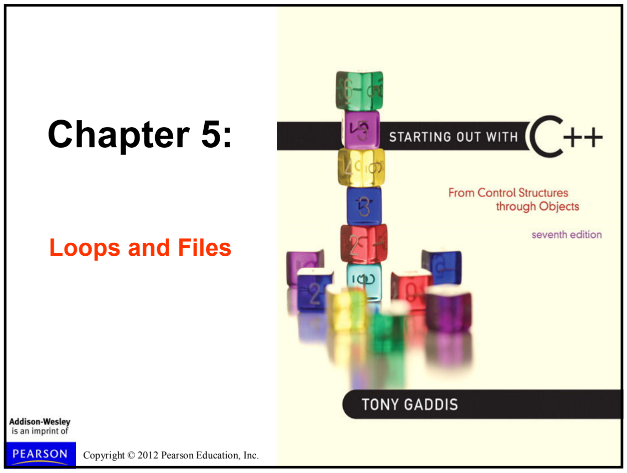
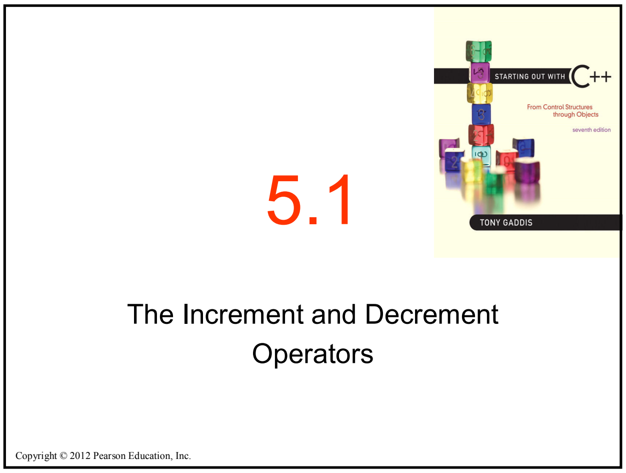


Chapter 5:

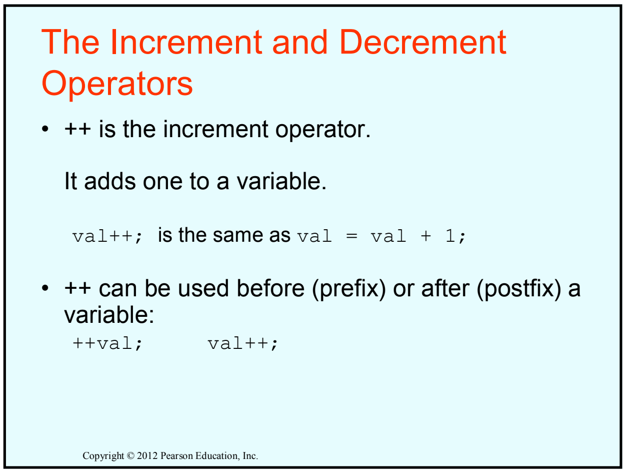
Loops and Files

Addison-Wesley
is an imprint of
PEARSON



5.1

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.

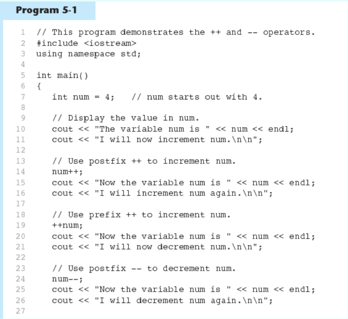
  It subtracts one from a variable.

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

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

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

Increment and Decrement Operators in Program 5-1

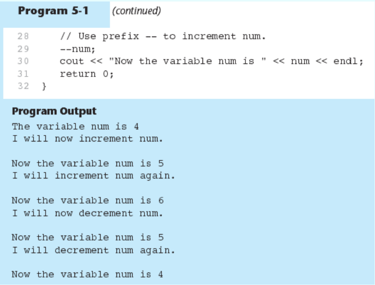


Program 5-1

```
// This program demonstrates the ++ and -- operators.
#include <iostream>
using namespace std;

int main()
{
   int num = 4;   // num starts out with 4.

   // Display the value in num.
   cout << "The variable num is " << num << endl;
   cout << "I will now increment num.\n\n";

   // Use postfix ++ to increment num.
   num++;
   cout << "Now the variable num is " << num << endl;
   cout << "I will increment num again.\n\n";

   // Use prefix ++ to increment num.
   ++num;
   cout << "Now the variable num is " << num << endl;
   cout << "I will now decrement num.\n\n";

   // Use postfix -- to decrement num.
   num--;
   cout << "Now the variable num is " << num << endl;
   cout << "I will decrement num again.\n\n";

```

Continued…

Increment and Decrement Operators in Program 5-1

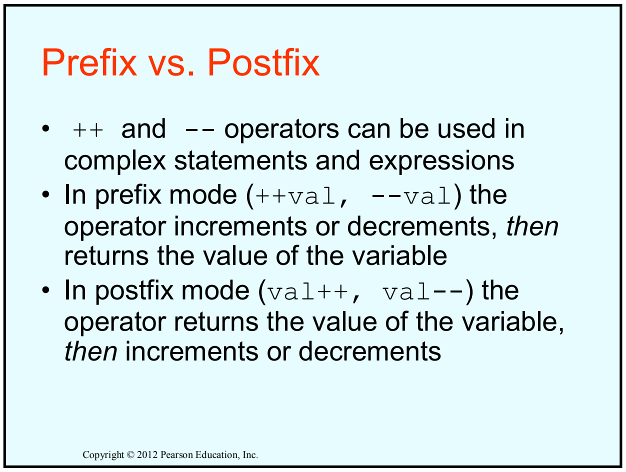


Program 5-1 *(continued)*

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

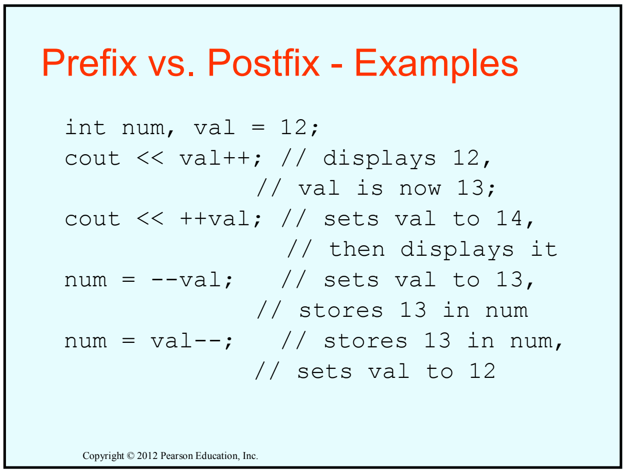
Program Output

```
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
```



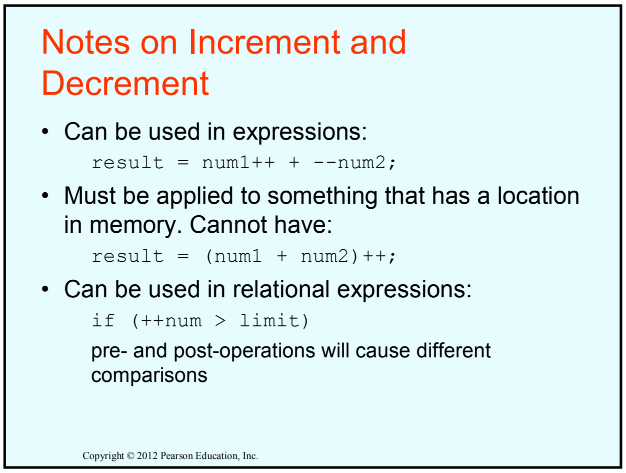
Prefix vs. Postfix

- `++` 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 and Decrement

- Can be used in expressions:

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

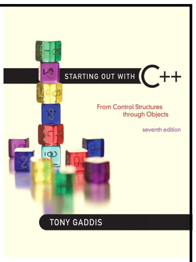
- Must be applied to something that has a location in memory. Cannot have:

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

- Can be used in relational expressions:

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

  pre- and post-operations will cause different comparisons



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;
```

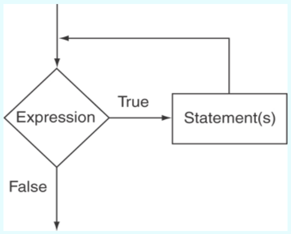
- *`statement;`* can also be a block of statements enclosed in `{ }`

The `while` Loop – How It Works

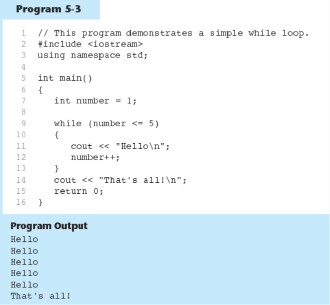
```
while (expression)
     statement;
```

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

The Logic of a `while` Loop



The `while` loop in Program 5-3



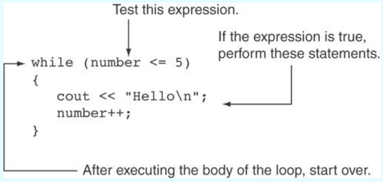
Program 5-3

```
 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 `while` Loop in Program 5-3 Lines 9 through 13 Works



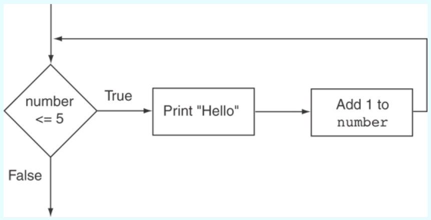
Test this expression.

If the expression is true, perform these statements.

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

After executing the body of the loop, start over.

Flowchart of the `while` Loop in Program 5-3



The `while` Loop is a Pretest Loop

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

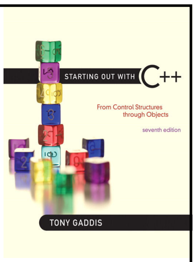
```
int number = 6;
while (number <= 5)
{
   cout << "Hello\n";
   number++;
}
```

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

Example of an Infinite Loop

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



5.3

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

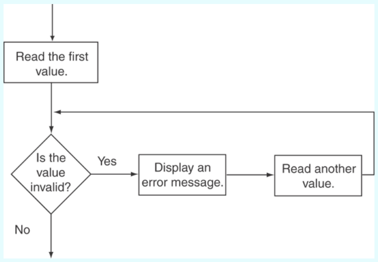
- Here's the general approach, in pseudocode:

```
Read an item of input.
While the input is invalid
    Display an error message.
    Read the input again.
End While
```

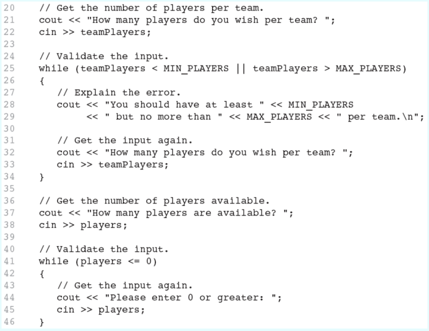
Input Validation Example

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

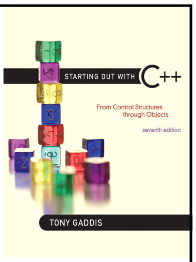
Flowchart for Input Validation



Input Validation in Program 5-5



```
// Get the number of players per team.
cout << "How many players do you wish per team? ";
cin >> teamPlayers;

// Validate the input.
while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)
{
   // Explain the error.
   cout << "You should have at least " << MIN_PLAYERS
        << " but no more than " << MAX_PLAYERS << " per team.\n";

   // Get the input again.
   cout << "How many players do you wish per team? ";
   cin >> teamPlayers;
}

// Get the number of players available.
cout << "How many players are available? ";
cin >> players;

// Validate the input.
while (players <= 0)
{
   // Get the input again.
   cout << "Please enter 0 or greater: ";
   cin >> players;
}
```



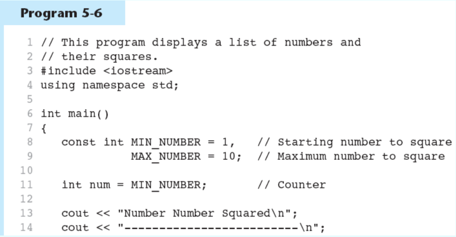
5.4

Counters

Counters

- <u>Counter</u>: 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 <u>*loop control variable*</u>)
- Must be initialized before entering loop

A Counter Variable Controls the Loop in Program 5-6

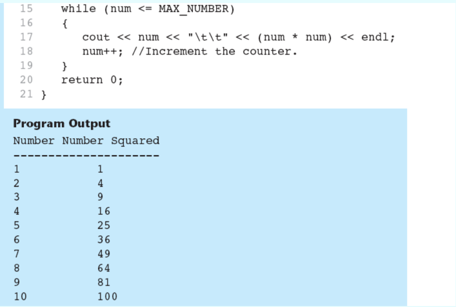


Program 5-6

```
// This program displays a list of numbers and
// their squares.
#include <iostream>
using namespace std;

int main()
{
   const int MIN_NUMBER = 1,   // Starting number to square
             MAX_NUMBER = 10;  // Maximum number to square

   int num = MIN_NUMBER;       // Counter

   cout << "Number Number Squared\n";
   cout << "-------------------------\n";
```

Continued…

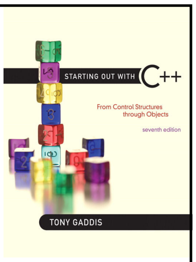
A Counter Variable Controls the Loop in Program 5-6



```
   while (num <= MAX_NUMBER)
   {
      cout << num << "\t\t" << (num * num) << endl;
      num++; //Increment the counter.
   }
   return 0;
}
```

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
```



5.5

The `do-while` Loop

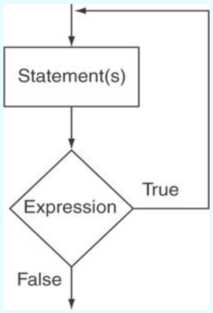
The `do-while` Loop

- `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

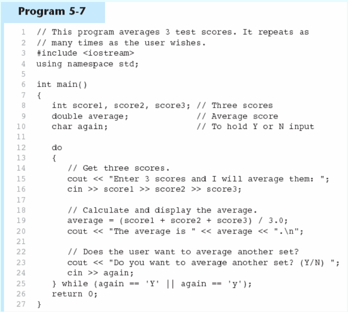


An Example `do-while` Loop

```
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.

A `do-while` Loop in Program 5-7

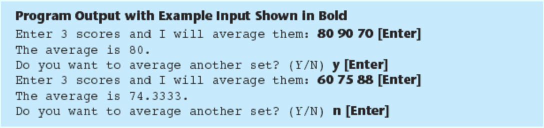


Program 5-7

```
// This program averages 3 test scores. It repeats as
// many times as the user wishes.
#include <iostream>
using namespace std;

int main()
{
   int score1, score2, score3; // Three scores
   double average;             // Average score
   char again;                 // To hold Y or N input

   do
   {
      // Get three scores.
      cout << "Enter 3 scores and I will average them: ";
      cin >> score1 >> score2 >> score3;

      // Calculate and display the average.
      average = (score1 + score2 + score3) / 3.0;
      cout << "The average is " << average << ".\n";

      // Does the user want to average another set?
      cout << "Do you want to average another set? (Y/N) ";
      cin >> again;
   } while (again == 'Y' || again == 'y');
   return 0;
}
```

Continued…

A `do-while` Loop in Program 5-7

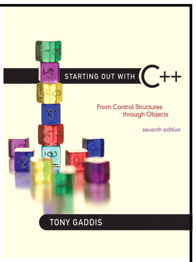


Program Output with Example Input Shown in Bold

```
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 *on pages* 245-246)



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 the `update` expression or after the `)`

`for` Loop - Mechanics

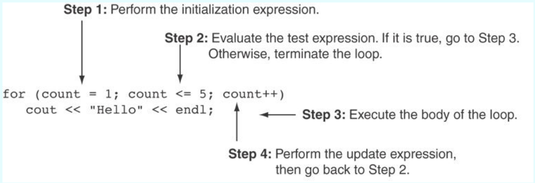
```
for(initialization; test; update)
          statement; // or block in { }
```

1) Perform *`initialization`*
2) Evaluate *`test`* expression
   - If `true`, execute *`statement`*
   - If `false`, terminate loop execution
3) 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



Step 1: Perform the initialization expression.

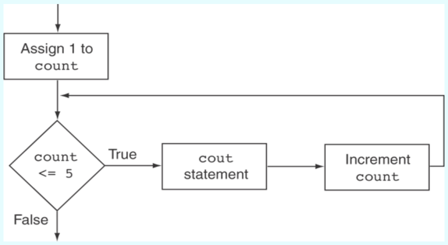
Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

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

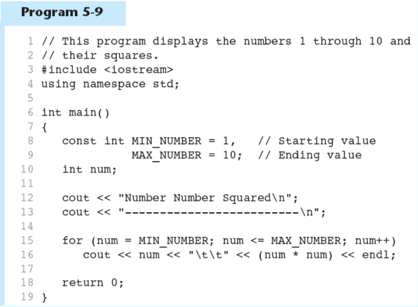
Step 3: Execute the body of the loop.

Step 4: Perform the update expression, then go back to Step 2.

Flowchart for the Previous Example



A `for` Loop in Program 5-9

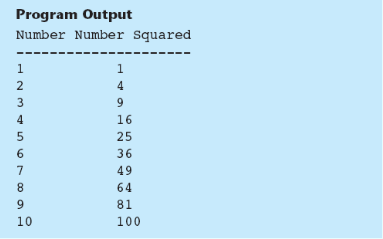


Program 5-9

```
 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    const int MIN_NUMBER = 1,   // Starting value
 9              MAX_NUMBER = 10;  // Ending value
10    int num;
11
12    cout << "Number Number Squared\n";
13    cout << "-------------------------\n";
14
15    for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
16       cout << num << "\t\t" << (num * num) << endl;
17
18    return 0;
19 }
```

Continued…

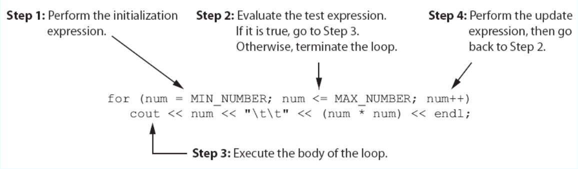
A `for` Loop in Program 5-9



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 15 through 16 in Program 5-9



Step 1: Perform the initialization expression.

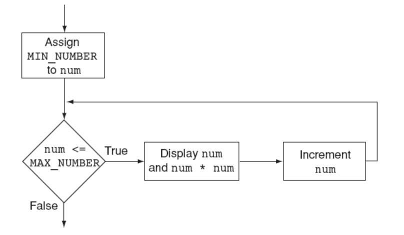
Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

Step 4: Perform the update expression, then go back to Step 2.

```
for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
   cout << num << "\t\t" << (num * num) << endl;
```

Step 3: Execute the body of the loop.

Flowchart for Lines 15 through 16 in Program 5-9



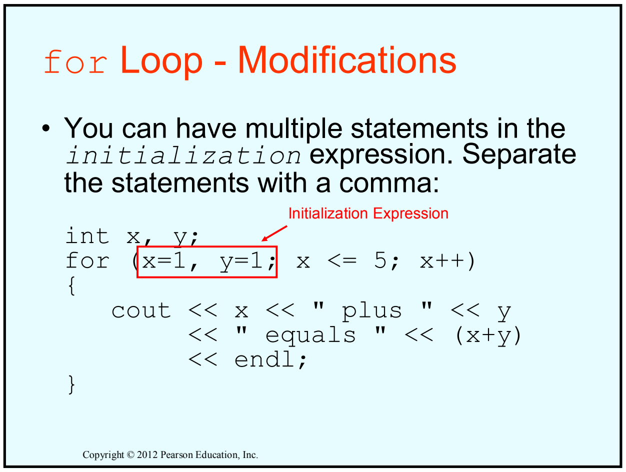
When to Use the `for` Loop

- 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;
```

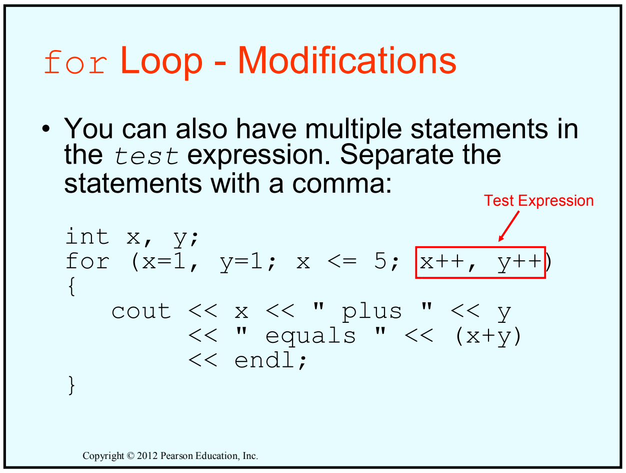


for Loop - Modifications

- 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;
}
```

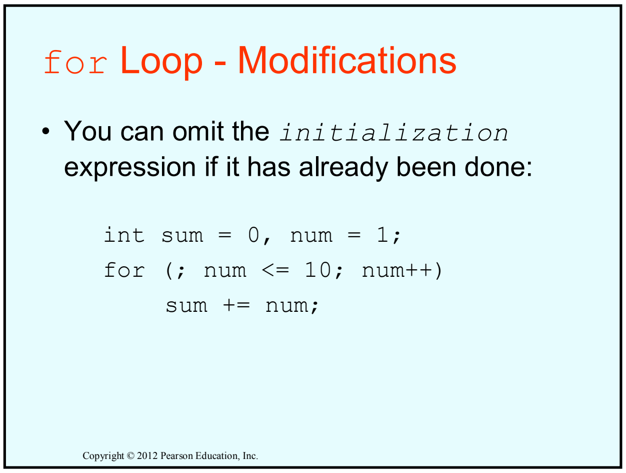


for Loop - Modifications

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

Test Expression

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



for Loop - Modifications

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

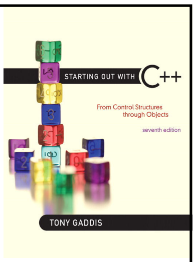
```
int sum = 0, num = 1;
for (; num <= 10; num++)
    sum += num;
```

`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

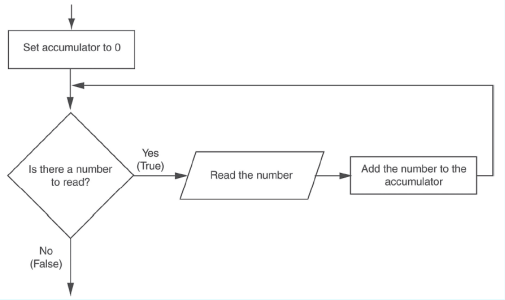
Keeping a Running Total

Keeping a Running Total

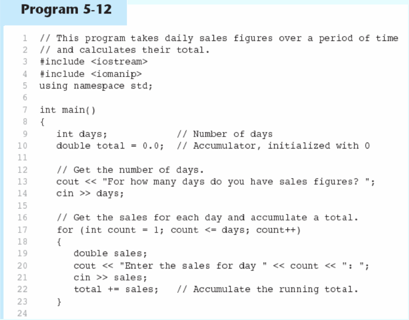
- <u>running total</u>: accumulated sum of numbers from each repetition of loop
- <u>accumulator</u>: variable that holds running total

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

Logic for Keeping a Running Total



A Running Total in Program 5-12

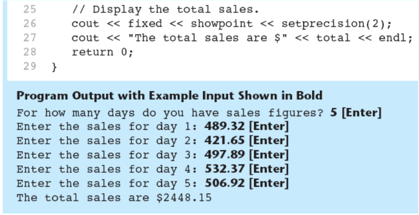


Program 5-12

```
// This program takes daily sales figures over a period of time
// and calculates their total.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int days;            // Number of days
   double total = 0.0;  // Accumulator, initialized with 0

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

   // Get the sales for each day and accumulate a total.
   for (int count = 1; count <= days; count++)
   {
      double sales;
      cout << "Enter the sales for day " << count << ": ";
      cin >> sales;
      total += sales;   // Accumulate the running total.
   }

```

Continued…

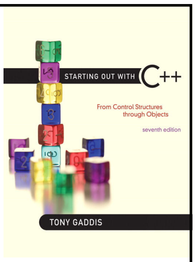
A Running Total in Program 5-12



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

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
```



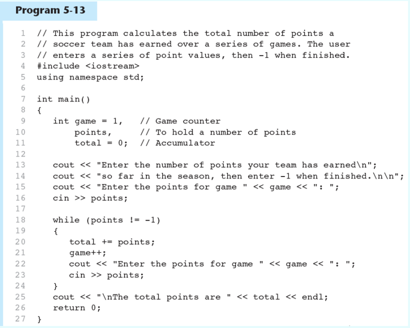
5.8

Sentinels

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

A Sentinel in Program 5-13

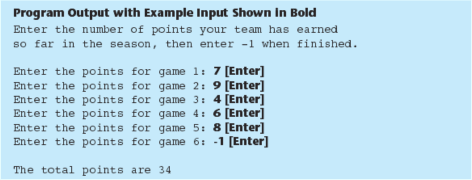


Program 5-13

```
// This program calculates the total number of points a
// soccer team has earned over a series of games. The user
// enters a series of point values, then -1 when finished.
#include <iostream>
using namespace std;

int main()
{
   int game = 1,    // Game counter
       points,      // To hold a number of points
       total = 0;   // Accumulator

   cout << "Enter the number of points your team has earned\n";
   cout << "so far in the season, then enter -1 when finished.\n\n";
   cout << "Enter the points for game " << game << ": ";
   cin >> points;

   while (points != -1)
   {
      total += points;
      game++;
      cout << "Enter the points for game " << game << ": ";
      cin >> points;
   }
   cout << "\nThe total points are " << total << endl;
   return 0;
}
```

Continued…

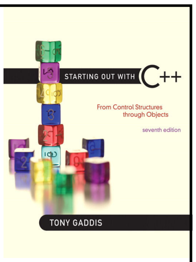
A Sentinel in Program 5-13



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

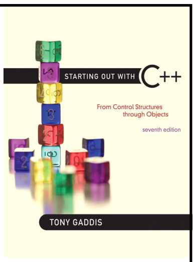


Deciding Which Loop to Use

Deciding Which Loop to Use

- The `while` loop is a conditional pretest loop
  - Iterates as long as a certain condition exits
  - Validating input
  - Reading lists of data terminated by a sentinel
- The `do-while` loop is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu
- The `for` loop is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

5.10



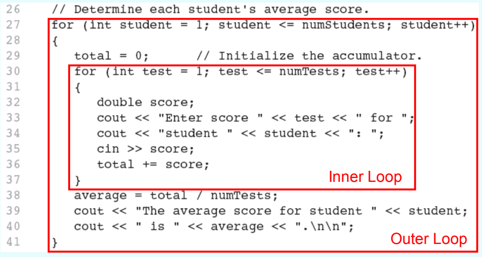
Nested Loops

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;
```

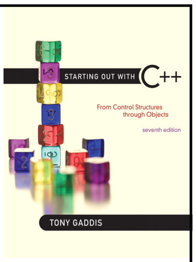
Nested `for` Loop in Program 5-14



```
// Determine each student's average score.
for (int student = 1; student <= numStudents; student++)
{
   total = 0;      // Initialize the accumulator.
   for (int test = 1; test <= numTests; test++)
   {
      double score;
      cout << "Enter score " << test << " for ";
      cout << "student " << student << ": ";
      cin >> score;
      total += score;
   }
   average = total / numTests;
   cout << "The average score for student " << student;
   cout << " is " << average << ".\n\n";
}
```

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.

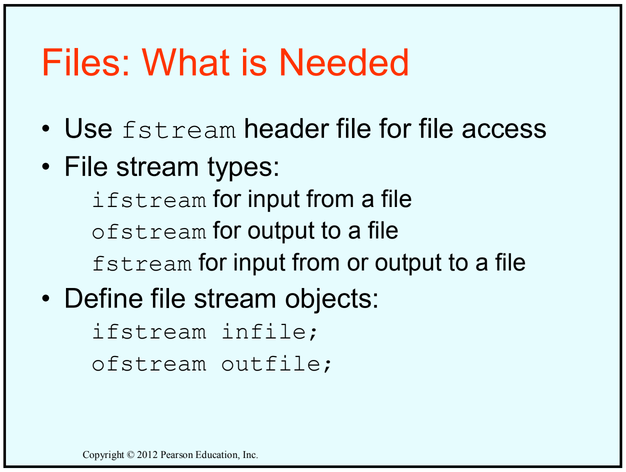


5.11

Using Files for Data Storage

Using Files for Data Storage

- Can use files instead of keyboard, monitor screen for program input, output
- Allows data to be retained between program runs
- Steps:
  - *Open* the file
  - *Use* the file (read from, write to, or both)
  - *Close* the file



Files: What is Needed

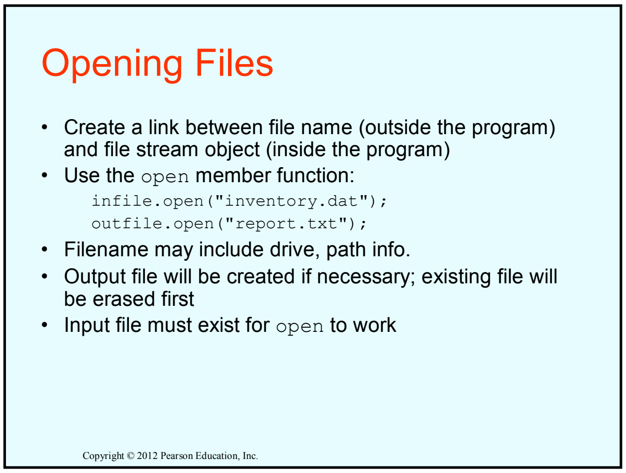
- Use `fstream` header file for file access
- File stream types:

  `ifstream` for input from a file

  `ofstream` for output to a file

  `fstream` for input from or output to a file
- Define file stream objects:

```
ifstream infile;
ofstream outfile;
```

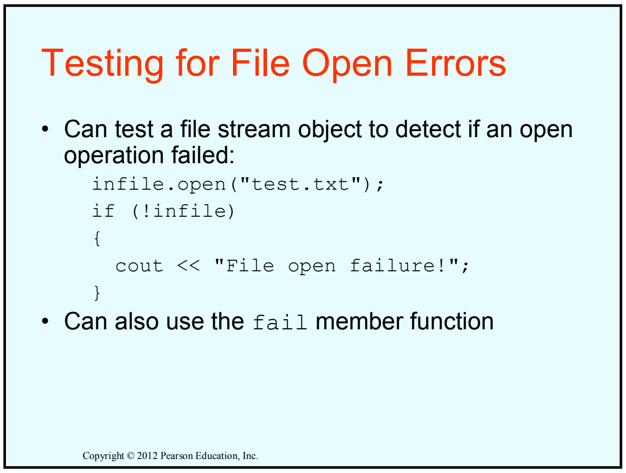


Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the `open` member function:

```
infile.open("inventory.dat");
outfile.open("report.txt");
```

- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for `open` to work

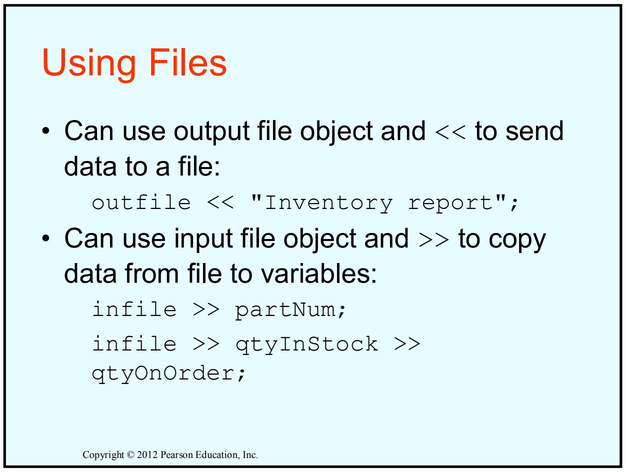


Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
if (!infile)
{
  cout << "File open failure!";
}
```

- Can also use the `fail` member function



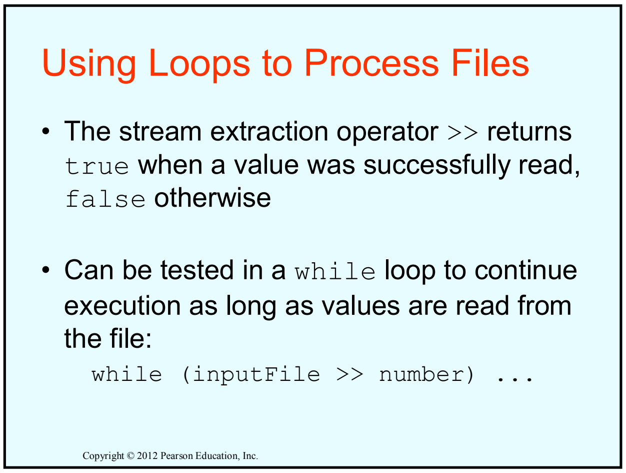
Using Files

- Can use output file object and << to send data to a file:

```
outfile << "Inventory report";
```

- Can use input file object and >> to copy data from file to variables:

```
infile >> partNum;
infile >> qtyInStock >>
qtyOnOrder;
```

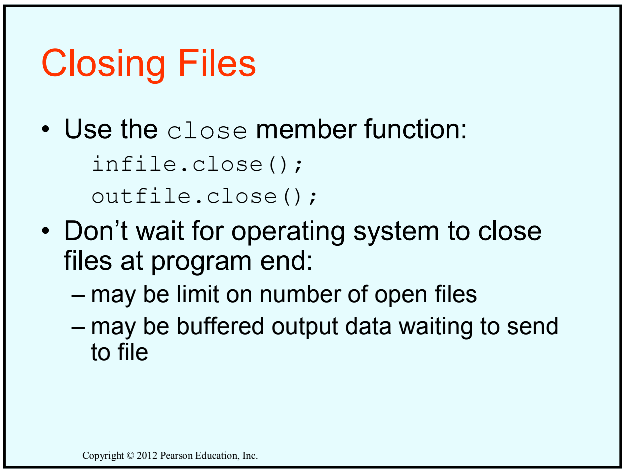


Using Loops to Process Files

- 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) ...
```



Closing Files

- Use the `close` member function:

```
infile.close();
outfile.close();
```

- Don’t wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

Letting the User Specify a Filename

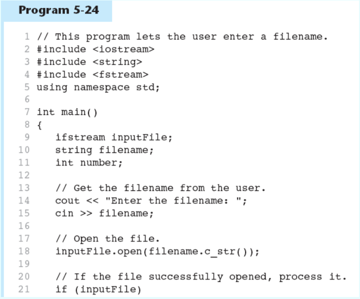
- The `open` member function requires that you pass the name of the file as a null-terminated string, which is also known as a *C-string*.
- *String literals are stored* in memory as null-terminated C-strings, but *string objects* are **not**.

Letting the User Specify a Filename

- `string` objects have a member function named `c_str`
  - It returns the contents of the object formatted as a null-terminated C-string.
  - Here is the general format of how you call the `c_str` function:

```
stringObject.c_str()
```

Letting the User Specify a Filename in Program 5-24

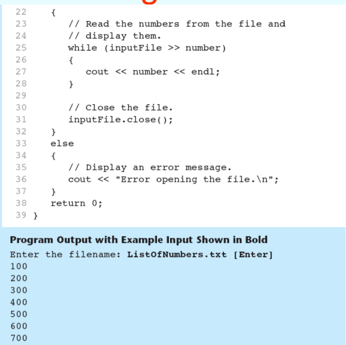


Program 5-24

```
// This program lets the user enter a filename.
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
   ifstream inputFile;
   string filename;
   int number;

   // Get the filename from the user.
   cout << "Enter the filename: ";
   cin >> filename;

   // Open the file.
   inputFile.open(filename.c_str());

   // If the file successfully opened, process it.
   if (inputFile)
```

Continued…

Letting the User Specify a Filename in Program 5-24

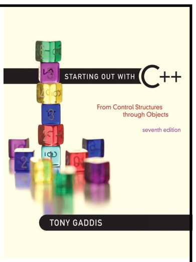


```
   {
      // Read the numbers from the file and
      // display them.
      while (inputFile >> number)
      {
         cout << number << endl;
      }

      // Close the file.
      inputFile.close();
   }
   else
   {
      // Display an error message.
      cout << "Error opening the file.\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter the filename: ListOfNumbers.txt [Enter]
100
200
300
400
500
600
700
```

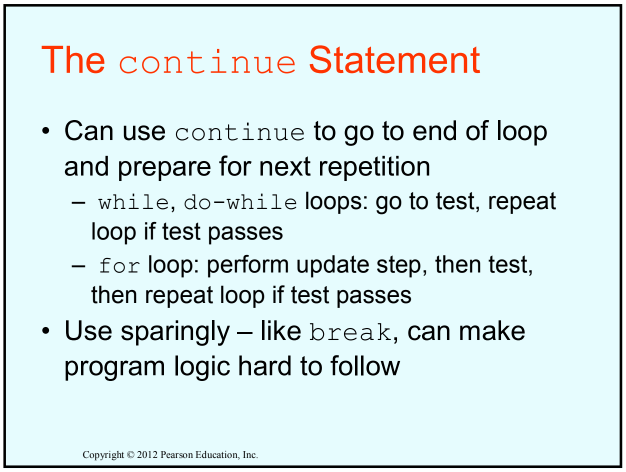
5.12



Breaking and Continuing a Loop

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



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