4.1 Arithmetic Operators

The *arithmetic operators*—operators that perform addition, subtraction, multiplication, and division—are the workhorses of many programming languages, including C. Table 4.1 shows C's arithmetic operators.

Table 4.1
Arithmetic Operators

Unary	*Binary*	
	Additive	*Multiplicative*
+ unary plus - unary minus	+ addition - subtraction	* multiplication / division % remainder

The additive and multiplicative operators are said to be ***binary*** because they require *two* operands. The ***unary*** operators require *one* operand:

```
i = +1;    /* + used as a unary operator */
j = -i;    /* - used as a unary operator */
```

The unary + operator does nothing; in fact, it didn't even exist in K&R C. It's used primarily to emphasize that a numeric constant is positive.

The binary operators probably look familiar. The only one that might not is `%`, the remainder operator. The value of `i % j` is the remainder when `i` is divided by `j`. For example, the value of `10 % 3` is 1, and the value of `12 % 4` is 0.

Q&A

The binary operators in Table 4.1—with the exception of `%`—allow either integer or floating-point operands, with mixing allowed. When `int` and `float` operands are mixed, the result has type `float`. Thus, `9 + 2.5f` has the value 11.5, and `6.7f / 2` has the value 3.35.

The `/` and `%` operators require special care:

- The `/` operator can produce surprising results. When both of its operands are integers, the `/` operator "truncates" the result by dropping the fractional part. Thus, the value of `1 / 2` is 0, not 0.5.
- The `%` operator requires integer operands; if either operand is not an integer, the program won't compile.

undefined behavior ➤4.4

- Using zero as the right operand of either `/` or `%` causes undefined behavior.

Q&A

- Describing the result when `/` and `%` are used with negative operands is tricky. The C89 standard states that if either operand is negative, the result of a division can be rounded either up or down. (For example, the value of `-9 / 7` could be either –1 or –2). If `i` or `j` is negative, the sign of `i % j` in C89 depends on the implementation. (For example, the value of `-9 % 7` could be either –2 or 5). In C99, on the other hand, the result of a division is always truncated toward zero (so `-9 / 7` has the value –1) and the value of `i % j` has the same sign as `i` (hence the value of `-9 % 7` is –2).

C99