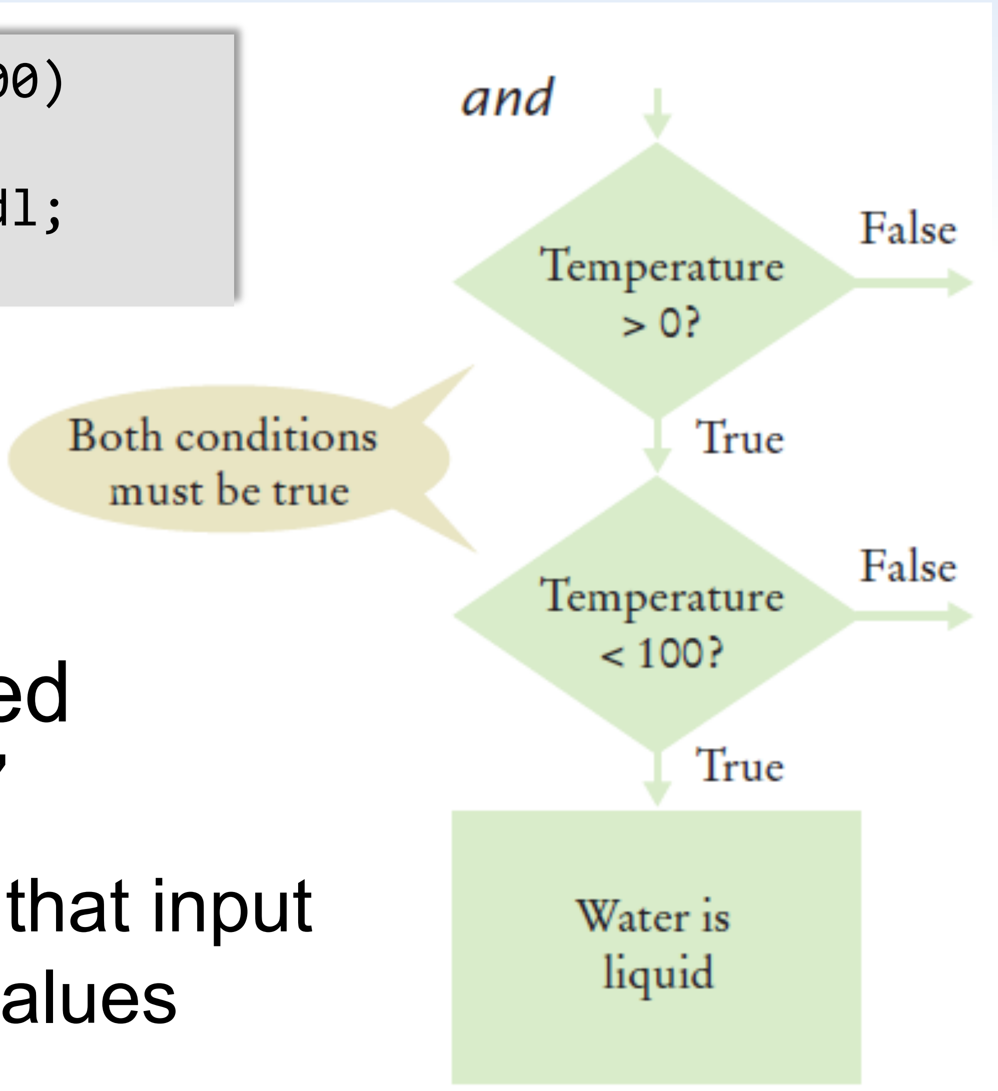
Chapter 6 and 7

Loops++

and Flowchart

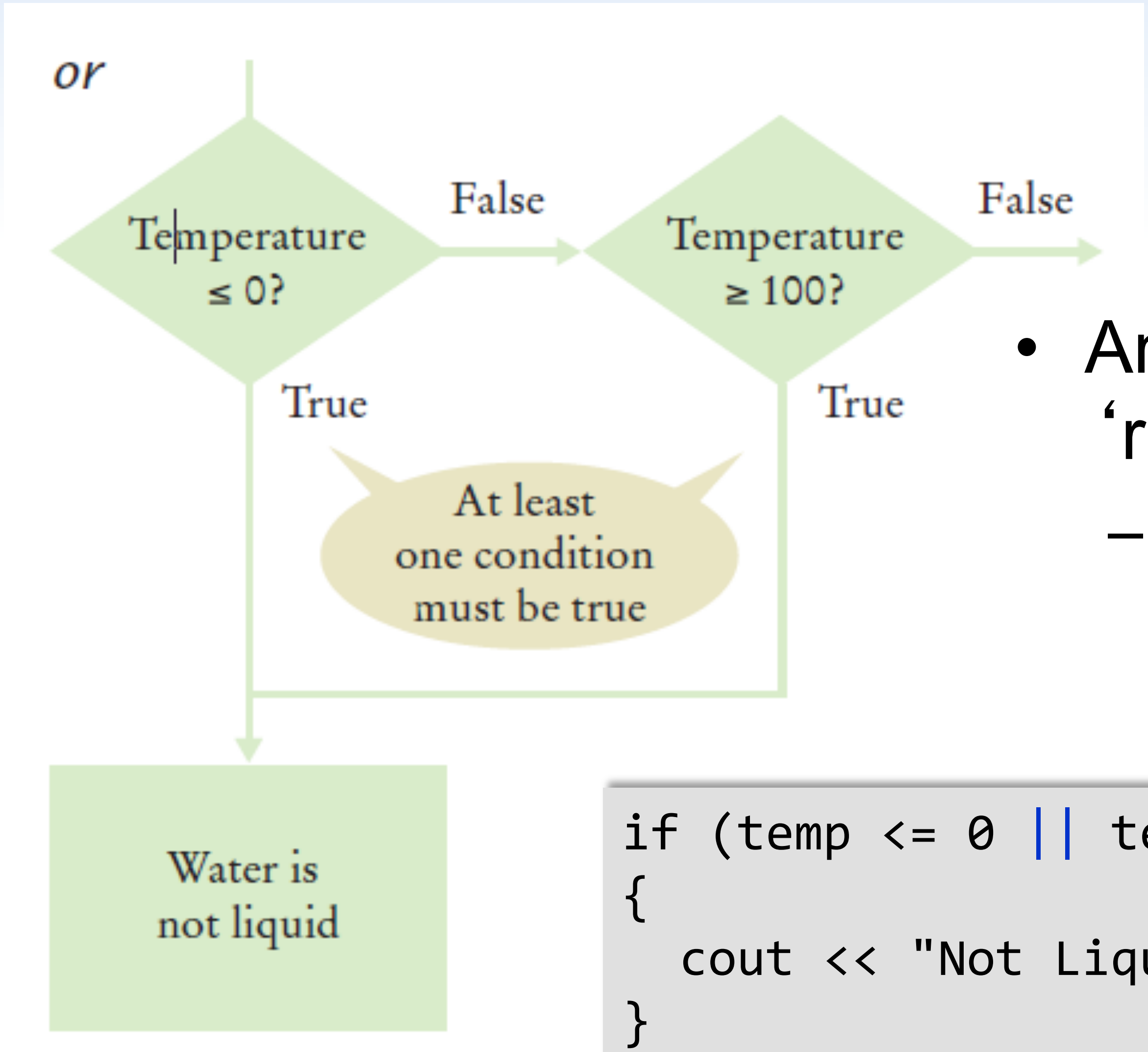
```
if (temp > 0 && temp < 100)
{
  cout << "Liquid" << endl;
}
```



- This is often called ‘range checking’
  - Used to validate that input is between two values

Copyright © 2013 by John Wiley & Sons. All rights reserved.

or Flowchart



- Another form of ‘range checking’
  - Checks if value is outside a range

```
if (temp <= 0 || temp >= 100)
{
   cout << "Not Liquid" << endl;
}
```

Copyright © 2013 by John Wiley & Sons. All rights reserved.

The switch Statement
(multiway branching)

- An example of a switch statement:

```
switch (option)
{
   case 'A':
      aCount++;     // optional
      break;        // optional
   case 'B':
      bCount++;     // optional
      break;        // optional
   case 'C':
      cCount++;     // optional
      break;        // optional
   default:         // optional
      cerr << "bad option!" << endl;  // optional
      break;        // optional
}
```

The switch Statement

- The expression of a `switch` statement must result in an *integral type*, meaning an integer (`short`, `int`, `long`) or a `char`
- It cannot be a `bool` value or a floating point value (`float` or `double`)
- The implicit boolean condition in a `switch` statement is equality
- You cannot perform relational checks with a `switch` statement

Loops

- *Repetition statements* allow us to execute a statement multiple times
- Often they are referred to as *loops*
- Like conditional statements, they are controlled by boolean expressions
- C++ has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*
- The programmer should choose the right kind of loop for the situation

Two Types of Loops

Count controlled loops

Repeat a statement or block a specified number of times

Event-controlled loops

Repeat a statement or block until a condition within the loop body changes that causes the repetition to stop

While Statement

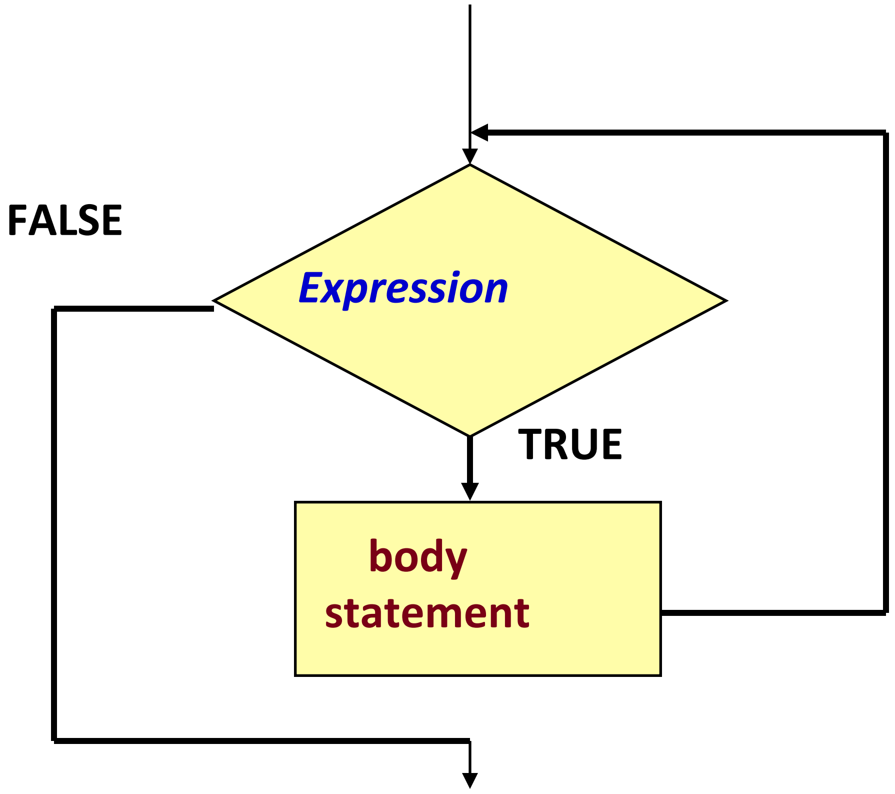
SYNTAX

```
while (Expression)
{        .
         .           // loop body
         .
}
```

Loop body can be a single statement, a null statement, or a block

- When the expression is tested and found to be false, the loop is exited and control passes to the statement that follows the loop body

WHILE LOOP



Count-Controlled Loops

Count-controlled loops contain:

- An initialization of the loop control variable
- An expression to test if the proper number of repetitions has been completed
- An update of the loop control variable to be executed with each iteration of the body

Count-Controlled Loop Example

```
int   count;          // Loop-control variable

count  =  4;          // Initialize loop variable

while(count > 0)      // Test expression
{
    cout  << count  << endl; // Repeated action

    count --; // Update loop variable
}
cout  << “Done” << endl;
```

Types of Event-Controlled Loops

- **Sentinel controlled**

  Keep processing data until a special value that is not a possible data value is entered to indicate that processing should stop

- **End-of-file controlled**

  Keep processing data as long as there is more data in the file

- **Flag controlled**

  Keep processing data until the value of a flag changes in the loop body

Example

myInfile contains 100 blood pressures

Use a while loop to read the 100 blood pressures and find their total

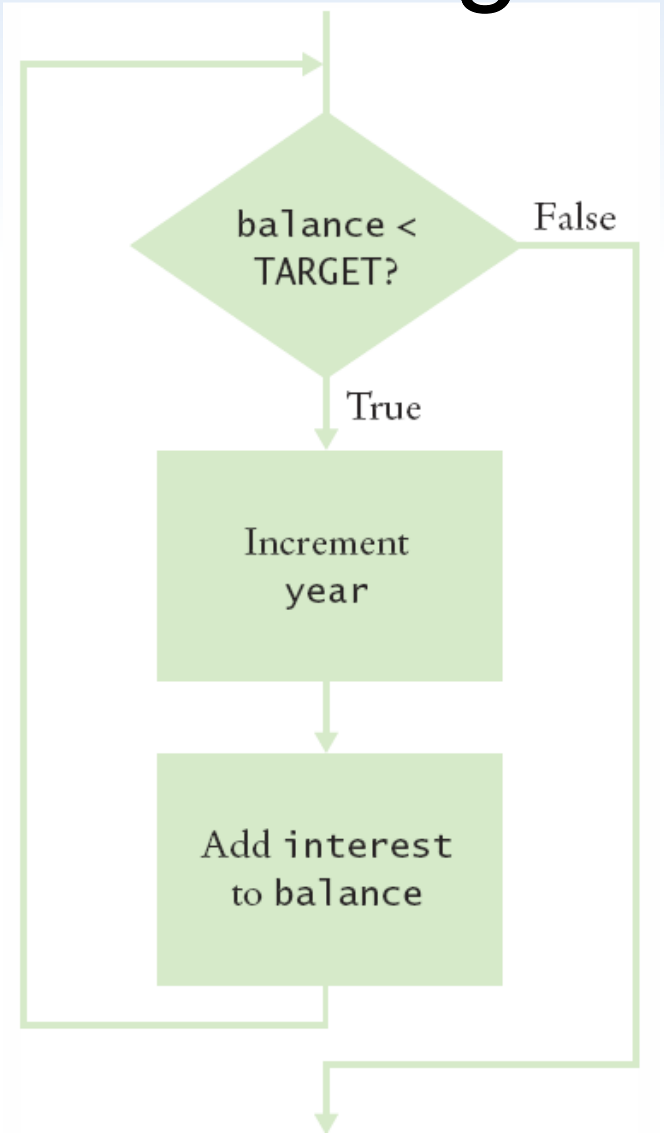
Examples of Kinds of Loops

Count controlled loop	Read exactly 100 blood pressures from a file
End-of-file controlled loop	Read all the blood pressures from a file no matter how many are there

Examples of Kinds of Loops

Sentinel controlled loop	Read blood pressures until a special value selected by you(like -1) is read
Flag controlled loop	Read blood pressures until a dangerously high BP(200 or more) is read

An flag controlled `while` Loop



A loop executes instructions repeatedly while a condition is true.

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE/100;
   balance = balance + interest;
}
```

Copyright © 2013 by John Wiley & Sons. All rights reserved.

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25)
{
   cout << count << endl;
   count = count - 1;
}
```

- This loop will continue executing until interrupted (Control-C) or until an underflow error occurs

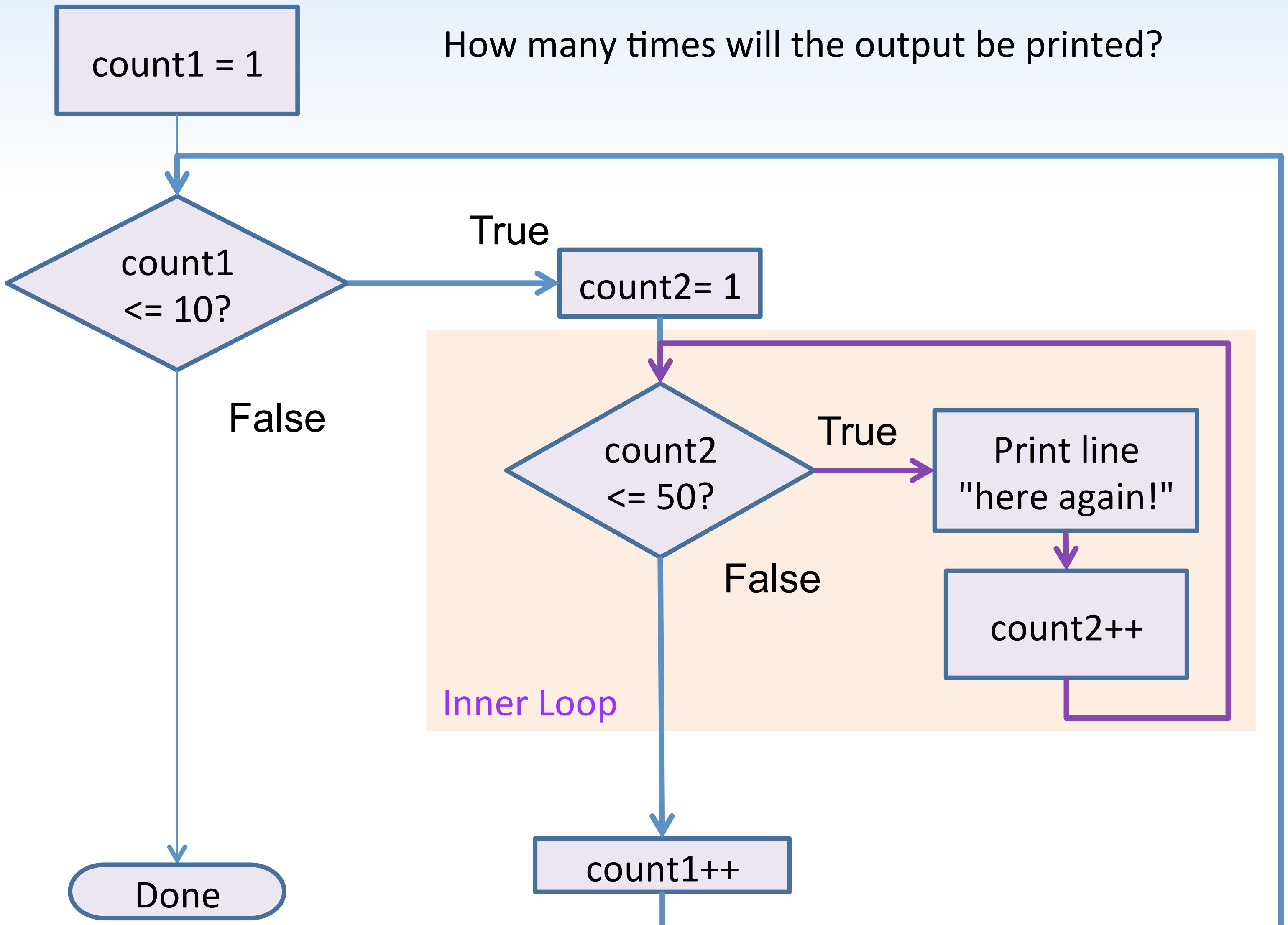
Nested Loops

- How many times will the output be printed?

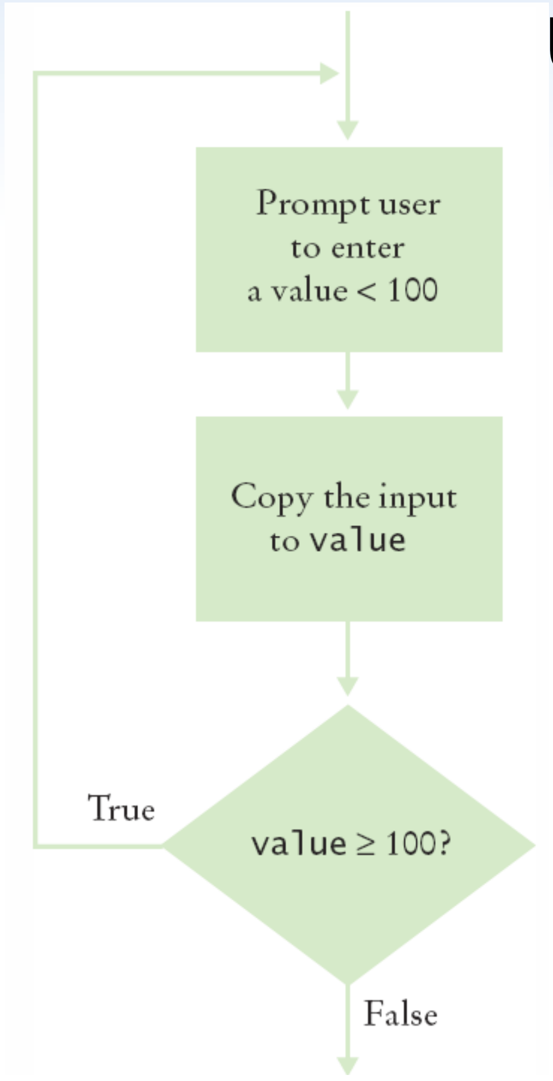
```
count1 = 1;
while (count1 <= 10)
{
   count2 = 1;
   while (count2 <= 50)
   {
      cout << "Here again" << endl;
      count2++;
   }
   count1++;
}
```

Flowchart of a Nested Loop

How many times will the output be printed?



4.4 The do Loop



Use a do loop when you want to:

– Execute the body at least once

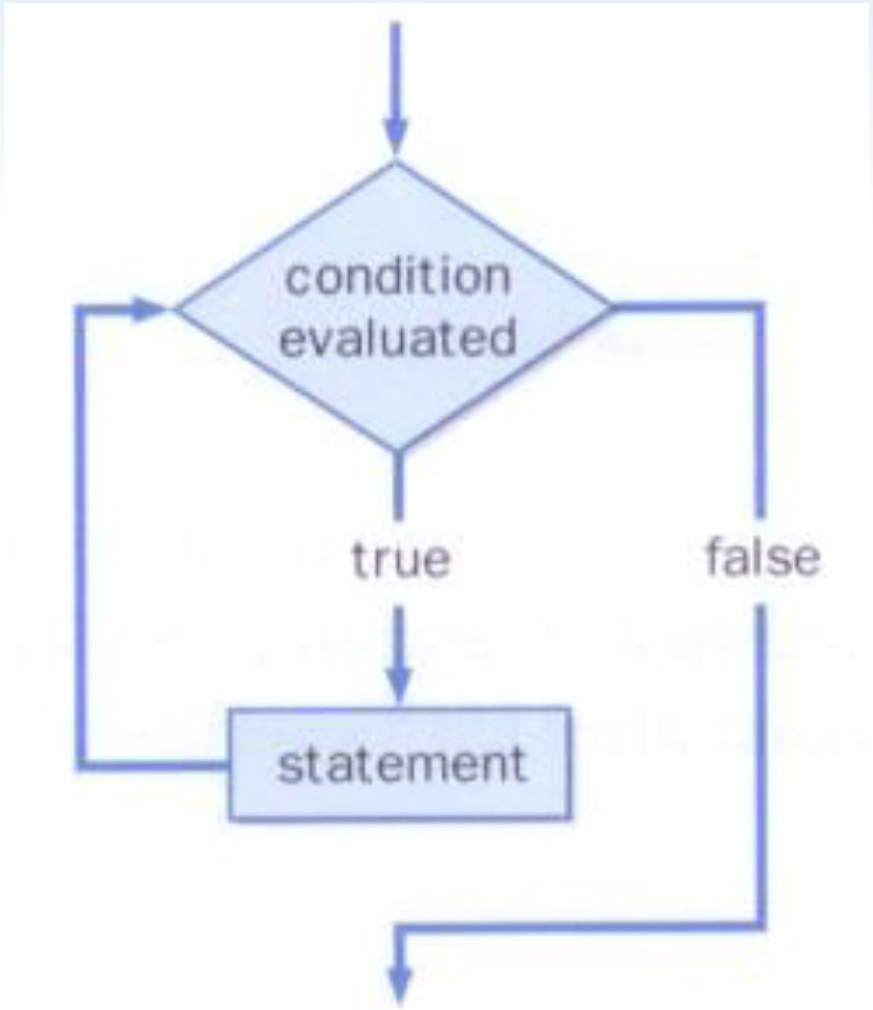
– Test the condition AFTER your first loop

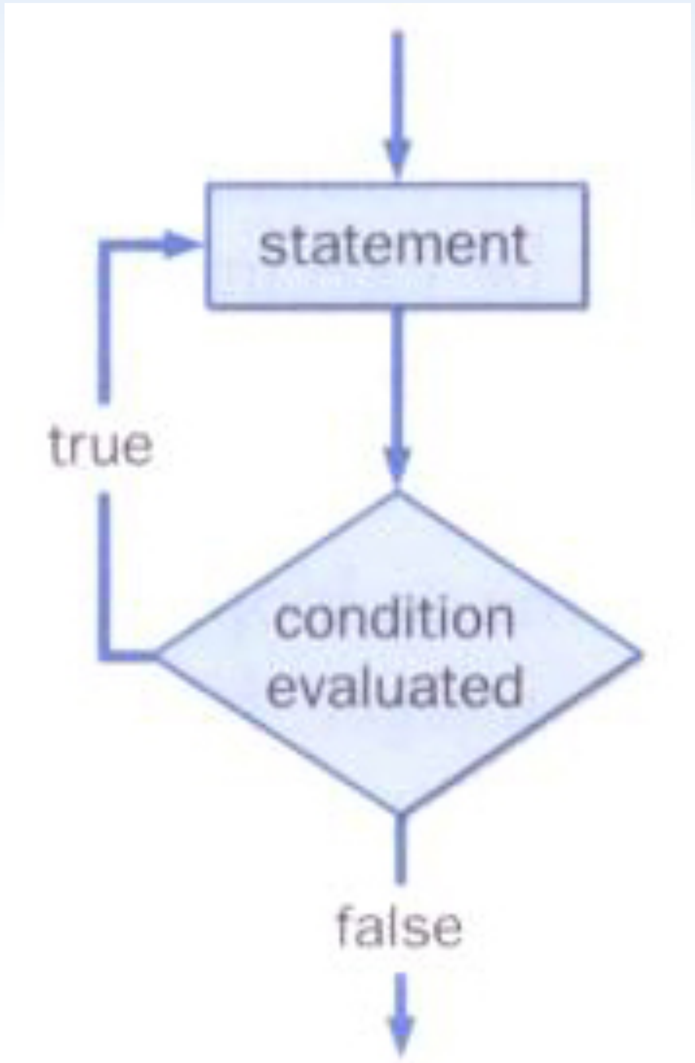
```
int i = 1;  // initialize
final int FINGERS = 5;
do
{
   // paint finger
   i++; // update
}
while (i <= FINGERS);  // test
```

Note the semicolon at the end!

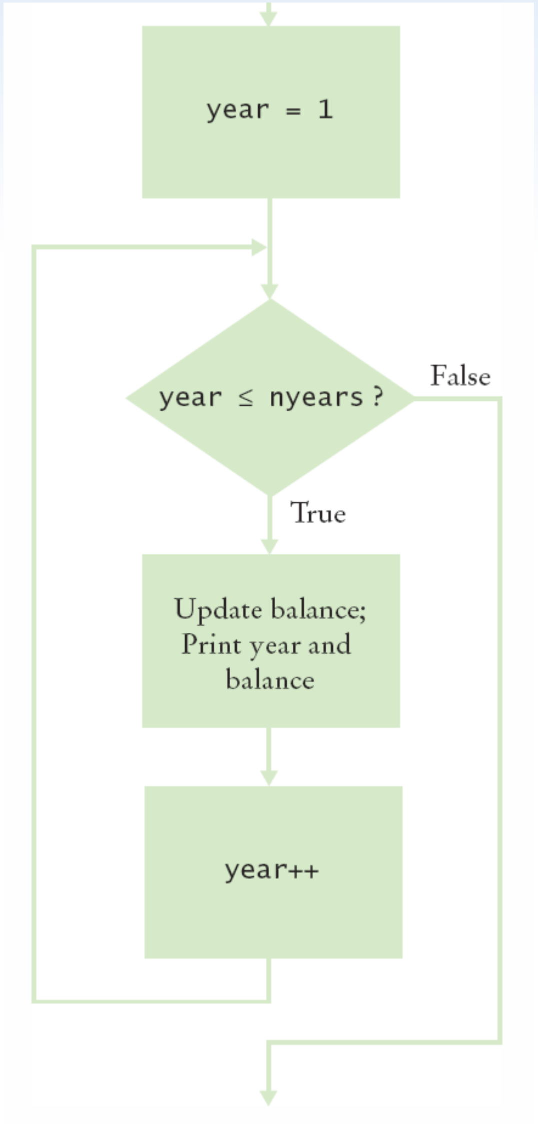
Copyright © 2013 by John Wiley & Sons. All rights reserved.

Comparing while and do Loops





A for Loop (count controlled)



- Print the balance at the end of each year for a number of years

Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

```
for (int year = 1; year <= nyears; year++)
{
    Update balance.
    Print year and balance.
}
```

Copyright © 2013 by John Wiley & Sons. All rights reserved.

The for Loop

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

The for Loop

- An example of a for loop:

```
for (int count=1; count <= 5; count++)
   System.out.println (count);
```

- The initialization section can be used to declare a variable

- Like a while loop, the condition of a for loop is tested prior to executing the loop body

- Therefore, the body of a for loop will execute zero or more times

Steps to Writing a Loop

Planning:

1. Decide what work to do inside the loop
2. Specify the loop condition
3. Determine loop type
4. Setup variables before the first loop
5. Process results when the loop is finished
6. Trace the loop with typical examples

Coding:

7. Implement the loop in C++

Copyright © 2013 by John Wiley & Sons. All rights reserved.

Lab 3

- Loops