Operators

An operator is a symbol that instructs C to perform some operation, or action, on one or more operands. An operand is something that an operator acts on. In C, all operands are expressions. C operators fall into several categories:

* The assignment operator
* Mathematical operators
* Relational operators
* Logical operators

The Assignment Operator

The assignment operator is the equal sign (=). Its use in programming is somewhat different from its use in regular math. If you write x = y; in a C program, it doesn't mean "x is equal to y." Instead, it means "assign the value of y to x." In a C assignment statement, the right side can be any expression, and the left side must be a variable name. Thus, the form is as follows:

variable = expression;

When executed, expression is evaluated, and the resulting value is assigned to variable.

Mathematical Operators

C's mathematical operators perform mathematical operations such as addition and subtraction. C has two unary mathematical operators and five binary mathematical operators.

Unary Mathematical Operators

The unary mathematical operators are so named because they take a single operand. C has two unary mathematical operators, listed here.

C's unary mathematical operators.

Operator	*Symbol*	*Action*	*Examples*
Negate	-	Negates the value	-x
Increment	++	Increments the operand by one	++x, x++
Decrement	--	Decrements the operand by one	--x, x--

The increment and decrement operators can be used only with variables, not with constants. The operation performed is to add one to or subtract one from the operand. In other words, the statements ++x; --y; are the equivalent of these statements:

x = x + 1; y = y - 1;

You should note from Table 4.1 that either unary operator can be placed before its operand (prefix mode) or after its operand (postfix mode). These two modes are not equivalent. They differ in terms of when the increment or decrement is performed:

* When used in prefix mode, the increment and decrement operators modify their operand before it's used.

* When used in postfix mode, the increment and decrement operators modify their operand after it's used.

An example should make this clearer. Look at these two statements:

```
x = 10;          y = x++;
```

After these statements are executed, x has the value 11, and y has the value 10. The value of x was assigned to y, and then x was incremented. In contrast, the following statements result in both y and x having the value 11. x is incremented, and then its value is assigned to y.

```
x = 10;          y = ++x;
```

Remember that = is the assignment operator, not a statement of equality. As an analogy, think of = as the "photocopy" operator. The statement y = x means to copy x into y. Subsequent changes to x, after the copy has been made, have no effect on y.

Binary Mathematical Operators

C's binary operators take two operands. The binary operators, which include the common mathematical operations found on a calculator, are listed.

C's binary mathematical operators

Operator	*Symbol*	*Action*	*Example*
Addition	+	Adds two operands	x + y
Subtraction	-	Subtracts the second operand from the first operand	x –y
Multiplication	*	Multiplies two operands	x * y
Division	/	Divides the first operand by the second operand	x / y
Modulus	%	Gives the remainder	x % y

Operator Precedence and Parentheses

In an expression that contains more than one operator, what is the order in which operations are performed? The importance of this question is illustrated by the following assignment statement:

```
x = 4 + 5 * 3;
```

Performing the addition first results in the following, and x is assigned the value 27. In contrast, if the multiplication is performed first, you have the following, and x is assigned the value 19.

Clearly, some rules are needed about the order in which operations are performed. This order, called operator precedence, is strictly spelled out in C. Each operator has a specific precedence. When an expression is evaluated, operators with higher precedence are

performed first. Below is the list of precedence of C's mathematical operators. Number 1 is the highest precedence and thus is evaluated first.

The precedence of C's mathematical operators

Operators	*Relative Precedence*
++ --	1
* / %	2
+ -	3

Looking at it, you can see that in any C expression, operations are performed in the following order:

* Unary increment and decrement
* Multiplication, division, and modulus
* Addition and subtraction

If an expression contains more than one operator with the same precedence level, the operators are performed in left-to-right order as they appear in the expression. For example, in the following expression, the % and * have the same precedence level, but the % is the leftmost operator, so it is performed first:

12 % 5 * 2

Returning to the previous example, you see that the statement x = 4 + 5 * 3; assigns the value 19 to x because the multiplication is performed before the addition.

What if the order of precedence doesn't evaluate your expression as needed? Using the previous example, what if you wanted to add 4 to 5 and then multiply the sum by 3? C uses parentheses to modify the evaluation order. A subexpression enclosed in parentheses is evaluated first, without regard to operator precedence. Thus, you could write
x = (4 + 5) * 3;

The expression 4 + 5 inside parentheses is evaluated first, so the value assigned to x is 27. You can use multiple and nested parentheses in an expression. When parentheses are nested, evaluation proceeds from the innermost expression outward. Look at the following complex expression: x = 25 - (2 * (10 + (8 / 2)));

You might want to use parentheses in some expressions for the sake of clarity, even when they aren't needed for modifying operator precedence. Parentheses must always be in pairs, or the compiler generates an error message.

Order of Subexpression Evaluation

As was mentioned in the previous section, if C expressions contain more than one operator with the same precedence level, they are evaluated left to right. For example, in the expression w * x / y * z, w is multiplied by x, the result of the multiplication is then divided by y, and the result of the division is then multiplied by z.

Across precedence levels, however, there is no guarantee of left-to-right order. Look at this expression: w * x / y + z / y

Because of precedence, the multiplication and division are performed before the addition. However, C doesn't specify whether the subexpression w * x / y is to be evaluated before or after z / y. It might not be clear to you why this matters. Look at another example:

w * x / ++y + z / y

If the left subexpression is evaluated first, y is incremented when the second expression is evaluated. If the right expression is evaluated first, y isn't incremented, and the result is different. Therefore, you should avoid this sort of indeterminate expression in your programming.

Compound Assignment Operators

C's compound assignment operators provide a shorthand method for combining a binary mathematical operation with an assignment operation. For example, say you want to increase the value of x by 5, or, in other words, add 5 to x and assign the result to x. You could write `x = x + 5;`

Using a compound assignment operator, which you can think of as a shorthand method of assignment, you would write

`x += 5;`

In more general notation, the compound assignment operators have the following syntax (where op represents a binary operator):

exp1 op= exp2

This is equivalent to writing exp1 = exp1 op exp2;

You can create compound assignment operators using the five binary mathematical operators discussed earlier.

Examples of compound assignment operators

When You Write This...	It Is Equivalent To This
x *= y	x = x * y
y -= z + 1	y = y - z + 1
a /= b	a = a / b
x += y / 8	x = x + y / 8
y %= 3	y = y % 3

The compound operators provide a convenient shorthand, the advantages of which are particularly evident when the variable on the left side of the assignment operator has a

long name. As with all other assignment statements, a compound assignment statement is an expression and evaluates to the value assigned to the left side.

Relational Operators

C's relational operators are used to compare expressions, asking questions such as, "Is x greater than 100?" or "Is y equal to 0?" An expression containing a relational operator evaluates to either true (1) or false (0). C's six relational operators are listed here.

Shows some examples of how relational operators might be used. These examples use literal constants, but the same principles hold with variables.

C's relational operators

Operator	*Symbol*	*Question Asked*	*Example*
Equal	= =	Is operand 1 equal to operand 2?	x = = y
Greater than	>	Is operand 1 greater than operand 2?	x > y
Less than	<	Is operand 1 less than operand 2?	x < y
Greater than or equal	>=	Is operand 1 greater than or equal to operand 2?	x >= y
Less than or equal to	<=	Is operand 1 less than or equal to operand 2?	x <= y
Not equal	!=	Is operand 1 not equal to operand 2?	x != y

Relational operators in use

Expression	How It Reads	What It Evaluates To
5 = = 1	Is 5 equal to 1?	0 (false)
5 > 1	Is 5 greater than 1?	1 (true)
5 != 1	Is 5 not equal to 1?	1 (true)
(5 + 10) = = (3 * 5)	Is (5 + 10) equal to (3 * 5)?	1 (true)

Logical Operators

Sometimes you might need to ask more than one relational question at once. For example, "If it's 7:00 a.m. and a weekday and not my vacation, ring the alarm." C's logical operators let you combine two or more relational expressions into a single expression that evaluates to either true or false. We list C's three logical operators.

C's logical operators

Operator	Symbol	Example
AND	&&	exp1 && exp2
OR	||	exp1 || exp2
NOT	!	!exp1

The way these logical operators work is explained.

Expression	*What It Evaluates To*
(exp1 && exp2)	True (1) only if both exp1 and exp2 are true; false (0) otherwise
(exp1 || exp2)	True (1) if either exp1 or exp2 is true; false (0) only if both are false
(!exp1)	False (0) if exp1 is true; true (1) if exp1 is false

You can see that expressions that use the logical operators evaluate to either true or false, depending on the true/false value of their operand(s). Below we show some actual code examples.

Code examples of C's logical operators

Expression	*What It Evaluates To*
(5 == 5) && (6 != 2)	True (1), because both operands are true
(5 > 1) || (6 < 1)	True (1), because one operand is true
(2 == 1) && (5 == 5)	False (0), because one operand is false
!(5 == 4)	True (1), because the operand is false

You can create expressions that use multiple logical operators. For example, to ask the question "Is x equal to 2, 3, or 4?" you would write (x == 2) || (x == 3) || (x == 4)

The logical operators often provide more than one way to ask a question. If x is an integer variable, the preceding question also could be written in either of the following ways:

(x > 1) && (x < 5)
(x >= 2) && (x <= 4)

The Conditional Operator
The conditional operator is C's only ternary operator, meaning that it takes three operands. Its syntax is exp1 ? exp2 : exp3;

If exp1 evaluates to true (that is, nonzero), the entire expression evaluates to the value of exp2. If exp1 evaluates to false (that is, zero), the entire expression evaluates as the value of exp3. For example, the following statement assigns the value 1 to x if y is true and assigns 100 to x if y is false:

x = y ? 1 : 100;

Comma Operator

- the comma operator combines several expressions into a single statement and returns the value of the rightmost expression:
- a = 0, b = 3, a += b, b = 0;
  sets a to 3, b to 0 and returns 0
- it is rarely used outside the for statement
- the comma operator has the lowest precedence of any operator

Operator Precedence Revisited

List of all the C operators in order of decreasing precedence. Operators on the same line have the same precedence.

Level	Operators
1	( ) [ ] → .
2	! ~ ++ -- * (indirection) & (address-of) (type) sizeof + (unary) - (unary)
3	* (multiplication) / %
4	+ -
5	<< >>
6	< <= > >=
7	= = !=
8	& (bitwise AND)
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= ^= |= <<= >>=
15	,