
Ordering Information:

**C How to Program, 3/e**

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**.

To learn more about our **C/C++ training courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com**.

*Note from the Authors*: This article is an excerpt from Chapter 3, Section 3.12 of *C How to Program, 3/e.* In this article, we discuss unary increment and decrement operators in the C language. This is an introductory-level article and readers should be familiar with basic C concepts. The code examples included in this article show readers examples using the DEITEL™ signature LIVE-CODE™ Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

## 3.12 Increment and Decrement Operators

C also provides the unary *increment operator*, **++**, and the unary *decrement operator*, **--**, which are summarized in Fig. 3.12. If a variable **c** is incremented by 1, the increment operator **++** can be used rather than the expressions **c = c + 1** or **c += 1**. If increment or decrement operators are placed before a variable, they are referred to as the *preincrement* or *predecrement operators,* respectively. If increment or decrement operators are placed after a variable, they are referred to as the *postincrement* or *postdecrement operators,* respectively. Preincrementing (predecrementing) a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears. Postincrementing (postdecrementing) the variable causes the current value of the variable to be used in the expression in which it appears, then the variable value is incremented (decremented) by 1.

Figure 3.13 demonstrates the difference between the preincrementing and the postincrementing versions of the **++** operator. Postincrementing the variable **c** causes it to be incremented after it is used in the **printf** statement. Preincrementing the variable **c** causes it to be incremented before it is used in the **printf** statement..

| Operator | Sample expression | Explanation |
|---|---|---|
| ++ | ++a | Increment **a** by 1 then use the new value of **a** in the expression in which **a** resides. |
| ++ | a++ | Use the current value of **a** in the expression in which **a** resides, then increment **a** by 1. |
| -- | --b | Decrement **b** by 1 then use the new value of **b** in the expression in which **b** resides. |
| -- | b-- | Use the current value of **b** in the expression in which **b** resides, then decrement **b** by 1. |

Fig. 3.12 The increment and decrement operators

```
1  /* Fig. 3.13: fig03_13.c
2     Preincrementing and postincrementing */
3  #include <stdio.h>
4
5  int main()
6  {
7     int c = 5;
8
9     printf( "%d\n", c );
10    printf( "%d\n", c++ );     /* postincrement */
11    printf( "%d\n\n", c );
12
13    c = 5;
14    printf( "%d\n", c );
```

Fig. 3.13 Showing the difference between preincrementing and postincrementing. (part 1 of 2)

```
15      printf( "%d\n", ++c );     /* preincrement */
16      printf( "%d\n", c );
17
18      return 0;   /* successful termination */
19   }
```

```
5
5
6

5
6
6
```

**Fig. 3.13** Showing the difference between preincrementing and postincrementing. (part 2 of 2)

The program displays the value of **c** before and after the **++** operator is used. The decrement operator (**--**) works similarly.

### Good Programming Practice 3.10

*Unary operators should be placed directly next to their operands with no intervening spaces.*

The three assignment statements in Fig. 3.10

```
passes = passes + 1;
failures = failures + 1;
student = student + 1;
```

can be written more concisely with assignment operators as

```
passes += 1;
failures += 1;
student += 1;
```

with preincrement operators as

```
++passes;
++failures;
++student;
```

or with postincrement operators as

```
passes++;
failures++;
student++;
```

It is important to note here that when incrementing or decrementing a variable in a statement by itself, the preincrement and postincrement forms have the same effect. It is only when a variable appears in the context of a larger expression that preincrementing and postincrementing have different effects (and similarly for predecrementing and postdecrementing).

Only a simple variable name may be used as the operand of an increment or decrement operator.

### Common Programming Error 3.10

*Attempting to use the increment or decrement operator on an expression other than a simple variable name, e.g., writing* `++(x + 1)` *is a syntax error.*

### Good Programming Practice 3.11

*The ANSI standard generally does not specify the order in which an operator's operands will be evaluated (although we will see exceptions to this for a few operators in Chapter 4). Therefore the programmer should avoid using statements with increment or decrement operators in which a particular variable being incremented or decremented appears more than once.*