

UNIT 4

Operators and Expressions

(3 Hours) — C Programming (CACS151)

4.1 Introduction to Operators and Expressions

An operator is a symbol that instructs the compiler to perform a specific mathematical, relational, or logical operation on one or more operands. An expression is a combination of operators and operands that, when evaluated, produces a single value. C provides a rich set of operators, grouped into arithmetic, relational, logical, assignment, increment/decrement, conditional, bitwise, comma, and sizeof categories.

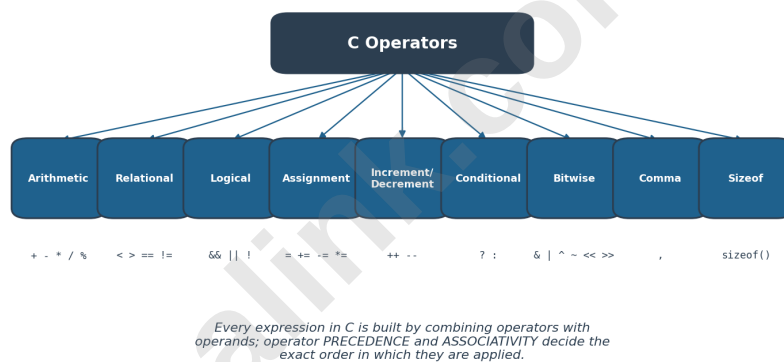


Figure 4.1 — Overview of the major categories of C operators

4.2 Arithmetic Operator

Arithmetic operators are used to perform basic mathematical operations on numeric operands. C supports the five arithmetic operators shown below; both operands may be of any numeric data type (int, float, double, etc.).

Operator	Meaning	Example (a=10, b=3)
+	Addition	$a + b \rightarrow 13$
-	Subtraction	$a - b \rightarrow 7$
*	Multiplication	$a * b \rightarrow 30$

Operator	Meaning	Example (a=10, b=3)
/	Division	a / b → 3 (integer division)
%	Modulus (remainder)	a % b → 1

Note: The modulus operator (%) works only with integer operands, and the result of integer division truncates any fractional part rather than rounding.

4.3 Relational Operator

Relational operators compare two operands and produce an integer result: 1 (true) if the relation holds, or 0 (false) otherwise. They are most commonly used to build the conditions in if statements and loops.

Operator	Meaning	Example (a=5, b=8)
>	Greater than	a > b → 0
<	Less than	a < b → 1
>=	Greater than or equal to	a >= b → 0
<=	Less than or equal to	a <= b → 1
==	Equal to	a == b → 0
!=	Not equal to	a != b → 1

4.4 Logical Operator

Logical operators combine the results of two or more relational or boolean expressions and return a single true/false (1/0) value. They are essential for building compound conditions.

Operator	Name	Description
&&	Logical AND	True only if both operands are true
	Logical OR	True if at least one operand is true
!	Logical NOT	Reverses the logical state of its operand

Example — Logical Operators

```
int age = 20, hasID = 1;
if (age >= 18 && hasID) {
    printf("Entry allowed");
}
```

4.5 Assignment Operator

The assignment operator (=) stores the value of the right-hand side expression into the variable on the left-hand side. C also provides compound (shorthand) assignment operators that combine an arithmetic or bitwise operation with assignment.

Operator	Example	Equivalent to
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

4.6 Increment/Decrement Operator

The increment (++) and decrement (--) operators increase or decrease the value of a variable by 1. Each can be used in prefix form (before the variable) or postfix form (after the variable) — the difference lies in when the value is updated relative to when it is used in the surrounding expression.

Form	Example	Behaviour
Prefix ++	++a	a is incremented first, then the new value is used
Postfix ++	a++	the current value of a is used first, then a is incremented
Prefix --	--a	a is decremented first, then the new value is used
Postfix --	a--	the current value of a is used first, then a is decremented

Example — Prefix vs Postfix

```
int a = 5;
int x = ++a; /* a becomes 6, x = 6 */
int b = 5;
int y = b++; /* y = 5, then b becomes 6 */
```

4.7 Conditional Operator

The conditional (ternary) operator `?:` is C's only operator that takes three operands. It provides a compact alternative to a simple if-else statement, evaluating one of two expressions based on a condition.

Example — Conditional Operator

```
int a = 10, b = 20, max;
max = (a > b) ? a : b; /* max = 20 */
```

Note: The general form is: condition ? expression_if_true : expression_if_false;

4.8 Bitwise Operator

Bitwise operators act directly on the individual bits of integer operands, making them useful for low-level tasks such as flag manipulation, masking, and hardware-level programming.

Operator	Name	Description
<code>&</code>	Bitwise AND	Sets each bit to 1 only if both corresponding bits are 1
<code> </code>	Bitwise OR	Sets each bit to 1 if at least one corresponding bit is 1
<code>^</code>	Bitwise XOR	Sets each bit to 1 if the corresponding bits differ
<code>~</code>	Bitwise NOT (Complement)	Inverts every bit of the operand
<code><<</code>	Left Shift	Shifts bits left, filling with 0s (multiplies by powers of 2)
<code>>></code>	Right Shift	Shifts bits right (divides by powers of 2 for unsigned values)

a = 89	0	1	0	1	1	0	0	1
b = 53	0	0	1	1	0	1	0	1
AND (&)	0	0	0	1	0	0	0	1
OR ()	0	1	1	1	1	1	0	1
XOR (^)	0	1	1	0	1	1	0	0

Bitwise operators act directly on the individual binary bits of integer operands.

Figure 4.2 — Bit-level result of AND, OR, and XOR on two 8-bit values

4.9 Comma Operator

The comma operator (,) allows two or more expressions to be evaluated in sequence within a single statement, with the overall value of the comma expression being the value of the last (right-most) expression. It is most often seen inside the initialization and update parts of a for loop.

Example — Comma Operator in a for Loop

```
for (i = 0, j = 10; i < j; i++, j--) {
    printf("%d %d\n", i, j);
}
```

4.10 Sizeof Operator

The sizeof operator is a compile-time unary operator that returns the size, in bytes, occupied by a data type or a variable. It is frequently used for portable memory allocation and for writing code that is independent of a particular machine's data-type sizes.

Example — Sizeof Operator

```
printf("%zu", sizeof(int)); /* commonly 4 */
printf("%zu", sizeof(double)); /* commonly 8 */
int arr[10];
printf("%zu", sizeof(arr)); /* 10 * sizeof(int) */
```

4.11 Operator Precedence and Associativity

When an expression contains more than one operator, C must decide which operator to apply first. Precedence determines which operator types are evaluated before others; associativity determines the order of evaluation among operators that share the same precedence level (left-to-right or right-to-left).

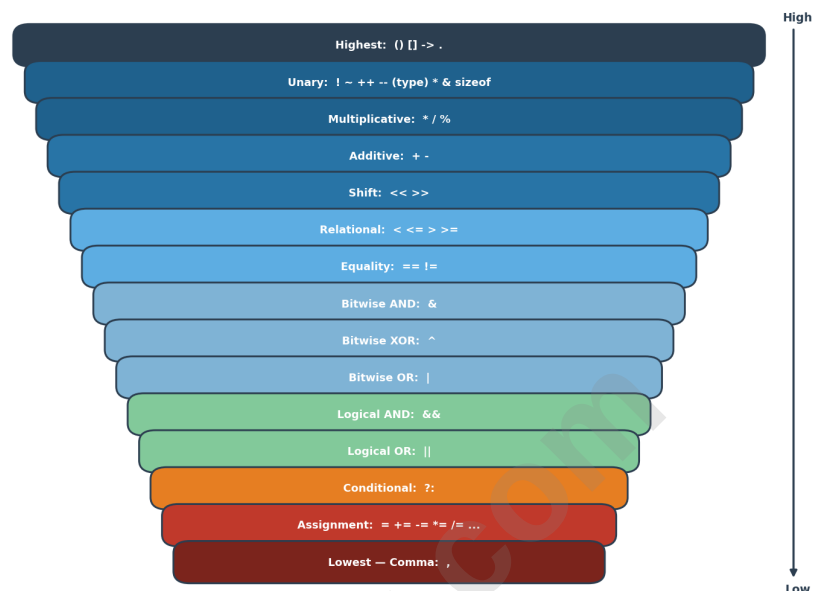


Figure 4.3 — Simplified operator precedence hierarchy (highest at top)

- Operators higher in the hierarchy are evaluated before operators lower down.
- Most binary operators (arithmetic, relational, logical, bitwise) associate left-to-right.
- Unary operators and the assignment operator associate right-to-left.
- Parentheses () can always be used to force a specific evaluation order, overriding default precedence.

4.12 Expressions and its Evaluation

An expression is evaluated by repeatedly applying operators to their operands according to precedence and associativity rules, until a single final value remains. This process can be visualized as an expression tree, where leaves are operands and internal nodes are operators, evaluated bottom-up.

Expression: $a + b * c - d / e$

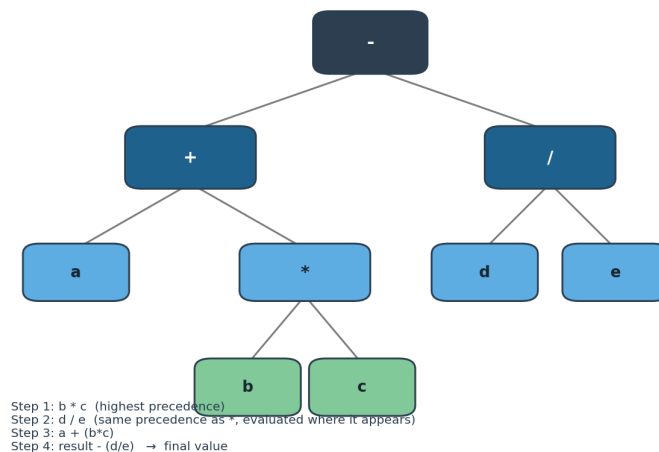


Figure 4.4 — Evaluation tree for $a + b * c - d / e$

Worked Example — Evaluate $4 + 6 * 2 - 8 / 4$

Step 1 (multiplication): $6 * 2 = 12$

Step 2 (division): $8 / 4 = 2$

Step 3 (addition): $4 + 12 = 16$

Step 4 (subtraction): $16 - 2 = 14$

Final result = 14

4.13 Type Casting in Expression

Type casting is the explicit conversion of a value from one data type to another. C also performs implicit type conversion (sometimes called type promotion) automatically when operands of different types appear together in an expression, but an explicit cast gives the programmer direct control over the conversion.

Example — Explicit Type Casting

```
int a = 7, b = 2;
```

```
float result = (float) a / b; /* result = 3.500000 */
```

```
/* Without the cast, a / b would perform integer  
division and give 3, discarding the fraction. */
```

Note: The general syntax for an explicit cast is: (type_name) expression;

4.14 Program Statement

A program statement is a complete instruction that the C compiler can execute, always terminated by a semicolon (;). Statements are the basic building blocks of a C program's body, and expressions become statements once followed by a semicolon.

Statement Type	Example
Expression statement	<code>a = b + c;</code>
Null statement	<code>;</code>
Compound statement (block)	<code>{ a = 1; b = 2; }</code>
Control statement	<code>if (a > b) { ... }</code>
Declaration statement	<code>int x, y;</code>

Key Takeaways — Unit 4

- ✓ Arithmetic, relational, and logical operators handle numeric computation and condition building
- ✓ Assignment and compound-assignment operators store and update values in one step
- ✓ Increment/decrement operators differ in prefix vs postfix timing of the update
- ✓ The conditional operator `?:` offers a compact alternative to if-else
- ✓ Bitwise operators manipulate individual bits, useful for low-level and flag-based logic
- ✓ The comma operator sequences expressions; `sizeof` returns the byte size of a type or variable
- ✓ Precedence and associativity together determine the exact order operators are applied in an expression
- ✓ Type casting — implicit or explicit — controls how mixed-type values are converted during evaluation
- ✓ Every executable instruction in C is a program statement, terminated by a semicolon

— *End of Unit 4* —