Introduction to Programming in C++
Seventh Edition

Chapter 8: More on the Repetition Structure

Objectives

- Include a posttest loop in pseudocode
- Include a posttest loop in a flowchart
- Code a posttest loop using the C++ `do while` statement
- Nest repetition structures
- Raise a number to a power using the `pow` function

Posttest Loops

- Repetition structures can be either pretest or posttest loops
- Pretest loop – condition evaluated before instructions are processed
- Posttest loop – condition evaluated after instructions are processed
- Posttest loop's instructions are always processed at least once
- Pretest loop's instructions may never be processed

Posttest Loops (cont'd.)

Problem specification

Sherri is standing an unknown number of steps away from the Burlington Fountain. Write the instructions that direct Sherri to walk from her current location to the fountain.

Illustration A

Illustration B

Solution 1 – pretest loop

```
repeat while (you are not directly in front of the fountain)
      walk forward
end repeat
```

condition

works when Sherri is zero or more steps away from the fountain

Solution 2 – posttest loop

```
repeat
      walk forward
end repeat while (you are not directly in front of the fountain)
```

condition

works only when Sherri is at least one step away from the fountain

Figure 8-1 Problem specification, illustrations, and solutions containing pretest and posttest loops

Posttest Loops (cont'd.)

Modified Solution 2 – posttest loop within a selection structure

```
if (you are not directly in front of the fountain)
    repeat
        walk forward
    end repeat while (you are not directly in front of the fountain)
end if
```

works when Sherri is zero or more steps away from the fountain

Figure 8-2 Selection structure added to Solution 2 from Figure 8-1

Flowcharting a Posttest Loop

- Decision symbol in a flowchart (a diamond) representing a repetition structure contains the loop condition
- Decision symbol appears at the top of a pretest loop, but at the bottom of a posttest loop

Flowcharting a Posttest Loop (cont'd.)

Problem specification

The Wheels & More store has several part-time employees; each earns $10 per hour. The store manager wants a program that calculates and displays the weekly gross pay amount for as many employees as needed without having to run the program more than once. Because the number of hours an employee worked can be a positive number only, the store manager will indicate that he is finished with the program by entering a negative number (in this case, –1) as the number of hours.

Input	Processing	Output
pay rate ($10 per hour) hours worked	Processing items: none	gross pay

```
Algorithm 1 (pretest loop):
1. enter the hours worked
2. repeat while (the hours worked are not equal to -1)
        calculate the gross pay by multiplying
        the hours worked by the pay rate
        display the gross pay
        enter the hours worked
   end repeat
```

Figure 8-3 Wheels & More problem specification & algorithms (continues)

Flowcharting a Posttest Loop (cont'd.)

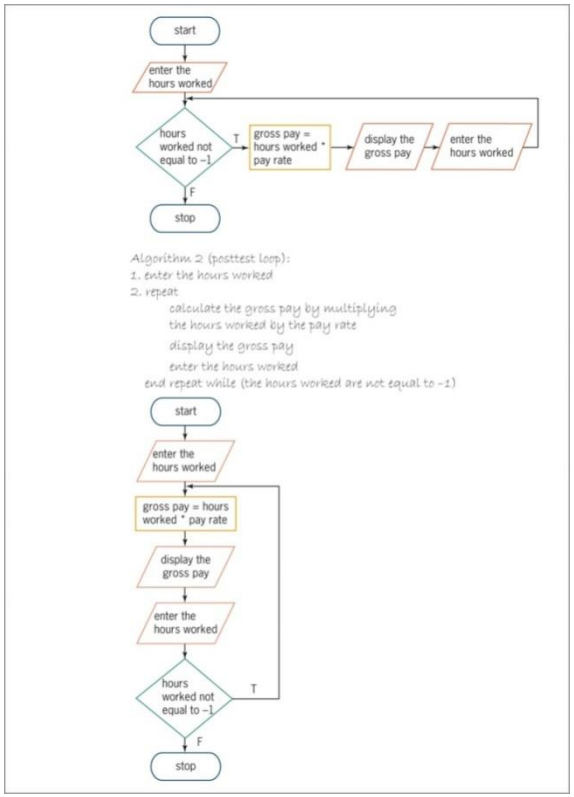

Figure 8-3 Wheels & More problem specification & algorithms (continued)

Flowcharting a Posttest Loop (cont'd.)

pay rate	hours worked	gross pay
10		

Figure 8-4 Input and output items entered in the desk-check table

pay rate	hours worked	gross pay
10	15	150

Figure 8-5 First hours worked and gross pay amounts recorded in the desk-check table

pay rate	hours worked	gross pay	
10	~~15~~	~~150~~	
	~~8~~	80	
	-1		sentinel value

Figure 8-7 Current status of the desk-check table

The `do while` Statement

- `do while` statement is used to code posttest loops in C++
- Syntax:

```
do {
    one or more statements to be processed one time,
    and thereafter as long as the condition is true
} while (condition);
```

- Some programmers use a comment (such as: `//begin loop`) to mark beginning of loop

The `do while` Statement (cont'd.)

- Programmer must provide loop *condition*
  - Must evaluate to a Boolean value
  - May contain variables, constants, functions, arithmetic operators, comparison operators, and logical operators
- Programmer must also provide statements to be executed when *condition* evaluates to true
- Braces are required around statements if there are more than one

The `do while` Statement (cont'd.)

HOW TO Use the `do while` Statement

Syntax

```
do   //begin loop
{
     one or more statements to be processed one time, and thereafter
     as long as the condition is true
}    while (condition);
```

the statement ends with a semicolon

Example 1

```
int age = 0;

cout << "Enter an age greater than 0: ";
cin >> age;
do   //begin loop
{
     cout << "You entered " << age << endl << endl;
     cout << "Enter an age greater than 0: ";
     cin >> age;
}    while (age > 0);
```

priming read

update read

semicolon

Figure 8-9 How to use the `do while` statement

The do while Statement (cont'd.)

Example 2

```
char makeEntry = ' ';
double sales = 0.0;

cout << "Enter a sales amount? (Y/N) ";
cin >> makeEntry;
do   //begin loop
{
     cout << "Enter the sales: ";
     cin >> sales;
     cout << "You entered " << sales << endl << endl;
     cout << "Enter a sales amount? (Y/N) ";
     cin >> makeEntry;
}    while (makeEntry == 'Y' || makeEntry == 'y');
```

priming read

update read

semicolon

Figure 8-9 How to use the do while statement (cont'd.)

The `do while` Statement (cont'd.)

IPO chart information	C++ instructions
Input	
pay rate ($10 per hour)	`const double RATE = 10.0;`
hours worked	`double hours = 0.0;`
Processing	
none	
Output	
gross pay	`double gross = 0.0;`
Algorithm	
1. enter the hours worked	`cout << "First hours worked (-1 to stop): ";` `cin >> hours;`
2. repeat	`do //begin loop` `{`
calculate the gross pay by multiplying the hours worked by the pay rate	`gross = hours * RATE;`
display the gross pay	`cout << "Gross pay: $" << gross;` `cout << endl << endl;`
enter the hours worked	`cout << "Next hours worked (-1 to stop): ";` `cin >> hours;`
end repeat while (the hours worked are not equal to -1)	`} while (hours != -1);`

Figure 8-10 IPO chart information and C++ instructions for the Wheels & More program

The `do while` Statement (cont'd.)

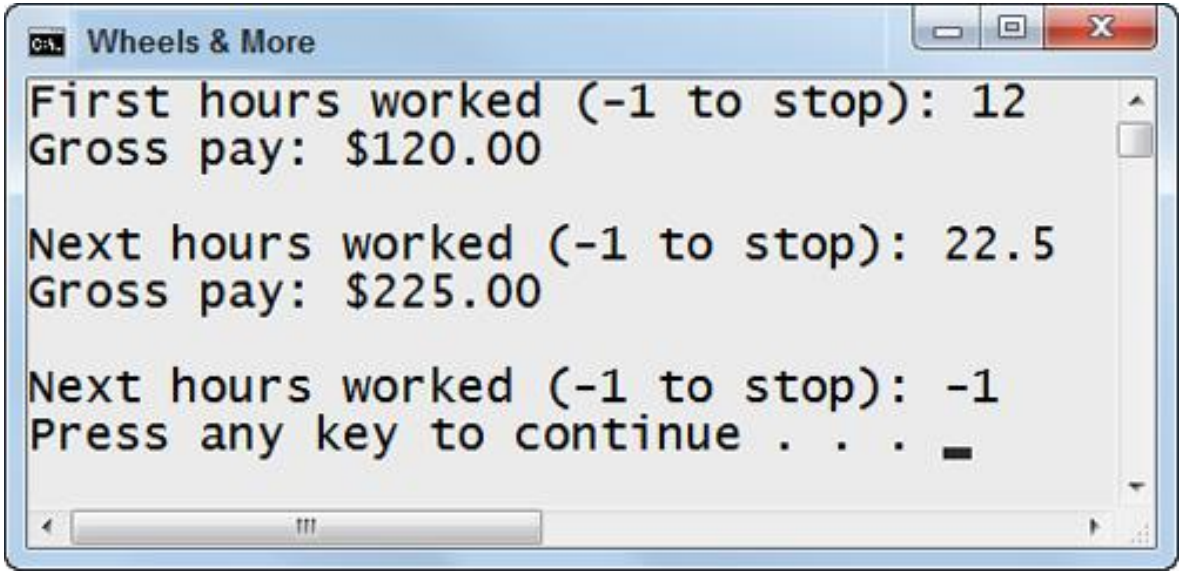

Figure 8-11 A sample run of the Wheels & More program

Nested Repetition Structures

- Like selection structures, repetition structures can be nested
- You can place one loop (the inner, or **nested loop**) inside another loop (the outer loop)
- Both loops can be pretest loops or posttest loops, or the two loops may be different types
- Programmer decides whether a problem requires a nested loop by analyzing the problem specification

Nested Repetition Structures (cont'd.)

```
1. start minutes at 0
2. repeat while (minutes are less than 60)
        start seconds at 0
        repeat while (seconds are less than 60)
            move second hand 1 position, clockwise
            add 1 to seconds
        end repeat
        move minute hand 1 position, clockwise
        add 1 to minutes
    end repeat
```

Figure 8-12 Logic used by a clock's minute and second hands

Nested Repetition Structures (cont'd.)

Problem specification

A waitress named Trixie works at a local diner. The diner just opened for the day and there are customers already sitting at several of the tables. Write the instructions that direct Trixie to go over to each table that needs to be waited on and tell the customers about the daily specials.

```
repeat for (each table that needs to be waited on)
    go to a table that needs to be waited on
    tell the customers at the table about the daily specials
end repeat
```

follow these instructions for each table

Figure 8-13 Problem specification and solution that requires a loop

Nested Repetition Structures (cont'd.)

Problem specification

A waitress named Trixie works at a local diner. The diner just opened for the day and there are customers already sitting at several of the tables. Write the instructions that direct Trixie to go over to each table that needs to be waited on and tell the customers about the daily specials. While at each table, Trixie should take each customer's order. She then should submit the order for the entire table to the cook.

```
repeat for (each table that needs to be waited on)
    go to a table that needs to be waited on
    tell the customers at the table about the daily specials
    repeat for (each customer at the table)
        ask the customer for his or her order
        record the order on the order slip for that table
    end repeat
    go over to the cook at the counter
    tear the appropriate order slip from the order pad
    give the order slip to the cook
end repeat
```

follow these instructions for each table

follow these instructions for each customer at the current table

Figure 8-14 Modified problem specification and solution that requires a nested loop

The Asterisks Program

- Simple program that prints asterisks to the screen in different patterns
- First version contains a single loop
- Second version contains a nested loop
  - Prints out three lines of five asterisks each using a nested for loop to print each line

The Asterisks Program (cont'd.)

Problem specification

Create a program that displays an asterisk on three separate lines on the computer screen, like this:

```
*
*
*
```

IPO chart information	C++ instructions
Input	
none	
Processing	
number of lines (counter: 1 to 3)	this variable is created and initialized in the for clause
Output	
asterisk (on each of 3 lines)	

Figure 8-15 Problem specification IPO chart information and C++ instructions for the asterisks program

The Asterisks Program (cont'd.)

Algorithm

```
repeat while (number of lines is less than 4)

    display an asterisk
    position the cursor on the next line
end repeat
```

```
for (int line = 1;
line < 4; line += 1)
{
    cout << '*';
    cout << endl;
}   //end for
```

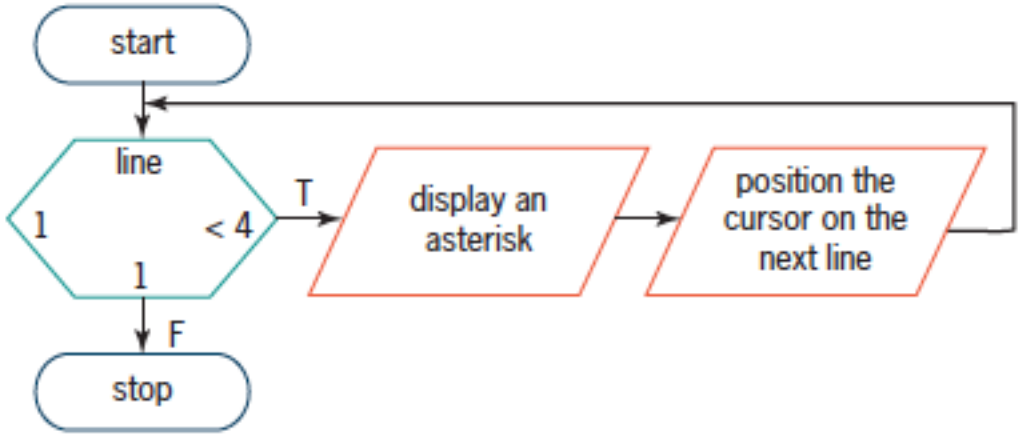

Figure 8-15 IPO chart information and C++ instructions for the asterisks program (cont'd.)

The Asterisks Program (cont'd.)

line
1
2
3
4

Figure 8-16 Completed desk-check table for the asterisks program

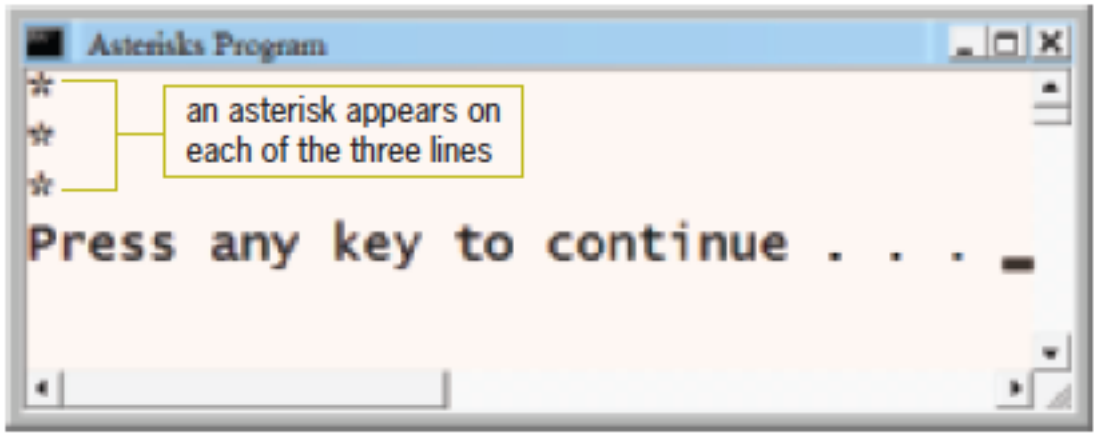

Figure 8-17 Sample run of the asterisks program

The Asterisks Program (cont'd.)

Problem specification

Create a program that displays two asterisks on each of three separate lines on the computer screen, like this:

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

IPO chart information	C++ instructions
Input	
none	
Processing	
number of lines (counter: 1 to 3)	this variable is created and initialized in the for clause
number of asterisks (counter: 1 to 2)	this variable is created and initialized in the for clause

Figure 8-18 Problem specification, IPO chart information, and C++ instructions for the modified asterisks program

The Asterisks Program (cont'd.)

Output

asterisk (2 on each of 3 lines)

Algorithm

```
repeat while (number of
lines is less than 4)

      repeat while (number
      of asterisks is less than 3)

            display an asterisk
      end repeat
      position the cursor on the next line
end repeat
```

```
for (int line = 1;
line < 4; line += 1)
{
    for (int numAsterisks = 1;
    numAsterisks < 3;
    numAsterisks += 1)
        cout << '*';
    //end for
    cout << endl;
}   //end for
```

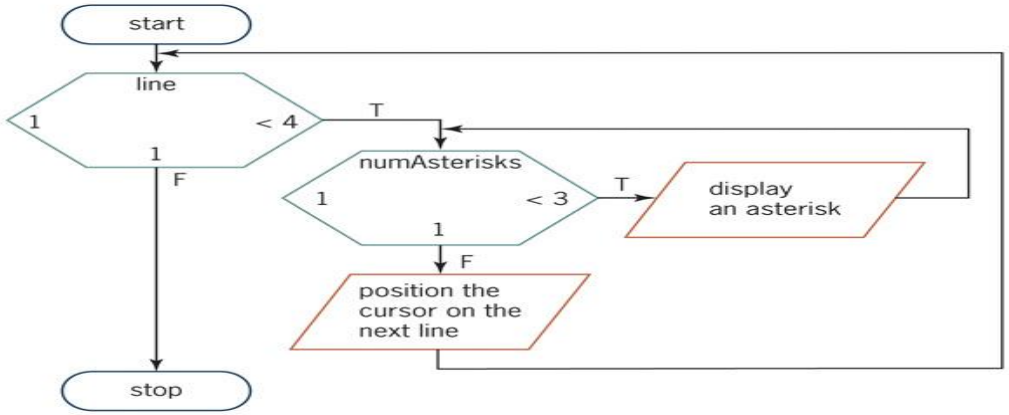

Figure 8-18 Problem specification, IPO chart information, and C++ instructions for the modified asterisks program (cont.)

The Asterisks Program (cont'd.)

line	numAsterisks
1	1

Output

```
*
```

Figure 8-19 Desk-check table and output after the nested loop's `cout` statement is processed the first time

line	numAsterisks
1	~~1~~
	2

Output

```
**
```

Figure 8-20 Desk-check table and output after the nested loop's `cout` statement is processed the second time

The Asterisks Program (cont'd.)

line	numAsterisks
4	4
2	2
	5
	1

Output

```
**
*
```

Figure 8-21 Current status of the desk-check table and output

line	numAsterisks
4	4
2	2
	5
	4
	2
	3

Output

```
**
**
```

Figure 8-22 Desk-check table and output after the nested loop ends the second time

The Asterisks Program (cont'd.)

line	numAsterisks
4	4
2	2
3	3
	4
	2
	3
	4
	2
	3

Output

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

Figure 8-23 Desk-check table and output after the nested loop ends the third time

The Asterisks Program (cont'd.)

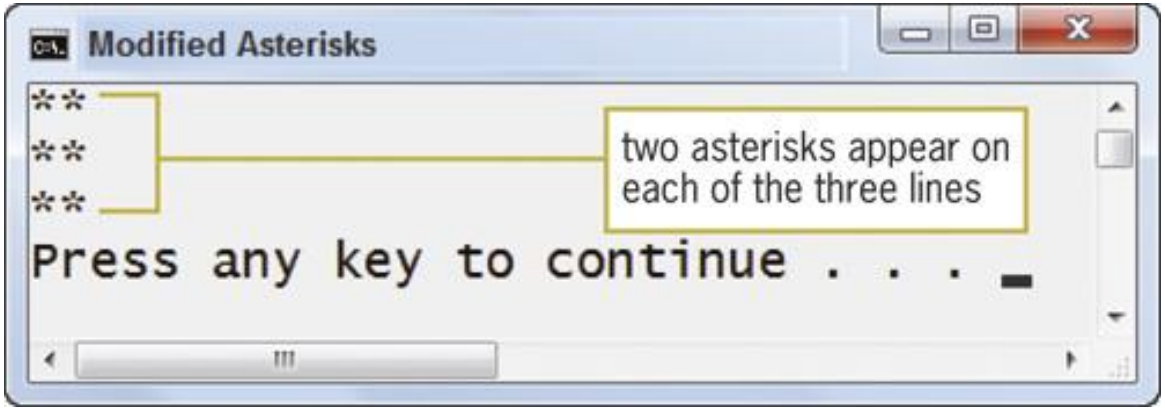

Figure 8-24 Sample run of the modified asterisks program

The Savings Calculator Program

- Calculate the value of a one-time deposit into a savings account after some period of time
- Program uses an exponential formula to calculate the total account value some number of years after the initial investment given the investment amount and the annual interest rate

The Savings Calculator Program (cont'd.)

Problem specification

For your 21st birthday, your grandmother opens a savings account for you and deposits $1,000 into the account. The savings account pays a 2% interest on the account balance. If you don't deposit any more money into the account, and you don't withdraw any money from the account, how much will your savings account be worth at the end of 1 through 5 years? Create a program that gives you the answers. You can calculate the answers using the following formula: $b = p * (1 + r)^n$. In the formula, p is the principal (the amount of the deposit), r is the annual interest rate, n is the number of years, and b is the balance in the savings account at the end of the nth year.

Example 1
$b = 1000 * (1 + .02)^1$
$b = \$1020$

Example 2
$b = 1000 * (1 + .02)^3$
$b = \$1061.21$ (rounded to two decimal places)

Input
principal (1000)
annual interest rate (2%)
number of years (counter: 1 to 5)

Processing
Processing items: none

Algorithm:
repeat
    calculate the account
    balance = principal
    * (1 + annual interest rate)^number of years

    display the current number
    of years and account balance

    add 1 to the number of years
end repeat while (number of years
is less than 6)

Output
account balance
(at end of each of
the 5 years)

principal	annual interest rate	years	account balance
1000	.02	~~1~~	~~1020.00~~
		~~2~~	~~1040.40~~
		~~3~~	~~1061.21~~
		~~4~~	~~1082.43~~
		~~5~~	1104.08
		6	

Figure 8-25 Problem specification, sample calculations, IPO chart, and desk-check table for the savings calculator program

The Savings Calculator Program (cont'd.)

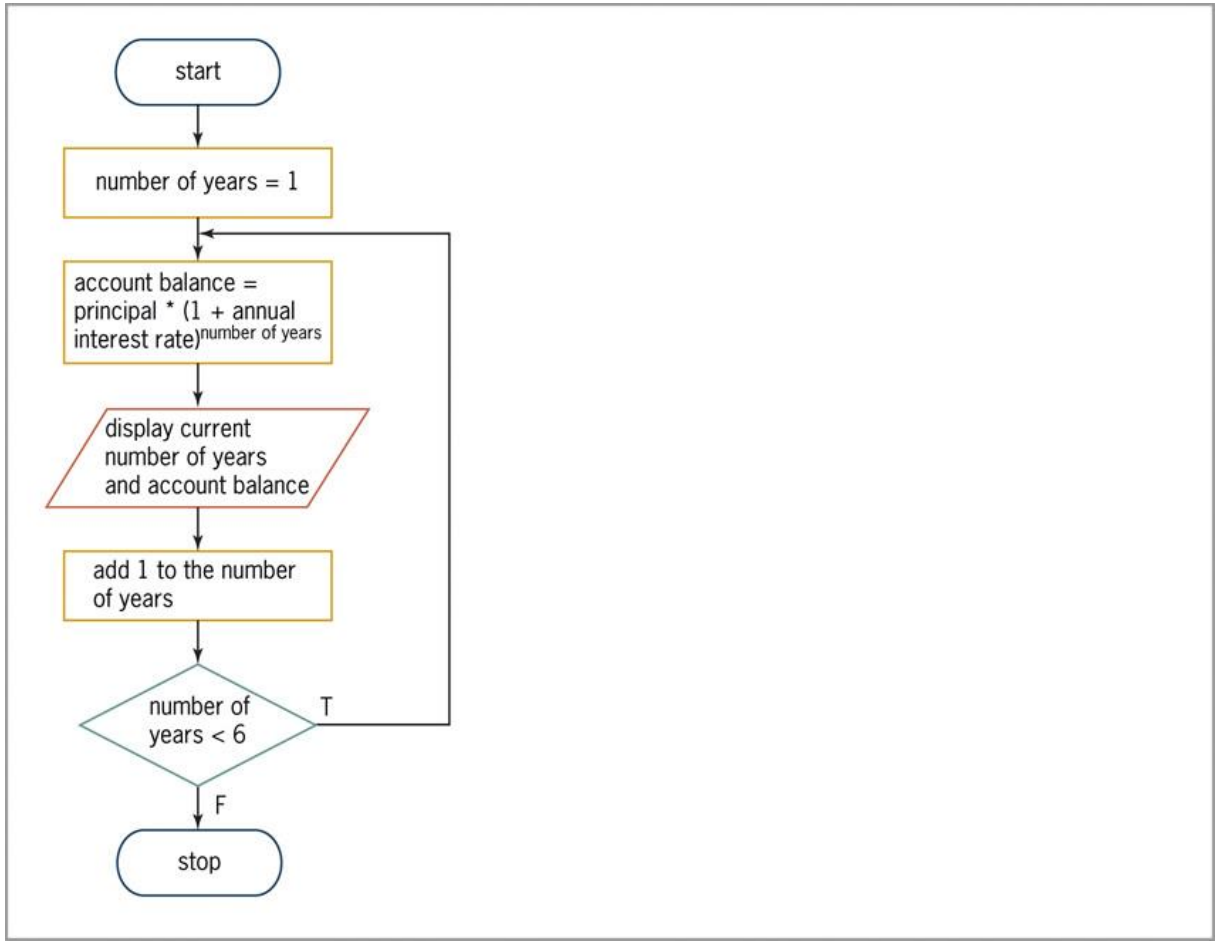

Figure 8-26 Flowchart for the savings calculator program

The `pow` Function

- The **`pow` function** is a convenient tool to raise a number to a power (**exponentiation)**
- The `pow` function raises a number to a power and returns the result as a `double` number
- Syntax is `pow(x, y)`, in which `x` is the base and `y` is the exponent
- At least one of the two arguments must be a `double`
- Program must contain the `#include <cmath>` directive to use the `pow` function

The pow Function (cont'd.)

HOW TO Use the pow Function

Syntax

pow(*x*, *y*) — requires the `#include <cmath>` directive

Example 1

```
double cube = 0.0;
cube = pow(4.0, 3);
```

The assignment statement assigns the number 64.0, which is 4.0 raised to the third power, to the `cube` variable.

Example 2

```
cout << pow(100, .5);
```

The statement displays the number 10, which is 100 raised to the .5 power. The `pow(100, .5)` expression is equivalent to finding the square root of the number 100.

Example 3

```
double area = 0.0;
double radius = 5.0;
area = 3.14 * pow(radius, 2.0);
```

The assignment statement raises the value stored in the `radius` variable to the second power; in other words, it squares the value. The result is 25.0. The assignment statement then multiplies the 25.0 by 3.14 and assigns the product (78.5) to the `area` variable.

Figure 8-27 How to use the `pow` function

Coding the Savings Calculator Program

- Solution to the savings calculator problem and its corresponding C++ code (following slides)
- Code uses the `pow` function to calculate the total account value some number of years after the initial investment given a fixed annual interest rate of 2%

Coding the Savings Calculator Program (cont'd.)

IPO chart information	C++ instructions
Input	
principal (1000)	`int principal = 1000;`
annual interest rate (2%)	`double rate = .02;`
number of years (counter: 1 to 5)	`int years = 1;`
Processing	
none	
Output	
account balance (at end of each of the 5 years)	`double balance = 0.0;`

Figure 8-28 IPO chart information and C++ instructions for the savings calculator program

Coding the Savings Calculator Program (cont'd.)

Algorithm

repeat	do //begin loop {
calculate the account balance = principal * (1 + annual interest rate)number of years	balance = principal * pow(1 + rate, years);
display the current number of years and account balance	cout << "Year " << years << ":" << endl; cout << " $" << balance << endl;
add 1 to the number of years	years += 1;
end repeat while (number of years is less than 6)	} while (years < 6);

Figure 8-28 IPO chart information and C++ instructions for the savings calculator program

Coding the Savings Calculator Program (cont'd.)

```
1  //Savings Calculator.cpp - displays the balance
2  //in a savings account at the end of 1 through 5 years
3  //Created/revised by <your name> on <current date>
4
5  #include <iostream>
6  #include <iomanip>
7  #include <cmath>
8  using namespace std;
9
10 int main()
11 {
12     int principal  = 1000;
13     double rate    = .02;
14     int years      = 1;          //counter
15     double balance = 0.0;
16
17     //display output with two decimal places
18     cout << fixed << setprecision(2);
19
20     do  //begin loop
21     {
22         balance = principal * pow(1 + rate, years);
23         cout << "Year " << years << ":" << endl;
24         cout << "     $" << balance << endl;
25         //update years counter
26         years += 1;
27     }   while (years < 6);
28
29     //system("pause");
30     return 0;
31 }   //end of main function
```

required for the pow function

if your C++ development tool requires this statement, delete the two forward slashes

Figure 8-29 Savings calculator program

Coding the Savings Calculator Program (cont'd.)

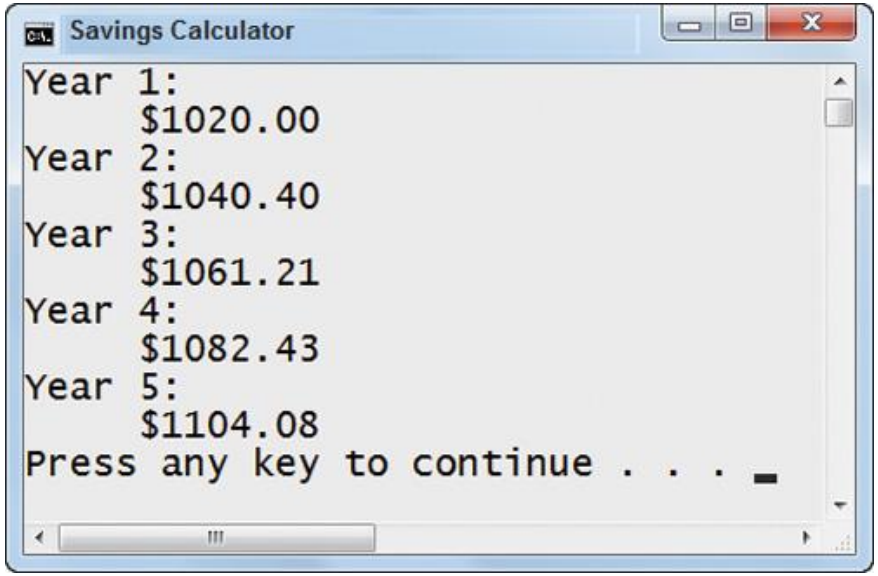

Figure 8-30 Sample run of savings calculator program

Modifying the Savings Calculator Program

- Savings calculator program is modified to calculate the total account value for interest rates of 2%, 3%, and 4%

Modifying the Savings Calculator Program (cont'd.)

IPO chart information	C++ instructions
Input	
principal (1000)	`int principal = 1000;`
annual interest rate (counter: 2% to 4%)	this variable is created and initialized in the nested for clause
number of years (counter: 1 to 5)	`int years = 1;`
Processing	
none	
Output	
account balance (at end of each of the 5 years)	`double balance = 0.0;`

Figure 8-31 IPO chart information and C++ instructions for the modified savings calculator program

Modifying the Savings Calculator Program (cont'd.)

Algorithm

```
repeat
    display the current
    number of years

    repeat for (annual interest
    rate from 2% to 4%)

        calculate the account
        balance = principal
        * (1 + annual interest
        rate)^number of years

        display the current rate
        with no decimal places

        display the current account
        balance with two decimal places
    end repeat
    add 1 to the number of years
end repeat while (number of years is
less than 6)
```

```
do    //begin loop
{
    cout << "Year " << years <<
    ":" << endl;

    for (double rate = .02;
    rate < .05; rate += .01)
    {
        balance = principal *
        pow(1 + rate, years);

        cout << fixed <<
        setprecision(0);
        cout << "   Rate " << rate
        * 100 << "%: $";
        cout << setprecision(2) <<
        balance << endl;
    }  //end for
    years += 1;
}  while (years < 6);
```

Figure 8-31 Modified IPO chart information and C++ instructions (cont'd.)

Modifying the Savings Calculator Program (cont'd.)

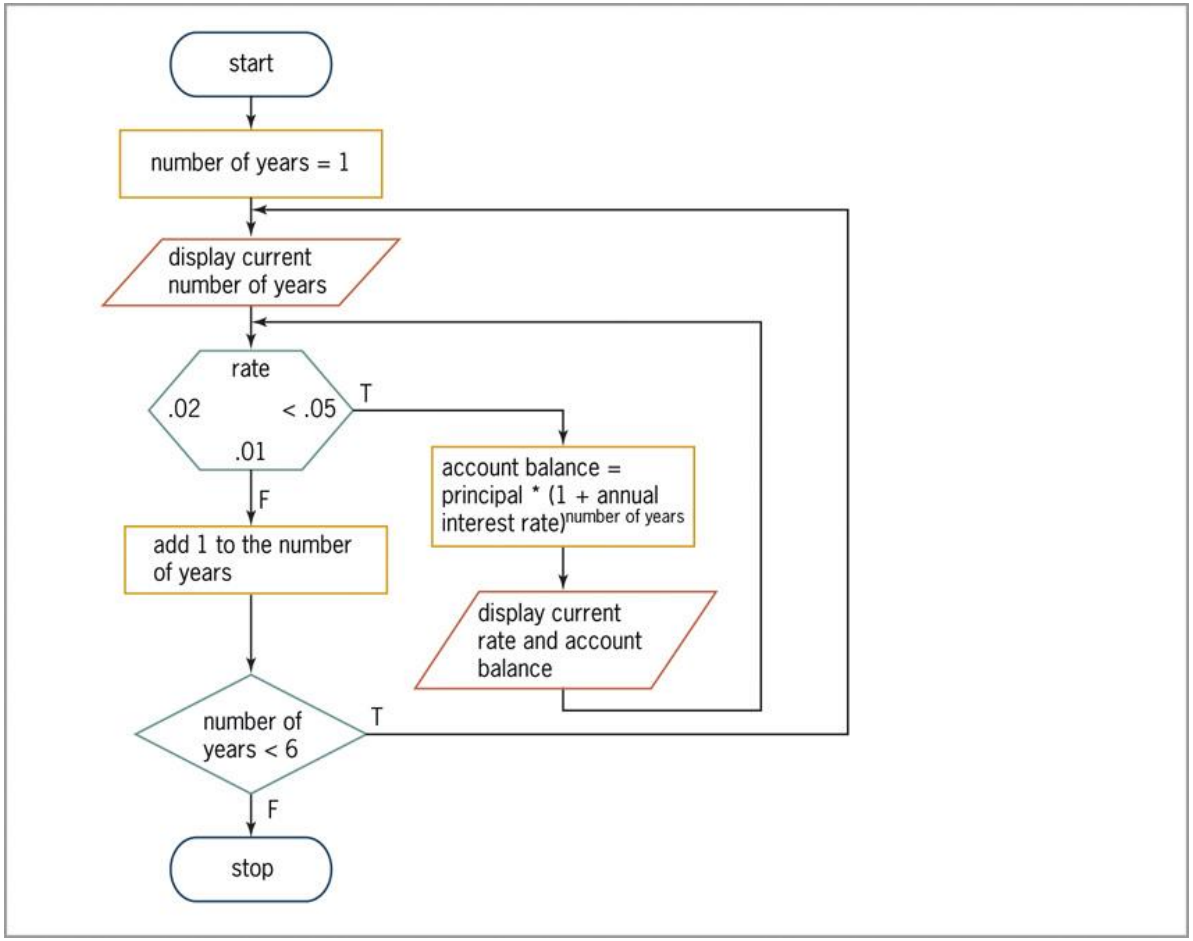

Figure 8-32 Flowchart for the modified savings calculator program

Modifying the Savings Calculator Program (cont'd.)

```
//Modified Savings Calculator.cpp - displays the balance
//in a savings account at the end of 1 through 5 years
//using interest rates of 2%, 3%, and 4%
//Created/revised by <your name> on <current date>

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

int main()
{
    int principal  = 1000;
    int years      = 1;             //counter
    double balance = 0.0;

    do  //begin loop
    {
        cout << "Year " << years << ":" << endl;

        for (double rate = .02; rate < .05; rate += .01)
        {
             balance = principal * pow(1 + rate, years);
             //display rate with zero decimal places
             cout << fixed << setprecision(0);
             cout << "   Rate " << rate * 100 << "%: $";
             //display balance with two decimal places
             cout << setprecision(2) << balance << endl;
        }    //end for

        //update years counter
        years += 1;
    }   while (years < 6);

    //system("pause");
    return 0;
}   //end of main function
```

if your C++ development tool requires this statement, delete the two forward slashes

Figure 8-33 Modified savings calculator program

Modifying the Savings Calculator Program (cont'd.)

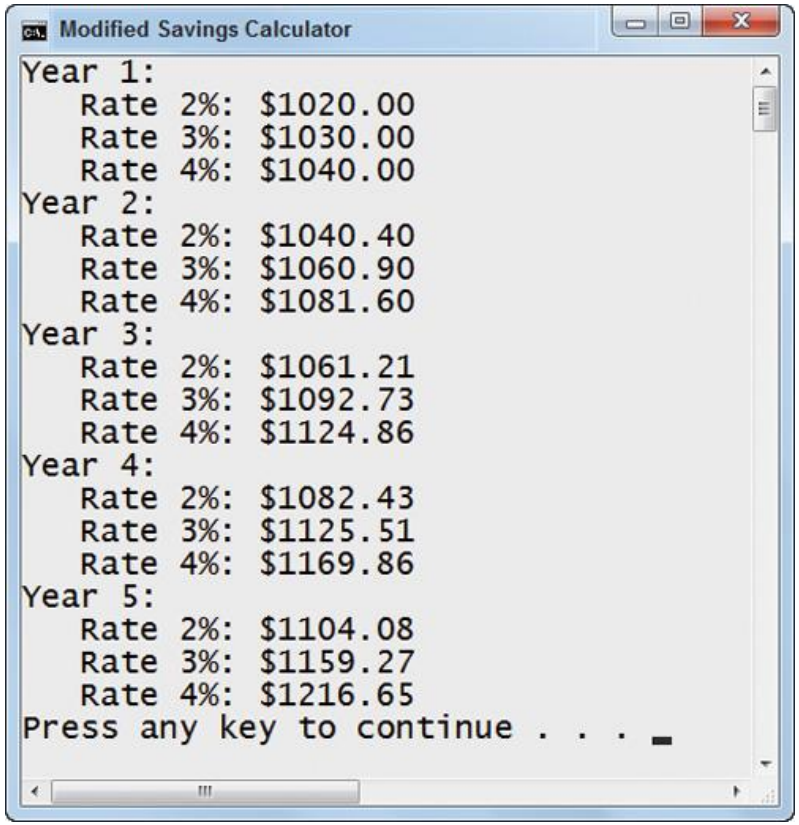

Figure 8-34 Sample run of modified savings calculator program

Summary

- A repetition structure can be either a pretest loop or a posttest loop
- In a pretest loop, the loop condition is evaluated *before* the instructions in the loop are processed
- In a posttest loop, the evaluation occurs *after* the instructions within the loop are processed
- Use the `do while` statement to code a posttest loop in C++
- Use either the `while` statement or the `for` statement to code a pretest loop in C++

Summary (cont'd.)

- Repetition structures can be nested, which means one loop (called the inner or nested loop) can be placed inside another loop (called the outer loop)
- For nested repetition structures to work correctly, the entire inner loop must be contained within the outer loop
- You can use the built-in C++ `pow` function to raise a number to a power
- The `pow` function returns the result as a `double`

Lab 8-1: Stop and Analyze

- Study the program in Figure 8-36 and answer the questions
- The program displays the total sales made in Region 1 and the total sales made in Region 2

Lab 8-2: Plan and Create

Problem specification

Last month, Mrs. Johansen began teaching multiplication to the students in her second grade class. She wants a program that displays one or more multiplication tables. A sample multiplication table is shown below. The x entries represent the number entered by the user and are called the multiplicand. The numbers 1 through 9 are called the multiplier. The y entries represent the product, which is the result of multiplying the multiplicand (x) by the multiplier (the numbers 1 through 9).

multiplicand

multiplier

product

Table format	Sample table using a multiplicand of 2
x * 1 = y	2 * 1 = 2
x * 2 = y	2 * 2 = 4
x * 3 = y	2 * 3 = 6
x * 4 = y	2 * 4 = 8
x * 5 = y	2 * 5 = 10
x * 6 = y	2 * 6 = 12
x * 7 = y	2 * 7 = 14
x * 8 = y	2 * 8 = 16
x * 9 = y	2 * 9 = 18

Figure 8-37 Problem specification for Lab 8-2

Lab 8-3: Modify

- Modify the program in Lab 8-2
- Change both loops to posttest loops
- Use the program to display the multiplication tables for the multiplicands 6, 9, and 2

Lab 8-4: Desk-Check

- Desk-check the code in Figure 8-43
- What will the code display on the computer screen?

```
int number = 1;

while (number < 3)
{
      cout << number << ' ';
      for (int x = 1; x <= 4; x += 1)
            cout << number + x << ' ';
      //end for
      number += 1;
      cout << endl;
}     //end while
```

Figure 8-43 Code for Lab 8-4

Lab 8-5: Debug

- Follow the instructions for starting C++ and opening the Lab8-5.cpp file
- Debug the program