Objectives

- In this chapter, you will:
  - Learn about repetition (looping) control structures
  - Explore how to construct and use counter-controlled, sentinel-controlled, flag-controlled, and EOF-controlled repetition structures
  - Examine break and continue statements
  - Discover how to form and use nested control structures

Objectives (cont’d.)

- In this chapter, you will (cont’d.):
  - Learn how to avoid bugs by avoiding patches
  - Learn how to debug loops

Why Is Repetition Needed?

- Repetition allows efficient use of variables
- Can input, add, and average multiple numbers using a limited number of variables
- For example, to add five numbers:
  - Declare a variable for each number, input the numbers and add the variables together
  - Create a loop that reads a number into a variable and adds it to a variable that contains the sum of the numbers
**while Looping (Repetition) Structure**

- **Syntax of the while statement:**
  ```
  while (expression)
  statement
  ```
- **statement** can be simple or compound
- **expression** acts as a decision maker and is usually a logical expression
- **statement** is called the body of the loop
- The parentheses are part of the syntax

---

**while Looping (Repetition) Structure (cont’d.)**

Consider the following C++ program segment (Assume that `i` is an int variable)

```cpp
i = 0;  //Line 1
while (i <= 50)  //Line 2
  cout << i << “ “;  //Line 3
  i = i + 2;  //Line 4
```

Sample Run:

```
0 2 4 6 8 10
```

---

**while Looping (Repetition) Structure (cont’d.)**

- `i` in Example 5-1 is called the **loop control variable (LCV)**
- **Infinite loop**: continues to execute endlessly
  - Avoided by including statements in loop body that assure the exit condition is eventually false
while Looping (Repetition) Structure (cont’d.)

**EXAMPLE 5-2**
Consider the following C++ program segment:

```c++
int i = 20;  //Line 1
while (i > 20)  //Line 2
    i--;  //Line 3
cout << i << endl;  //Line 5
```

It is easy to overlook the difference between this example and Example 5-1. In this example, in Line 1, i is set to 20. Because 1 is 20, the expression 1 < 20 in the while statement (Line 2) evaluates to false. Because initially the loop entry condition, 1 < 20, is false, the body of the while loop never executes. Hence, no values are output, and the value of i remains 20.

Case 1: Counter-Controlled while Loops

- When you know exactly how many times the statements need to be executed
  — Use a counter-controlled while loop
```c++
counter = 0;  //initialize the loop control variable
while (counter < N) //test the loop control variable
    {...
        counter++;  //update the loop control variable
    }
```

Case 2: Sentinel-Controlled while Loops

- Sentinel variable is tested in the condition
- Loop ends when sentinel is encountered
```c++
cin >> variable;  //initialize the loop control variable
while (variable != sentinel) //test the loop control variable
    {...
        cin >> variable;  //update the loop control variable
    }
```

Example 5-5: Telephone Digits

- Example 5-5 provides an example of a sentinel-controlled loop
- The program converts uppercase letters to their corresponding telephone digit
Case 3: Flag-Controlled while Loops

- **Flag-controlled while loop:** uses a bool variable to control the loop
  
  ```
  found = false; //initialize the loop control variable
  while (!found) //test the loop control variable
  {
    ...
    if (expression)
      found = true; //update the loop control variable
    ...
  }
  ```

Number Guessing Game

- Example 5-6 implements a number guessing game using a flag-controlled while loop
- Uses the function `rand` of the header file `cstdlib` to generate a random number
  - `rand()` returns an int value between 0 and 32767
  - To convert to an integer >= 0 and < 100:
    - `rand() % 100`

Case 4: EOF-Controlled while Loops

- **End-of-file (EOF)-controlled while loop:** when it is difficult to select a sentinel value
- The logical value returned by `cin` can determine if there is no more input
**eof Function**

- The function `eof` can determine the end of file status.
- `eof` is a member of data type `istream`.
- Syntax for the function `eof`:
  ```cpp
  istreamVar.eof()
  ```
- `istreamVar` is an input stream variable, such as `cin`.

**More on Expressions in while Statements**

- The expression in a `while` statement can be complex.
  
  Example:
  ```cpp
  while ((noOfGuesses < 5) && (!isGuessed))
  {
    ...
  }
  ```

**Programming Example: Fibonacci Number**

- Consider the following sequence of numbers:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- Called the Fibonacci sequence.
- Given the first two numbers of the sequence (say, `a1` and `a2`):
  - `n`th number `a_n`, `n >= 3`, of this sequence is given by:
    ```cpp
    a_n = a_{n-1} + a_{n-2}
    ```

**Programming Example: Fibonacci Number (cont’d.)**

- Fibonacci sequence:
  - `n`th Fibonacci number
    - `a2` = 1
    - `a1` = 1
  - Determine the `n`th number `a_n`, `n >= 3`
Programming Example: Fibonacci Number (cont’d.)

• Suppose \(a_2 = 6\) and \(a_1 = 3\)
  
  \[- \quad a_3 = a_2 + a_1 = 6 + 3 = 9\]
  
  \[- \quad a_4 = a_3 + a_2 = 9 + 6 = 15\]

• Write a program that determines the \(n^{th}\) Fibonacci number, given the first two numbers

Programming Example: Input and Output

• Input: first two Fibonacci numbers and the desired Fibonacci number
  
  • Output: \(n^{th}\) Fibonacci number

Programming Example: Problem Analysis and Algorithm Design

• Algorithm:
  
  – Get the first two Fibonacci numbers
  – Get the desired Fibonacci number
    
    • Get the position, \(n\), of the number in the sequence
  – Calculate the next Fibonacci number
    
    • Add the previous two elements of the sequence
  – Repeat Step 3 until the \(n^{th}\) Fibonacci number is found
  – Output the \(n^{th}\) Fibonacci number

Programming Example: Variables

\[
\begin{align*}
\text{int } & \text{previous1; } //\text{variable to store the first Fibonacci number} \\
\text{int } & \text{previous2; } //\text{variable to store the second Fibonacci number} \\
\text{int } & \text{current; } //\text{variable to store the current Fibonacci number} \\
\text{int } & \text{counter; } //\text{loop control variable} \\
\text{int } & \text{nthFibonacci; } //\text{variable to store the desired Fibonacci number}
\end{align*}
\]
Programming Example: Main Algorithm

- Prompt the user for the first two numbers—that is, `previous1` and `previous2`
- Read (input) the first two numbers into `previous1` and `previous2`
- Output the first two Fibonacci numbers
- Prompt the user for the position of the desired Fibonacci number

Programming Example: Main Algorithm (cont’d.)

- Read the position of the desired Fibonacci number into `nthFibonacci`
  - if (`nthFibonacci` == 1)
    The desired Fibonacci number is the first Fibonacci number; copy the value of `previous1` into `current`
  - else if (`nthFibonacci` == 2)
    The desired Fibonacci number is the second Fibonacci number; copy the value of `previous2` into `current`
  - else calculate the desired Fibonacci number as follows:
    - Start by determining the third Fibonacci number
    - Initialize `counter` to 3 to keep track of the calculated Fibonacci numbers.
    - Calculate the next Fibonacci number, as follows:
      ```
      current = previous2 + previous1;
      ```
    - Assign the value of `previous2` to `previous1`
    - Assign the value of `current` to `previous2`
    - Increment `counter`
    - Repeat until Fibonacci number is calculated:
      ```
      while (counter <= nthFibonacci) {
          current = previous2 + previous1;
          previous1 = previous2;
          previous2 = current;
          counter++;
      }
      ```
Programming Example: Main Algorithm (cont’d.)

• Output the \texttt{n}thFibonacci number, which is current

\textbf{for Looping (Repetition) Structure (cont’d.)}

• \texttt{for} loop: called a counted or indexed \texttt{for} loop
• Syntax of the \texttt{for} statement:

  \begin{Verbatim}
  \texttt{for} (initial statement; loop condition; update statement) \\
  \hspace{1cm} \texttt{statement}
  \end{Verbatim}

• The initial statement, loop condition, and update statement are called \texttt{for} loop control statements

\textbf{for Looping (Repetition) Structure (cont’d.)}

\textbf{EXAMPLE 5-9}

The following \texttt{for} loop print the first 10 nonnegative integers:

\begin{verbatim}
for (i = 0; i < 10; i++)
  cout \ll i \ll \text{" \text{"} }
time
\end{verbatim}

The initial statement, \texttt{i = 0}, initializes the \texttt{i} variable \texttt{i} to 0. Next, the loop condition, \texttt{i < 10} is evaluated. Because \texttt{0 < 10} is true, the print statement executes and outputs 0. The update statement, \texttt{i++}, then executes, which sets the value of \texttt{i} to \texttt{1}. Once again, the loop condition is evaluated, which is still true, and so on. When \texttt{i} becomes 10, the loop condition evaluates to false, the \texttt{for} loop terminates, and the statement following the \texttt{for} loop executes.
for Looping (Repetition) Structure (cont’d.)

1. The following for loop outputs "Hello!" and a star (on separate lines) five times:
   ```
   for (i = 1; i <= 5; i++)
   
   cout << "Hello!" << endl;
   cout << '*' << endl;
   
   3.
   ```

2. Consider the following for loop:
   ```
   for (i = 1; i <= 5; i++)
   
   cout << "Hello!" << endl;
   cout << '*' << endl;
   
   This loop outputs "Hello!" five times and the star only once.
   ```

for Looping (Repetition) Structure (cont’d.)

• The following is a semantic error:

```plaintext
for (i = 0; i < 1; i++)
```

• The following is a legal (but infinite) for loop:

```plaintext
for (;;) cout << "Hello!" << endl;
```
do...while Looping (Repetition) Structure

• Syntax of a do...while loop:

```
do statement
while (expression);
```

• The statement executes first, and then the expression is evaluated
  – As long as expression is true, loop continues
• To avoid an infinite loop, body must contain a statement that makes the expression false

do...while Looping (Repetition) Structure (cont’d.)

• The statement can be simple or compound
• Loop always iterates at least once

Example 5.18

```
i = 10
while (i <= 20)
    if (i % 2 == 0)
        printf("%d is even\n", i);
    i = i + 2;
```

The output of the code is:
10 12 14 16 18
After 20 is output, the statement:
```
i = i + 2;
```
changes the value of i to 22 and so i <= 20 becomes false, which halts the loop.
do...while Looping (Repetition) Structure (cont’d.)

Choosing the Right Looping Structure

- All three loops have their place in C++
  - If you know or can determine in advance the number of repetitions needed, the for loop is the correct choice
  - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a while loop
  - If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a do...while loop

break and continue Statements

- break and continue alter the flow of control
- break statement is used for two purposes:
  - To exit early from a loop
  - Can eliminate the use of certain (flag) variables
  - To skip the remainder of a switch structure
- After break executes, the program continues with the first statement after the structure

break and continue Statements (cont’d.)

- continue is used in while, for, and do...while structures
- When executed in a loop
  - It skips remaining statements and proceeds with the next iteration of the loop
Nested Control Structures

• To create the following pattern:
  *
  **
  ***
  ****
  *****

• We can use the following code:
  ```cpp
  for (i = 1; i <= 5; i++)
  {
    for (j = 1; j <= i; j++)
      cout << '*';
    cout << endl;
  }
  ```

Nested Control Structures (cont’d.)

• What is the result if we replace the first for statement with this?
  ```cpp
  for (i = 5; i >= 1; i--)
  ```

• Answer:
  ****
  ***
  **
  *

Avoiding Bugs by Avoiding Patches

• Software patch
  – Piece of code written on top of an existing piece of code
  – Intended to fix a bug in the original code
• Some programmers address the symptom of the problem by adding a software patch
• Should instead resolve underlying issue

Debugging Loops

• Loops are harder to debug than sequence and selection structures
• Use loop invariant
  – Set of statements that remains true each time the loop body is executed
• Most common error associated with loops is off-by-one
Summary

- C++ has three looping (repetition) structures:
  - while, for, and do...while
  - while, for, and do are reserved words
  - while and for loops are called pretest loops
  - do...while loop is called a posttest loop
  - while and for may not execute at all, but
do...while always executes at least once

Summary (cont’d.)

- while: expression is the decision maker,
  and statement is the body of the loop
- A while loop can be:
  - Counter-controlled
  - Sentinel-controlled
  - EOF-controlled
- In the Windows console environment, the
  end-of-file marker is entered using Ctrl+z

Summary (cont’d.)

- for loop: simplifies the writing of a counter-
  controlled while loop
  - Putting a semicolon at the end of the for loop is
    a semantic error
- Executing a break statement in the body of a
  loop immediately terminates the loop
- Executing a continue statement in the body of a
  loop skips to the next iteration