Chapter 5: Loops and Files

The Increment and Decrement Operators

++ is the increment operator.
It adds one to a variable.

val++; is the same as val = val + 1;

++ can be used before (prefix) or after (postfix) a variable:
++val; val++;

The Increment and Decrement Operators

-- is the decrement operator.
It subtracts one from a variable.

val--; is the same as val = val - 1;

-- can be also used before (prefix) or after (postfix) a variable:
--val; val--;

Increment and Decrement Operators in Program 5-1

Program 5-1

```cpp
int main()
{
    int var = 5;
    cout << "The variable var is " << var << endl;
    var = var + 1;
    cout << "The variable var is " << var << endl;
    return 0;
}
```

Program Output

The variable var is 5
I will increment var.
How the variable var is 6
I will increment var again.
How the variable var is 7
I will increment var again.
How the variable var is 8

Prefix vs. Postfix

++ and -- operators can be used in complex statements and expressions

In prefix mode (++var, --var) the operator increments or decrements, then returns the value of the variable

In postfix mode (var++, var--) the operator returns the value of the variable, then increments or decrements
Prefix vs. Postfix - Examples

```cpp
int num, val = 12;
cout << val++; // displays 12,
    // val is now 13;
cout << ++val; // sets val to 14,
    // then displays it
num = --val; // sets val to 13,
    // stores 13 in num
num = val--; // stores 13 in num,
    // sets val to 12
```

Notes on Increment and Decrement

- Can be used in expressions:
  ```cpp
  result = num1++ + --num2;
  ```
- Must be applied to something that has a location in memory. Cannot have:
  ```cpp
  result = (num1 + num2)++;
  ```
- Can be used in relational expressions:
  ```cpp
  if (++num > limit)
      pre- and post-operations will cause different comparisons
  ```

Introduction to Loops: The while Loop

- **Loop**: a control structure that causes a statement or statements to repeat
- General format of the while loop:
  ```cpp
  while (expression)
      statement;
  ```
  *statement*; can also be a block of statements enclosed in `{ }`

The while Loop – How It Works

```cpp
while (expression)
    statement;
```
- **expression** is evaluated
  - if true, then **statement** is executed, and **expression** is evaluated again
  - if false, then the loop is finished and program statements following **statement** execute

The Logic of a while Loop

![Diagram of a while loop with conditions and flow](diagram.png)
The while loop in Program 5-3

```cpp
// This program demonstrates a simple while loop.
int main()
{
    int number = 6;
    while (number <= 5)
    {
        cout << "Hello\n";
        number++;
    }
    return 0;
}
```

Flowchart of the while Loop in Program 5-3

The while Loop is a Pretest Loop

*expression* is evaluated before the loop executes. The following loop will never execute:

```cpp
int number = 6;
while (number <= 5)
{
    cout << "Hello\n";
    number++;
}
```

How the while Loop in Program 5-3 Lines 9 through 13 Works

- **Test this expression:**
  ```cpp
  while (number <= 5) {
    cout << "Hello\n";
    number++;
  }
  ```
- **If the expression is true, perform these statements.**
- **After executing the body of the loop, start over.**

Watch Out for Infinite Loops

- The loop must contain code to make *expression* become false
- Otherwise, the loop will have no way of stopping
- Such a loop is called an infinite loop, because it will repeat an infinite number of times

Example of an Infinite Loop

```cpp
int number = 1;
while (number <= 5)
{
    cout << "Hello\n";
}
```
Using the while Loop for Input Validation

- Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.

- The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.

Using the while Loop for Input Validation

• Here's the general approach, in pseudocode:

  Read an item of input.
  While the input is invalid
    Display an error message.
    Read the input again.
  End While

Input Validation Example

```cpp
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
  cout << "Invalid Entry!"
      << "Enter a number less than 10: ";
  cin >> number;
}
```

Flowchart for Input Validation

Input Validation in Program 5-5

```cpp
// Get the number of players per team.
int numPlayers = 0;
cin >> numPlayers;

// Validate the input.
while (numPlayers < 10 || numPlayers > 20)
{
  cout << "Invalid Entry!
      Enter a number between 10 and 20: ";
  cin >> numPlayers;
}

int totalPlayers = numPlayers * numTeams;
if (totalPlayers < 10 || totalPlayers > 40)
{
  cout << "Invalid Entry!
      Enter a number between 10 and 40: ";
  cin >> totalPlayers;
}
```
5.4

Counters

- **Counter**: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the *loop control variable*)
- Must be initialized before entering loop

A Counter Variable Controls the Loop in Program 5-6

```c
/* This program displays a list of numbers and
their squares.
* @include <iostream>
* using namespace std;

int main()
{
  const int MAX_NUMBER = 10; // Maximum number to square

  int num = MAX_NUMBER; // Counter
  cout << "Number Squared: ";
  cout << "---------------------" << endl;

  while (num <= MAX_NUMBER)
  {
    cout << num << "\n" << (num * num) << endl;
    num++; // Increment the counter.
  }

  return 0;
}
```

The do-while Loop

- **do-while**: a posttest loop – execute the loop, then test the expression
- General Format:
  ```c
  do
  statement; // or block in {}
  while (expression);
  ```
- Note that a semicolon is required after `(expression)`
The Logic of a do-while Loop

An Example do-while Loop

```cpp
int x = 1;
do {
  cout << x << endl;
} while (x < 0);
```

Although the test expression is false, this loop will execute one time because do-while is a posttest loop.

A do-while Loop in Program 5-7

```cpp
int x = 1;
do {
  cout << x << endl;
} while (x < 0);
```

A do-while Loop in Program 5-7

```cpp
int x = 1;
do {
  cout << x << endl;
} while (x < 0);
```

A do-while Loop in Program 5-7

```cpp
int x = 1;
do {
  cout << x << endl;
} while (x < 0);
```

do-while Loop Notes

- Loop always executes at least once
- Execution continues as long as `expression` is true, stops repetition when `expression` becomes false
- Useful in menu-driven programs to bring user back to menu to make another choice (see Program 5-8 on pages 245-246)

5.6

The for Loop
The for Loop

- Useful for counter-controlled loop
- General Format:
  ```
  for(initialization; test; update)
  statement; // or block in { }
  ```
- No semicolon after the update expression or after the

for Loop - Mechanics

for(initialization; test; update)
statement; // or block in { }

1) Perform initialization
2) Evaluate test expression
   - If true, execute statement
   - If false, terminate loop execution
3) Execute update, then re-evaluate test expression

for Loop - Example

```cpp
int count;
for (count = 1; count <= 5; count++)
  cout << "Hello" << endl;
```

A Closer Look at the Previous Example

1. Perform the initialization expression.
2. Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.
3. Execute the body of the loop.

Step 3: Execute the body of the loop.

```
for (count = 1; count <= 5; count++)
  cout << "Hello" << endl;
```

Step 4: Perform the update expression, then go back to Step 2.

A for Loop in Program 5-9

```cpp
// This program displays the numbers 1 through 10 and their squares.
// their squares...
#include <iostream>
using namespace std;

int main()
{
  // Initialize the variable count
  cout << "Enter number: " << count << " \n";
  // Read the number from the user
  cin >> number;
  // Display the square of the number
  cout << number << " squared is " << number * number << " \n";
  // Display the square of the number
  return 0;
}
```

Continued...
A for Loop in Program 5-9

Program Output

<table>
<thead>
<tr>
<th>Number</th>
<th>Number Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

A Closer Look at Lines 15 through 16 in Program 5-9

Step 1: Perform the initialization expression.
Step 2: Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.
Step 4: Perform the update expression, then go back to Step 2.

Flowchart for Lines 15 through 16 in Program 5-9

When to Use the for Loop

- In any situation that clearly requires
- an initialization
- a false condition to stop the loop
- an update to occur at the end of each iteration

The for Loop is a Pretest Loop

- The for loop tests its test expression before each iteration, so it is a pretest loop.

```cpp
for (count = 11; count <= 10; count++)
    cout << "Hello" << endl;
```

for Loop - Modifications

- You can have multiple statements in the initialization expression. Separate the statements with a comma:

```cpp
int x, y;
for (x=1, y=1; x <= 5; x++)
    cout << x << " plus " << y << " equals " << (x+y) << endl;
```
for Loop - Modifications

- You can also have multiple statements in the test expression. Separate the statements with a comma:

```c
int x, y;
for (x=1, y=1; x <= 5; x++, y++)
{
    cout << x << " plus " << y << " equals " << (x+y) << endl;
}
```

- You can omit the initialization expression if it has already been done:

```c
int sum = 0, num = 1;
for (; num <= 10; num++)
    sum += num;
```

- You can declare variables in the initialization expression:

```c
int sum = 0;
for (int num=0; num <= 10; num++)
    sum += num;
```

The scope of the variable num is the for loop.

5.7 Keeping a Running Total

- Running total: accumulated sum of numbers from each repetition of loop
- Accumulator: variable that holds running total

```c
int sum=0, num=1; // sum is the while (num <= 10) // accumulator
{
    sum += num;
    num++;
}
cout << "Sum of numbers 1 – 10 is" << sum << endl;
```
5.8 Sentinels

- **sentinel**: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, e.g., -999 for a test score
- Used to terminate input when user may not know how many values will be entered
5.9  Deciding Which Loop to Use

- **The while loop** is a conditional pretest loop
  - Iterates as long as a certain condition exists
  - Validating input
  - Reading lists of data terminated by a sentinel

- **The do-while loop** is a conditional posttest loop
  - Always iterates at least once
  - Repeating a menu

- **The for loop** is a pretest loop
  - Built-in expressions for initializing, testing, and updating
  - Situations where the exact number of iterations is known

5.10  Nested Loops

- A **nested loop** is a loop inside the body of another loop

- **Inner (inside), outer (outside) loops:**

  ```cpp
  for (row=1; row<=3; row++) //outer
    for (col=1; col<=3; col++)//inner
      cout << row * col << endl;
  ```

- **Nested for Loop in Program 5-14**

  ```cpp
  // Determine each student's average score.
  for (list student = list student; student++)
  {
    total = 0; // Initialize the accumulator.
    for (list test = list test; test++)
    {
      double score;
      test << "Enter score " << test << " for ",
      test << "student " << student << ": ";
      cin >> score;
      total += score;
    } //Inner Loop
    average = total / numTests;
    cout << "The average score for student " << student;
    cout << " is " << average << ", the average is: ", average;
  } //Outer Loop
  ```

- **Nested Loops - Notes**

  - Inner loop goes through all repetitions for each repetition of outer loop
  - Inner loop repetitions complete sooner than outer loop
  - Total number of repetitions for inner loop is product of number of repetitions of the two loops.
5.11 Using Files for Data Storage

Can use files instead of keyboard, monitor screen for program input, output

Allows data to be retained between program runs

Steps:
- Open the file
- Use the file (read from, write to, or both)
- Close the file

Hey!

Files: What is Needed

- Use fstream header file for file access
- File stream types:
  - ifstream for input from a file
  - ofstream for output to a file
  - fstream for input from or output to a file
- Define file stream objects:
  - ifstream infile;
  - ofstream outfile;

Opening Files

- Create a link between file name (outside the program) and file stream object (inside the program)
- Use the open member function:
  - infile.open("inventory.dat");
  - outfile.open("report.txt");
- Filename may include drive, path info.
- Output file will be created if necessary; existing file will be erased first
- Input file must exist for open to work

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:
  - infile.open("test.txt");
  - if (!infile)
    - cout << "File open failure!";
- Can also use the fail member function

Using Files

- Can use output file object and << to send data to a file:
  - outfile << "Inventory report";
- Can use input file object and >> to copy data from file to variables:
  - infile >> partNum;
  - infile >> qtyInStock >> qtyOnOrder;
Using Loops to Process Files

- The stream extraction operator `>>` returns true when a value was successfully read, false otherwise.
- Can be tested in a while loop to continue execution as long as values are read from the file:
  ```cpp
  while (inputFile >> number) ...
  ```

Closing Files

- Use the `close` member function:
  ```cpp
  infile.close();
  outfile.close();
  ```
- Don't wait for operating system to close files at program end:
  - may be limit on number of open files
  - may be buffered output data waiting to send to file

Letting the User Specify a Filename

- In many cases, you will want the user to specify the name of a file for the program to open.
- In C++ 11, you can pass a `string` object as an argument to a file stream object's open member function.

Letting the User Specify a Filename in Program 5-24

- In Program 5-24, you can pass a string object as an argument to a file stream object's open member function.

Using the `c_str` Member Function in Older Versions of C++

- Prior to C++ 11, the open member function requires that you pass the name of the file as a null-terminated string, which is also known as a C-string.
- *String literals are stored* in memory as null-terminated C-strings, but *string objects are not.*
Using the `c_str` Member Function in Older Versions of C++

- String objects have a member function named `c_str`.
- It returns the contents of the object formatted as a null-terminated C-string.
- Here is the general format of how you call the `c_str` function:
  ```c
  stringObject.c_str()
  ```
- Line 18 in Program 5-24 could be rewritten in the following manner:
  ```c
  inputFile.open(filename.c_str());
  ```

Breaking Out of a Loop

- Can use `break` to terminate execution of a loop.
- Use sparingly if at all – makes code harder to understand and debug.
- When used in an inner loop, terminates that loop only and goes back to outer loop.

The `continue` Statement

- Can use `continue` to go to end of loop and prepare for next repetition.
- `while`, `do-while` loops: go to test, repeat loop if test passes.
- `for` loop: perform update step, then test, then repeat loop if test passes.
- Use sparingly – like `break`, can make program logic hard to follow.