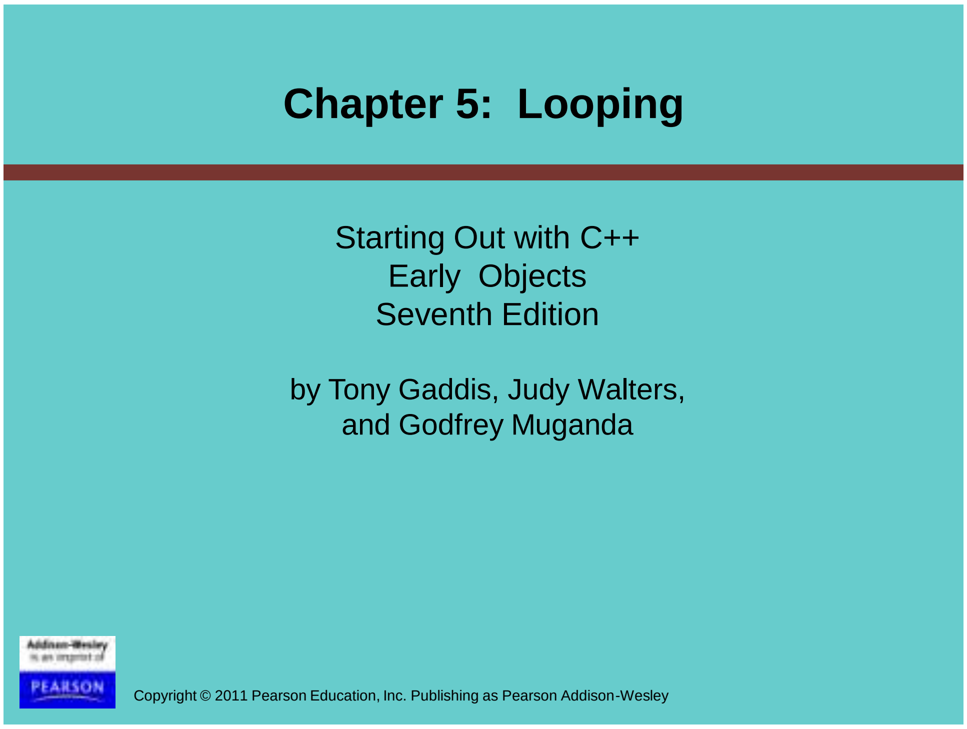


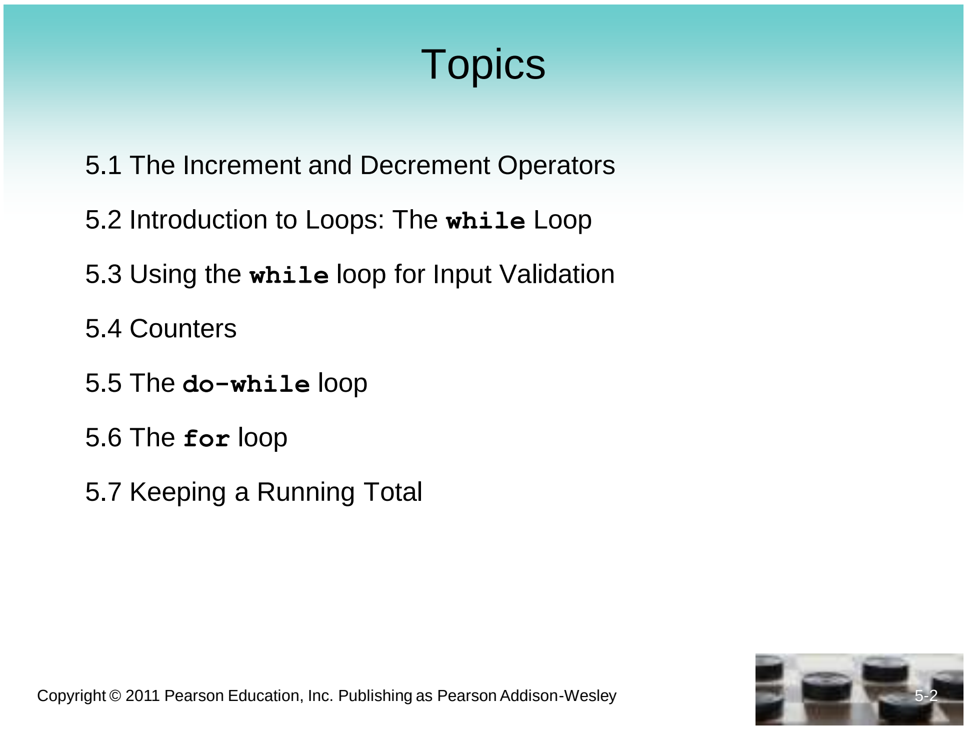
Chapter 5: Looping

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley is an imprint of PEARSON

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Topics

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Topics (continued)

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.1 The Increment and Decrement Operators

- **`++`** adds one to a variable

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

- **`--`** subtracts one from a variable

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

- can be used in prefix mode (before) or postfix mode (after) a variable

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Prefix Mode

- **`++val`** and **`--val`** increment or decrement the variable, *then* return the new value of the variable.
- It is this returned new value of the variable that is used in any other operations within the same statement

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Prefix Mode Example

```
int x = 1, y = 1;

x = ++y;          // y is incremented to 2
                  // Then 2 is assigned to x
cout << x
  << "   " << y; // Displays 2   2

x = --y;          // y is decremented to 1
                  // Then 1 is assigned to x
cout << x
  << "   " << y; // Displays 1 1
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Postfix Mode

- **`val++`** and **`val--`** return the old value of the variable, *then* increment or decrement the variable
- It is this returned old value of the variable that is used in any other operations within the same statement

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Postfix Mode Example

```
int x = 1, y = 1;

x = y++;        // y++ returns a 1
                // The 1 is assigned to x
                // and y is incremented to 2
cout << x
  << "  " << y; // Displays 1  2

x = y--;        // y-- returns a 2
                // The 2 is assigned to x
                // and y is decremented to 1
cout << x
  << "  " << y; // Displays 2 1
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Increment & Decrement Notes

- Can be used in arithmetic expressions
  ```
  result = num1++ + --num2;
  ```
- Must be applied to something that has a location in memory. Cannot have
  ```
  result = (num1 + num2)++; // Illegal
  ```
- Can be used in relational expressions
  ```
  if (++num > limit)
  ```
- Pre- and post-operations will cause different comparisons

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.2 The **while** Loop

- Loop: part of program that may execute > 1 time (*i.e.,* it repeats)
- format:
  ```
  while (condition)
  {
    statement(s);
  }
  ```
  No ; here
- The **{ }** can be omitted if there is only one statement in the body of the loop

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

How the **`while`** Loop Works

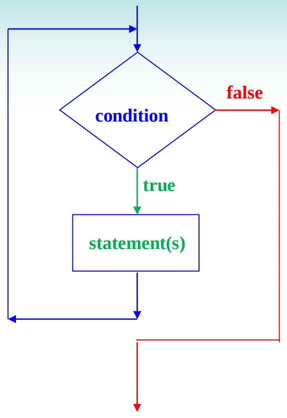
```
while (condition)
{
  statement(s);
}
```

condition is evaluated

- if it is true, the ***statement(s)*** are executed, and then ***condition*** is evaluated again
- if it is false, the loop is exited

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`while` Loop Flow of Control



Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`while` Loop Example

```
int val = 5;

while (val >= 0)
{
    cout << val << "   ";
    val--;
}
```

- produces output:

```
5  4  3  2  1  0
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`while` Loop is a Pretest Loop

- **`while`** is a pretest loop (***`condition`*** is evaluated <u>before</u> the loop executes)

- If the condition is initially false, the statement(s) in the body of the loop are never executed

- If the condition is initially true, the statement(s) in the body continue to be executed until the condition becomes false

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Exiting the Loop

- The loop must contain code to allow ***`condition`*** to eventually become **`false`** so the loop can be exited
- Otherwise, you have an infinite loop (*i.e.*, a loop that does not stop)
- Example infinite loop:

```
x = 5;
while (x > 0)     // infinite loop because
   cout << x;     // x is always > 0
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Common Loop Errors

- Don't forget the { } :

```
int numEntries = 1;
while (numEntries <=3)
   cout << "Still working … ";
   numEntries++; // not in the loop body
```

- Don't use = when you mean to use ==

```
while (numEntries = 3)   // always true
{
   cout << "Still working … ";
   numEntries++;
}
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.3 Using the **`while`** Loop for Input Validation

Loops are an appropriate structure for validating user input data

1. Prompt for and read in the data.
2. Use a **`while`** loop to test if data is valid.
3. Enter the loop only if data is <u>not</u> valid.
4. Inside the loop, display error message and prompt the user to re-enter the data.
5. The loop will not be exited until the user enters valid data.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Input Validation Loop Example

```
cout << "Enter a number (1-100) and"
     << " I will guess it. ";
cin  >> number;
while (number < 1 || number > 100)
{
   cout << "Number must be between 1 and 100."
        << " Re-enter your number. ";
   cin  >> number;
}
// Code to use the valid number goes here.
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.4 Counters

- Counter: variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (loop control variable)
- Must be initialized before entering loop
- May be incremented/decremented either inside the loop or in the loop test

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Letting the User Control the Loop

- Program can be written so that user input determines loop repetition
- Can be used when program processes a list of items, and user knows the number of items
- User is prompted before loop. Their input is used to control number of repetitions

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

User Controls the Loop Example

```
int num, limit;
cout << "Table of squares\n";
cout << "How high to go? ";
cin  >> limit;
cout << "\n\nnumber square\n";
num = 1;
while (num <= limit)
{   cout << setw(5) << num << setw(6)
         << num*num << endl;
    num++;
}
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.5 The `do-while` Loop

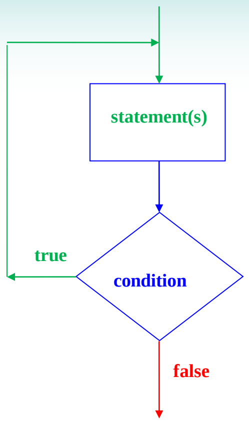
- **`do-while`**: a post test loop (***`condition`*** is evaluated <u>after</u> the loop executes)
- Format:

```
do
{
    1 or more statements;
} while (condition);
```

Notice the required ;

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`do-while` Flow of Control



Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`do-while` Loop Notes

- Loop always executes at least once
- Execution continues as long as ***`condition`*** is **`true`**; the loop is exited when ***`condition`*** becomes **`false`**
- Useful in menu-driven programs to bring user back to menu to make another choice

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.6 The **`for`** Loop

- Pretest loop that executes zero or more times
- Useful for counter-controlled loop

Required ;

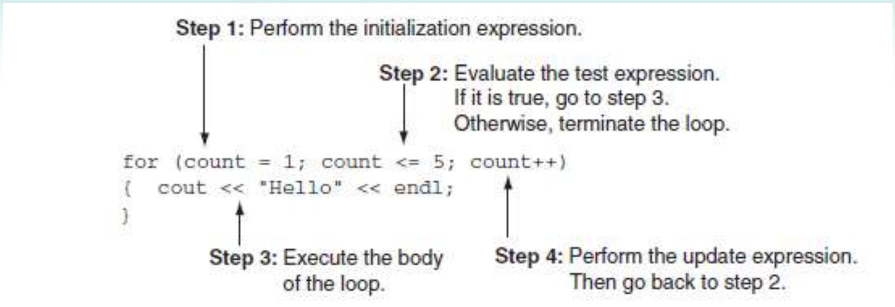
- Format:

```
for( initialization; test; update )
{    1 or more statements;
}
```

No ; goes here

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`for` Loop Mechanics



Step 1: Perform the initialization expression.

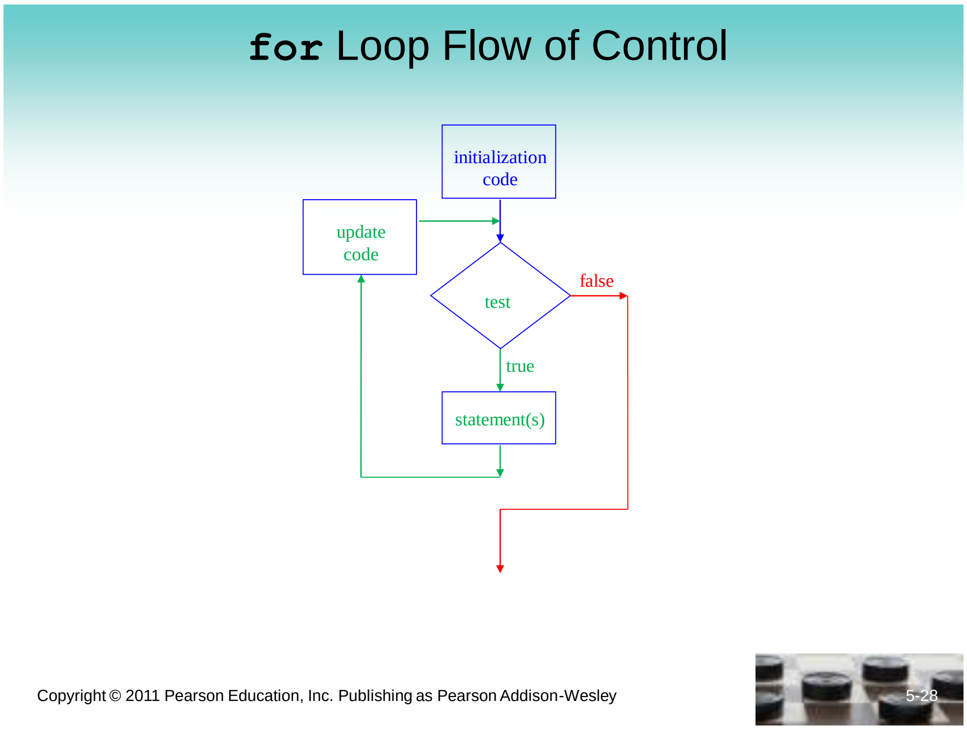
Step 2: Evaluate the test expression. If it is true, go to step 3. Otherwise, terminate the loop.

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

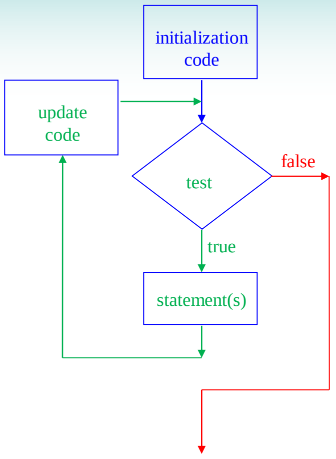
Step 3: Execute the body of the loop.

Step 4: Perform the update expression. Then go back to step 2.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



`for` Loop Flow of Control



Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
for (number = 1; number <= 5; number++)
        cout << number << " ";

cout << endl;
```

This loop will produce the following output:

```
1 2 3 4 5
```

```
// This program uses a for loop to display the numbers 1-5
// and their squares.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
      int num;
      cout << "Number Square\n";
      cout << "--------------\n";

      for (num = 1; num <= 5; num++)
              cout << setw(4) << num << setw(7) << (num * num)
              << endl;
      return 0;
}
```

Program Output

```
Number Squared
------------------
1         1
2         4
3         9
4         16
5         25
```

The for Loop is a Pretest Loop

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

Because the variable `count` is initialized to a value that makes the test expression false from the beginning, this loop terminates as soon as it begins.

Avoid Modifying the Counter Variable in the Body of the for Loop

The following loop, for example, increments x twice for each iteration:

```
for (x = 1; x <= 10; x++)
{
        cout << x << endl;
        x++; // Wrong!
}
```

Other Forms of the Update Expression

Here is a loop that displays all the even numbers from 2 through 100 by adding 2 to its counter:

```
for (num = 2; num <= 100; num += 2 )
        cout << num << endl;
```

And here is a loop that counts backward from 10 down to 0:

```
for (num = 10; num >= 0; num-- )
        cout << num << endl;
```

Defining a Variable in the for Loop's Initialization Expression

Not only may the counter variable be initialized in the initialization expression, it may be defined there as well.

```
for (int num = 1; num <= 5; num++)
   cout << setw(4) << num << setw(7)
        << (num * num) << endl;
```

When a variable is defined in the initialization expression of a for loop, the scope of the variable is limited to the loop. This means you cannot access the variable in statements outside the loop.

```
for (int count = 1; count <= 10; count++)
        cout << count << endl;

cout << "count is now " << count << endl; // ERROR!
```

Creating a User-Controlled for Loop

```
// Get the final counter value
cout << "How many times should the loop execute? ";
cin >> finalValue;

for (int num = 1; num <= finalValue; num++)
{
        // Statements in the loop body go here.
}
```

for Loop Example

```
int sum = 0, num;
for (num = 1; num <= 10; num++)
   sum += num;
cout << "Sum of numbers 1 - 10 is "
      << sum << endl;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

for Loop Notes

- If ***test*** is false the first time it is evaluated, the body of the loop will not be executed
- The update expression can increment or decrement by any amount
- Variables used in the initialization section should not be modified in the body of the loop

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

`for` Loop Modifications

- Can define variables in initialization code
  - Their scope is the **`for`** loop
- Initialization and update code can contain more than one statement
  - Separate statements with commas
- Example:

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

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

More **`for`** Loop Modifications
(These are NOT Recommended)

- Can omit ***`initialization`*** if already done

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

- Can omit ***`update`*** if done in loop

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

- Can omit ***`test`*** – may cause an infinite loop

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

- Can omit loop body if all work is done in header ??

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.7 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;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
int numDays;                    // Number of days
double dailySales,              // The sales amount for a single day
       totalSales = 0.0,        // Accumulator, initialized with 0
       averageSales;            // The average daily sales amount

// Get the number of days
cout << "For how many days do you have sales figures? ";
cin >> numDays;

// Get the sales for each day and accumulate a total
for (int day = 1; day <= numDays; day++) // day is the counter
{
     cout << "Enter the sales for day " << day << ": ";
     cin >> dailySales;
     totalSales += dailySales; // Accumulate the running total
}
// Compute the average daily sales
averageSales = totalSales / numDays;
```

5.8 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

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Sentinel Example

```
int total = 0;
cout << "Enter points earned "
     << "(or -1 to quit): ";
cin  >> points;
while (points != -1) // -1 is the sentinel
{
   total += points;
   cout << "Enter points earned: ";
   cin  >> points;
}
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.9 Using a Loop to Read Data From a File

- A Loop can be used to read in each piece of data from a file
- It is not necessary to know how much data is in the file
- Several methods exist to test for the end of the file

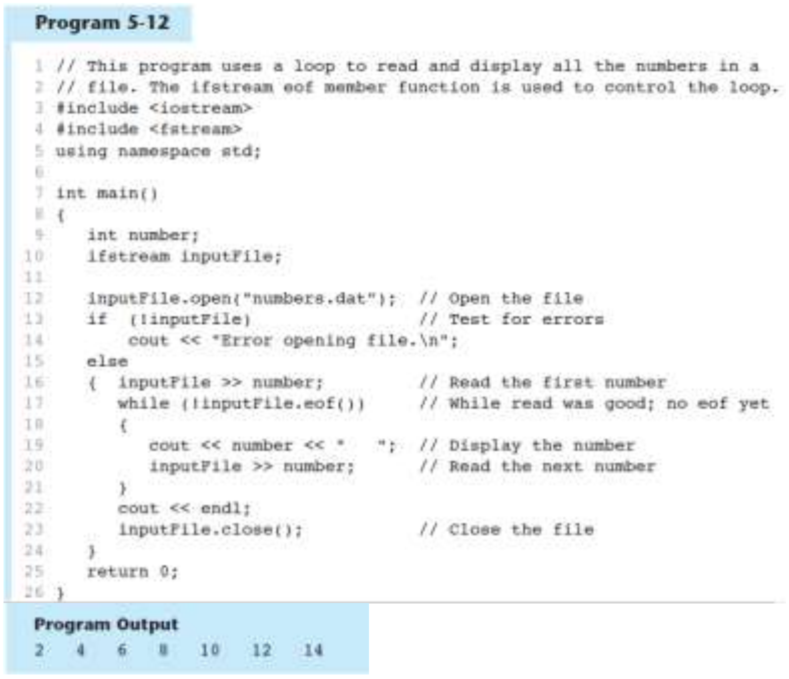
Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Test for the End of a File

- **`eof()`** member function returns **`true`** when the previous read encountered the end of file; returns **`false`** otherwise
- Example:

```
datafile >> score;
while (!datafile.eof())
{
   sum += score;
   datafile >> score;
}
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 5-12

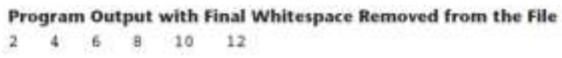
```
// This program uses a loop to read and display all the numbers in a
// file. The ifstream eof member function is used to control the loop.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   int number;
   ifstream inputFile;

   inputFile.open("numbers.dat");  // Open the file
   if  (!inputFile)                // Test for errors
       cout << "Error opening file.\n";
   else
   {  inputFile >> number;         // Read the first number
      while (!inputFile.eof())     // While read was good; no eof yet
      {
         cout << number << "   "; // Display the number
         inputFile >> number;      // Read the next number
      }
      cout << endl;
      inputFile.close();           // Close the file
   }
   return 0;
}
```

Program Output

```
2   4   6   8   10   12   14
```

Problems Using **`eof()`**

- For the **`eof()`** function to work correctly using this method, there must be a whitespace (space, tab, or [Enter]) after the last piece of data
- Otherwise the end of file will be encountered when reading the final data value and it will not be processed



Program Output with Final Whitespace Removed from the File

```
2   4   6   8   10   12
```

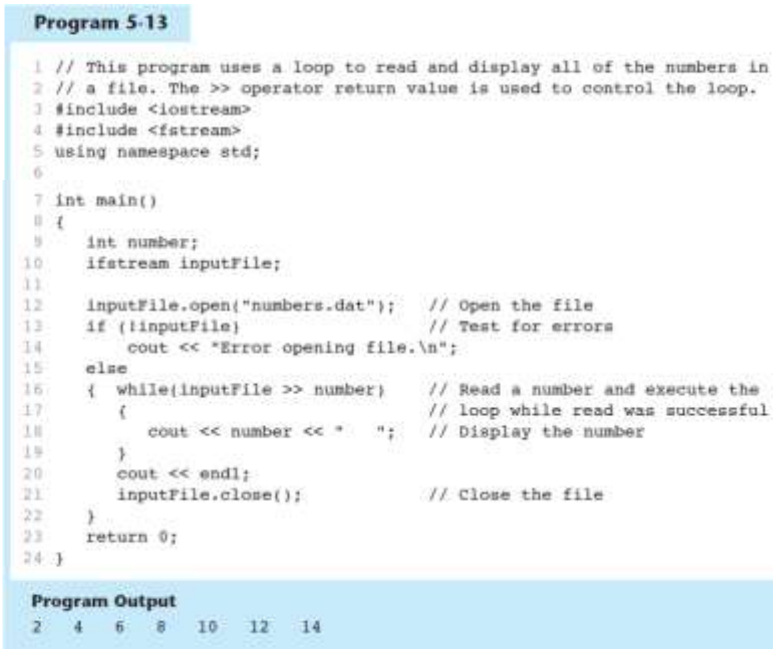
Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Using the **>>** Operation

- The stream extraction operator (**>>**) returns a value indicating if a read is successful
- This can be tested to find the end of file since the read “fails” when there is no more data
- Example:

```
while (datafile >> score)
   sum += score;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 5-13

```
// This program uses a loop to read and display all of the numbers in
// a file. The >> operator return value is used to control the loop.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   int number;
   ifstream inputFile;

   inputFile.open("numbers.dat");   // Open the file
   if (!inputFile)                  // Test for errors
      cout << "Error opening file.\n";
   else
   {  while(inputFile >> number)    // Read a number and execute the
      {                             // loop while read was successful
         cout << number << "   ";   // Display the number
      }
      cout << endl;
      inputFile.close();            // Close the file
   }
   return 0;
}
```

Program Output

```
2   4   6   8   10   12   14
```

5.10 Deciding Which Loop to Use

- `while`: pretest loop (loop body may not be executed at all)
- `do-while`: post test loop (loop body will always be executed at least once)
- `for`: pretest loop (loop body may not be executed at all); has initialization and update code; is useful with counters or if precise number of repetitions is known

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.11 Nested Loops

- A nested loop is a loop inside the body of another loop
- Example:

```
for (row = 1; row <= 3; row++)
{
   for (col = 1; col <= 3; col++)
   {
      cout << row * col << endl;
   }
}
```

outer loop

inner loop

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Notes on Nested Loops

- Inner loop goes through all its 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. In previous example, inner loop repeats 9 times

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

25. Write a nested loop that displays the following ouput:

```
*****
*****
*****
```

18. Rectangle Display
Write a program that asks the user for two positive integers between 2 and 10 to use for the length and width of a rectangle. If the numbers are different, the larger of the two numbers should be used for the length and the smaller for the width. The program should then display a rectangle of this size on the screen using the character ‘X’. For example, if the user enters either 2 5 or 5 2, the program should display the following:

```
XXXXX
XXXXX
```

Note: Better to use rows and columns instead of length and width.

5.12 Breaking Out of a Loop

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

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.13 The **`continue`** Statement

- Can use **`continue`** to go to end of loop and prepare for next repetition
  - **`while`** and **`do-while`** loops go to test and repeat the loop if test condition is true
  - **`for`** loop goes to update step, then tests, and repeats loop if test condition is true
- Use sparingly – like **`break`**, can make program logic hard to follow

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

5.14 Creating Good Test Data

- When testing a program, the quality of the test data is more important than the quantity.
- Test data should show how different parts of the program execute
- Test data should evaluate how program handles:
  - normal data
  - data that is at the limits the valid range
  - invalid data

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

3.12 Introduction to Files

- Can use a file instead of keyboard for program input
- Can use a file instead of monitor screen for program output
- Files are stored on secondary storage media, such as disk
- Files allow data to be retained between program executions

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

There are five steps that must be taken when a file is used by a program:

1. Include the header file needed to perform file input/output.
2. Define a file stream object.
3. Open the file.
4. Use the file.
5. Close the file.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Step 1: Include the header file needed to perform file input/output.

The file fstream contains all the definitions necessary for file operations. It is included with the following statement:

```
#include <fstream>
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Step 2: Define a file stream object.

We will need to define one or more file stream objects. They are called stream objects because a file can be thought of as a stream of data. File stream objects work very much like cin and cout objects.

Streams of data can be sent to a file stream object, which writes the data to a file. Data that is read from a file flows from a file stream object into other variables.

The fstream header file contains definitions for the data types ofstream, ifstream, and fstream.

Before a C++ program can work with a file, it must define an object of one of these data types.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

ofstream Output file stream. This data type can be used to open *output* files and write data to them.

ifstream Input file stream. This data type can be used to open existing input files and read data from them into memory.

fstream File stream. This data type can be used to open files, write data to them, and read data from them. With the fstream data type,
data may be copied from variables into a file, or from a file into variables.

In this section we only discuss the ofstream and ifstream types. The fstream type is covered in Chapter 13.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Here are example statements that define ofstream and ifstream objects:

```
ofstream outputFile;
ifstream inputFile;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Step 3: Open the file.

Outside of the C++ program, a file is identified by its name. Inside a C++ program, however, a file is identified by a stream object. The object and the file name are linked when the file is opened.

```
inputFile.open("customer.dat"); // Open an input file
```

It is also possible to define a file stream object and open a file all in one statement. Here is an example:

```
ifstream inputFile("customer.dat");
```

When no path is given, the program will look for the file in a default directory.

If the file you want to open is not in the default directory, you will need to specify its location as well as its name.

```
outputFile.open("C:\\data\\invetory.dat");
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

Step 4: Use the file.

Writing information to a file

```
outputFile << "I love C++ programming";
```

As you can see, the statement looks like a cout statement, except the file stream object name replaces cout. Here is a statement that writes both a string and the contents of a variable to a file:

```
outputFile << "Price: " << Price;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
// This program uses the << operator to write information to a file.
#include <iostream>
#include <fstream> // Needed to use files
using namespace std;

int main()
{
        ofstream outputFile;
        outputFile.open("demofile.txt");

        cout << "Now writing information to the file.\n";
        // Write 3 great names to the file
        outputFile << "Bach\n";
        outputFile << "Beethoven\n";
        outputFile << "Mozart\n";

        // Close the file
        outputFile.close();
        cout << "Done.\n";
        return 0;
}
```

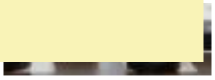
Program Screen Output

```
Now writing information to the file.
Done.
```

Output to File demofile.txt

```
Bach
Beethoven
Mozart
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addiso



```
// This program uses the >> operator to read information from a file.
#include <iostream>
#include <fstream> // Needed to use files
#include <string>
using namespace std;

int main()
{
        ifstream inFile;
        string name;

        inFile.open("demofile.txt");
        cout << "Reading information from the file.\n\n";

        inFile >> name; // Read name 1 from the file
        cout << name << endl; // Display name 1

        inFile >> name; // Read name 2 from the file
        cout << name << endl; // Display name 2

        inFile >> name; // Read name 3 from the file
        cout << name << endl; // Display name 3
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
    inFile.close(); // Close the file
    cout << "\nDone.\n";
    return 0;
}
```

Program Screen Output

```
Reading information from the file.

Bach
Beethoven
Mozart

Done.
```

When the >> operator extracts data from a file, it expects to read pieces of data that are separated by whitespace characters (spaces, tabs, or newlines).

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
// This program uses the >> operator to read rectangle dimensions
// from a file. It demonstrates that, as with cin, more than one
// value can be read in from a file with a single statement.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    ifstream inFile;
    int length, width;

    inFile.open("dimensions.txt");
    cout << "Reading dimensions of 4 rectangles from the file.\n\n";

    // Process rectangle 1
    inFile >> length >> width;
    cout << "Area of rectangle 1: " << (length * width) << endl;

    // Process rectangle 2
    inFile >> length >> width;
    cout << "Area of rectangle 2: " << (length * width) << endl;
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

```
    // Process rectangle 3
    inFile >> length >> width;
    cout << "Area of rectangle 3: " << (length * width) << endl;

    // Process rectangle 4
    inFile >> length >> width;
    cout << "Area of rectangle 4: " << (length * width) << endl;

    // Close the file
    inFile.close();
    cout << "Done.\n";

    return 0;
}
```

```
Reading dimensions of 4 rectangles from the file.

Area of rectangle 1: 20
Area of rectangle 2: 35
Area of rectangle 3: 120
Area of rectangle 4: 24

Done
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

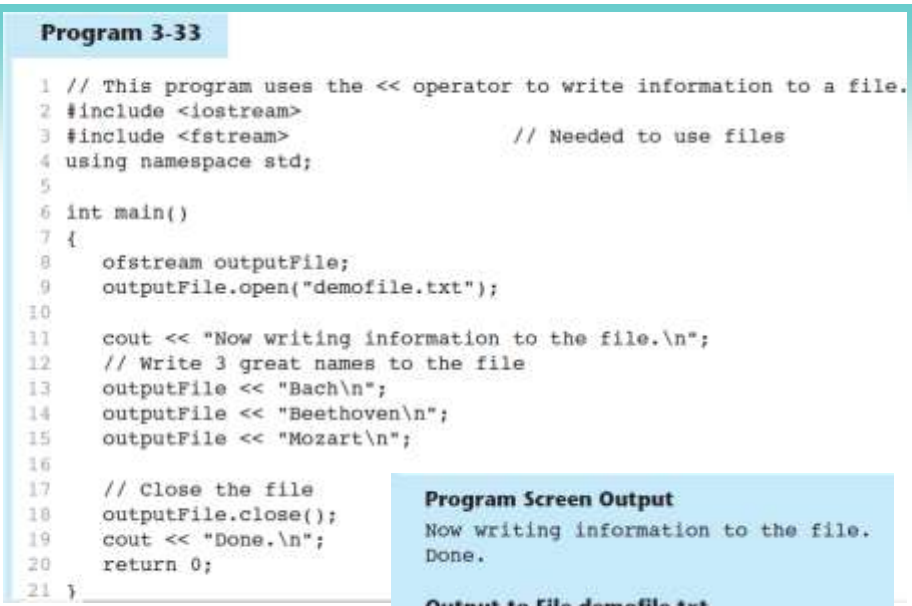
Step 5: Close the file.

Most operating systems temporarily store information in a *file buffer* before it is written to a file. A file buffer is a small holding section of memory that file-bound information is first written to. When the buffer is filled, all the information stored there is written to the file. This technique improves the system’s performance.

Closing a file causes any unsaved information that may still be held in a buffer to be saved to its file. This means the information will be in the file if you need to read it later in the same program.

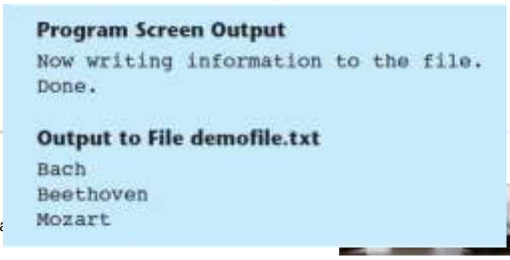
```
outputFile.close();
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Program 3-33

```
// This program uses the << operator to write information to a file.
#include <iostream>
#include <fstream>                    // Needed to use files
using namespace std;

int main()
{
   ofstream outputFile;
   outputFile.open("demofile.txt");

   cout << "Now writing information to the file.\n";
   // Write 3 great names to the file
   outputFile << "Bach\n";
   outputFile << "Beethoven\n";
   outputFile << "Mozart\n";

   // Close the file
   outputFile.close();
   cout << "Done.\n";
   return 0;
}
```



Program Screen Output

```
Now writing information to the file.
Done.
```

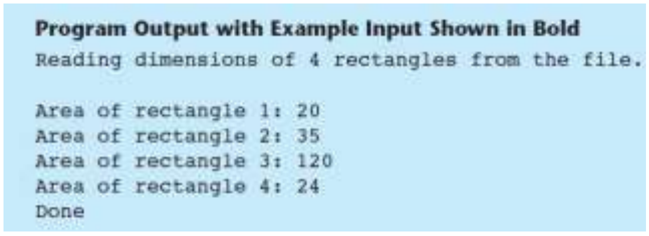
Output to File demofile.txt

```
Bach
Beethoven
Mozart
```

Copyright © 2011 Pearson Education, Inc. Publishing a

Program 3-35, finds the area of four rectangles, illustrates reading data from a text file named dimensions.txt, which was previously created with a text editor. Here is a sample of the file's contents. Each pair of numbers is the length and width of a different rectangle.

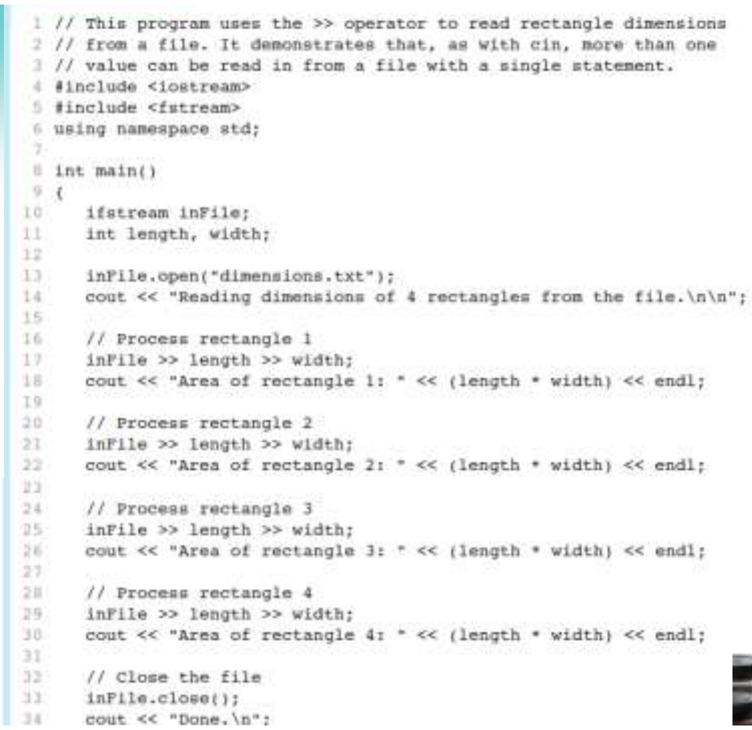
```
10    2
5     7
6     20
8     3
```



Program Output with Example Input Shown in Bold

```
Reading dimensions of 4 rectangles from the file.

Area of rectangle 1: 20
Area of rectangle 2: 35
Area of rectangle 3: 120
Area of rectangle 4: 24
Done
```

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



```
// This program uses the >> operator to read rectangle dimensions
// from a file. It demonstrates that, as with cin, more than one
// value can be read in from a file with a single statement.
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ifstream inFile;
   int length, width;

   inFile.open("dimensions.txt");
   cout << "Reading dimensions of 4 rectangles from the file.\n\n";

   // Process rectangle 1
   inFile >> length >> width;
   cout << "Area of rectangle 1: " << (length * width) << endl;

   // Process rectangle 2
   inFile >> length >> width;
   cout << "Area of rectangle 2: " << (length * width) << endl;

   // Process rectangle 3
   inFile >> length >> width;
   cout << "Area of rectangle 3: " << (length * width) << endl;

   // Process rectangle 4
   inFile >> length >> width;
   cout << "Area of rectangle 4: " << (length * width) << endl;

   // Close the file
   inFile.close();
   cout << "Done.\n";
```

23. Using Files—Storing and Retrieving Numbers

Part 1

Write a program that asks the user to enter five floating-point numbers. The program should create a file and save all five numbers to the file.

Part 2

Write a program that opens the file created by Part 1, reads the five numbers, and displays them. The program should also calculate and display the sum of the five numbers.

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

25. Using Files—Average Rainfall Modification
Write a program that calculates the average monthly rainfall for three months. The program reads its input from a file.
The program should display a message similar to the following:

The average monthly rainfall for June, July, and August was 6.72 inches.

Enter Sample data to test your program in file rainfall.dat.
June 6.0
July 7.0
August 7.16

Note: You may need to use *cin.ignore()* here.

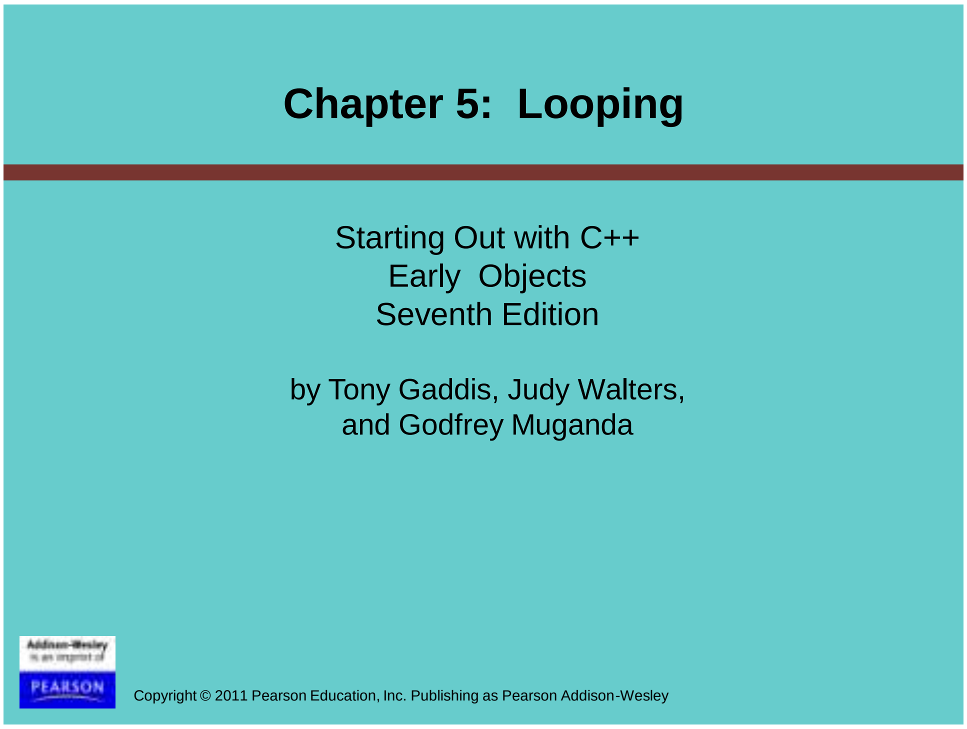
Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

HW

Lab 5

1. Students should read the Pre-lab Reading Assignment before coming to lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab. (photocopy or copy/paste, answer then print and bring to class.)

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley



Chapter 5: Looping

Starting Out with C++
Early Objects
Seventh Edition

by Tony Gaddis, Judy Walters,
and Godfrey Muganda

Addison-Wesley is an imprint of PEARSON

Copyright © 2011 Pearson Education, Inc. Publishing as Pearson Addison-Wesley