Chapter 5: Repetition Statements
In this chapter, you will learn about:

- Basic loop structures
- while loops
- Interactive while loops
- for loops
- Loop programming techniques
Objectives (continued)

- Nested loops
- do while loops
- Common programming errors
Basic Loop Structures

- Repetition structure has four required elements:
  - Repetition statement
  - Condition to be evaluated
  - Initial value for the condition
  - Loop termination

- Repetition statements include:
  - `while`
  - `for`
  - `do while`
Basic Loop Structures (continued)

• The condition can be tested
  – At the beginning: **Pretest** or entrance-controlled loop
  – At the end: **Posttest** or exit-controlled loop

• Something in the loop body must cause the condition to change, to avoid an **infinite loop**, which never terminates
• Pretest loop: Condition is tested first; if false, statements in the loop body are never executed

• **while** and **for** loops are pretest loops

---

**Figure 5.1** A pretest loop
• Posttest loop: Condition is tested after the loop body statements are executed; loop body always executes at least once

• **do while** is a posttest loop

**Figure 5.2 A posttest loop**
Fixed-Count Versus Variable-Condition Loops

- **Fixed-count loop**: Loop is processed for a fixed number of repetitions
- **Variable-condition loop**: Number of repetitions depends on the value of a variable
while Loops

• **while statement** is used to create a **while loop**
  – Syntax:
    
    ```
    while (expression)
    statement;
    ```

• Statements following the expressions are executed as long as the expression condition remains true (evaluates to a non-zero value)
while Loops (continued)

Program 5.1

#include <iostream>
using namespace std;

int main()
{
    int count;

    count = 1; // initialize count
    while (count <= 10)
    {
        cout << count << " ";
        count++;
        // increment count
    }

    return 0;
}
Interactive while Loops

- Combining interactive data entry with the `while` statement provides for repetitive entry and accumulation of totals.
Interactive while Loops (cont’d)

Figure 5.7 Accumulation flow of control
• **Sentinel**: A data value used to signal either the start or end of a data series
  
  • Use a sentinel when you don’t know how many values need to be entered
break and continue Statements

- **break statement**
  - Forces an immediate break, or exit, from `switch`, `while`, `for`, and `do-while` statements
  - Violates pure structured programming, but is useful for breaking out of loops when an unusual condition is detected
Example of a `break` statement:

```cpp
while (count <= 10)
{
    cout << "Enter a number: ";
    cin >> num;
    if (num > 76)
    {
        cout << "You lose!\n";
        break;        // break out of the loop
    }
    else
    {
        cout << "Keep on trucking!\n";
        count++;    
    }
// break jumps to here
```
• A `continue` statement where invalid grades are ignored, and only valid grades are added to the total:

```cpp
while (count < 30) {
    cout << "Enter a grade: ";
    cin >> grade
    if(grade < 0 || grade > 100) continue;
    total = total + grade;
    count++;
}
```
break and continue Statements (cont’d)

• **continue statement**
  – Applies to **while**, **do-while**, and **for** statements; causes the next iteration of the loop to begin immediately
  – Useful for skipping over data that should not be processed in this iteration, while staying within the loop
The Null Statement

- **Null statement**
  - Semicolon with nothing preceding it
    - ;
  - Do-nothing statement required for syntax purposes only
for Loops

• **for** statement: A loop with a fixed count condition that handles alteration of the condition
  – Syntax:
    
    ```
    for (initializing list; expression; altering list)
    
    statement;
    ```

• **Initializing list**: Sets the starting value of a counter

• **Expression**: Contains the maximum or minimum value the counter can have; determines when the loop is finished
for Loops (continued)

- **Altering list**: Provides the increment value that is added or subtracted from the counter in each iteration of the loop
- If initializing list is missing, the counter initial value must be provided prior to entering the for loop
- If altering list is missing, the counter must be altered in the loop body
- Omitting the expression will result in an infinite loop
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
    const int MAXCOUNT = 5;
    int count;

    cout << "NUMBER SQUARE ROOT\n";
    cout << "------ ---------\n";

    cout << setiosflags(ios::showpoint);
    for (count = 1; count <= MAXCOUNT; count++)
        cout << setw(4) << count
            << setw(15) << sqrt(double(count)) << endl;

    return 0;
}
for Loops (cont’d)

Figure 5.10 for loop flowchart.
A Closer Look: Loop Programming Techniques

• These techniques are suitable for pretest loops (for and while):
  – Interactive input within a loop
    • Includes a cin statement within a while or for loop
  – Selection within a loop
    • Using a for or while loop to cycle through a set of values to select those values that meet some criteria
A Closer Look: Loop Programming Techniques (continued)

Program 5.13

```cpp
#include <iostream>
using namespace std;

// This program computes the positive and negative sums of a set
// of MAXNUMS user-entered numbers
int main()
{
    const int MAXNUMS = 5;
    int i;
    double usenum, positiveSum, negativeSum;
```
A Closer Look: Loop Programming Techniques (continued)

```cpp
positiveSum = 0; // this initialization can be done in the declaration
negativeSum = 0; // this initialization can be done in the declaration
for (i = 1; i <= MAXNUMS; i++)
{
    cout << "Enter a number (positive or negative) : ";
    cin >> usenum;
    if (usenum > 0)
        positiveSum = positiveSum + usenum;
    else
        negativeSum = negativeSum + usenum;
}

cout << "The positive total is " << positiveSum << endl;
cout << "The negative total is " << negativeSum << endl;

return 0;
```
• Evaluating functions of one variable
  – Used for functions that must be evaluated over a range of values
  – Noninteger increment values can be used
A Closer Look: Loop Programming Techniques (continued)

Program 5.14

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
    int x, y;

    cout << "x value   y value\n" << "-------    -------\n";

    for (x = 2; x <= 6; x++)
    {
        y = 10 * pow(x, 2.0) + 3 * x - 2;
        cout << setw(4) << x
             << setw(11) << y << endl;
    }

    return 0;
}
```
• **Interactive loop control**
  – Variable is used to control the loop repetitions
  – Provides more flexibility at run-time

• **Random numbers and simulation**
  – Pseudorandom generator used for simulators
  – C++ functions: `rand()`; `srand()`
A Closer Look: Loop Programming Techniques (continued)

Program 5.16

```c++
#include <iostream>
#include <iomanip>
using namespace std;

// This program displays a table of numbers with their squares and cubes, starting from the number 1. The final number in the table is input by the user.

int main()
{
    int num, final;

    cout << "Enter the final number for the table: ";
    cin >> final;
    cout << "NUMBER SQUARE CUBE\n";
    cout << "------- ------ ----\n";

    for (num = 1; num <= final; num++)
        cout << setw(3) << num
             << setw(8) << num * num
             << setw(7) << num * num * num << endl;

    return 0;
}
```
A Closer Look: Loop Programming Techniques (continued)

Program 5.17

```cpp
#include <iostream>
#include <cmath>
#include <ctime>
using namespace std;

// This program generates 10 pseudorandom numbers
// with C++'s rand() function

int main()
{
    const int NUMBERS = 10;
    double randvalue;
    int i;

    srand(time(NULL));  // generates the first seed value
    for (i = 1; i <= NUMBERS; i++)
    {
        randvalue = rand();
        cout << randvalue << endl;
    }

    return 0;
}
```
Nested Loops

- **Nested loop**: A loop contained within another loop
  - All statements of the inner loop must be completely contained within the outer loop; no overlap allowed
  - Different variables must be used to control each loop
  - For each single iteration of the outer loop, the inner loop runs through all of its iterations
Nested Loops (continued)

Figure 5.12 For each \( i, j \) loops.
#include <iostream>
using namespace std;

int main()
{
    const int MAXI = 5;
    const int MAXJ = 4;
    int i, j;

    for (i = 1; i <= MAXI; i++)    // start of outer loop
    {
        cout << "\nI is now " << i << endl;    //
        for (j = 1; j <= MAXJ; j++)    // start of inner loop
            cout << " J = " << j;    // end of inner loop
        cout << endl;
    }
    return 0;
}
do while Loops

• **do while** loop is a posttest loop
  – Loop continues while the condition is true
  – Condition is tested at the end of the loop
  – Syntax:
    
    ```
    do
    statement;
    while (expression);
    ```

• All statements are executed at least once in a posttest loop
do while Loops

Figure 5.13 The do while loop structure.
Figure 5.14 The do statement’s flow of control.
Validity Checks

- Useful in filtering user-entered input and providing data validation checks

```cpp
    do
    {
        cout << "Enter an identification number: ";
        cin >> id_num;
    }
    while (id_num < 1000 || id_num > 1999);
```

- Can enhance with `if-else` statement
Common Programming Errors

• Making the “off by one” error: loop executes one too many or one too few times
• Using the assignment operator (=) instead of the equality comparison operator (==) in the condition expression
• Testing for equality with floating-point or double-precision operands; use an epsilon value instead
Common Programming Errors (continued)

- Placing a semicolon at the end of the `for` clause, which produces a null loop body
- Using commas instead of semicolons to separate items in the `for` statement
- Changing the value of the control variable
- Omitting the final semicolon in a `do` statement
• Loop: A section of repeating code, whose repetitions are controlled by testing a condition

• Three types of loops:
  – while
  – for
  – do while

• Pretest loop: Condition is tested at beginning of loop; loop body may not ever execute; ex., while, for loops
Summary (continued)

- Posttest loop: Condition is tested at end of loop; loop body executes at least once; ex., `do while`
- Fixed-count loop: Number of repetitions is set in the loop condition
- Variable-condition loop: Number of repetitions is controlled by the value of a variable