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--;

Prefix vs. Postfix

++ and -- operators can be used in complex statements and expressions
In prefix mode (++val, -val) the operator increments or decrements, then returns the value of the variable
In postfix mode (val++, val--) the operator returns the value of the variable, then increments or decrements
Prefix vs. Postfix - Examples

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

5.2

Introduction to Loops: The while Loop

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

The while Loop – How It Works

```
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

- Expression
- True
  - Statement(s)
- False
The while loop in Program 5-3

```cpp
Program 5-3
// This program demonstrates a simple while loop.
// Assume we want to print "Hello,
// 1", "Hello,
// 2", ..., "Hello,
// 5".
int number = 1;
while (number <= 5)
{
    cout << "Hello, \n" << number; // Make sure there's a newline after every output.
    number++;
}
```

Output:
```
Hello,
1
Hello,
2
Hello,
3
Hello,
4
Hello,
5
```

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

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

Test this expression:
```cpp
while (number <= 5) {
    cout << "Hello\n";
    number++;
}
```

If the expression is true, perform these statements.

A flowchart of the while Loop in Program 5-3

Flowchart picture showing logic:
- `number` is true?
  - `Print "Hello"`
  - `Add 1 to number` (and loop back)
- `Number is false`?

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++;
}
```

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";
}
```
5.3

Using the \textit{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.

\textbf{Input Validation Example}

\begin{verbatim}
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!
" << "Enter a number less than 10: ";
cin >> number;
}
\end{verbatim}
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++
// Program 5-6

1: 1 // This program displays a list of numbers and
2: 2 their squares.
3: 3 #include <iostream>
4: 4 using namespace std;
5: 5 6 int main()
7: 7 { // Start of function
8: 8 1 // count <= MAX_NUMBER = 1, // starting number to square
9: 9 2 int MAX_NUMBER = 10; // Maximum number to square
10: 10 3 int num = MAX_NUMBER; // Counter
11: 11 4 cout << "Number Number Squared\n";
12: 12 5 cout << "-------------------\n";
13: 13 6 while (num <= MAX_NUMBER) { // do-while loop
14: 14 7 cout << num << num << "\n" << num * num << endl;
15: 15 8 num++; // Increment the counter.
16: 16 9 }
17: 17 10 return 0;
18: 18 11}
19: 19
```

---

**A Counter Variable Controls the Loop in Program 5-6**

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>

---

5.5

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

A do-while Loop in Program 5-7

5.6

The for Loop
The for Loop

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

for Loop - Mechanics

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

Flowchart for the Previous Example

A for Loop in Program 5-9

Program 5-9

```cpp
1 // This program displays the numbers 1 through 10 and
2 // their squares.
3 #include <iostream>
4 using namespace std;
5 int main()
6 {
7  for (int i = 1; i <= 10; i++)
8      cout << i << " squared is " << i * i << endl;
9  return 0;
10 }
```

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

<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

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.
- The following loop will never iterate:

  ```
  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:

  ```
  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:

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

**Test Expression**

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

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

**for Loop - Modifications**

- You can declare variables in the *initialization* expression:

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

The scope of the variable `num` is the for loop.

---

**Keeping a Running Total**

- **running total**: accumulated sum of numbers from each repetition of loop
- **accumulator**: variable that holds running total

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

---

**Logic for Keeping a Running Total**

- Set accumulator to 0
- Is there a number to read?
  - Yes (That)
  - No (Stop)
- Read the number
- Add the number to the accumulator
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 exits
  - 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 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.