02_04_C Program Control

Objectives

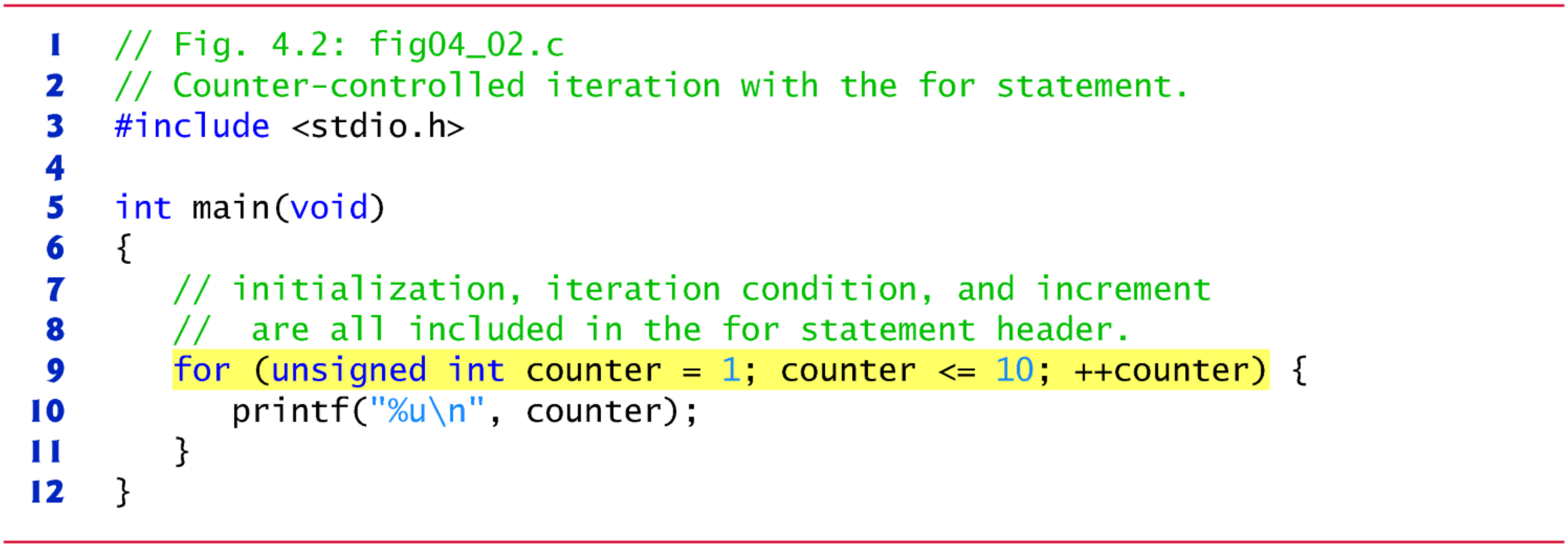
In this chapter, you'll learn:

- The essentials of counter-controlled iteration.
- To use the `for` and `do`…`while` iteration statements to execute statements repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To use the `break` and `continue` statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions in control statements.
- To avoid the consequences of confusing the equality and assignment operators.

```c
// Fig. 4.1: fig04_01.c
// Counter-controlled iteration.
#include <stdio.h>

int main(void)
{
   unsigned int counter = 1; // initialization

   while (counter <= 10) { // iteration condition
      printf ("%u\n", counter);
      ++counter; // increment
   }
}
```

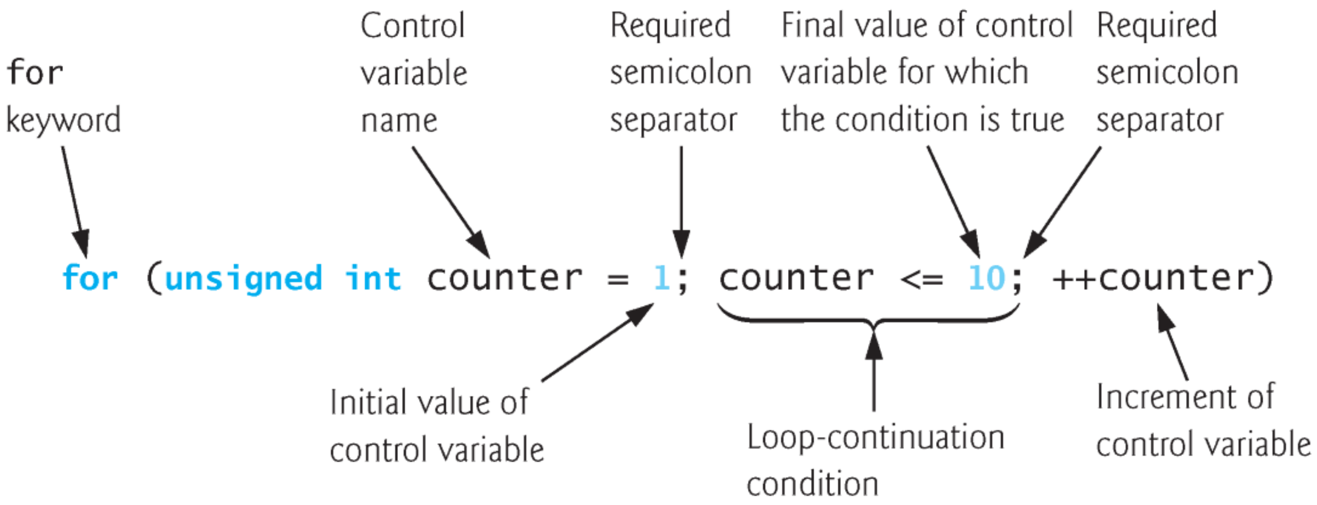
```
1
2
3
4
5
6
7
8
9
10
```

Fig. 4.1 | Counter-controlled iteration.



```c
// Fig. 4.2: fig04_02.c
// Counter-controlled iteration with the for statement.
#include <stdio.h>

int main(void)
{
   // initialization, iteration condition, and increment
   //  are all included in the for statement header.
   for (unsigned int counter = 1; counter <= 10; ++counter) {
      printf("%u\n", counter);
   }
}
```

Fig. 4.2 | Counter-controlled iteration with the `for` statement.



for
keyword

Control
variable
name

Required
semicolon
separator

Final value of control
variable for which
the condition is true

Required
semicolon
separator

```
for (unsigned int counter = 1; counter <= 10; ++counter)
```

Initial value of
control variable

Loop-continuation
condition

Increment of
control variable

Fig. 4.3 | `for` statement header components.

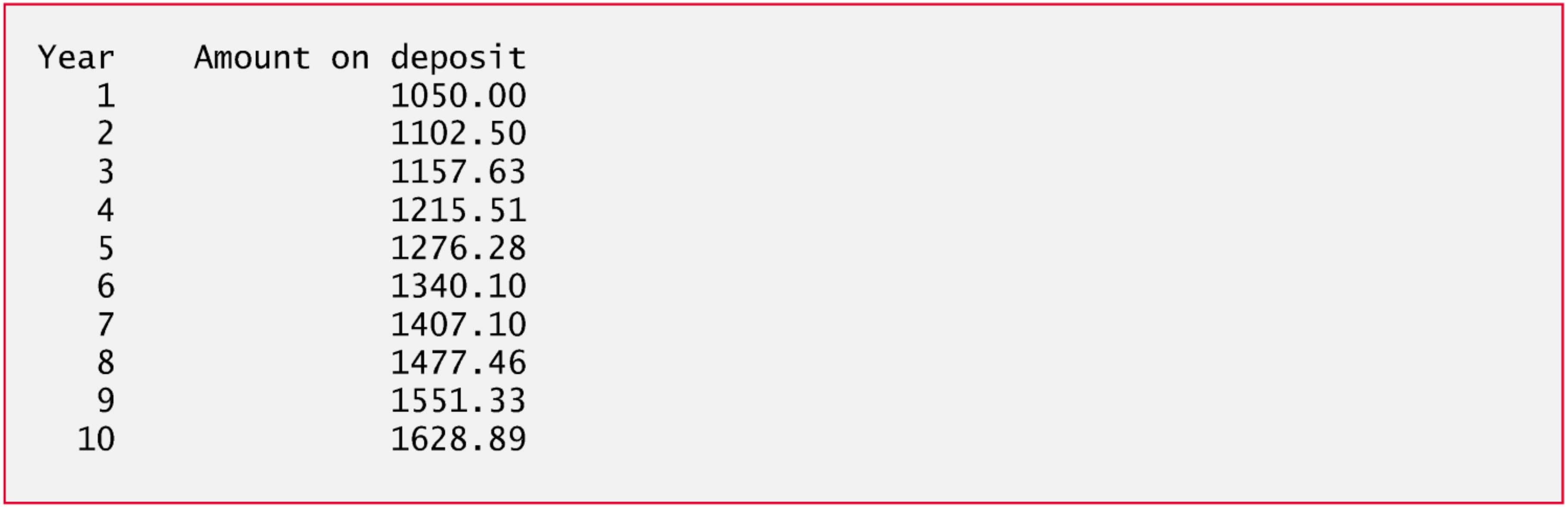
```c
// Fig. 4.5: fig04_05.c
// Summation with for.
#include <stdio.h>

int main(void)
{
   unsigned int sum = 0; // initialize sum

   for (unsigned int number = 2; number <= 100; number += 2) {
      sum += number; // add number to sum
   }

   printf("Sum is %u\n", sum);
}
```

```
Sum is 2550
```

Fig. 4.5 | Summation with `for`.

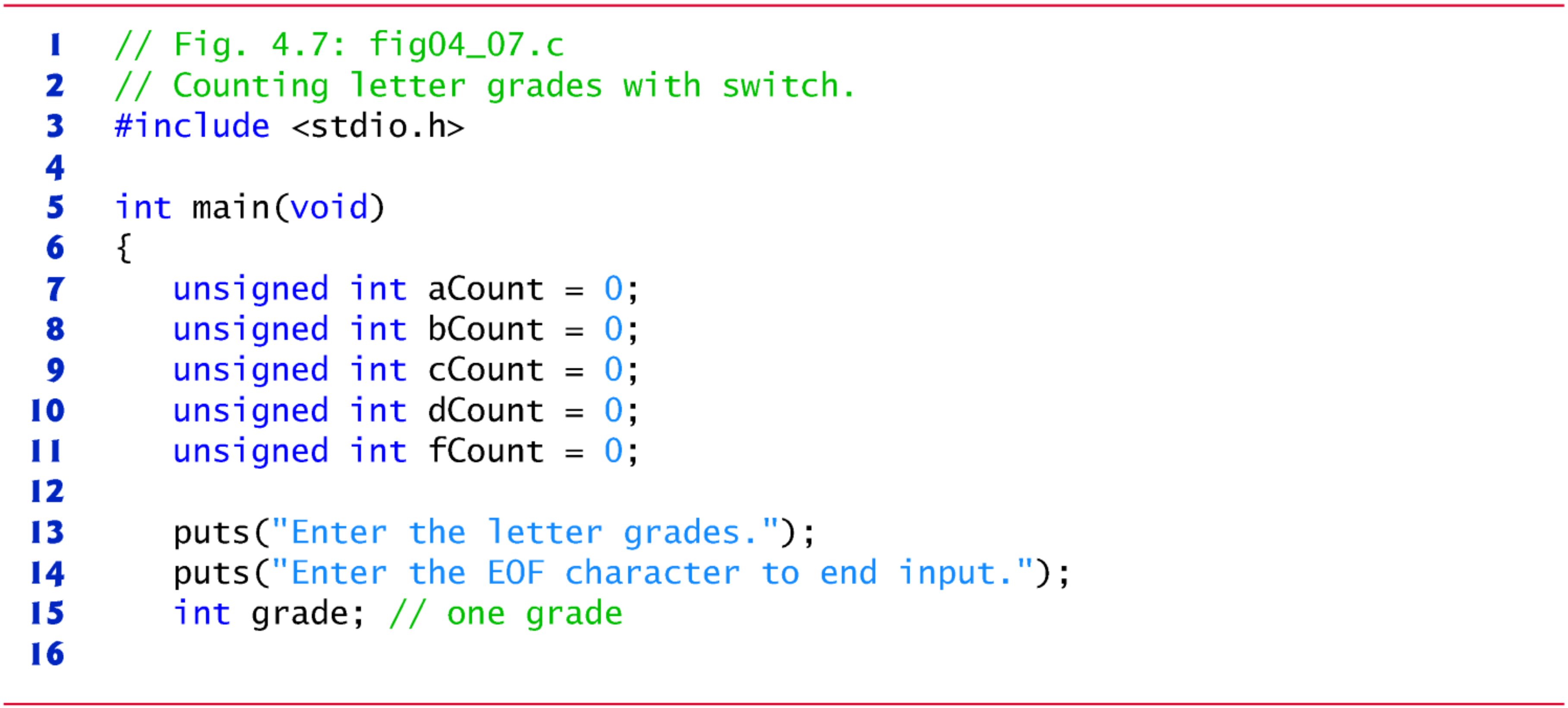
```c
// Fig. 4.6: fig04_06.c
// Calculating compound interest.
#include <stdio.h>
#include <math.h>

int main(void)
{
   double principal = 1000.0; // starting principal
   double rate = .05; // annual interest rate

   // output table column heads
   printf("%4s%21s\n", "Year", "Amount on deposit");

   // calculate amount on deposit for each of ten years
   for (unsigned int year = 1; year <= 10; ++year) {

      // calculate new amount for specified year
      double amount = principal * pow(1.0 + rate, year);

      // output one table row
      printf("%4u%21.2f\n", year, amount);
   }
}
```

Fig. 4.6 | Calculating compound interest. (Part 1 of 2.)



```
Year    Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89
```

Fig. 4.6 | Calculating compound interest. (Part 2 of 2.)



```c
// Fig. 4.7: fig04_07.c
// Counting letter grades with switch.
#include <stdio.h>

int main(void)
{
   unsigned int aCount = 0;
   unsigned int bCount = 0;
   unsigned int cCount = 0;
   unsigned int dCount = 0;
   unsigned int fCount = 0;

   puts("Enter the letter grades.");
   puts("Enter the EOF character to end input.");
   int grade; // one grade

```

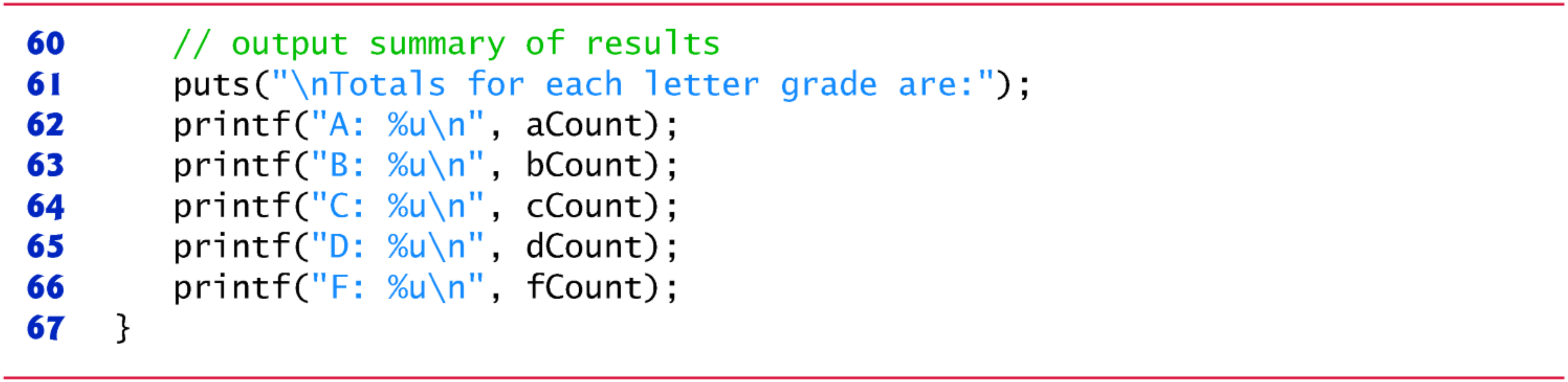
Fig. 4.7 | Counting letter grades with switch. (Part 1 of 5.)

```
   // loop until user types end-of-file key sequence
   while ((grade = getchar()) != EOF) {

      // determine which grade was input
      switch (grade) { // switch nested in while

         case 'A': // grade was uppercase A
         case 'a': // or lowercase a
            ++aCount;
            break; // necessary to exit switch

         case 'B': // grade was uppercase B
         case 'b': // or lowercase b
            ++bCount;
            break;

         case 'C': // grade was uppercase C
         case 'c': // or lowercase c
            ++cCount;
            break;

```

Fig. 4.7 | Counting letter grades with `switch`. (Part 2 of 5.)

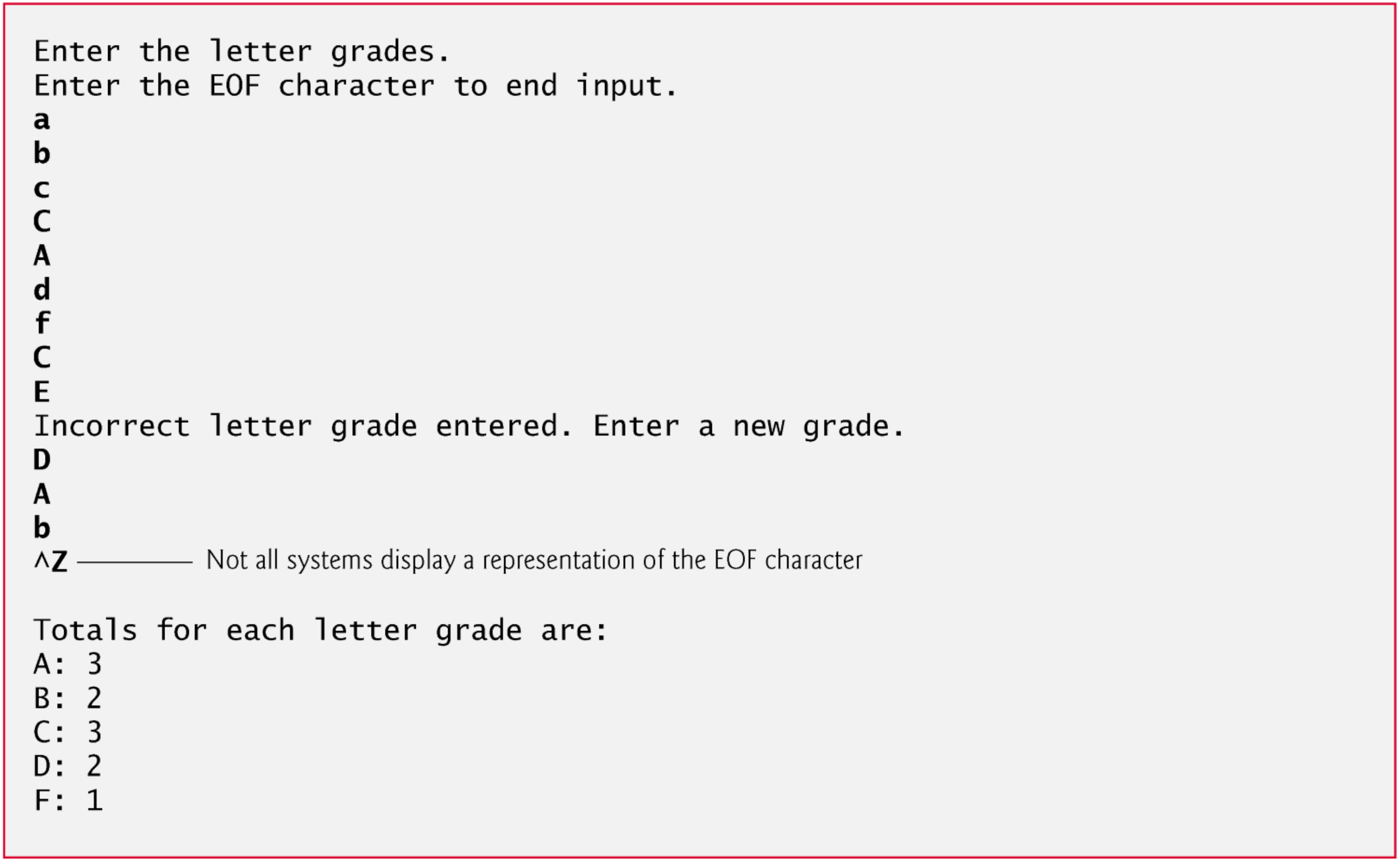
```c
         case 'D': // grade was uppercase D
         case 'd': // or lowercase d
            ++dCount;
            break;

         case 'F': // grade was uppercase F
         case 'f': // or lowercase f
            ++fCount;
            break;

         case '\n': // ignore newlines,
         case '\t': // tabs,
         case ' ': // and spaces in input
            break;

         default: // catch all other characters
            printf("%s", "Incorrect letter grade entered.");
            puts(" Enter a new grade.");
            break; // optional; will exit switch anyway
      }
   } // end while

```

Fig. 4.7 | Counting letter grades with `switch`. (Part 3 of 5.)



```
   // output summary of results
   puts("\nTotals for each letter grade are:");
   printf("A: %u\n", aCount);
   printf("B: %u\n", bCount);
   printf("C: %u\n", cCount);
   printf("D: %u\n", dCount);
   printf("F: %u\n", fCount);
}
```

Fig. 4.7 | Counting letter grades with `switch`. (Part 4 of 5.)



```
Enter the letter grades.
Enter the EOF character to end input.
a
b
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b
^Z

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Not all systems display a representation of the EOF character

Fig. 4.7 | Counting letter grades with `switch`. (Part 5 of 5.)

```c
// Fig. 4.9: fig04_09.c
// Using the do...while iteration statement.
#include <stdio.h>

int main(void)
{
   unsigned int counter = 1; // initialize counter

   do {
      printf("%u  ", counter);
   } while (++counter <= 10);
}
```

```
1  2  3  4  5  6  7  8  9  10
```

Fig. 4.9 | Using the `do...while` iteration statement.

```c
// Fig. 4.11: fig04_11.c
// Using the break statement in a for statement.
#include <stdio.h>

int main(void)
{
   unsigned int x; // declared here so it can be used after loop

   // loop 10 times
   for (x = 1; x <= 10; ++x) {

      // if x is 5, terminate loop
      if (x == 5) {
         break; // break loop only if x is 5
      }

      printf("%u ", x);
   }

   printf("\nBroke out of loop at x == %u\n", x);
}
```

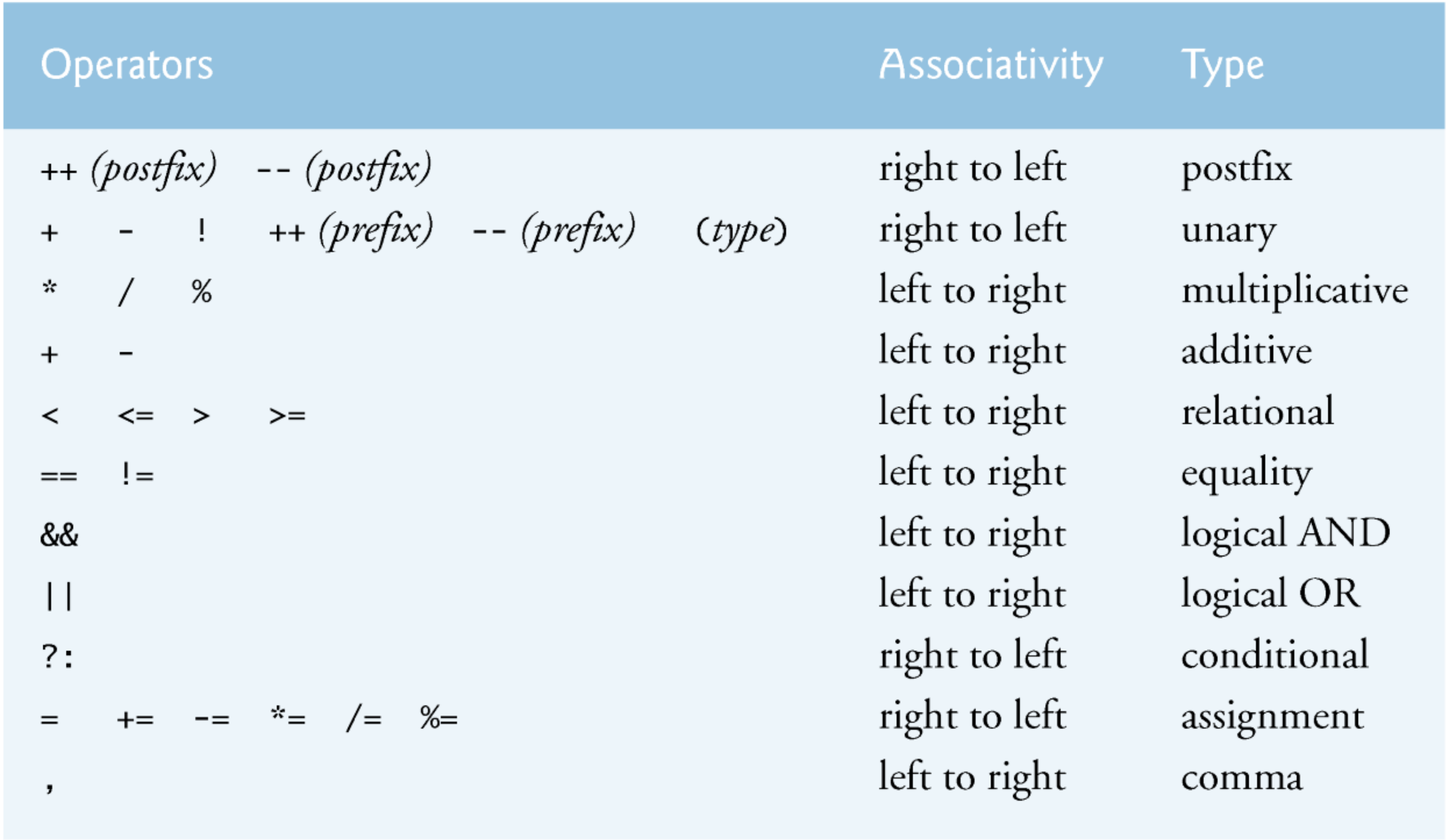
```
1 2 3 4
Broke out of loop at x == 5
```

Fig. 4.11 | Using the `break` statement in a `for` statement.

```c
// Fig. 4.12: fig04_12.c
// Using the continue statement in a for statement.
#include <stdio.h>

int main(void)
{
   // loop 10 times
   for (unsigned int x = 1; x <= 10; ++x) {

      // if x is 5, continue with next iteration of loop
      if (x == 5) {
         continue; // skip remaining code in loop body
      }

      printf("%u ", x);
   }

   puts("\nUsed continue to skip printing the value 5");
}
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

Fig. 4.12 | Using the `continue` statement in a `for` statement.



Operators	Associativity	Type
++ *(postfix)* -- *(postfix)*	right to left	postfix
+ - ! ++ *(prefix)* -- *(prefix)* (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Fig. 4.16 | Operator precedence and associativity.