Two Patterns for Input*

Jeffrey D. Oldham

2000 Jan 19

In many cases, a program's input is line-based. That is, every input line has the same format. We present two programming patterns to deal with line-based input.

1 Examples of Line-Based Input

I define line-based input as input that consists of lines each having similar format. For example, a file may consist of lines each containing a name and age. Another input may consist of lines of text. Another input may have lines with a name, beginning hours, and ending hours.

2 Process Line-Based Input on a Per Line Basis

If the input is line-based, usually the program consists of processing each line and optionally printing output per line.

Suppose we wish to determine the name of the oldest person given line-based input of name and age. As each line is read, we compare the age of the oldest person seen so far with the age specified on the line, updating the age and associated name if the line's age is larger.

Consider writing a program to count the number of characters and lines in a file. We can read the entire line (not the line's fields), incrementing the number of lines seen so far and augmenting the character count by the number of characters on that line.

3 Two Patterns for Writing Line-Based C++ Code

I currently can conceive of two large classes of line-based input:

- breaking the line into fields and
- reading an entire line as a string.

To decide which pattern to use, ask if the line's format can be specified as having several different fields. See the examples below for more explanation.

3.1 Treating the Line as a Set of Fields

Consider our example of computing the name of the oldest person given a list of lines each having a name and an age.

Here is an example program:

```
#include <iostream>
#include <string>
#include <stdlib.h>             // has EXIT_SUCCESS

int main() {
  string inputName, nameOfOldest;
  int inputAge;
  int ageOfOldest = -1;         // signals no value yet

  // each input line looks like:
  //             <name>       <age>
  while (cin >> inputName >> inputAge) {
    if (inputAge > ageOfOldest) {
      nameOfOldest = inputName;
      ageOfOldest = inputAge;
    }
  }
  // No more input when we reach here.
```

```
  cout << "The oldest person " << nameOfOldest << " is " <<
    ageOfOldest << " years old.\n";

  return EXIT_SUCCESS;
}
```

The important part of the program is the while loop. Since each line has two fields, the input in the loop's conditional will have two variables, one for each field. In the body, we process the line. The code after the loop is executed only after all of the input has been read.[1]

Here are rules for this pattern:

- Use a while loop.
- The loop's conditional should be an input statement with as many variables as fields on the line.
- The loop's body should contain code to process the line's fields.
- The code just after the loop is executed after reading all the input.

3.2 Treating a Line as a Single String

For this pattern, we do not break the line into fields. For example, the program counting the number of characters and lines in a file need not break the line into fields.

Here is an example program:

```
#include <iostream>
#include <string>
#include <stdlib.h>             // has EXIT_SUCCESS

int main() {
  int numberOfCharacters = 0;
  int numberOfLines = 0;
  string currentLine;

  // Read entire line as a string.
```

[1] The other reason for exiting the loop is that the input did not have the desired format. We will omit discussing this here.

```
  while (getline(cin, currentLine)) {
    ++numberOfLines;
    numberOfCharacters += (currentLine.length() + 1 /* newline character */);
  }
  // No more input when we reach here.
  cout << "The number of lines is " << numberOfLines << ".\n";
  cout << "The number of characters is " << numberOfCharacters << ".\n";

  return EXIT_SUCCESS;
}
```

The important part of the program is the while loop. Each line is read using the getline() function. In the body, we process the line. The code after the loop is executed only after all of the input has been read.

Here are the rules for this pattern:

- Use a while loop.
- The loop's conditional should be a getline() function.
- The loop's body should contain code to process that line.
- The code just after the loop is executed after reading all the input.