Unit 3 Lesson 9 Repetition Statements (Loops)

Mr. Dave Clausen

La Cañada High School

Introduction to Loops

- We all know that much of the work a computer does is repeated many times.
- When a program repeats a group of statements a given number of items, the repetition is accomplished using a *loop*.
- This will be our third category of structures: *iteration structures*.
- Loops are *iteration structures*.
- Each loop or pass through a group of statements is called an *iteration*.

Repetition Statements

- Our third control structure: iteration or repetition (completes our three control structures: sequence, selection, iteration)
- Two main categories of repetition:
  - definite loop
    - repeats a predetermined number of times
  - indefinite loop
    - repeats a number of times that has not been predetermined.

Repetition Forms

- Three loop types:
  - for<a definite number of times> <do action>
  - while<condition is true> <do action>
  - do<action> while <condition is true>
- Three basic constructs
  - A variable is assigned some value.
  - The value of the variable changes at some point in the loop.
  - The loop repeats until the variable reaches a predetermined value, the program then executes the next statement after the loop.

Pretest Loops

- Pretest Loop (Entrance Controlled Loops)
  - a loop where the control condition (Boolean expression) is tested BEFORE the loop.
  - If the condition is true, the loop is executed.
  - If the condition is false the loop is not executed
  - Therefore, it is possible that these loops may not be executed at all (when the condition is False)
  - There are two pretest loops
    - for loop
    - while loop

Post Test Loops

- Post Test Loops (exit-controlled loop)
  - a loop where the control condition (Boolean expression) is tested AFTER the loop has been executed.
  - If the condition is true, the loop is executed again.
  - If the condition is false the loop is not executed again.
  - Therefore, this type of loop will always be executed at least once.
  - There is one post test loop: do…while

Fixed repetition loops

- Fixed repetition loop
  - a loop used when you know in advance how many repetitions need to be executed, or when you ask the user how many repetitions are needed.
  - also known as a definite loop:
    - The programmer knows, or the user chooses the definite number of repetitions necessary to solve the problem.
  - the “for” loop is:
    - a fixed repetition loop
    - and a pretest loop

Variable Condition Loops

- Variable Condition Loops
  - needed to solve problems where the conditions change within the body of the loop.
  - Also called indefinite loops:
    - the loop repeats an indefinite number of iterations until some condition is met, or while some condition is met.
    - The loop terminates depending upon conditions involving sentinel values, Boolean flags, arithmetic expressions, end of line, or end of file markers.
    - While and do…while loops are variable condition loops.

The for Loop

- The *for* loop repeats one or more statements a specified number of times.
- A *for* loop is difficult to read the first time you see one.
- Like an if statement, the *for* loop uses parentheses.
- In the parentheses are three items called parameters, which are needed to make a *for* loop work.
- Each parameter in a *for* loop is an expression.

Figure 9-1

- Figure 9-1 shows the format of a *for* loop.

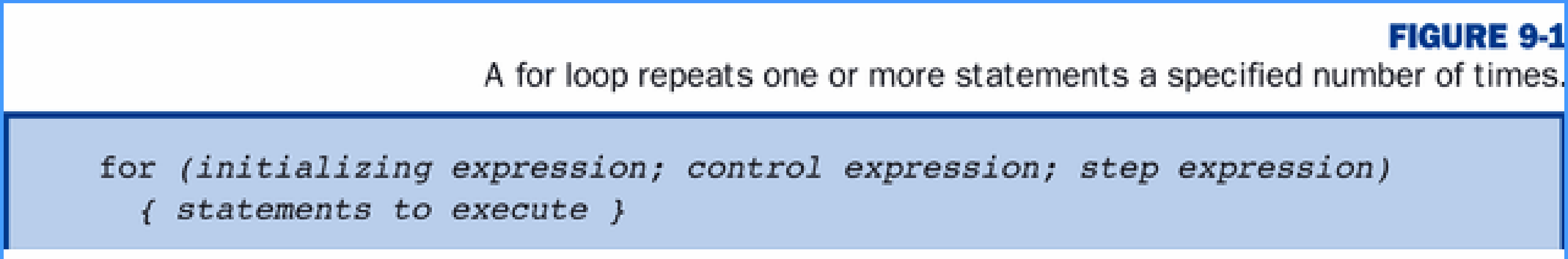


FIGURE 9-1

A for loop repeats one or more statements a specified number of times.

```
for (initializing expression; control expression; step expression)
  { statements to execute }
```

The for Loop

- General form:

  for(<initialization expression>; <termination or control conditon>; <update or step expression>)

  <statement>

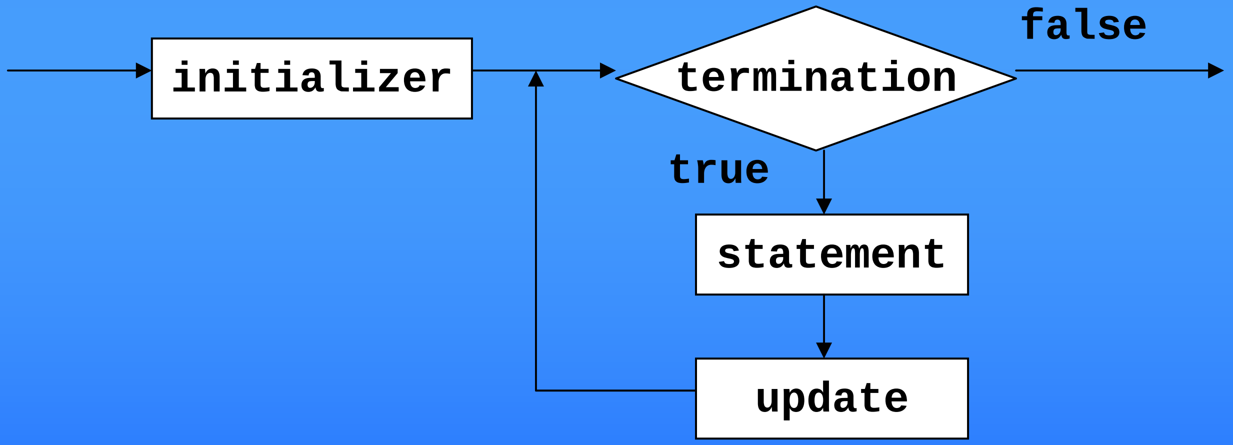
```
for(counter = 1; counter <= 10; counter++)//Loop Heading
   cout<< counter << endl;                 //Loop body
```

Syntax and Semantics of the ***`for`*** *Loop*

```
for (<initializer>; <termination>; <update>)
   <statement>
```

Loop header

Loop body



initializer

termination

false

true

statement

update

The for Loop Internal Logic

- The control variable is assigned an initial value in the initialization expression
- The termination condition is evaluated
- If termination condition is true
  - the body of the loop is executed and the update expression is evaluated
- If the termination condition is false
  - program control is transferred to the first statement following the loop.

Code List 9-1

```
// forloop.cpp            forloop.txt

# include <iostream.h>


int main( )
{
    int counter ; // counter variable
    for (counter = 1; counter <= 3; counter ++)
      cout << counter << endl;
    return 0;
}
```

Increment Operator

- The Increment operator adds 1 to the variable
- Instead of x = x + 1 you can write as + +x
  - if the + + occurs before the x (+ + x) it is called a prefix operator
  - if the + + occurs after the x (x+ +) it is called a postfix operator
- Our text uses the prefix operator
  - the prefix executes faster on most compilers

Decrement Operator

- The Decrement operator subtracts 1 from the variable
- Instead of x = x - 1 you can write as --x
  - if the -- occurs before the x (-- x) it is called a prefix operator
  - if the -- occurs after the x (x--) it is called a postfix operator
- Our text uses the prefix operator
  - the prefix executes faster on most compilers

Counting Backward and Other Tricks

- A counter variable can also count backward by having the step expression decrement the value rather than increment it.
- The program in Code List 9-2 counts backward from 10 to 1.
  - The counter is initialized to 10.
  - With each iteration, the decrement operator subtracts 1 from the counter.

Code List 9-2

```
// backward.cpp                    backward.txt

#include <iostream.h>

int main ( )
{
    int counter ; // counter variable
    for(counter = 10; counter >= 0; counter --)
      cout << counter << end1;
    cout << “”End of loop.\n”;
    return 0;
}
```

Code List 9-3

```
// dblstep.cpp                dblstep.txt
#include <iostream.h>

int main ( )
{
    int counter ; // counter variable
    for (counter = 1; counter <= 100; counter = counter + counter )
       cout << counter << end1;
    return 0;
}
```

Scope of Loop Control Variable

- The loop control variable must be declared before it is used.
  - The rules for the scope of the variable apply here.
- If the variable is only going to be used as a loop counter, and for nothing else…
  - You can limit it’s scope by declaring it when it is initialized in the loop

```
for(int counter = 1; counter <=10; ++ counter )
   cout<< counter <<endl; // counter is only
                          // referenced in the loop
```

For Loops

- For loops can count down (decrement)

```
for(int counter=20; counter>=15; --counter)
    cout<< counter << endl;
```

- For loops can count by factors other than one

```
for(int counter=2; counter<=10; counter=counter+2)
    cout<< counter << endl;
```

  - Style
    - Indent the body of the loop, use blank lines before and after, and use comments.

For Statement Flexibility

- The for statement gives you a lot of flexibility.
- As you have already seen, the step expression can increment, decrement, or count in other ways.

Table 9-1

- Some more examples of for statements are shown in Table 9-1.

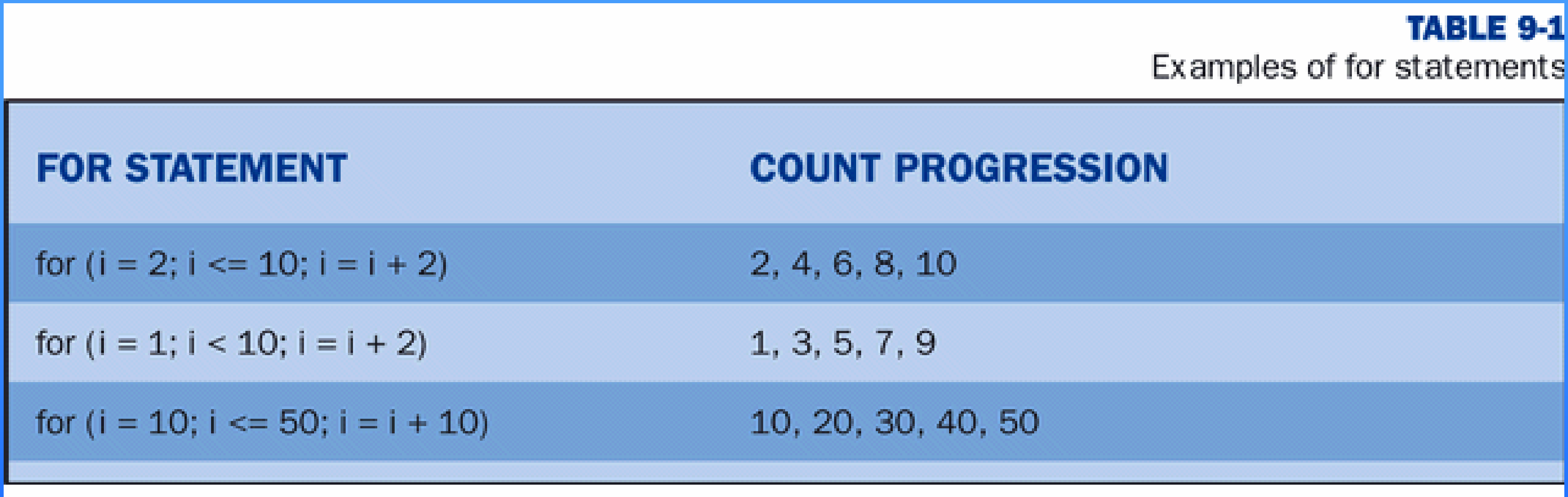


TABLE 9-1
Examples of for statements

FOR STATEMENT	COUNT PROGRESSION
for (i = 2; i <= 10; i = i + 2)	2, 4, 6, 8, 10
for (i = 1; i < 10; i = i + 2)	1, 3, 5, 7, 9
for (i = 10; i <= 50; i = i + 10)	10, 20, 30, 40, 50

Accumulator

- An accumulator is a variable used to keep a running total or sum of successive values of another variable
  - i.e. sum = sum + grade;
  - you should initialize the value of the accumulator before the loop: sum = 0;
  - the accumulator statement occurs in the body of the loop

```
//lcv means loop control variable
sum=0;
for(lcv = 1; lcv <= 100; ++lcv)
   sum = sum + lcv;
```

Using a Statement Block in a for Loop

- If you need to include more than one statement in the loop, place all the statements that are to be part of the loop inside braces {curly brackets}.
- The statements in the braces will be repeated each time the loop iterates.
- The statements that follow the braces are not part of the loop.
- In Code List 9-4, an output statement has been added inside the loop of the backward.cpp program.
- The phrase inside loop will appear with each iteration of the loop.

Code-List 9-4

```
// backward2.cpp              backward2.txt
#include <iostream.h>
int main ( )
{
    int i; // counter variable
    for( i = 10; i >= 0; i--)
    {
        cout << i << endl;
        cout << “Inside Loop\n”;
    }
    cout << “End of loop.\n”;
    return 0;
}
```

Errors with for Loops

- Do NOT place a ; (semicolon) directly after the command *for* in a *for* loop:
- Don’t do this for example:

```
for(int i = 1; i <= 10; i ++) ;  //Don’t do this!
     cout << i << end1;
```

- This will prevent any lines of code within the loop from being repeated or iterated.
- This will result in a logic error, the compiler will NOT tell you that there is a syntax error.

While Loops

- A while loop is similar to a for loop.
- While loops are sometimes better suited for many loops other than count controlled loops.
- In a *while* loop, something inside the loop triggers the loop to stop.
- For example, a while loop may be written to ask the user to enter a series of numbers while the number is not –999.

The while loop

- The while loop repeats a statement or group of statements as long as a control expression is true.
- Unlike a for loop, a while loop usually does not use a counter variable.
- The control expression in a while loop can be any valid expression.
- The program in Code List 9-5 uses a while loop to repeatedly divide a number by 2 until the number is less than or equal to 1.

Code List 9-5

```
// while1.cpp                    while1.txt
#include <iostream.h>
int main ( )
{
    float number;
    cout << “Please enter the number to divide:”;
    cin >> number;
    while (number > 1.0)
    {
      cout << number << endl;
      number = number / 2.0;
    }
    return 0;
}
```

While Loops

- General form:

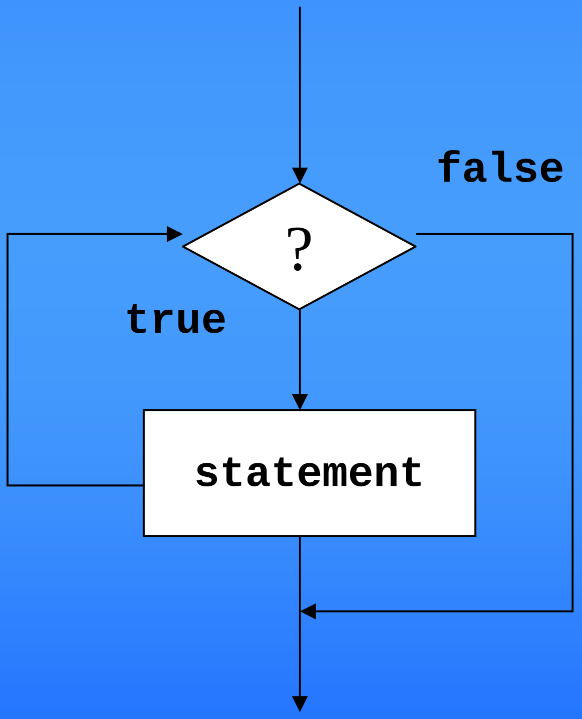
```
while (<Boolean expression>)
    <statement>
```

  - The parentheses around the Boolean is required.
  - If the condition is true the body of the loop is executed again.
  - If the loop condition is false, the program continues with the first statement after the loop.
  - A while loop may not be executed… why?

Syntax and Semantics of **`while`** *Statements*

```
while (<Boolean expression>)
   <statement>
```

```
while (<Boolean expression>)
{
   <statement 1>
   .
   <statement n>
}
```



While Loops: Discussion

- The condition can be any valid Boolean Expression
- The Boolean Expression must have a value PRIOR to **entering** the loop.
- The body of the loop can be a compound statement or a simple statement.
- The loop control condition needs to change in the loop body
  - If the condition is true and the condition is not changed or updated, an **infinite** loop could result.
  - If the condition is true and never becomes false, this results in an **infinite** loop also.

While Tests Before the Loop

- In a while loop, the control expression is tested before the statements in the loop begin.
- Figure 9-3 shows a flowchart of the program in Code List 9-5.
- If the number provided by the user is less than or equal to 1, the statements in the loop are never executed.

Figure 9-3

FIGURE 9-3

A while loop tests the control expression before the loop begins.

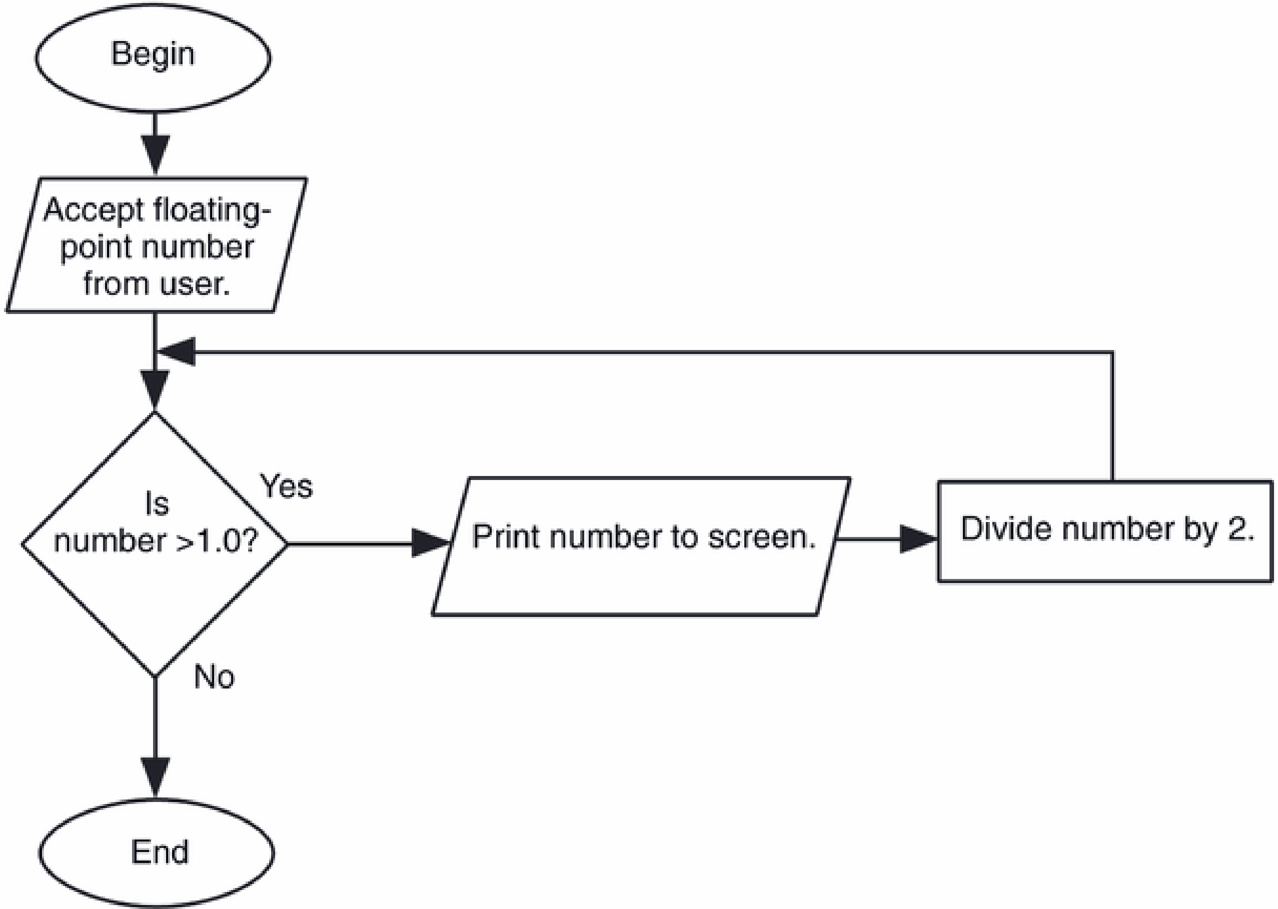


Figure 9-4

- Comparison of a *for* loop with a *while* loop to accomplish the same task in a count controlled loop.

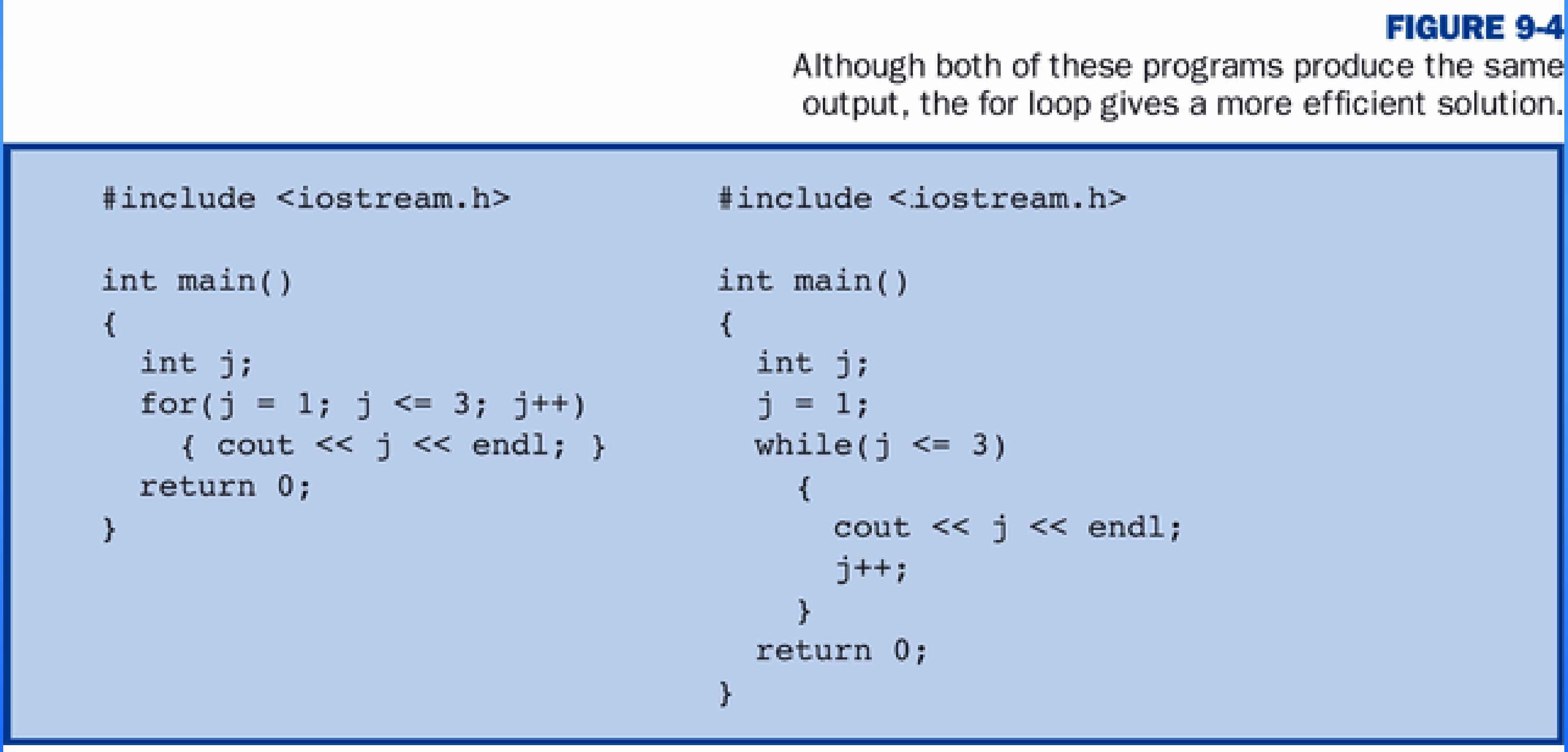


FIGURE 9-4

Although both of these programs produce the same output, the for loop gives a more efficient solution.

```
#include <iostream.h>

int main()
{
  int j;
  for(j = 1; j <= 3; j++)
    { cout << j << endl; }
  return 0;
}
```

```
#include <iostream.h>

int main()
{
  int j;
  j = 1;
  while(j <= 3)
    {
      cout << j << endl;
      j++;
    }
  return 0;
}
```

The while Loop Accumulator

Write code that computes the sum of the numbers between 1 and 10.

```
int counter = 1;
int sum = 0;
while (counter <= 10)
{
   sum = sum + counter;
   counter = counter + 1;
}
```

Sentinel Values and Counters

- Sentinel Value
  - A value that determines the end of a set of data, or the end of a process in an indefinite loop.

    P309ex1.cpp P309ex1.txt
  - While loops may be repeated an indefinite number of times.
    - It is common to count the number of times the loop repeats.
    - Initialize this “counter” before the loop
    - Increment the counter inside the loop

Errors with while Loops

- Do NOT place a **;** (semicolon) directly after the command *while* in a *while* loop:

```
int counter = 1;
while(counter <= 10) ;  //Don’t do this!
{
    cout << counter << end1;
    counter ++;
}
```

- This will prevent any lines of code within the loop from being repeated or iterated.
- This will result in a logic error, the compiler will NOT tell you that there is a syntax error.
- This could also result in an infinite loop.

The do while Loop

- The last iteration structure in C++ is the *do while* loop.
- A *do while* loop repeats a statement or group of statements as long as a control expression is true that is checked at the end of the loop.
- **Because the control expression is tested at the end of the loop, a *do while* loop is executed at least one time.**
- Code List 9-6 shows and example of a *do while* loop.

Code List 9-6

```
// dowhile.cpp                    dowhile.txt
#include <iostream.h>
int main ( )
{
     double number, squared;
     do
     {
         cout << “Enter a number (Enter -999 to quit):”;
         cin >> number;
         squared = number * number;
         cout << number << “squared is “ << squared << endl;
     }while (number!= -999);
     return 0;
}
```

do…while loops

- General form:

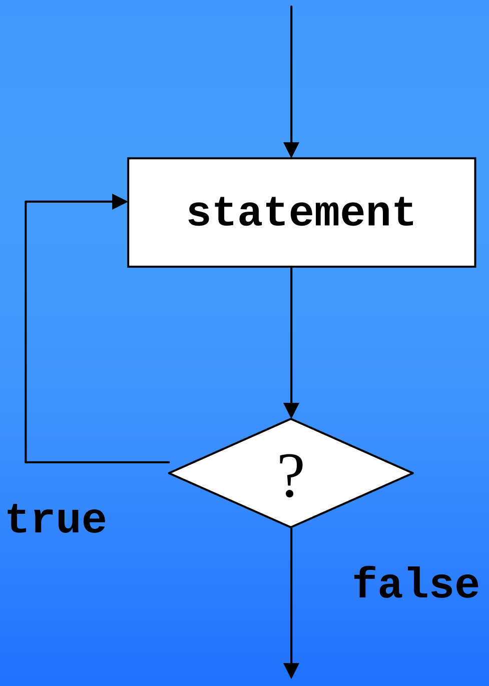
```
do
{
    <statement>
 }while (<Boolean expression>);
```

  - The Boolean expression must have a value before it is executed at the **end** of the loop.
  - If the loop condition is true, control is transferred back to the top of the loop.
  - If the loop condition is false, the program continues with the first statement after the loop.
  - A do...while loop will always be executed at least once… why?

*Syntax and Semantics of do…***`while`** *Statements*

```
do
{
   <statement>
}while (<Boolean expression>);

do
{
   <statement 1>
   .
   <statement n>
} while (<Boolean expression>);
```



do…while Loops: Discussion

- The condition can be any valid Boolean Expression
- The Boolean Expression must have a value PRIOR to **exiting** the loop.
- The body of the loop is treated as a compound statement even if it is a simple statement. { }
- The loop control condition needs to eventually change to FALSE in the loop body
  - If the condition never becomes false, this results in an **infinite** loop.

Errors with do while Loops

- Do NOT place a **;** (semicolon) directly after the command *do* in a *do while* loop:

```
int counter = 1;
do;  //Don’t do this!
{
   cout << counter << end1;
   counter ++;
} while(counter <= 10);
```

- This will result in a syntax error.

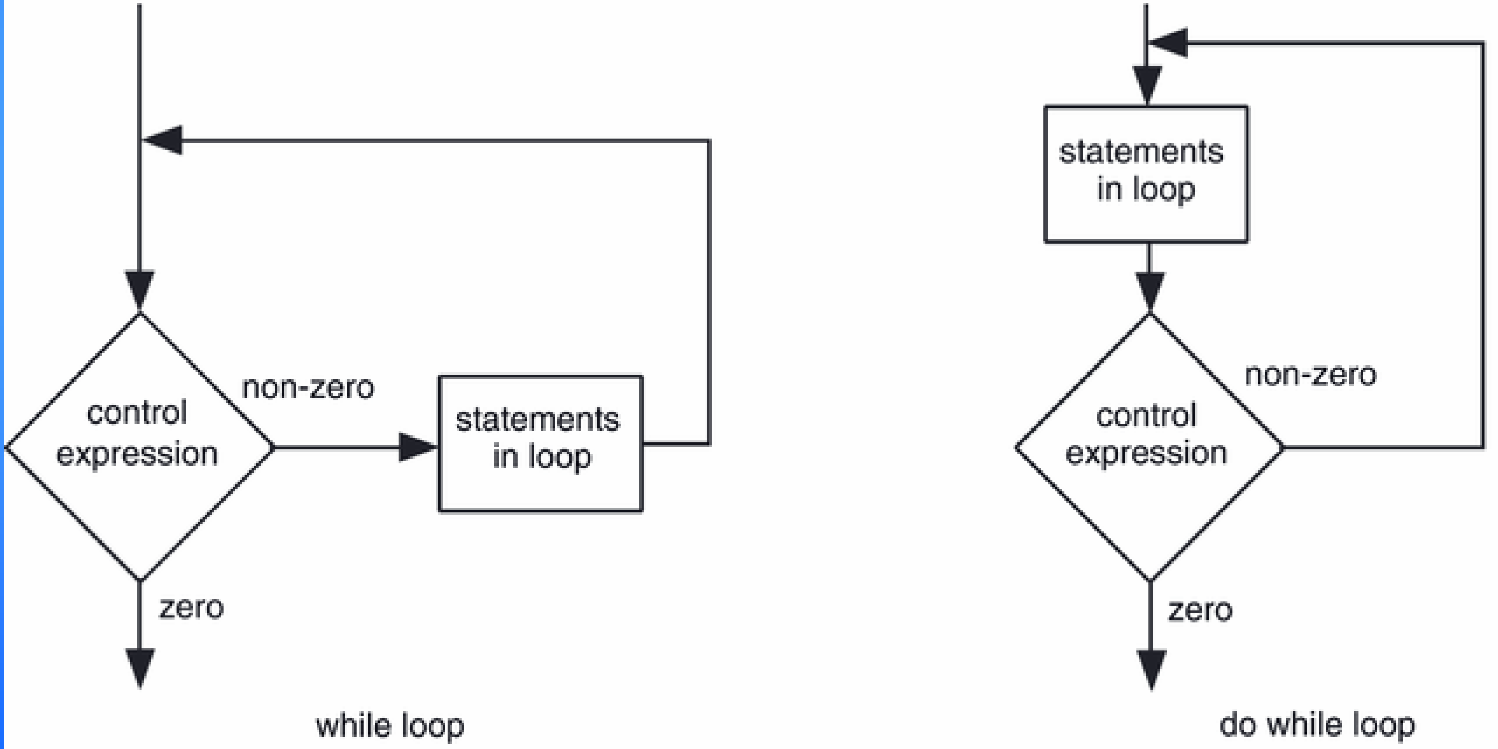
Comparing while with do while

- To help illustrate the difference between a while and a do while loop, compare the two flowcharts in figure 9-5.
- Use a while loop when you need to test the control expression **before** the loop is executed the first time.
- Use a do while loop when the statements in the loop need to be executed at least once.

Figure 9-5

- Pretest vs. Post Test indefinite loops.

FIGURE 9-5
The difference between a while loop and a do while loop is where the control expression is tested.



Choosing which loop to use.

- for loop
  - when a loop is to be executed a predetermined number of times.
- while loop
  - a loop repeated an indefinite number of times
  - check the condition before the loop
  - a loop that might not be executed (reading data)
- do...while
  - a loop repeated an indefinite number of times
  - check the condition at the end of the loop

Designing Correct Loops

- Initialize all variables properly
  - Plan how many iterations, then set the counter and the limit accordingly
- Check the logic of the termination condition
- Update the loop control variable properly

Off-by-One Error

```
int counter = 1;
while (counter <= 10)
{                 // Executes 10 passes
   <do something>
   counter++;
}
```

```
int counter = 1;
while (counter < 10)
{                 // Executes 9 passes
   <do something>
   counter++;
}
```

Infinite Loop

```
int counter = 1;
while (counter <= 10)
{                       // Executes 5 passes
   <do something>
   counter = counter + 2;
}
```

```
int counter = 1;
while (counter != 10)
{                       //Infinite Loop
   <do something>
   counter = counter + 2;
}
```

In general, avoid using **`!=`** in loop termination conditions.

Error Trapping

```
//”primed” while loop
cout<<"Enter a score between  ”<<low_double<<“ and  “<<high_double;
cin>>score;
while((score < low_double) || (score > high_double))
{
     cout<<“Invalid score, try again.”;

     //update the value to be tested in the Boolean Expression

      cout<<"Enter a score between  ”<<low_double<<“ and “<<high_double;
      cin>>score;
}
```

break and continue

- For this class **do not** use a **break** statement to terminate a loop.
- Only use break statements in a switch structure.
- Do not use continue in a loop either.
- Instead, use compound Boolean expressions to terminate loops.

Preferred Code List 9-7

```
// dowhilenobreak.cpp                    dowhilenobreak.txt
#include <iostream.h>
int main()
{
    double num, squared;
    do
    {
          cout << "Enter a number (Enter 0 to quit): ";
          cin >> num;
          if (num != 0.0)
          {
            squared = num * num;
            cout << num << " squared is " << squared << endl;
          }
    }while (num!=0);
 return 0;
}
```

Code List 9-7 using while

```
// whilenobreak.cpp                    whilenobreak.txt
#include <iostream.h>
int main()
{
    double num, squared;
    cout << "Enter a number (Enter 0 to quit): ";
    cin >> num;
    while (num!=0)
    {
      squared = num * num;
      cout << num << " squared is " << squared << endl;
      cout << "Enter a number (Enter 0 to quit): ";
      cin >> num;
    }
 return 0;
}
```

Nested Loops

- Nested loop
  - when a loop is one of the statements within the body of another loop.

```
for (k=1; k<=5; ++k)
   for (j=1; j<=3; ++j)
      cout<<(k+j)<<endl;
```

Multab.cpp

Multab.txt

  - Each loop needs to have its own level of indenting.
  - Use comments to explain each loop
  - Blank lines around each loop can make it easier to read

Code List 9-9

```
//nestloop.cpp                    nestloop.txt
#include <iostream.h>
int main()
{
     int i,j;
     cout << "BEGIN\n";
     for(i = 1; i <= 3; i++)
     {
          cout << " Outer loop: i = " << i << endl;
          for(j = 1; j <= 4; j++)
               cout << "     Inner loop: j = " << j << endl;
     }
     cout << "END\n";
     return 0;
}
```

Repetition and Selection

- The use of an if statement within a loop to look for a certain condition in each iteration of the loop.
  - Examples:
    - to generate a list of Pythagorean Triples
    - to perform a calculation for each employee
    - to find prime numbers
      - let’s look at our Case Study program for Chapter 6

primes.cpp primes.txt