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 \( val = val + 1; \)
- **--** is the decrement operator.
  - It subtracts one from a variable.
  - \( val--; \) is the same as \( val = val - 1; \)
- **++** can be used before (prefix) or after (postfix) a variable:
  - \( ++val; \)
  - \( val++; \)
- **--** 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

```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 showing the logic of a while loop with decision points for True and False]
The while loop in Program 5-3

Program 5-3

// This program demonstrates a simple while loop.
#include <iostream>
using namespace std;

int main()
{
  int number = 1;
  while (number <= 5)
  {
    cout << "Hello\n";
    number++;
  }
  return 0;
}

Program Output:
Hello
Hello
Hello
Hello
That's all!

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

The while Loop is a Pretest Loop

The 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";
}
```
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.

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

```
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

![Flowchart](image)

Input Validation in Program 5-5

```
// Get the number of players per team.
short num_players = get_num_players_per_team();
while (num_players > MAX_PLAYERS)
{
  // Validate the input.
  while (num_players > MAX_PLAYERS) // num_players > MAX_PLAYERS
  {
    // Display the error.
    cout << "You should have at least " << num_players << " players per team."
  }
}
```
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

**Program 5.6**

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

int main()
{
    const int MIN_NUMBER = 1; // starting number to square
    const int MAX_NUMBER = 10; // maximum number to square

    cout << "Counter Numbers Squared."
    for(int num = MIN_NUMBER; num <= MAX_NUMBER; num++)
    {
        cout << num << " " << num * num << endl;
    }
    return 0;
}
```

### The do-while Loop

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

Statement(s)

Expression

True

False

An Example do-while Loop

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

Program 5-7:

```cpp
// This program averages 3 exam scores. It prompts an
// exam score, asks if more exams will be given.
// Possible choices are yes, no, or another.
// Last exam is an exam score.

bool more; // This is a boolean variable.
int exam; // This is an integer variable.
int exam1, exam2, exam3; // These names are not important.
string choice; // This is another string.

int average = (exam1 + exam2 + exam3) / 3; // This averages the exam scores.

if (choice == "Y") {
    more = true;
    exam = exam1;
} 
else if (choice == "N") {
    more = false;
    exam = exam1;
} 
else { // This is another condition.
    more = true;
    exam = exam1;
}
```

A do-while Loop in Program 5-7

Program Output with Example Input Shown in Bold

Would you like to average another set? [Y/N] Y [Enter]

The average is 85.

Do you want to average another set? [Y/N] Y [Enter]

The average is 88.

Do you want to average another set? [Y/N] N [Enter]

The average is 82.

A do-while Loop in Program 5-7

Continued...

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

```java
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

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

A Closer Look at the Previous Example

A for Loop in Program 5-9

```c
Program 5-9
```
A for Loop in Program 5-9

<table>
<thead>
<tr>
<th>Program Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

A 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;
```

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:

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

Logic for Keeping a Running Total

- Set accumulator to 0
- Read the number
- Add the number to the accumulator
- Is there a number to read?
  - Yes (Yes)
  - No (No, Finish)
5.8 Sentinels

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

A Sentinel in Program 5-13

Program 5-13

```
// This program calculates the total number of points a player scores in a game.
// Enter the number of games played and the number of points scored per game.
// The program calculates the total number of points.
// The output is in the format: total points = <total points>

1: int main()
2: {
3:     // Variables
4:     int game, score, total;
5:     total = 0; // Initialize total to zero
6: 
7:     // Input
8:     cout << "Enter the number of games: " << flush;
9:     cin >> game;
10:    
11:     // Calculate total points
12:     while (game > 0)
13:     {
14:         cout << "Enter the points for game: " << flush;
15:         cin >> score;
16:         total += score; // Add scores to total
17:         game--;
18:     }
19:     
20:     // Output
21:     cout << "The total points are: " << total << endl;
22:     return 0;
23: }
```

Program Output with Example Input Shown in Bold

- Enter the number of games: 3
- Enter the points for game 1: 6
- Enter the points for game 2: 8
- Enter the points for game 3: 5

The total points are 19

A Running Total in Program 5-12

Program 5-12

```
// Display the total sales for each day
// The program asks for the number of days
// and the sales for each day.
// The total sales for each day is calculated.
// The output is in the format: \
// Day 1: sales = <day 1 sales>
// Day 2: sales = <day 2 sales>
// Day 3: sales = <day 3 sales>
// 
// The total sales are <total sales>

1: void main()
2: {
3:     // Initialize total to zero
4:     double total = 0;
5: 
6:     // Ask for the number of days
7:     cout << "Enter the number of days: " << flush;
8:     cin >> days;
9: 
10:    // Calculate total sales
11:    for (int day = 1; day <= days; day++)
12:    {
13:        // Display sales for each day
14:        cout << "Day " << day << " sales = " << sales << endl;
15:    }
16: 
17:    // Calculate total sales
18:    total += sales;
19: 
20:    // Display total sales
21:    cout << "The total sales are: " << total << endl;
22: }
```

Program Output with Example Input Shown in Bold

- Enter the number of days: 3
- Enter sales for day 1: 456
- Enter sales for day 2: 423
- Enter sales for day 3: 756

The total sales are 1635
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:

```
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

```
// Determine each student's average score.
for (int student = 1; student <= numStudents; student++)
{
  total = 0; // Initialize the accumulator.
  for (int test = 1; test <= numTests; test++)
    {  
      double score;
      cout << "Enter score " << test << " for ",
      cout << "student " << student << ": ";
      cin >> score;
      total += score;
    }
  average = total / numTests;
  cout << "The average score for student " << student;
  cout << " is " << average << ".\n\";
```

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.
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

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

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:

  ```
  while (inputFile >> number) ...
  ```

Closing Files

- Use the `close` member function:

  ```
  inFile.close();
  outFile.close();
  ```

- Don't wait for the 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

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 \texttt{c\_str} Member Function in Older Versions of C++

- \texttt{string} objects have a member function named \texttt{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 \texttt{c\_str} function:

\begin{verbatim}
stringObject.c\_str()
\end{verbatim}

- Line 18 in Program 5-24 could be rewritten in the following manner:

\begin{verbatim}
inputFile.open(filename.c\_str());
\end{verbatim}

---

Breaking and Continuing a Loop

5.12

Breaking Out of a Loop

- Can use \texttt{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 \texttt{continue} Statement

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