

STARTING OUT WITH

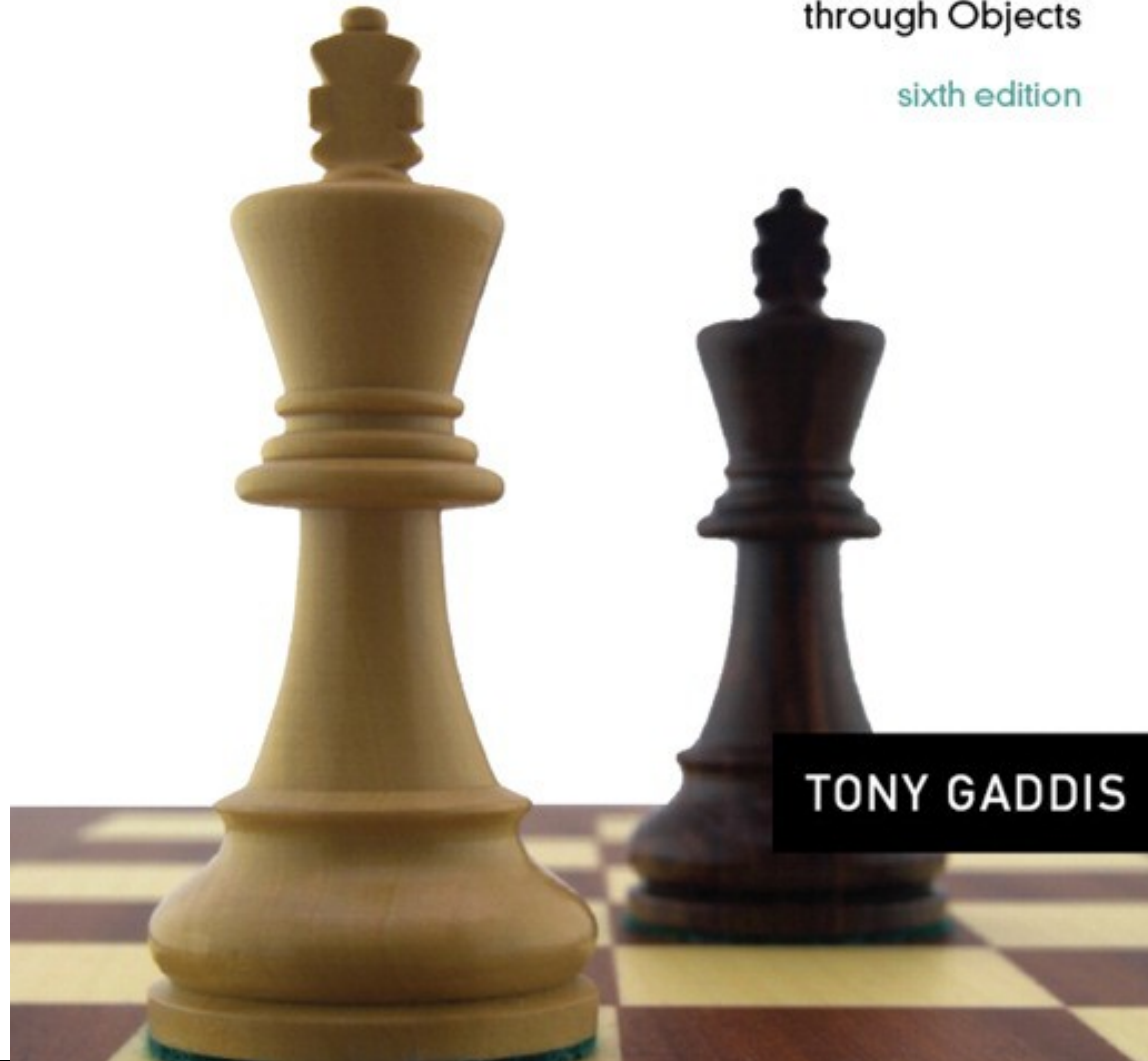
C++

From Control Structures
through Objects

sixth edition

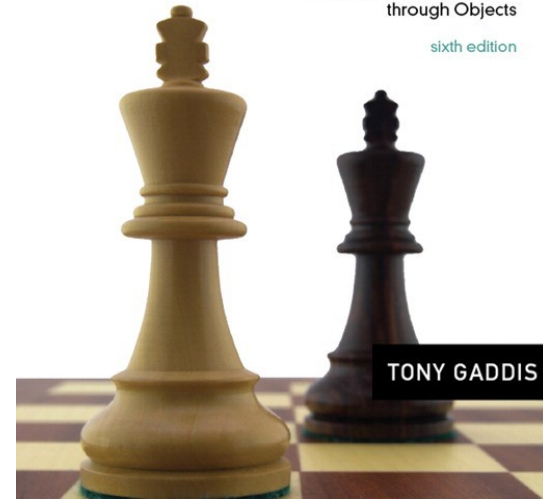
Chapter 5:

Looping



TONY GADDIS

5.1



TONY GADDIS

The Increment and Decrement Operators

The Increment and Decrement Operators



- `++` is the increment operator.

It adds one to a variable.

`val++;` is the same as `val = val + 1;`

- `++` can be used before (prefix) or after (postfix) a variable:

`++val;` `val++;`

The Increment and Decrement Operators



- `--` is the decrement operator.

It subtracts one from a variable.

`val--;` is the same as `val = val - 1;`

- `--` can be also used before (prefix) or after (postfix) a variable:

`--val;` `val--;`



Program 5-1

```
1 // This program demonstrates the ++ and -- operators.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int num = 4;    // num starts out with 4.
8
9     // Display the value in num.
10    cout << "The variable num is " << num << endl;
11    cout << "I will now increment num.\n\n";
12
13    // Use postfix ++ to increment num.
14    num++;
15    cout << "Now the variable num is " << num << endl;
16    cout << "I will increment num again.\n\n";
17
18    // Use prefix ++ to increment num.
19    ++num;
20    cout << "Now the variable num is " << num << endl;
21    cout << "I will now decrement num.\n\n";
22
23    // Use postfix -- to decrement num.
24    num--;
25    cout << "Now the variable num is " << num << endl;
26    cout << "I will decrement num again.\n\n";
27
```



Program 5-1 *(continued)*

```
28      // Use prefix -- to increment num.
29      --num;
30      cout << "Now the variable num is " << num << endl;
31      return 0;
32  }
```

Program Output

```
The variable num is 4
I will now increment num.

Now the variable num is 5
I will increment num again.

Now the variable num is 6
I will now decrement num.

Now the variable num is 5
I will decrement num again.

Now the variable num is 4
```

Prefix vs. Postfix



- `++` and `--` operators can be used in complex statements and expressions
- In prefix mode (`++val`, `--val`) the operator increments or decrements, *then* returns the value of the variable
- In postfix mode (`val++`, `val--`) the operator returns the value of the variable, *then* increments or decrements

Prefix vs. Postfix - Examples



```
int num, val = 12;
cout << val++; // displays 12,
               // val is now 13;
cout << ++val; // sets val to 14,
               // then displays it
num = --val;   // sets val to 13,
               // stores 13 in num
num = val--;   // stores 13 in num,
               // sets val to 12
```


Notes on Increment, Decrement



- Can be used in expressions:

```
result = num1++ + --num2;
```

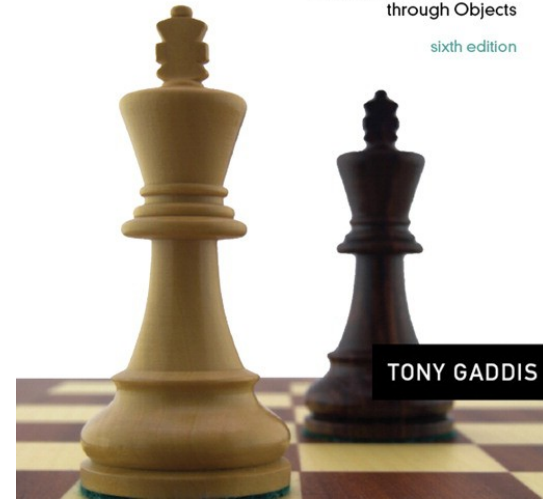
- Must be applied to something that has a location in memory. Cannot have:

```
result = (num1 + num2)++;
```

- Can be used in relational expressions:

```
if (++num > limit)
```

pre- and post-operations will cause different comparisons



TONY GADDIS

5.2

Introduction to Loops: The `while` Loop

Introduction to Loops:

The `while` Loop



- Loop: a control structure that causes a statement or statements to repeat
- General format of the `while` loop:

```
while (expression)  
    statement;
```
- *statement*; can also be a block of statements enclosed in { }

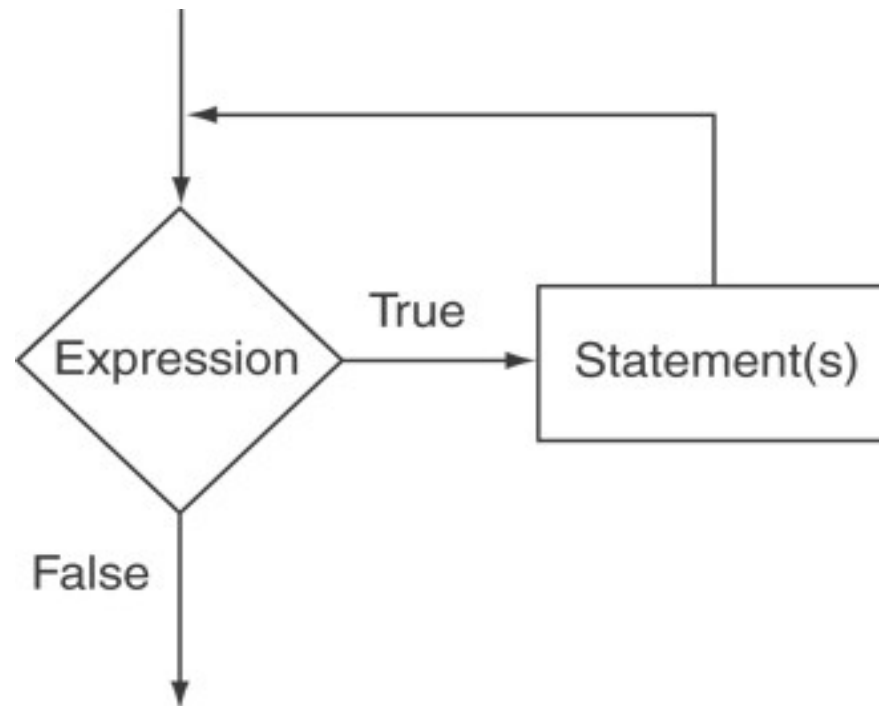
while Loop – How It Works



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

- *expression* is evaluated
 - if `true`, then *statement* is executed, and *expression* is evaluated again
 - if `false`, then the the loop is finished and program statements following *statement* execute

The Logic of a `while` Loop





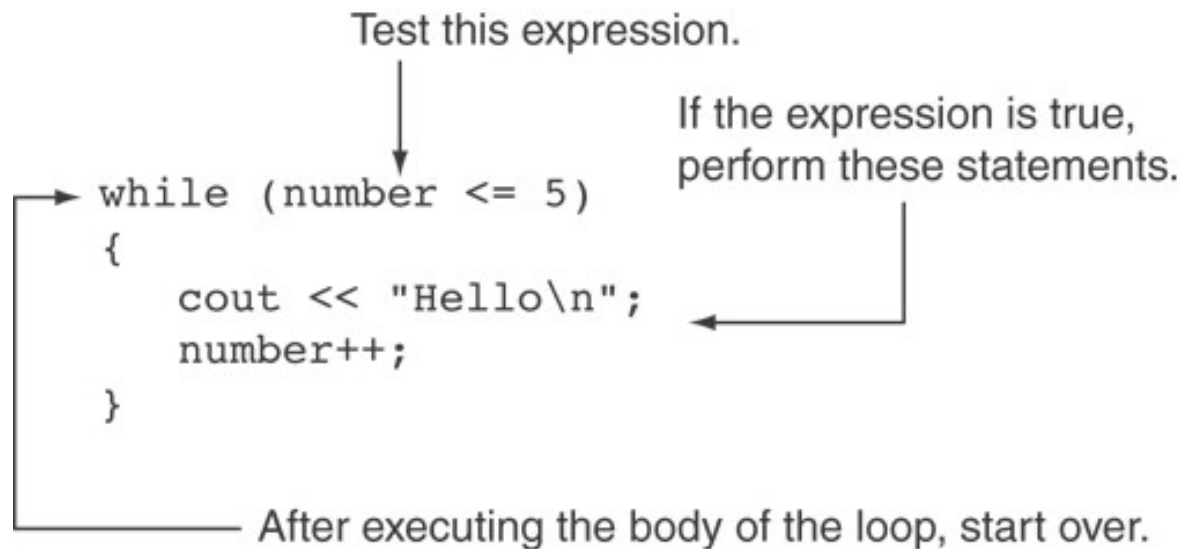
Program 5-3

```
1 // This program demonstrates a simple while loop.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int number = 1;
8
9     while (number <= 5)
10    {
11        cout << "Hello\n";
12        number++;
13    }
14    cout << "That's all!\n";
15    return 0;
16 }
```

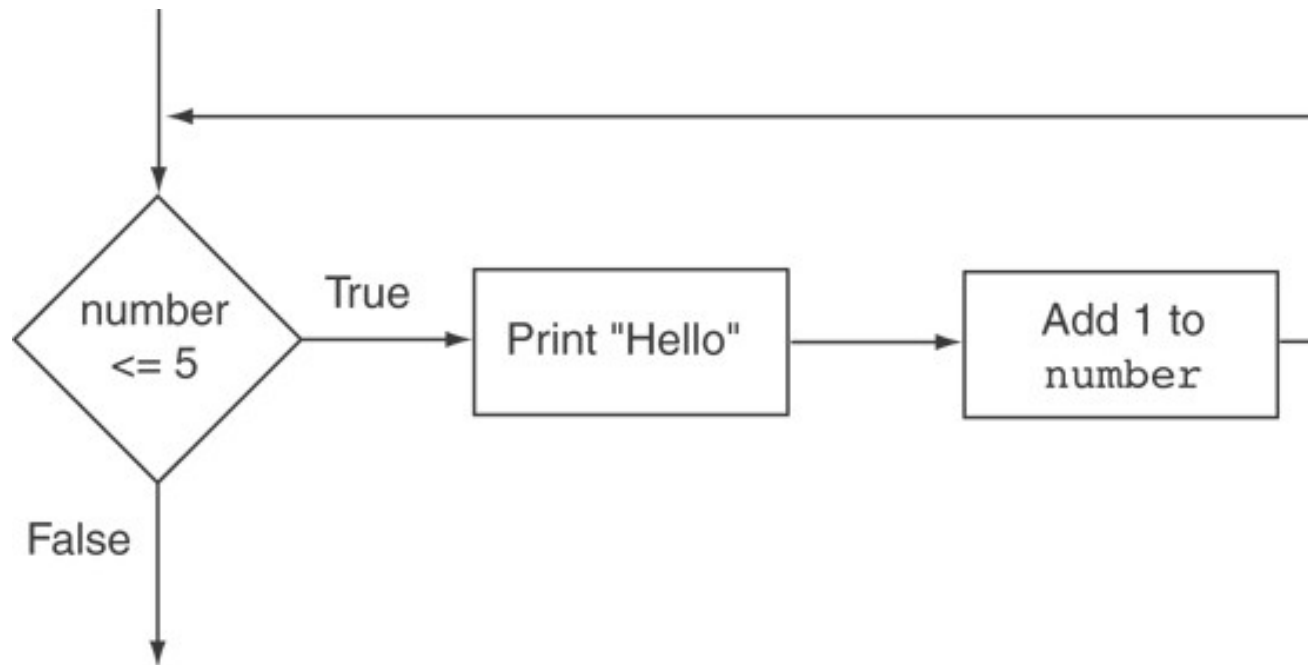
Program Output

```
Hello
Hello
Hello
Hello
Hello
That's all!
```

How the Loop in Lines 9 through 13 Works



Flowchart of the Loop



while is a Pretest Loop



- *expression* is evaluated before the loop executes. The following loop will never execute:

```
int number = 6;
while (number <= 5)
{
    cout << "Hello\n";
    number++;
}
```

Watch Out for Infinite Loops



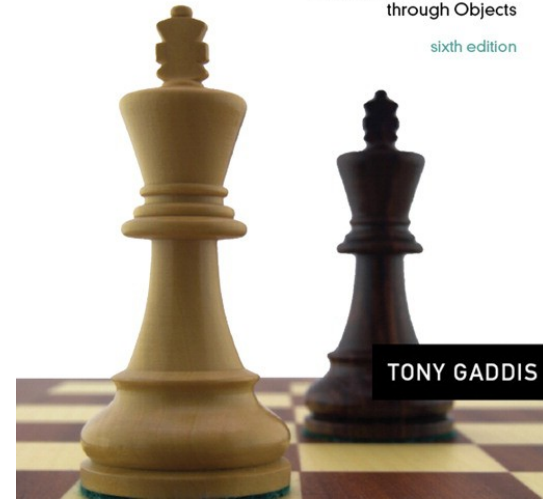
- The loop must contain code to make *expression* become false
- Otherwise, the loop will have no way of stopping
- Such a loop is called an *infinite loop*, because it will repeat an infinite number of times

An Infinite Loop



```
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```

5.3



TONY GADDIS

Using the `while` Loop for Input Validation

Using the `while` Loop for Input Validation



- Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.
- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Using the `while` Loop for Input Validation



- Here's the general approach, in pseudocode:

Read an item of input.

While the input is invalid

Display an error message.

Read the input again.

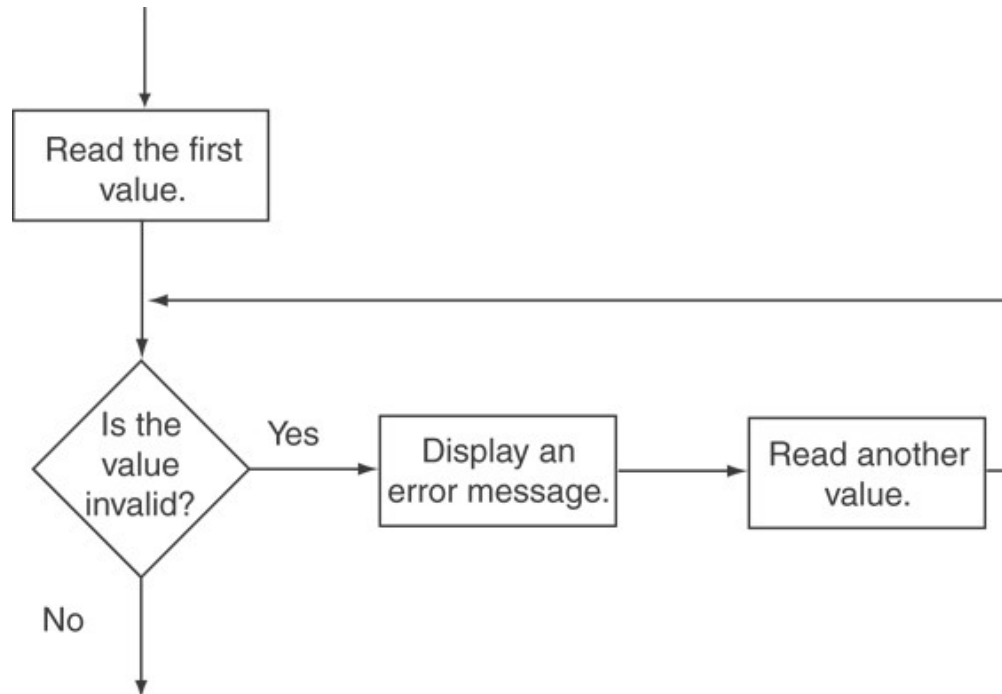
End While

Input Validation Example



```
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!"
        << "Enter a number less than 10: ";
    cin >> number;
}
```

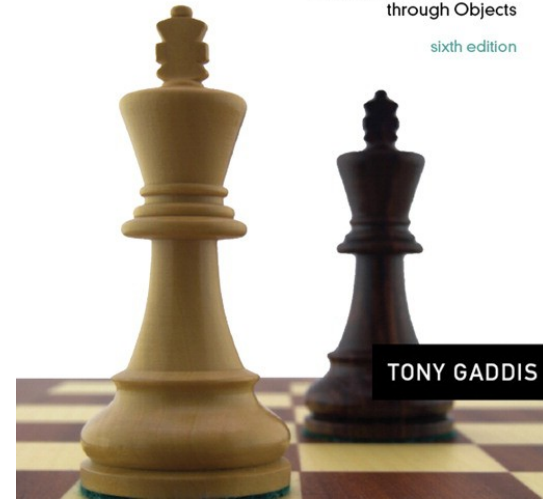
Flowchart



Input Validation Example from Program 5-5



```
29      // Get the number of players available.
30      cout << "How many players are available? ";
31      cin >> players;
32
33      // Validate the input.
34      while (players <= 0)
35      {
36          cout << "Please enter a positive number: ";
37          cin >> players;
38      }
```



TONY GADDIS

5.4

Counters

Counters



- Counter: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the loop control variable)
- Must be initialized before entering loop



Program 5-6

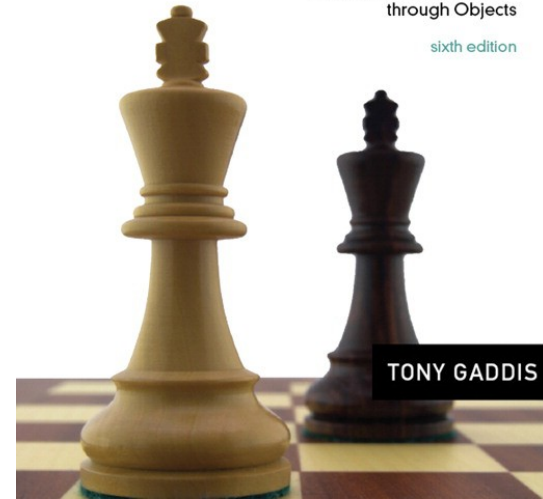
```
1  // This program displays the numbers 1 through 10 and
2  // their squares.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      int num = 1; //Initialize the counter.
9
10     cout << "Number Number Squared\n";
11     cout << "-----\n";
12     while (num <= 10)
13     {
14         cout << num << "\t\t" << (num * num) << endl;
15         num++; //Increment the counter.
16     }
17     return 0;
18 }
```



Program Output

Number	Number Squared
--------	----------------

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100



TONY GADDIS

5.5

The do-while Loop

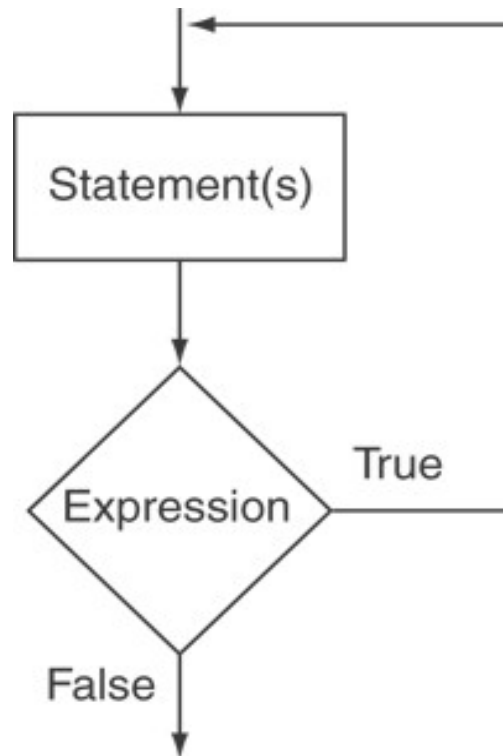
The do-while Loop



- do-while: a posttest loop – execute the loop, then test the `expression`
- General Format:

```
do
    statement; // or block in { }
while (expression);
```
- Note that a semicolon is required after (*expression*)

The Logic of a do-while Loop



do-while Example



```
int x = 1;
do
{
    cout << x << endl;
} while(x < 0);
```

Although the test expression is false, this loop will execute one time because `do-while` is a posttest loop.



Program 5-7

```
1 // This program averages 3 test scores. It repeats as
2 // many times as the user wishes.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int score1, score2, score3; // Three scores
9     double average;             // Average score
10    char again;                  // To hold Y or N input
11
12    do
13    {
14        // Get three scores.
15        cout << "Enter 3 scores and I will average them: ";
16        cin >> score1 >> score2 >> score3;
17
18        // Calculate and display the average.
19        average = (score1 + score2 + score3) / 3.0;
20        cout << "The average is " << average << ".\n";
21
22        // Does the user want to average another set?
23        cout << "Do you want to average another set? (Y/N) ";
24        cin >> again;
25    } while (again == 'Y' || again == 'y');
26    return 0;
27 }
```



Program Output with Example Input Shown in Bold

Enter 3 scores and I will average them: **80 90 70** [Enter]

The average is 80.

Do you want to average another set? (Y/N) **y** [Enter]

Enter 3 scores and I will average them: **60 75 88** [Enter]

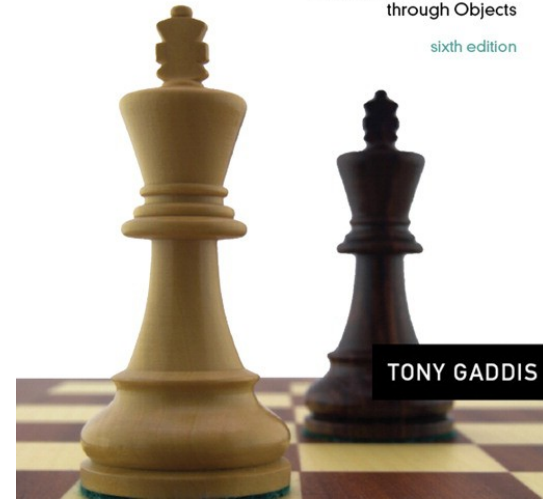
The average is 74.3333.

Do you want to average another set? (Y/N) **n** [Enter]

do-while Loop Notes



- Loop always executes at least once
- Execution continues as long as *expression* is true, stops repetition when *expression* becomes false
- Useful in menu-driven programs to bring user back to menu to make another choice (see Program 5-8 in the book)



TONY GADDIS

5.6

The for Loop

The for Loop



- Useful for counter-controlled loop
- General Format:

```
for(initialization; test; update)  
    statement; // or block in { }
```

- No semicolon after 3rd expression or after the)

for Loop - Mechanics



```
for(initialization; test; update)  
    statement; // or block in { }
```

- Perform *initialization*
- Evaluate *test* expression
 - If `true`, **execute** *statement*
 - If `false`, **terminate** loop execution
- 1) **Execute** *update*, then re-evaluate *test* expression

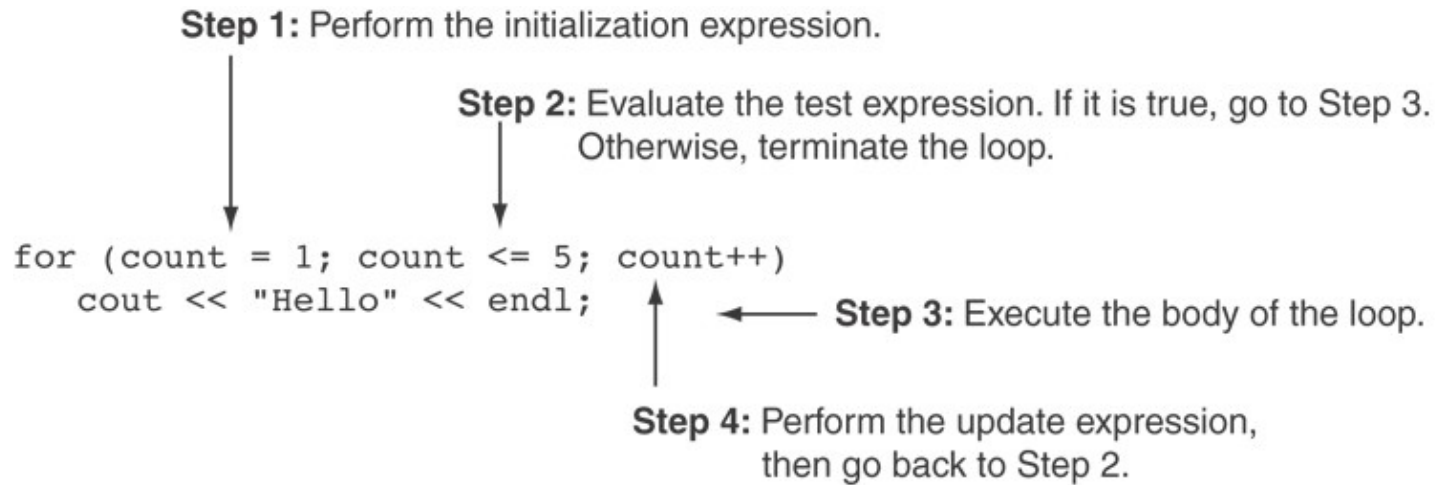
for Loop - Example



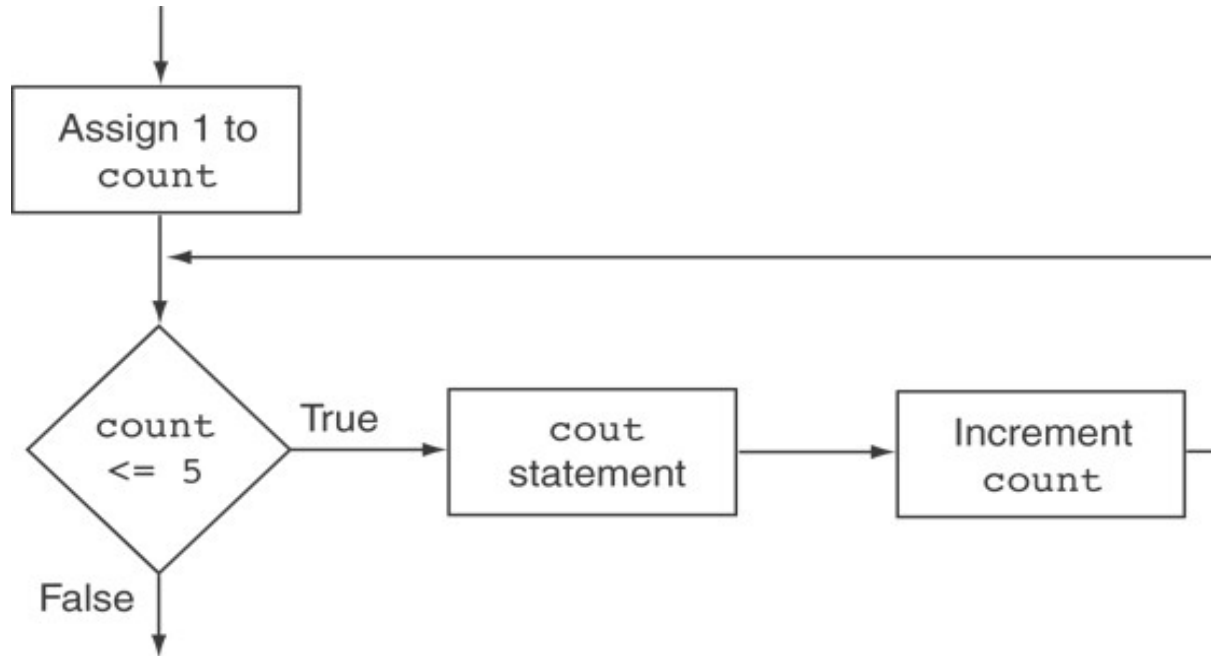
```
int count;
```

```
for (count = 1; count <= 5; count++)  
    cout << "Hello" << endl;
```


A Closer Look at the Previous Example



Flowchart for the Previous Example





Program 5-9

```
1  // This program displays the numbers 1 through 10 and
2  // their squares.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      int num;
9
10     cout << "Number      Number Squared\n";
11     cout << "-----\n";
12
13     for (num = 1; num <= 10; num++)
14         cout << num << "\t\t" << (num * num) << endl;
15     return 0;
16 }
```

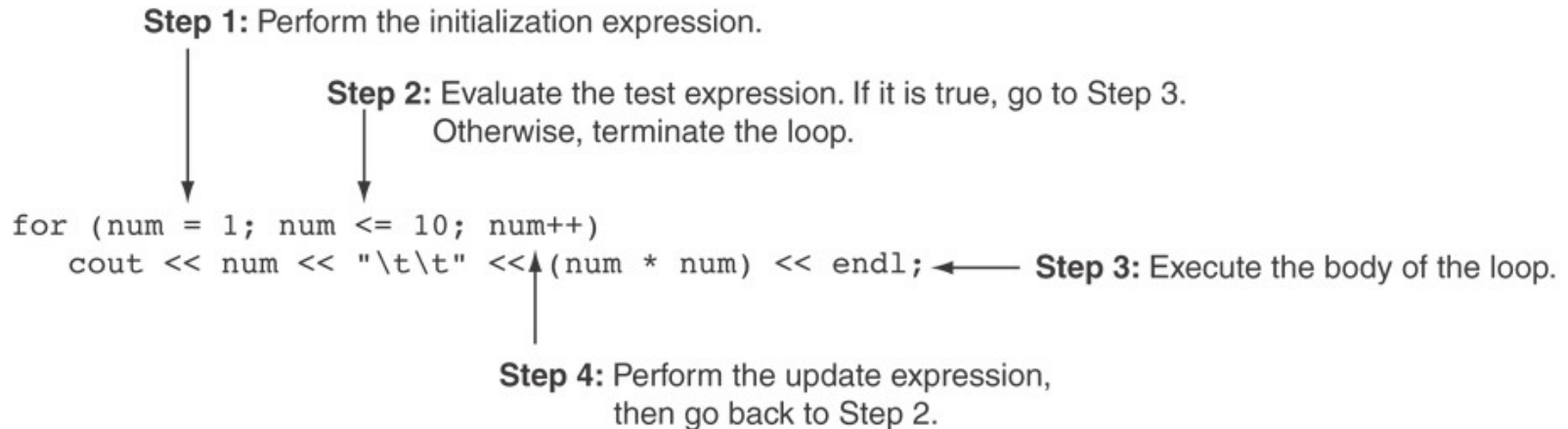


Program Output

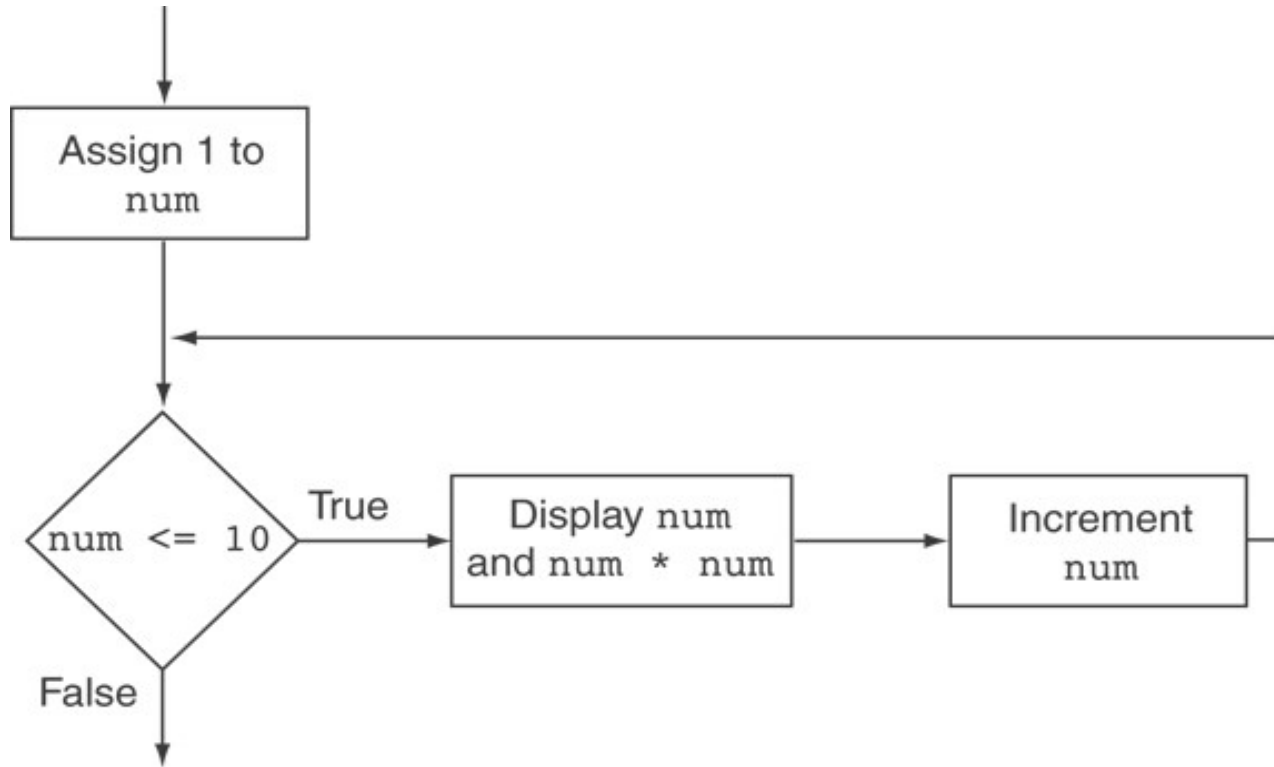
Number	Number Squared
--------	----------------

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

A Closer Look at Lines 13 through 14 in Program 5-9



Flowchart for Lines 13 through 14 in Program 5-9



When to Use the `for` Loop



- In any situation that clearly requires
 - an initialization
 - a false condition to stop the loop
 - an update to occur at the end of each iteration

The for Loop is a Pretest Loop



- The for loop tests its test expression before each iteration, so it is a pretest loop.
- The following loop will never iterate:

```
for (count = 11; count <= 10; count++)  
    cout << "Hello" << endl;
```


for Loop - Modifications



- You can have multiple statements in the *initialization* expression. Separate the statements with a comma:

Initialization Expression
↙

```
int x, y;  
for (x=1, y=1; x <= 5; x++)  
{  
    cout << x << " plus " << y  
        << " equals " << (x+y)  
        << endl;  
}
```

for Loop - Modifications



- You can also have multiple statements in the *test* expression. Separate the statements with a comma:

```
int x, y;  
for (x=1, y=1; x <= 5; x++, y++)  
{  
    cout << x << " plus " << y  
        << " equals " << (x+y)  
        << endl;  
}
```

Test Expression

An orange arrow originates from the text 'Test Expression' and points diagonally down and to the left, ending at the 'x++, y++' part of the for loop's test expression in the code block.

for Loop - Modifications



- You can omit the *initialization* expression if it has already been done:

```
int sum = 0, num = 1;  
for (; num <= 10; num++)  
    sum += num;
```

for Loop - Modifications

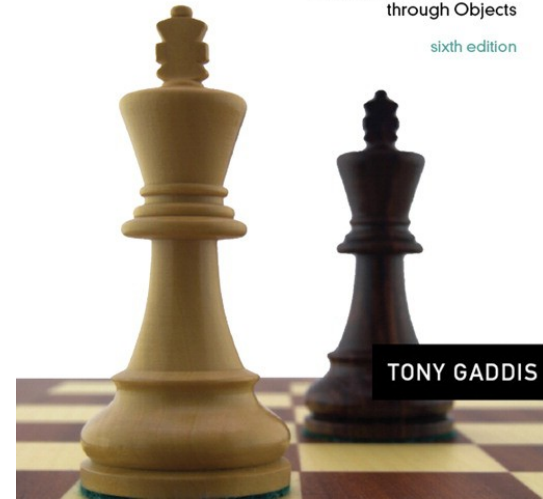


- You can declare variables in the *initialization* expression:

```
int sum = 0;  
for (int num = 0; num <= 10; num++)  
    sum += num;
```

The scope of the variable `num` is the `for` loop.

5.7



TONY GADDIS

Keeping a Running Total

Keeping a Running Total



- running total: accumulated sum of numbers from each repetition of loop
- accumulator: variable that holds running total

```
int sum=0, num=1; // sum is the
while (num <= 10) // accumulator
{
    sum += num;
    num++;
}
cout << "Sum of numbers 1 - 10 is"
      << sum << endl;
```



Program 5-12

```
1  // This program takes daily sales figures over a period of time
2  // and calculates their total.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9      int days;           // Number of days
10     double total = 0.0; // Accumulator, initialized with 0
11
12     // Get the number of days.
13     cout << "For how many days do you have sales figures? ";
14     cin >> days;
15
16     // Get the sales for each day and accumulate a total.
17     for (int count = 1; count <= days; count++)
18     {
19         double sales;
20         cout << "Enter the sales for day " << count << ": ";
21         cin >> sales;
22         total += sales; // Accumulate the running total.
23     }
24
```

(Program Continues)
5-55



Program 5-12 *(continued)*

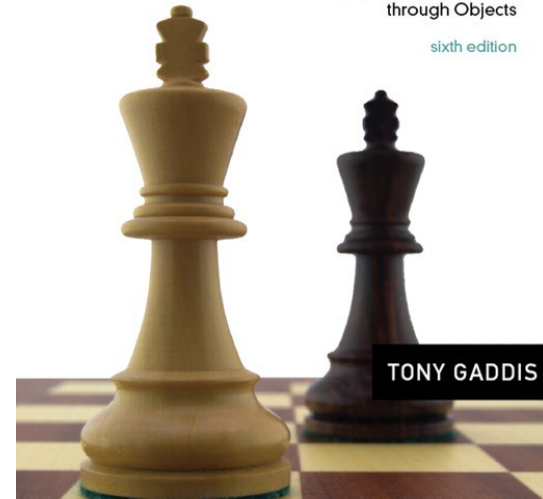
```
25      // Display the total sales.
26      cout << fixed << showpoint << setprecision(2);
27      cout << "The total sales are $" << total << endl;
28      return 0;
29  }
```

Program Output with Example Input Shown in Bold

```
For how many days do you have sales figures? 5 [Enter]
Enter the sales for day 1: 489.32 [Enter]
Enter the sales for day 2: 421.65 [Enter]
Enter the sales for day 3: 497.89 [Enter]
Enter the sales for day 4: 532.37 [Enter]
Enter the sales for day 5: 506.92 [Enter]
The total sales are $2448.15
```


5.8

Sentinels



TONY GADDIS

Sentinels



- sentinel: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, *e.g.*, -999 for a test score
- Used to terminate input when user may not know how many values will be entered



Program 5-13

```
1  // This program calculates the total number of points a
2  // soccer team has earned over a series of games. The user
3  // enters a series of point values, then -1 when finished.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int game = 1,    // Game counter
10         points,      // To hold a number of points
11         total = 0;   // Accumulator
12
13     cout << "Enter the number of points your team has earned\n";
14     cout << "so far in the season, then enter -1 when finished.\n\n";
15     cout << "Enter the points for game " << game << ": ";
16     cin >> points;
17
18     while (points != -1)
19     {
20         total += points;
21         game++;
22         cout << "Enter the points for game " << game << ": ";
23         cin >> points;
24     }
25     cout << "\nThe total points are " << total << endl;
26     return 0;
27 }
```

(Program Continues)



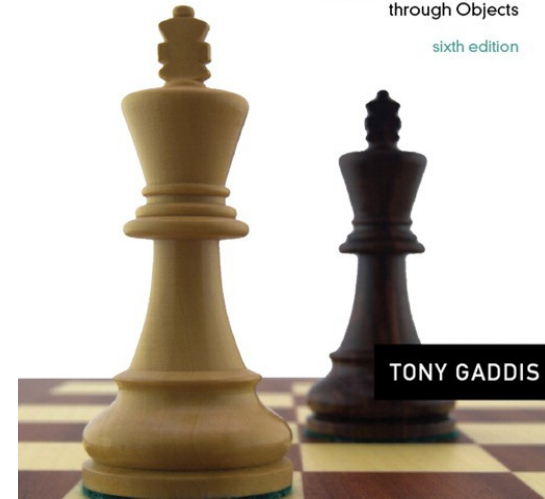
Program Output with Example Input Shown in Bold

Enter the number of points your team has earned
so far in the season, then enter -1 when finished.

Enter the points for game 1: **7 [Enter]**
Enter the points for game 2: **9 [Enter]**
Enter the points for game 3: **4 [Enter]**
Enter the points for game 4: **6 [Enter]**
Enter the points for game 5: **8 [Enter]**
Enter the points for game 6: **-1 [Enter]**

The total points are 34

5.9



TONY GADDIS

Using a Loop to Read Data from a File

Using a Loop to Read Data from a File



- The stream extraction operator `>>` returns `true` when a value was successfully read, `false` otherwise
- Can be tested in a `while` loop to continue execution as long as values are read from the file:

```
while (inputFile >> number) ...
```



Program 5-15

```
1  // This program displays all of the numbers in a file.
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main()
7  {
8      ifstream inputFile; // File stream object
9      int number;          // To hold a value from the file
10
11     inputFile.open("numbers.txt"); // Open the file.
12     if (!inputFile)                // Test for errors.
13         cout << "Error opening file.\n";
14     else
15     {
16         while (inputFile >> number) // Read a number
17         {
18             cout << number << endl; // Display the number.
19         }
20         inputFile.close();          // Close the file.
21     }
22     return 0;
23 }
```



TONY GADDIS

5.10

Deciding Which Loop to Use

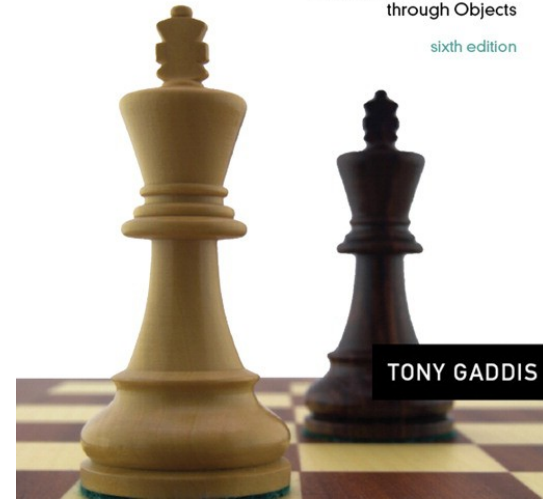
Deciding Which Loop to Use



- `while`: pretest loop; loop body may not be executed at all
- `do-while`: posttest loop; loop body will always be executed at least once
- `for`: pretest loop with initialization and update expression; useful with counters, or if precise number of repetitions is needed

5.11

Nested Loops



TONY GADDIS

Nested Loops



- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```
for (row=1; row<=3; row++) //outer
    for (col=1; col<=3; col++) //inner
        cout << row * col << endl;
```

Lines from Program 5-16



```
22     // Determine each student's average score.
23     for (int student = 1; student <= numStudents; student++)
24     {
25         total = 0;           // Initialize the accumulator.
26         for (int test = 1; test <= numTests; test++)
27         {
28             int score;
29             cout << "Enter score " << test << " for ";
30             cout << "student " << student << ": ";
31             cin >> score;
32             total += score;
33         }
34         average = total / numTests;
35         cout << "The average score for student " << student;
36         cout << " is " << average << ".\n\n";
37     }
```

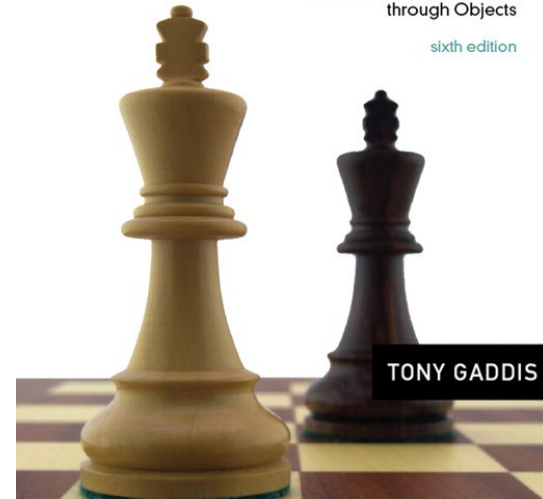
Nested Loops - Notes



- Inner loop goes through all repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops.

5.12

Breaking Out of a Loop



TONY GADDIS

Breaking Out of a Loop



- Can use `break` to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand and debug
- When used in an inner loop, terminates that loop only and goes back to outer loop



TONY GADDIS

5.13

The continue Statement

The `continue` Statement



- Can use `continue` to go to end of loop and prepare for next repetition
 - `while`, `do-while` loops: go to test, repeat loop if test passes
 - `for` loop: perform update step, then test, then repeat loop if test passes
- Use sparingly – like `break`, can make program logic hard to follow