Chapter 8
Repetition

3rd Edition

Computing Fundamentals with C++

Rick Mercer

Franklin, Beedle & Associates

Goals

- Use the Determinate Loop pattern to execute a set of statements a predetermined number of times
- Implement determinate loops with the `for` statement
- Recognize and use the Indeterminate Loop pattern to execute a set of statements until some event occurs to stop it (no more data, for example)
- Implement indeterminate loops with the C++ `while` statement

Repetitive Control

- The following algorithms involve repetition
    - Add the remaining flour ¼ cup at a time whipping until smooth
    - While there are more burger/fries/soda orders, sum each item. Apply tax. Display Total.
    - Compute a course grade for every student
    - While the ATM is running, process another customer, and allow many transactions
    - Microwave the food until the timer reaches 0, the cancel button is hit, or the door is opened

Why is repetition needed?

- To take advantage of the computer's speed to perform the same tasks faster
- To avoid writing the same statements over and over again (shorter programs)
- To visit all elements in a collection of objects
- To make programs general enough to handle various sized collections of data
- Consider code intended to average exactly 100 numbers (next slide):

Crazy way to average 100 values

```
double number;
double sum = 0;
cout << "Enter number: "; // <-Repeat these three
cin >> number;            // <- statements for each
sum = sum + number;       // <- number in the set
cout << "Enter number: ";
cin >> number;
sum = sum + number;

// ...291 statements deleted ...

cout << "Enter number: ";
cin >> number;
sum = sum + number;
double average = sum / 100;
cout << "Average: " << average << endl;
```

How many statements are required for 100 inputs?_____

What changes are necessary to average 200 inputs? _____

The Determinate Loop Pattern

- There is a better way
- We often need to perform some action a specific number of times:
  - Produce 89 paychecks
  - Count down to 0 (take 1 second of the clock)
  - Send grade reports to 75,531 students
- The *Determinate Loop* pattern repeats some action a specific number of times

Pattern:	Determinate Loop
Problem:	Do something exactly n times, where n is known in advance.

Algorithm	determine n
	repeat the following n times {
	    perform these actions
	}

Code Example:	`cout << "Enter n: ";`
	`cin >> n;`
	`for (int count = 1; count <= n; count++) {`
	`  cout << "Enter number: "; // Repeat`
	`  cin >> number;            // these`
	`  sum = sum + number;        // statements`
	`}`

Determinate Loops

- This template repeats a process n times

```
n = how often we must repeat the process
for (int i = 1; i <= n; i = i + 1) {
    the process to be repeated
}
```

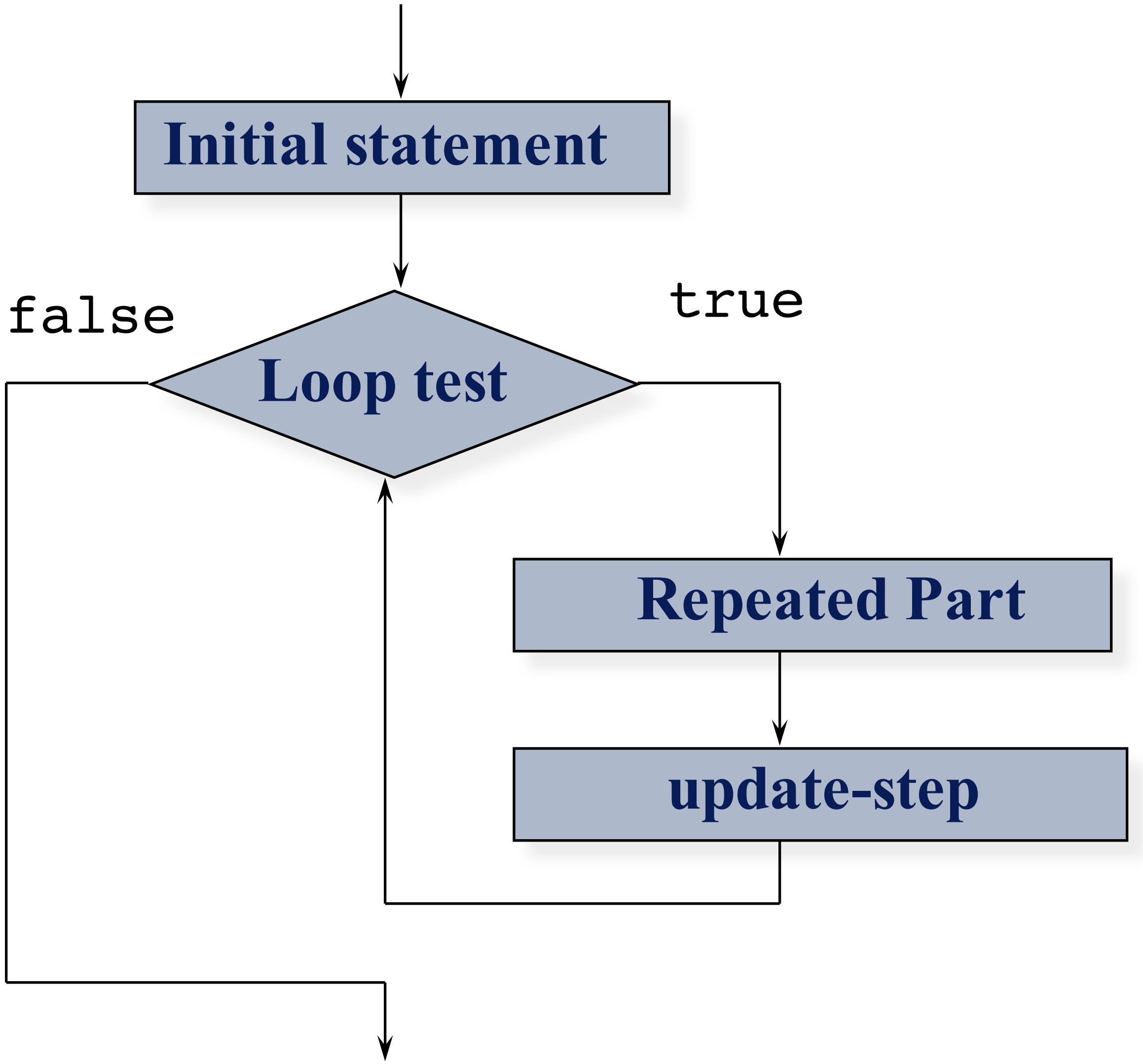
- *Determinate Loops* must know the number of repetitions ***before*** they begin
  - Know exactly how many employees, or students, or whatever, that must be processed

The for loop

```
for  ( initial statement; loop-test;  update-step )    {
  repeated-part
}
```

- When a `for` loop is encountered
  - the *initial-statement* is executed, usually `int i = 0;`
    - The *initial-statement* is only executed once, when the loop is entered
  - the *loop-test* evaluates to true or false
  - if the *loop-test* is false, the for loop is terminated
  - if *loop-test* is true, the *repeated-part* is executed and the *update-step* executes

Flow Chart View of a `for` loop



Use a `for` loop to produce an average

```
int n;
double number;
double sum = 0.0;
// Get a value for the number of iterations
cout << "How many numbers? ";
cin >> n;

for(int count = 1; count <= n; count = count + 1) {
   // Repeat the same three statements n times
   cout << "Enter number: ";
   cin >> number;
   sum = sum + number;
 }

// Compute and display the average
double average = sum / n;
cout << "Average: " << average;
```

Operators `++` and `--`

- It is common to see determinate loops of this form where n is the number of repetitions

```
for(int count = 1; count <= n; count++)
```

- The unary `++` and `--` operators add 1 and subtract 1 from the values, respectively

```
int n = 0;
n++; // n is now 1, equivalent to n=n+1;
n++; // n is now 2
n--; // n is now 1
```

- The expression `count++;` is equivalent to the more verbose `count = count + 1;`

Other Assignment Operators

- C++ has several assignment operators in addition to `=`

  `n -= 2;` is the equivalent of `n = n - 2;`

  `sum += x;` is the equivalent of `sum = sum + x;`

- What is `sum` when a user enters 7 and 8?

```
int sum = 0;
int num = 0;
cout << "Enter a number: ";
cin >> num;   // user enters 7
sum += num;
cout << "Enter a number: ";
cin >> num;   // user enters 8
sum += num;
```

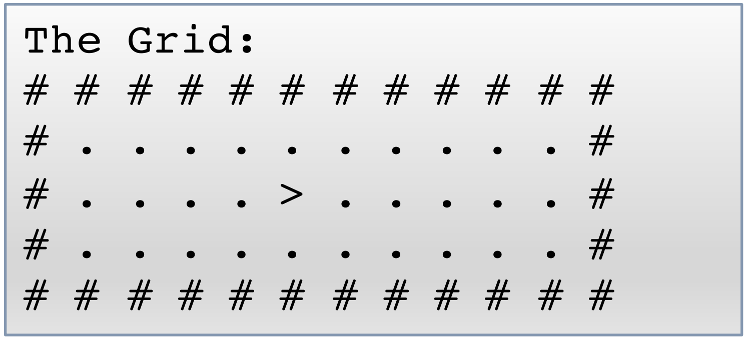
Determinate Loops with Grid Object

- This code surrounds the Grid with blocks

```
Grid g(7, 14, 4, 4, east);
g.display();
for (int row = 0; row < g.nRows(); row++) {
  g.block(row, 0);                  // block west col
  g.block(row, g.nColumns() - 1); // block east col
}

for (int col = 1; col < g.nColumns() - 1; col++) {
  g.block(0, col);                  // block north row
  g.block(g.nRows() - 1, col);      // block south row
}

g.display();
```



```
The Grid:
# # # # # # # # # # # #
# . . . . . . . . . . #
# . . . . > . . . . . #
# . . . . . . . . . . #
# # # # # # # # # # # #
```

The Determinate Loop Pattern
Find the Range of Test Scores

- Find the range of test scores where range is defined as the highest minus the lowest
- With the input of 4 test scores 80, 70, 100, and 90, what is the range _____?
- Prelude to the range problem:
    - Imagine finding the largest number in a list of thousands of numbers—we need a systematic method (we can’t just glance at the list)

Analysis

- Problem: Write a program that determines a range (highest-lowest) of test scores. The user must enter the number of tests to check
- Inputs: The number of test scores to scan, and the actual test scores
- Output: The range
- Name the objects?

__________ __________ ________ ________

Design

- Start with this algorithm
  1. Obtain the number of test scores
  2. Determine the range
  3. Display the range
- You might notice that the process step, "Determine the range", needs further refinement
- The first step is a prompt/input pattern and the third step is simply labeled output

Design (an Algorithm)

1. Obtain the number of test scores

```
cout << "Enter number of test scores: ";
cin >> n;
```

2. Determine the range: TBA
3. Display the range

```
cout << "Range = " << range;
```

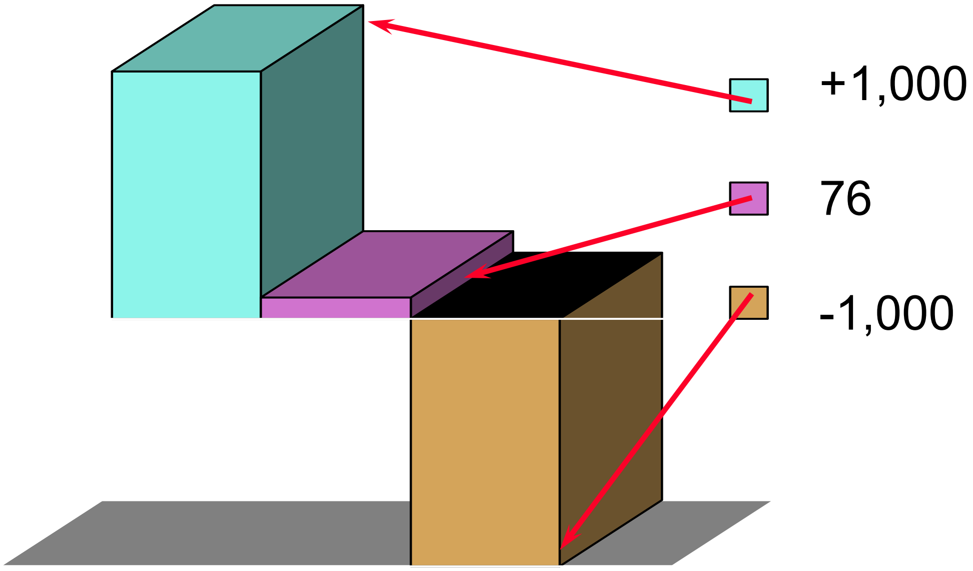
- Let us concentrate on the second step:
  - Determine the range
- Since range is defined as largest – smallest, we need to find the largest and smallest

Design

- We need the actual test scores for input to determine the largest and smallest
- As each new test score is input, we compare it to the highest so far, and also to the smallest so far
- But what do we compare the first test to?
    - How about something very large for the smallest
        - 1,000 will be the smallest so far
    - and something very small for the largest
        - -1,000 will be the largest so far
    - Then the first number (we'll use 76) is compared to these artificial values for largest and smallest

Design continued

- In a side by side comparison, we see a valid test score (76) is greater than the largest so far (-1000) and also less than the smallest so far (+1000)



Design

- Before reading tests from the user, initialize largest and smallest like this:

```
double largest = -1000;
double smallest = +1000;
```

- Then we need to do the following n times
  1) Input a test
  2) Compare test to largest and if necessary, store the test as the largest
  3) Compare to smallest and if necessary store it as the smallest
- Trace with inputs of 87, 91, 72 (range 91-72=19)

test	**?**	**87**	**91**	**72**
`largest`	`-1000`	`87`	`91`	`91`
`smallest`	`+1000`	`87`	`87`	`72`

Implementation

```
int n = 3;
int test;
int largest = -1000;
int smallest = 1000;
// 2. Determine the range
for (int counter = 1; counter <= n; counter++) {
  // The process to repeat n times
  cout << "Enter test: ";
  cin >> test;
  if (test > largest)
    largest = test;
  if (test < smallest)
    smallest = test;
}
int range = largest - smallest;
cout << range;
```

Dialog

```
Enter test: 87
Enter test: 91
Enter test: 72
19
```

Why bother?

- It should be noted, that this computer based range problem is more cumbersome than just scanning a small list of tests for the highest and lowest
- But imagine thousands of value stored in a file or a spreadsheet
- We could use the same pattern, but someone must somehow count the inputs before starting
- There must be a way to do this programmatically

Algorithmic Pattern
The Indeterminate Loop

- Determinate loops have a limitation
  - We must know n in advance
- Many situations repeat a set of statements, but we can not determine how many:
  - Processing report cards for every student in a school (or paychecks for all employees, or...)
  - Generating a bill for every customer
  - Playing a game until somebody wins

Some Events that terminate indeterminate loops

- An *indeterminate loop* repeats a process until some stopping event terminates the repetition
- There are many such events, but we'll focus on these events only:
  - User enters a special value indicating end of data.
  - A logical expression becomes false
  - The Grid mover hits the wall or an edge
  - The end of a file is encountered
- Indeterminate loops do not need to know n in advance

Pattern:	Indeterminate loop
Problem:	Some process must repeat an unknown number of times so some event is needed to terminate the loop.

Algorithm:	while(the termination event has not occurred) {
	    perform these actions
	    bring the loop closer to termination somehow
	}

Code Example:	`while(aGrid.frontIsClear()) {`
	  `myGrid.putDown();`
	  `myGrid.move();`
	`}`

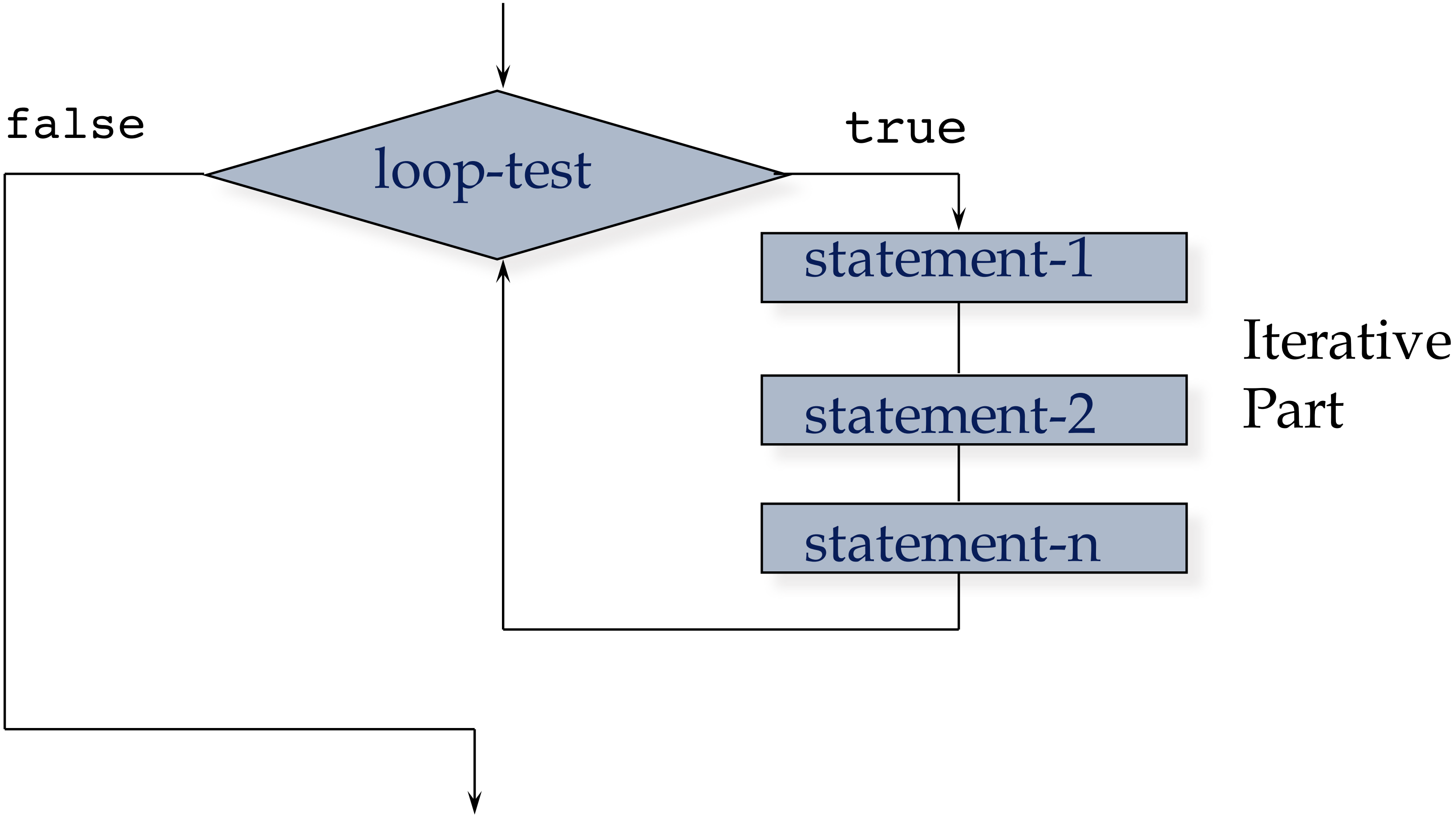
The while loop

- The indeterminate loop pattern can be implemented with the C++ `while` loop

```
while ( loop-test ) {
   repeated-part
}
```

- When a `while` statement is encountered the block executes *while* (as long as) the loop-test is true
- You need to determine the loop test, an expression that must eventually become false

Flow chart view of while-loop execution



while Statement as a Determinate Loop

- This loop terminates when counter <= n becomes false
- The event that terminates this loop is counter > n

```
int counter = 1;
int n = 4;
while (counter <= n) {
  cout << counter << " ";
  counter++;
}
```

- Output? _______________

Indeterminate Loop Pattern with `Grid`

```
Grid g(5, 10);
// assert: g is a 5x10 Grid surrounded by blocks
// with one opening and the mover in a random spot
while (g.frontIsClear()) {
  g.move(1);
}
g.display();
```

Output

```
# # # # # # # # # #
# . . . . . . . . #
# <               .
# . . . . . . . . #
# # # # # # # # # #
```

Indeterminate Loop Using a Sentinel

- A *sentinel* is a specific input from the user or a signal that there is no more data
  - The sentinel must be the same type of data
  - The sentinel must not be in the valid range of data
  - Example: Use -1 as the sentinel for test scores that can only be in the range of 0 through 100
- Enter test scores or -1 to quit:

```
80 95 76 82 56 100 45 86 -1
```

- A priming read could be used
  - The first input could be -1 or a valid number
  - The while loop test will check (see next slide)

Priming Read

- Read before the loop and at the end!

```
int sum = 0;
int test;
cout << "Enter data or -1 to quit" << endl;
cin >> test;
while (test != -1) {
  sum += test;
  cin >> test;
}
cout << sum;
```

Dialog

```
Enter data or -1 to quit
1  2  3
-1
6
```

Using `cin >>` as a Loop Test

- An input with `cin` evaluates to true or false

```
while (cin >> intObject)
```

- It can be part of the loop test to simplify the code

```
// Reading input can be part of a loop test
while ((cin >> test) && (test != -1)) {
  // Must have a valid int not equal to -1
  sum += test;
  n++; // n is count of test
}
```

Infinite Loops

- *Infinite loop:* a loop that never terminates
- Infinite loops are usually not desirable
- Below is an example of an infinite loop, there is no step that brings the loop closer to termination
    - Wait until you hear your fan turn on, or better yet, terminate the program

```
cin >> test;
while (test != -1) {
  sum += test;
  n++;
}
```

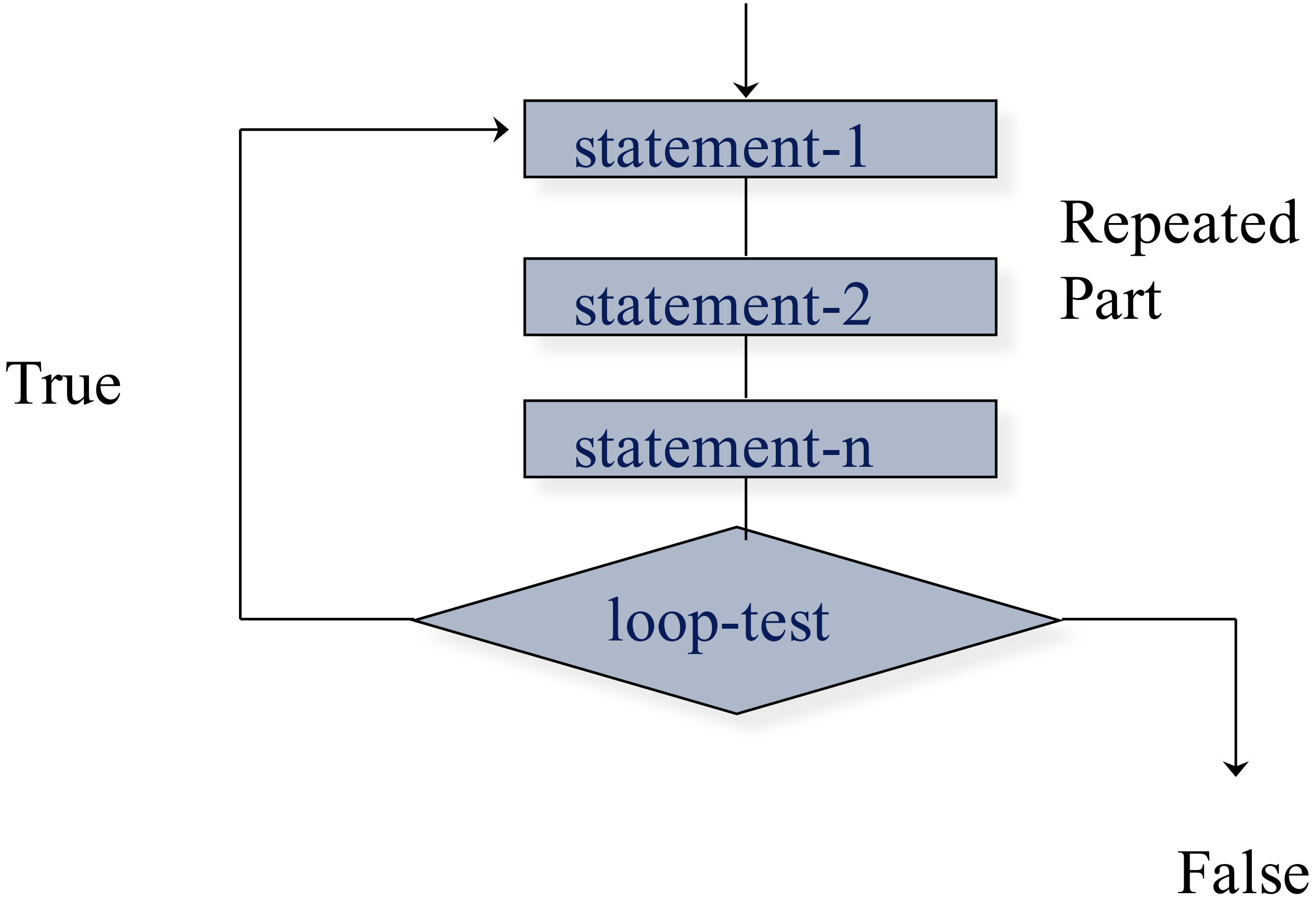
The do while Statement

- C++ also has a "post-test" loop
  - The loop test occurs at the end of the loop
- Use when you have to do something to initialize part of the loop test (or use `while` with `break`)

```
do {
   repeated-part
 } while ( loop-test ) ;
```

- The repeated part always executes at least once
  - a while loop executes zero times if the loop test is false immediately

Flow chart view of do while



Why another loop?

```
char nextOption() {
  // post: return an uppercase W, D, or Q
  char option = '?';
  do {
    cout << "W)ithdraw, D)eposit, or Q)uit: ";
    cin >> option; // wants w, W, d, D, q, or Q
    option = toupper(option); // need option in test
  } while( (option != 'W') && // a post test loop
           (option != 'D') &&
           (option != 'Q') );
  return option;
}

int main() {
  cout << nextOption();
  return 0;
}
```

Dialog:

```
W)ithdraw, D)eposit, or Q)uit: x
W)ithdraw, D)eposit, or Q)uit: y
W)ithdraw, D)eposit, or Q)uit: z
W)ithdraw, D)eposit, or Q)uit: w
W
```

Equivalent while loop

- The while loop repeats until the user enters an upper or lower case W, D, or Q using `break` to exit the loop

```
char nextOption() {
  // post: return an uppercase W, D, or Q
  char option;
  while (true) {
    cout << "W)ithdraw, D)eposit, or Q)uit: ";
    cin >> option;
    option = toupper(option);
    if (option=='W' || option=='D' || option=='Q')
      break; // a more positive way to stop
  }
  return option;
}
```

Loop Selection and Design

- The following outline is offered to help you choose and design loops in a variety of situations:
  - Determine which type of loop to use
  - Determine the loop-test
  - Write the statements to be repeated
  - Bring the loop one step closer to termination
  - Initialize objects if necessary

Determine Which Type of Loop to Use

- If the number of repetitions is known in advance or read as input, use a determinate for loop
- If the loop must stop when some event occurs, use an indeterminate while loop
- When the loop must always execute once (to validate input for example), use a do-while loop

Determine the Loop Test

- Try writing the conditions that must be true for the loop to terminate

```
inputName == "QUIT" // Termination condition
```

- The logical negation (with ! applied) can be used directly as the loop-test of a while loop:

```
while ( inputName != "QUIT") // logical negation
```

Write the Statements to be Repeated

- This is why the loop is being written

```
{
  cout << "Enter number: ";
  cin >> x;
  sum = sum + x;
  n++;
}
```

Bring the Loop one Step Closer to Termination

- To avoid an infinite loop, there should be at least one action in the loop body that brings it closer to termination.
  - Increment the counter by +1
  - Read data from an input stream with `cin >>`

Initialize Objects if Necessary

- Check to see if any objects used in either the body of the loop or the loop-test need to be initialized
- In this loop, which object(s) need to be initialized before this while loop is encountered? __________

```
int count, n;
double x, sum;
while (count <= n) {
  cout << "Enter a number: ";
  cin >> x;
  sum = sum + x;
  count++;
}
```