop.lee@gmail.com).