Standard Version of Starting Out with C++, 4th Edition

Chapter 5

Looping

Topics

5.1 The Increment and Decrement Operators

5.2 Introduction to Loops: The `while` Loop

5.3 Counters

5.4 Letting the User Control a Loop

5.5 Keeping a Running Total

5.6 Sentinels

5.7 Using a Loop to Read Data from a File

Topics

5.8 The `do-while` and `for` Loops

5.9 Deciding Which Loop to Use

5.10 Nested Loops

5.11 Breaking Out of a Loop

5.12 The `continue` Statement

5.13 Using Loops for Data Validation

5.1 The Increment and Decrement Operators

- `++`: add one to a variable

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

- `--`: subtract one from a variable

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

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

  ```
  ++val;      --val;
  ```

Prefix vs. Postfix

- `++` and `--` operators can be used in complex statements and expressions
- prefix (`++val, --val`): increment or decrement, then return the value of the variable
- postfix (`val++, val--`): return the value of the variable, then increment or decrement

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, 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: part of program that may execute > 1 time (repeats)
- `while` loop:

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

- `statement;` can also be a block of statements enclosed in `{ }`

`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 the loop is finished and program statements following `statement` execute

`while` Loop Example

```
int val = 5;
while (val <= 8)
      cout << val++ << endl;
```

- produces output:

```
5
6
7
8
```

`while` Loop Notes

- no `;` after `(expression)`
- `while` is a pretest loop – `expression` is evaluated before the loop executes
- loop must contain code to make `expression` become `false`
- Infinite loop: loop that does not stop

5.3 Counters

- Counter: variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (loop control variable)
- Must be initialized before entering loop
- May be incremented/decremented either inside the loop or in the loop test

5.4 Letting the User Control a Loop

- Program can be written so that user input determines loop repetition
- Used when program processes a list of items, and user knows the number of items
- User is prompted before loop. Their input is used to control number of repetitions

Letting the User Control a Loop - Example

```
int num = 1, limit;
cout << "Table of squares\n";
cout << "How high to go? ";
cin >> limit;
cout << "number square\n";
while (num <= limit)
{   cout << setw(5) << num << setw(6)
         << num*num << endl;
    num++;
}
```

5.5 Keeping a Running Total

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

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

5.6 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

5.7 Using a Loop to Read Data from a File

- `eof()` member function: returns `true` when the end of the file has been reached, `false` otherwise
- Can be tested in a `while` loop to continue execution until end of file:

```
while (!infile.eof()) ...
```

5.8 The `do-while` and `for` Loops

- `do-while`: a posttest loop – execute the loop, then test the `expression`
- Format:

```
do
   statement;  // or block in { }
while (expression);
```

- Note `;` after `(expression)`

`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

`for` Loop

- Useful for counter-controlled loop
- Format:

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

- No `;` after 3rd expression or after `)`

`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

```
int sum, num;

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

cout << "Sum of numbers 1 - 10 is"
     << sum << endl;
```

`for` Loop - Modifications

- Can omit `initialization` if already done:

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

- Can declare variables in `initialization`:

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

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

`for` Loop - Modifications

- Can omit `update` if done in loop:

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

- Can omit `test` – may get infinite loop:

```
for (sum = 0, num = 1; ; num++)
    sum += num;
```

5.9 Deciding Which Loop to Use

- `while`: pretest loop; loop body may not be executed at all
- `do-while`: posttest loop; loop body will always be executed at least once
- `for`: pretest loop with initialization and upate expression; useful with counters, or if precise number of repetitions is needed

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 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. In previous example, inner loop repeats 9 times

5.11 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

5.12 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

5.13 Using Loops for Data Validation

- Can design a loop to repeat execution until valid input is entered:

```
cout << "Enter a test score "
     << "in the range 0-100: ";
cin >> score;
while (score < 0 || score > 100)
{    cout << "Score out of range - "
          << "reenter: ";
     cin >> score;
}
```

Standard Version of Starting Out with C++, 4th Edition

Chapter 5

Looping