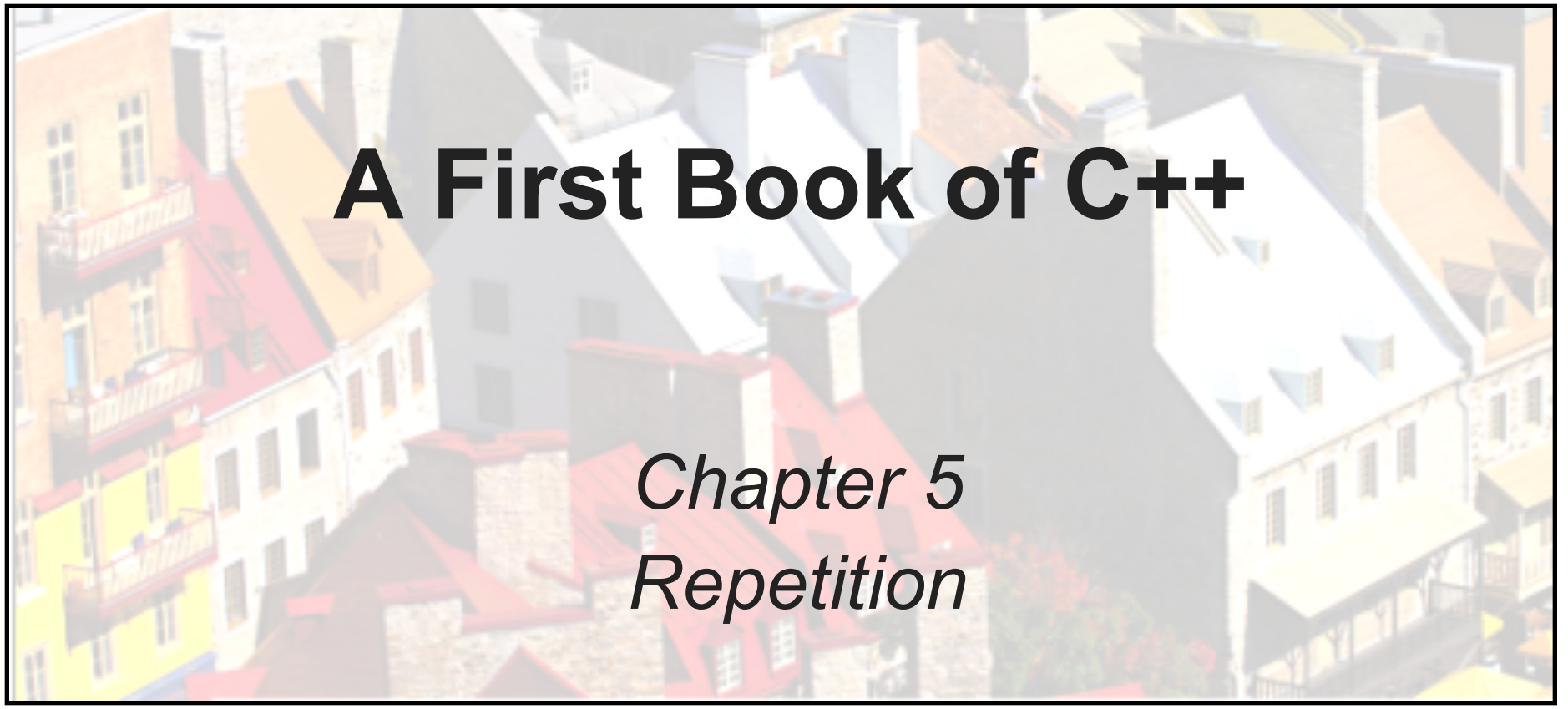

A First Book of C++

Chapter 5
Repetition

Objectives

- In this chapter, you will learn about:
  - The `while` Statement
  - Interactive `while` Loops
  - The `for` Statement
  - The `do while` Statement
  - Common Programming Errors
  - Additional Testing Techniques

The `while` Statement

- A general repetition statement
- Format:

```
while (expression)
      statement;
```

- Function:
  - `expression` is evaluated the same as an if-else statement
  - Statement following expression executed repeatedly as long as expression has non-zero value

The `while` Statement (cont'd.)

- Steps in execution of `while` statement:
  1. Test the expression
  2. If the expression has a non-zero (true) value
     a. Execute the statement following the parentheses
     b. Go back to step 1

     else
     a. Exit the `while` statement

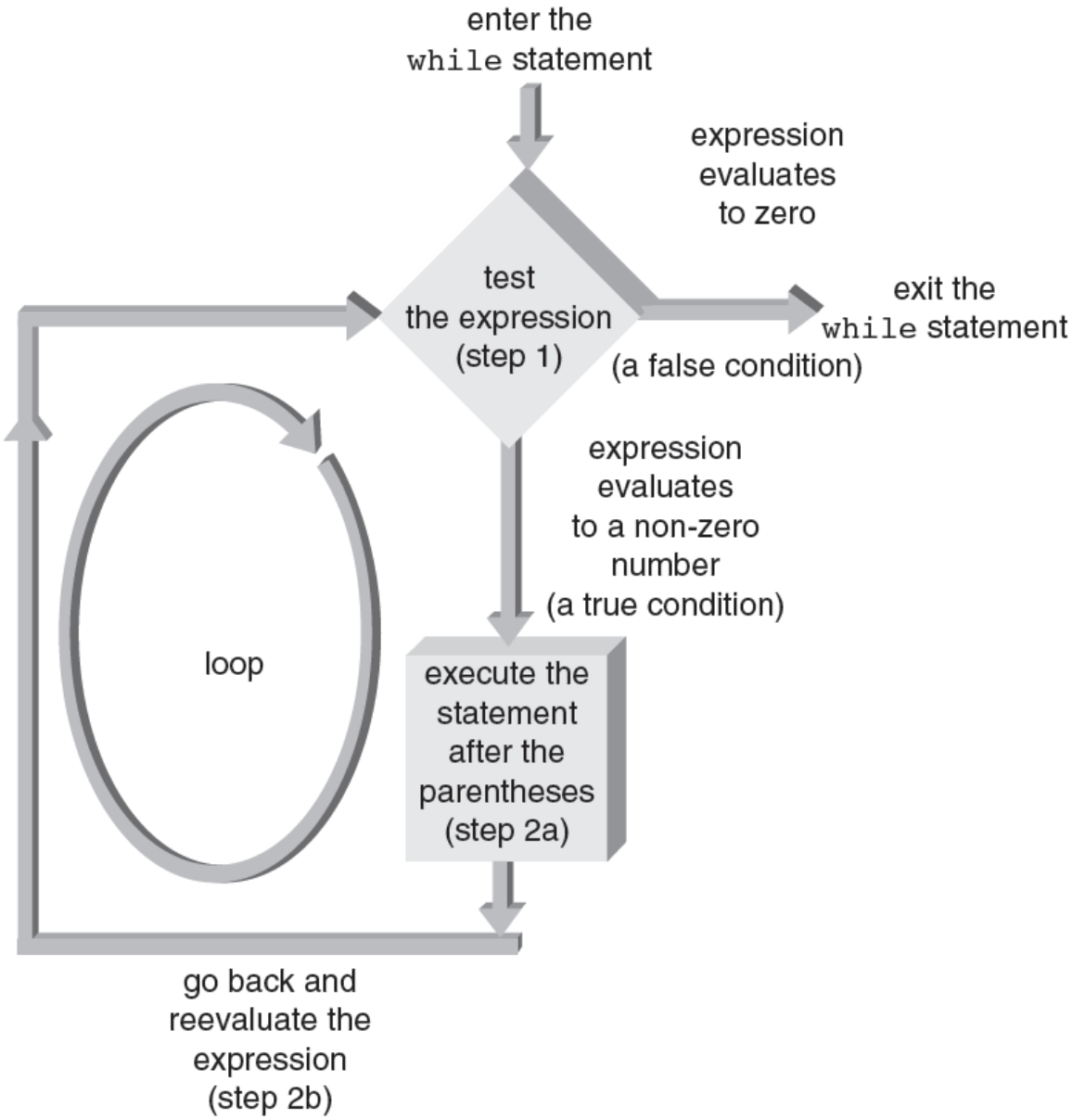

Figure 5.1 Anatomy of a `while` loop

The `while` Statement (cont'd.)

Program 5.1

```
#include <iostream>
using namespace std;

int main()
{
  int count;

  count = 1;              // initialize count
  while (count <= 10)
  {
    cout << count << " ";
    count++;              // increment count
  }

  return 0;
}
```

This is the output for Program 5.1:

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

The `while` Statement (cont'd.)

- Infinite loop: one that never ends
- Fixed-count loop: tested expression is a counter that checks for a fixed number of repetitions
- Variation: counter is incremented by a value other than 1
- Example:
  - Celsius-to-Fahrenheit temperature conversion
  - Display Fahrenheit and Celsius temperatures, from 5-50 degrees C, in 5 degree increments

The `while` Statement (cont'd.)

```
celsius = 5;                    // starting Celsius
value
while (celsius <= 50)
{
  fahren = (9.0/5.0) * celsius + 32.0;
  cout << setw(4) << celsius
       << setw(13) << fahren << endl;
  celsius = celsius + 5;
}
```

Interactive `while` Loops

- Combining interactive data entry with the repetition of a `while` statement
  - Produces very powerful and adaptable programs
- Example (Program 5.5): `while` statement accepts and displays four user-entered numbers
  - Numbers accepted and displayed one at a time

Program 5.5

```
#include <iostream>
using namespace std;

int main()
{
  const int MAXNUMS = 4;
  int count;
  double num;

  cout << "\nThis program will ask you to enter "
       << MAXNUMS << " numbers.\n";
  count = 1;

  while (count <= MAXNUMS)
  {
    cout << "\nEnter a number: ";
    cin  >> num;
    cout << "The number entered is " << num;
    count++;
  }
  cout << endl;

  return 0;
}
```

Interactive `while` Loops (cont'd.)

- Sample run of Program 5.5:

```
This program will ask you to enter 4
numbers.
Enter a number: 26.2
The number entered is 26.2
Enter a number: 5
The number entered is 5
Enter a number: 103.456
The number entered is 103.456
Enter a number: 1267.89
The number entered is 1267.89
```

Interactive `while` Loops (cont'd.)

- Example (Program 5.6): adding a single number to a total
  - A number is entered by the user
  - Accumulating statement adds the number to total

```
total = total + num;
```

  - A `while` statement repeats the process

Interactive `while` Loops (cont'd.)

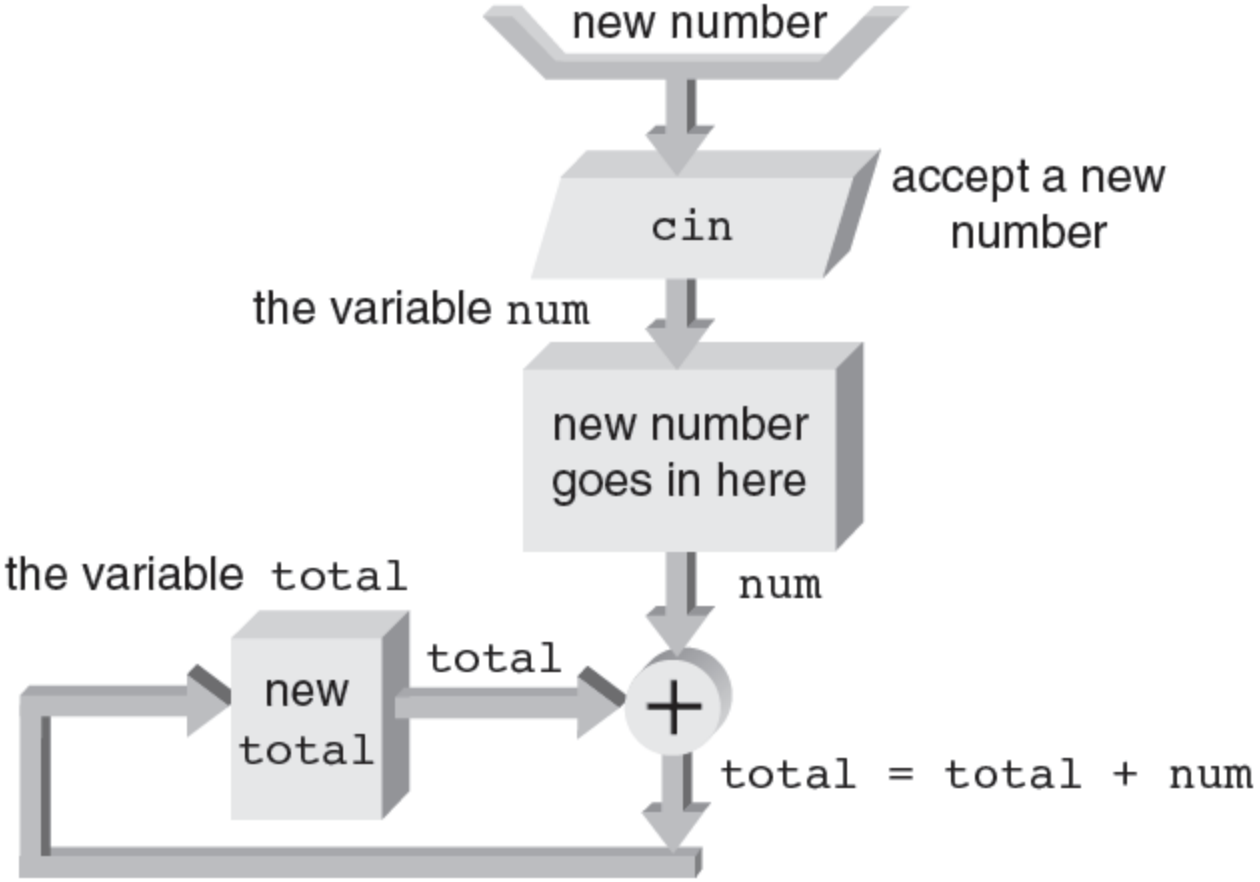

Figure 5.3 Accepting and adding a number to a total

Program 5.6

```
#include <iostream>
using namespace std;

int main()
{
  const int MAXNUMS = 4;
  int count;
  double num, total;

  cout << "\nThis program will ask you to enter "
       << MAXNUMS << " numbers.\n";
  count = 1;
  total = 0;

  while (count <= MAXNUMS)
  {
    cout << "\nEnter a number: ";
    cin  >> num;
    total = total + num;
    cout << "The total is now " << total;
    count++;
  }

  cout   << "\n\nThe final total is " << total << endl;

  return 0;
}
```

Interactive `while` Loops (cont'd.)

- Sample run of Program 5.6:

```
This program will ask you to enter 4 numbers.
Enter a number: 26.2
The total is now 26.2
Enter a number: 5
The total is now 31.2
Enter a number: 103.456
The total is now 134.656
Enter a number: 1267.89
The total is now 1402.546

The final total is 1402.546
```

Sentinels

- Sentinels
  - Data values used to signal the start or end of data series
- Values must be selected so as not to conflict with legitimate data values

`break` and `continue` Statements

- `break`: forces immediate exit from structures
  - Use in switch statements:
    - The desired case has been detected and processed
  - Use in `while`, `for`, and `do-while` statements:
    - An unusual condition is detected
- Format:

```
break;
```

`break` and `continue` Statements (cont'd.)

- `continue`: causes the next iteration of the loop to begin immediately
  - Execution transferred to the top of the loop
  - Applies only to `while`, `do-while`, and `for` statements
- Format:

```
continue;
```

The Null Statement

- Used where a statement is syntactically required but no action is called for
  - A do-nothing statement
  - Typically used with `while` or `for` statements

The `for` Statement

- Same function as the `while` statement but in different form

```
for (initializing list; expression; altering list)
   statement;
```

- Function: statement executed while expression has non-zero (true) value
- Components:
  - Initializing list: initial value of expression
  - Expression: a valid C++ expression
  - Altering list: statements executed at end of each `for` loop to alter value of expression

The `for` Statement (cont'd.)

- Components of `for` statement correspond to operations performed in `while` statement
  - Initialization
  - Expression evaluation
  - Altering of expression values
- Components of `for` statement are optional, but semicolons must always be present

The `for` Statement (cont'd.)

Program 5.9

```
#include <iostream>
using namespace std;
int main()
{
  int count;
  for (count = 2; count <= 20; count = count + 2)
    cout << count << " ";

  return 0;
}
```

This is the output of Program 5.9:

```
2 4 6 8 10 12 14 16 18 20
```

The `for` Statement (cont'd.)

- Program 5.9 modified: initializer outside `for` loop

Program 5.9a

```
#include <iostream>
using namespace std;

int main()
{
  int count;

  count = 2;    // initialized outside the for statement
  for ( ; count <= 20; count = count + 2)
    cout << count << " ";

  return 0;
}
```

The `for` Statement (cont'd.)

Program 5.10

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
  const int MAXNUMS = 10;
  int num;
  cout << endl;      // print a blank line
  cout << "NUMBER   SQUARE   CUBE\n"
       << "------   ------   ----\n";
  for (num = 1; num <= MAXNUMS; num++)
    cout << setw(3) << num << "       "
         << setw(3) << num * num << "      "
         << setw(4) << num * num * num << endl;

  return 0;
}
```

The `for` Statement (cont'd.)

- When Program 5.10 is run, the display produced is:

```
 NUMBER          SQUARE          CUBE

_ _ _ _ _ _    _ _ _ _ _ _ _    _ _ _ _

      1              1              1
      2              4              8
      3              9             27
      4             16             64
      5             25            125
      6             36            216
      7             49            343
      8             64            512
      9             81            729
     10            100           1000
```

Interactive `for` Loops

- Same effect as using `cin` object within a `while` loop
- Provides interactive user input

Program 5.11

```
#include <iostream>
using namespace std;

// This program calculates the average
// of MAXCOUNT user-entered numbers
int main()
{
  const int MAXCOUNT = 5;
  int count;
  double num, total, average;

  total = 0.0;

  for (count = 0; count < MAXCOUNT; count++)
  {
    cout << "Enter a number: ";
    cin  >> num;
    total = total + num;
  }

  average = total / count;
  cout << "The average of the data entered is " << average
       << endl;

  return 0;
}
```

Interactive `for` Loops (cont'd.)

- Program 5.11: `for` statement creates a loop
  - Loop executed five times
- Actions performed in each loop
  - User prompted to enter a number
  - Number added to the total

Interactive `for` Loops (cont'd.)

- Initialization variations:
  - Alternative 1: initialize `total` outside the loop and `count` inside the loop as in Program 5.11
  - Alternative 2: initialize both `total` and `count` inside loop

    ```
    for (total = 0.0, count = 0; count < MAXCOUNT;
    count++)
    ```

  - Alternative 3: initialize and declare both `total` and `count` inside loop

    ```
    for (double total = 0.0, int count = 0; count <
    MAXCOUNT; count++)
    ```

Nested Loops

- A loop contained within another loop
- Example:

```
for(i = 1; i <= 5; i++)        // start of outer loop
{
 cout << "\ni is now " << i << endl;

 for(j = 1; j <= 4; j++)     // start of inner loop
 {
    cout << " j = " << j;
 }                             // end of inner loop
}                              // end of outer loop
```

Nested Loops (cont'd.)

- Outer (first) loop:
  - Controlled by value of `i`
- Inner (second) loop:
  - Controlled by value of `j`
- Rules:
  - For each single trip through outer loop, inner loop runs through its entire sequence
  - Different variable to control each loop
  - Inner loop statements contained within outer loop

Nested Loops (cont'd.)

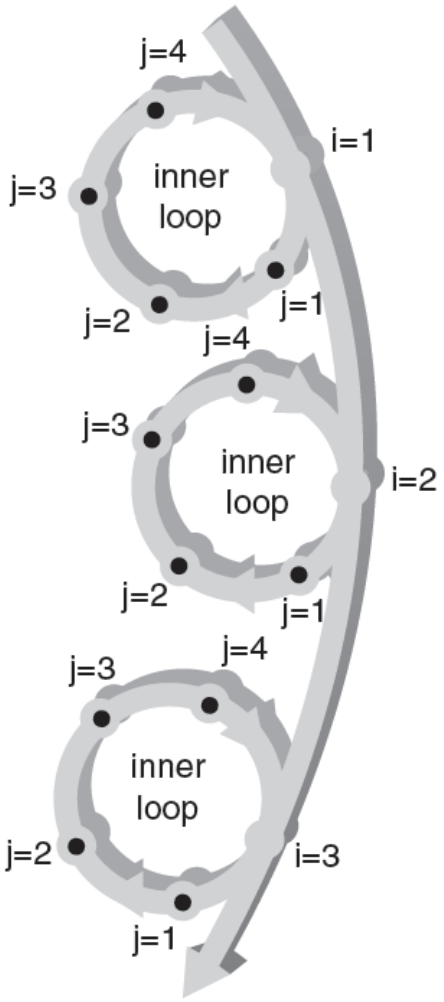

Figure 5.7 For each value of `i`, `j` loops four times

The `do-while` Statement

- A repetition statement that evaluates an expression at the end of the statement
  - Allows some statements to be executed before an expression is evaluated
  - `for` and `while` evaluate an expression at the beginning of the statement
- Format:

```
do
      statement;
while (expression);// don't forget final ;
```

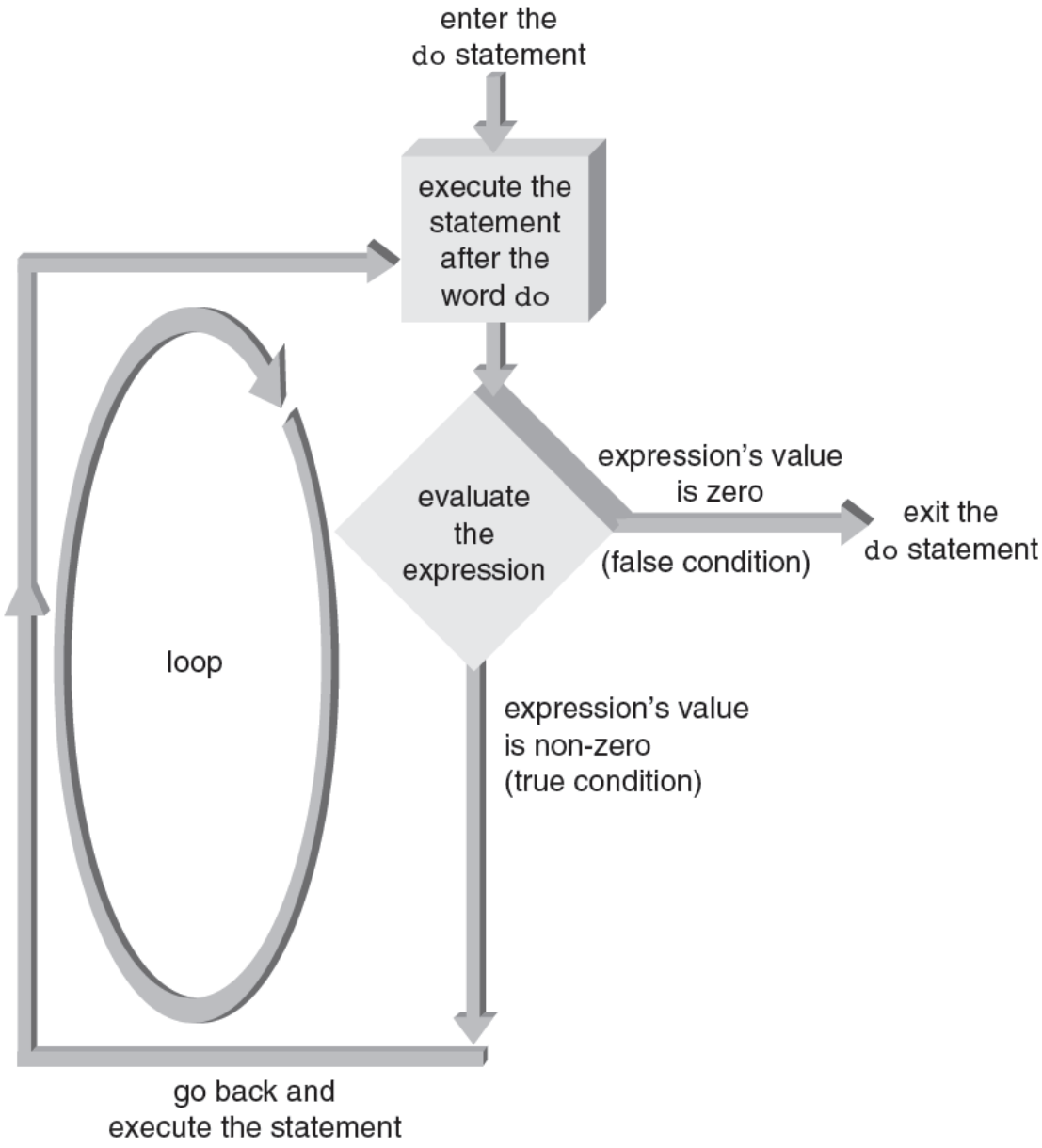

Figure 5.8 The do statement's flow of control

Validity Checks

- Provided by `do-while` statement through filtering of user-entered input
- Example:

```
do
{
  cout << "\nEnter an identification number: ";
  cin >> idNum;
  if (idNum < 100 || idNum > 1999)
  {
     cout << "\n An invalid number was just entered"
          << "\nPlease check the ID number and reenter";
  }
  else
     break; // break if a valid ID number was entered
} while(1); // this expression is always true
```

Common Programming Errors

- "Off by one" errors: loop executes one time too many or one time too few
  - Initial and final conditions to control loop must be carefully constructed
- Inadvertent use of assignment operator, =, in place of the equality operator, ==
  - This error is not detected by the compiler

Common Programming Errors (cont'd.)

- Using the equality operator when testing double-precision operands
  - Do not test expression (`fnum == 0.01`)
  - Replace by a test requiring absolute value of

    `(fnum - 0.01) < epsilon` for very small epsilon
- Placing a semicolon at end of the `for` statement parentheses:

```
for (count = 0; count < 10; count ++);
      total = total + num;
```

  - Creates a loop that executes 10 times and does nothing but increment count

Common Programming Errors (cont'd.)

- Using commas instead of semicolons to separate items in a `for` statement:

```
for (count = 1, count < 10, count ++)
```

  - Commas should be used to separate items within the separating and initializing lists
- Changing the value of the control variable used in the tested condition both inside the body of a `for` loop and in its altering list
- Omitting the final semicolon from the `do-while` statement

Summary

- `while`, `for`, and `do-while` statements create loops
  - These statements evaluate an expression
    - On the basis of the expression value, either terminate the loop or continue with it
  - Each pass through the loop is called a repetition or iteration
- `while` checks expression before any other statement in the loop
  - Variables in the tested expression must have values assigned before `while` is encountered

Summary (cont'd.)

- The `for` statement: fixed-count loops
  - Included in parentheses at top of loop:
    - Initializing expressions
    - Tested expression
    - Expressions that affect the tested expression
    - Other loop statements can also be included as part of the altering list

Summary (cont'd.)

- The `do-while` statement checks its expression at the end of the loop
  - Body of the loop must execute at least once
  - `do-while` loop must contain statement(s) that either:
    - Alter the tested expression's value or
    - Force a break from the loop