An Introduction to Programming with C++

Sixth Edition

Chapter 7
The Repetition Structure
Objectives

• Differentiate between a pretest loop and a posttest loop
• Include a pretest loop in pseudocode
• Include a pretest loop in a flowchart
• Code a pretest loop using the C++ while statement
• Utilize counter and accumulator variables
• Code a pretest loop using the C++ for statement
Repeating Program Instructions

- The **repetition structure**, or **loop**, processes one or more instructions repeatedly.
- Every loop contains a Boolean condition that controls whether the instructions are repeated.
- A **looping condition** says whether to continue looping through instructions.
- A **loop exit condition** says whether to stop looping through the instructions.
- Every looping condition can be expressed as a loop exit condition (its opposite).
Repeating Program Instructions (cont’d.)

- C++ uses looping conditions in repetition structures
- A repetition structure can be pretest or posttest
- In a pretest loop, the condition is evaluated before the instructions in the loop are processed
- In a posttest loop, the condition is evaluated after the instructions in the loop are processed
- In both cases, the condition is evaluated with each repetition
Repeating Program Instructions (cont’d.)

- The instructions in a posttest loop will always be processed at least once
- Instructions in a pretest loop may not be processed if the condition initially evaluates to false
- The repeatable instructions in a loop are called the loop body
- The condition in a loop must evaluate to a Boolean value
Repeating Program Instructions (cont’d.)

Figure 7-1 A problem that requires the sequence structure only

Robin is sitting at a table in a bookstore. Robin needs to sign a copy of her bestselling book on Robotics for a customer.

1. accept the book from the customer
2. place the book on the table
3. open the front cover of the book
4. sign your name on the first page
5. close the book
6. return the book to the customer
7. thank the customer
Repeating Program Instructions (cont’d.)

Robin is sitting at a table in a bookstore, attending her book signing. Customers are standing in line waiting for Robin to sign their copy of her bestselling book on Robotics. Robin needs to sign each customer’s book.

Figure 7-2 A problem that requires the sequence and repetition structures
Using a Pretest Loop to Solve a Real-World Problem

• Most loops have a condition and a loop body
• Some loops require the user to enter a special **sentinel value** to end the loop
• Sentinel values should be easily distinguishable from the valid data recognized by the program
• When a loop’s condition evaluates to true, the instructions in the loop body are processed
• Otherwise, the instructions are skipped and processing continues with the first instruction after the loop
Using a Pretest Loop to Solve a Real-World Problem (cont.)

• After each processing of the loop body (iteration), the loop’s condition is reevaluated to determine if the instructions should be processed again

• A **priming read** is an instruction that appears before a loop and is used to set up the loop with an initial value entered by the user

• An **update read** is an instruction that appears within a loop that allows the user to enter a new value at each iteration of the loop
Using a Pretest Loop to Solve a Real-World Problem (cont’d.)

Figure 7-3 Problem specification and IPO chart for the Carroll Cabinets program
Problem specification
In January of each year, Miller Incorporated pays a 5% bonus to each of its salespeople. The bonus is based on the amount of sales made by the salesperson during the previous year. The payroll clerk wants a program that calculates and displays the bonus amounts for as many salespeople as needed without having to run the program more than once. Because the sales amounts entered by the payroll clerk will always be positive numbers, the payroll clerk will indicate that he is finished with the program by entering a sales amount of -1 (a negative number 1).

Input
- bonus rate (5%) sales

Processing
- Processing items: none

Output
- bonus

Algorithm:
1. enter the sales
2. repeat while (the sales are not equal to -1) 
   - calculate the bonus by multiplying the sales by the bonus rate
   - display the bonus
   - enter the sales
end repeat

Figure 7-4 Problem specification and IPO chart for the Miller Incorporated program
Using a Pretest Loop to Solve a Real-World Problem (cont’d.)

Figure 7-5 Components of the algorithm from Figure 7-4
Flowcharting a Pretest Loop

• The diamond symbol in a flowchart is the decision symbol – represents repetition structures
• A diamond representing a repetition structure contains a Boolean condition
• The condition determines whether the instructions in the loop are processed
• A diamond representing a repetition structure has one flowline leading into it and two leading out
Flowcharting a Pretest Loop (cont’d.)

- The flowlines leading out are marked “T” (true) and “F” (false)
- The “T” line points to loop body
- The “F” line points to the instructions to be processed if the loop’s condition evaluates to false
- The flowline entering the diamond and symbols and flowlines of the true path form a circle, or loop
- This distinguishes a repetition structure from a selection structure in a flowchart
Flowcharting a Pretest Loop (cont’d.)

Figure 7-6 Miller Incorporated algorithm shown in flowchart form
Flowcharting a Pretest Loop (cont’d.)

Figure 7-7 Input and output items entered in the desk-check table

<table>
<thead>
<tr>
<th>bonus rate</th>
<th>sales</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-8 First sales amount recorded in the desk-check table

<table>
<thead>
<tr>
<th>bonus rate</th>
<th>sales</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>10000</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-9 First salesperson’s bonus information recorded in the desk-check table

<table>
<thead>
<tr>
<th>bonus rate</th>
<th>sales</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>10000</td>
<td>500</td>
</tr>
</tbody>
</table>
Flowcharting a Pretest Loop (cont’d.)

Figure 7-10 Second salesperson’s information recorded in the desk-check table

Figure 7-11 Completed desk-check table
The `while` Statement

• You can use the `while` statement to code a pretest loop in C++

• Syntax is:
  ```
  while (condition)
  one statement or a statement block to be processed as long as the condition is true
  ```

• Must supply looping condition (Boolean expression)

• Condition can contain constants, variables, functions, arithmetic operators, comparison operators, and logical operators
The \texttt{while} Statement (cont’d.)

- Must also provide loop body statements, which are processed repeatedly as long as condition is true.
- If more than one statement in loop body, must be entered as a statement block (enclosed in braces).
- Can include braces even if there is only one statement in the statement block.
- Good programming practice to include a comment, such as \texttt{//end while}, to mark the end of the \texttt{while} statement.
The **while** Statement (cont’d.)

- A loop whose instructions are processed indefinitely is called an **infinite loop** or **endless loop**
- You can usually stop a program that has entered an infinite loop by pressing Ctrl+c
The **while** Statement (cont’d.)

**HOW TO** Use the `while` Statement

**Syntax**

```
while (condition)
    either one statement or a statement block to be processed as long as the condition is true
//end while
```

**Example 1**

```cpp
int age = 0;

cout << "Enter age: ";
cin >> age;
while (age > 0)
{
    cout << "You entered " << age << endl;
    cout << "Enter age: ";
cin >> age;
} //end while
```

Figure 7-12 How to use the `while` statement
The **while** Statement (cont’d.)

**Example 2**

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

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

Figure 7-12 How to use the **while** statement (cont’d.)
Figure 7-13 IPO chart information and C++ instructions for the Miller Incorporated program
The **while** Statement (cont’d.)

Figure 7-14 A sample run of the Miller Incorporated program
Using Counters and Accumulators

- Some problems require you to calculate a total or average.
- To do this, you use a counter, accumulator, or both.
- A **counter** is a numeric variable used for counting something.
- An **accumulator** is a numeric variable used for accumulating (adding together) multiple values.
- Two tasks are associated with counters and accumulators: initializing and updating.
Using Counters and Accumulators (cont’d.)

- **Initializing** means assigning a beginning value to a counter or accumulator (usually 0) – happens once, before the loop is processed.

- **Updating** (or **incrementing**) means adding a number to the value of a counter or accumulator.

- A counter is updated by a constant value (usually 1).

- An accumulator is updated by a value that varies.

- Update statements are placed in the body of a loop since they must be performed at each iteration.
The Sales Express Program

- Example problem and program solution (following slides)
- Program takes in a sequence of sales amounts from the keyboard and outputs their average
- Uses a counter to keep track of the number of sales entered and an accumulator to keep track of the total sales
- Both are initialized to 0
- The loop ends when the user enters a sentinel value (-1)
The Sales Express Program (cont’d.)

Problem specification
Sales Express wants a program that displays the average amount the company sold during the prior year. The sales manager will enter each salesperson’s sales. The program will use a counter to keep track of the number of sales amounts entered and an accumulator to total the sales amounts. When the sales manager has finished entering the sales amounts, the program will calculate the average sales amount by dividing the value stored in the accumulator by the value stored in the counter. It then will display the average sales amount on the screen. The sales manager will indicate that she is finished with the program by entering a negative number as the sales amount. If the sales manager does not enter any sales amounts, the program should display the “No sales entered” message.

Figure 7-15 Problem specification for the Sales Express program
Figure 7-15 IPO chart information and C++ instructions for the Sales Express program (part 1)
The Sales Express Program (cont’d.)

3. if (the number of sales entered is
greater than 0)
   
calculate the average sales by
dividing the total sales by the
number of sales entered

display the average sales

else
   display “No sales entered”
   message
end if

if (numSales > 0)
{
   average = totalSales /
   numSales;

   cout << "Average: $" << average
   << endl;
}
else
   cout << "No sales entered"
   << endl;
   //end if

Figure 7-15 IPO chart information and C++ instructions for the Sales Express program (part 2)
The Sales Express Program (cont’d.)

Figure 7-16 Desk-check table after the first sales amount is entered

<table>
<thead>
<tr>
<th>sales</th>
<th>numSales</th>
<th>totalSales</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>30000.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7-17 Desk-check showing the first update to the counter and accumulator variables

<table>
<thead>
<tr>
<th>sales</th>
<th>numSales</th>
<th>totalSales</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1</td>
<td>30000.0</td>
<td>0.0</td>
</tr>
<tr>
<td>30000.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Sales Express Program (cont’d.)

Figure 7-18 Desk-check table after the second update to the counter and accumulator variables

<table>
<thead>
<tr>
<th>sales</th>
<th>numSales</th>
<th>totalSales</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>30000.0</td>
<td>1</td>
<td>30000.0</td>
<td>30000.0</td>
</tr>
<tr>
<td>40000.0</td>
<td>2</td>
<td>70000.0</td>
<td>35000.0</td>
</tr>
</tbody>
</table>

Figure 7-19 Completed desk-check for the Sales Express program
The Sales Express Program (cont’d.)

Figure 7-20 First sample run of the Sales Express program

Figure 7-21 Second sample run of the Sales Express program
Counter-Controlled Pretest Loops

• Loops can be controlled using a counter rather than a sentinel value
• These are called counter-controlled loops
• Example problem and program solution (following slides)
• Counter-controlled loop is used that totals the quarterly sales from three regions
• Loop repeats three times, once for each region, using a counter to keep track
Counter-Controlled Pretest Loops (cont’d.)

Problem specification
The sales manager at Jasper Music Company wants a program that allows him to enter the quarterly sales amount made in each of three regions: Region 1, Region 2, and Region 3. The program should calculate the total quarterly sales and then display the result on the screen. The program will use a counter to ensure that the sales manager enters exactly three sales amounts. It will use an accumulator to total the sales amounts.

Figure 7-22 Problem specification for the Jasper Music Company program
IPO chart information

**Input**
region’s quarterly sales

**Processing**
number of regions (counter: 1 to 3)

**Output**
total quarterly sales (accumulator)

**Algorithm**
1. repeat while (the number of regions is less than 4)

   enter the region’s quarterly sales

   add the region’s quarterly sales to the total quarterly sales

   add 1 to the number of regions

   end repeat

2. display the total quarterly sales

   cout << "Total quarterly sales: $" << totalSales << endl;

C++ instructions

```cpp
int regionSales = 0;

int numRegions = 1;

int totalSales = 0;

while (numRegions < 4)
{
    cout << "Enter region " << numRegions << "'s quarterly sales: ";
    cin >> regionSales;
    totalSales += regionSales;
    numRegions += 1;
}
```

Figure 7-22 IPO chart information and C++ instructions for the Jasper Music Company program
Counter-Controlled Pretest Loops (cont’d.)

Figure 7-23 Desk-check table after the variable declaration statements are processed

<table>
<thead>
<tr>
<th>regionSales</th>
<th>numRegions</th>
<th>totalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 7-24 Results of processing the loop body instructions the first time

<table>
<thead>
<tr>
<th>regionSales</th>
<th>numRegions</th>
<th>totalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2500</td>
</tr>
</tbody>
</table>

Figure 7-25 Results of processing the loop body instructions the second time

<table>
<thead>
<tr>
<th>regionSales</th>
<th>numRegions</th>
<th>totalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>1</td>
<td>2500</td>
</tr>
<tr>
<td>6000</td>
<td>2</td>
<td>8500</td>
</tr>
</tbody>
</table>
Counter-Controlled Pretest Loops (cont’d.)

**Figure 7-26** Results of processing the loop body instructions the third time

**Figure 7-27** A Sample run of the Jasper Music Company program
The \texttt{for} Statement

- The \texttt{for} statement can also be used to code any pretest loop in C++
- Commonly used to code counter-controlled pretest loops (more compact than \texttt{while} in this case)
- Syntax:
  
  \begin{verbatim}
  for ([initialization]; condition; [update])
  \end{verbatim}
  
  one statement or a statement block to be processed as long as the condition is true

- \textit{Initialization} and \textit{update} arguments are optional
The \texttt{for} Statement (cont’d.)

- \textit{Initialization} argument usually creates and initializes a counter variable.
- Counter variable is local to \texttt{for} statement (can only be used inside the loop).
- \textit{Condition} argument specifies condition that must be true for the loop body to be processed.
- \textit{Condition} must be a Boolean expression:
  - May contain variables, constants, functions, arithmetic operators, comparison operators, and logical operators.
The **for** Statement (cont’d.)

- Loop ends when *condition* evaluates to false
- *Update* argument usually contains an expression that updates the counter variable
- Loop body follows the **for** clause
  - Must be placed in braces if it contains more than one statement
  - Braces can be used even if the loop body contains only one statement
- Good programming practice to place a comment, such as `//end for`, to mark the end of a **for** loop
The **for** Statement (cont’d.)

**How TO** Use the **for** Statement

Syntax

```cpp
for (([initialization]; condition; [update]))
    either one statement or a statement block to be processed as long as the
    condition is true
//end for
```

Example 1: displays the numbers 1, 2, and 3 on separate lines on the screen

```cpp
for (int x = 1; x < 4; x += 1)
    cout << x << endl;
//end for
```

You also can use `x = x + 1`

Example 2: displays the numbers 3, 2, and 1 on separate lines on the screen

```cpp
for (int x = 3; x > 0; x = x - 1)
    cout << x << endl;
//end for
```

You also can use `x = x - 1`

**Figure 7-28** How to use the **for** statement