Appendix K: Stream Member Functions for Formatting

Chapter 3 introduced you to stream manipulators for formatting the appearance of data displayed by `cout`. Stream manipulators aren't the only way to format data with `cout`. Field width, precision, and format flags may also be modified by `cout`'s *member functions*. Here is an example of how the display field width can be set with a member function:

```
cout.width(5);
```

All of `cout`'s member functions are called this way. A period separates the word `cout` from the name of the member function. Any arguments needed by the function are placed in parentheses. In the statement above, `cout`'s member function `width` is called to set the field width to 5 spaces. This is equivalent to inserting the `setw(5)` manipulator before a variable in a `cout` statement. For example, the following code displays the contents of the variable x in a field of 8 spaces.

```
cout.width(8);
cout << x << endl;
```

There is also a member function named `precision`, which sets the precision of floating point numbers. The following example sets the precision to 2 decimal places:

```
cout.precision(2);
```

This setting will remain in effect until it is reset or until the end of the program. Format flags are set with the `setf` member function. Here is an example that sets the formatting of numbers to fixed point notation:

```
cout.setf(ios::fixed);
```

You can also send multiple flags to `setf` by separating them with the | symbol. Here is a statement that sets the flags for fixed point notation, decimal point displaying, and left-justification:

```
cout.setf(ios::fixed | ios::showpoint | ios::left);
```

Regardless of the way format flags are set, they may be turned off with the `unsetf` function. It works like `setf`, except the format flags you specify are disabled. The following statement turns off the `ios::fixed` and `ios::left` flags:

```
cout.unsetf(ios::fixed | ios::left);
```

Program K-1 demonstrates the `precision` and `setf` member functions.

Program K-1

```
1  // This program asks for sales figures for 3 days. The total
2  // sales is calculated and displayed in a table.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       double day1, day2, day3, total;
10
11      cout << "Enter the sales for day 1: ";
12      cin >> day1;
13      cout << "Enter the sales for day 2: ";
14      cin >> day2;
15      cout << "Enter the sales for day 3: ";
16      cin >> day3;
17
18      total = day1 + day2 + day3;
19
20      cout.precision(2);
21      cout.setf(ios::fixed | ios::showpoint);
22
23      cout << "\nSales Figures\n";
24      cout << "-------------\n";
25      cout << "Day 1: " << setw(8) << day1 << endl;
26      cout << "Day 2: " << setw(8) << day2 << endl;
27      cout << "Day 3: " << setw(8) << day3 << endl;
28      cout << "Total: " << setw(8) << total << endl;
29
30      return 0;
31 }
```

Program Output with Example Input Shown in Bold

```
Enter the sales for day 1: 2642.00 [Enter]
Enter the sales for day 2: 1837.20 [Enter]
Enter the sales for day 3: 1963.00 [Enter]

Sales Figures
-------------
Day 1: 2642.00
Day 2: 1837.20
Day 3: 1963.00
Total: 6442.20
```

Table K-1 summarizes the member functions we have discussed.

Table K-1

Member Function	Description
`width()`	Sets the display field width.
`precision()`	Sets the precision of floating point numbers.
`setf()`	Sets the specified format flags.
`unsetf()`	Disables, or turns off, the specified format flags.

The `cin` object also supports the `width` member function, which establishes an input field width. This is most helpful when `cin` is reading a string and storing it as a null-terminated C-string in a character array. When you are reading a string as input into a `char` array, `cin` has no way of knowing how large the array is. If the user types more characters than the array will hold, `cin` will store the string in the array anyway, overwriting whatever is in memory next to the array. An input field width solves this problem by telling `cin` the number of characters to read. Here is a statement declaring an array of 10 characters and using a `cin` statement to read no more characters than the array will hold:

```
char word[10];
cin.width(10);
cin >> word;
```

The field width specified is 10. This value will cause `cin` to read no more than nine characters of input, leaving room for the null character at the end. Program K-2 demonstrates.

Program K-2

```
1  // This program uses cin's width member function.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      char word[5];
8
9      cout << "Enter a word: ";
10     cin.width(5);
11     cin >> word;
12     cout << "You entered " << word << endl;
13     return 0;
14 }
```

Program Output with Example Input Shown in Bold

```
Enter a word: Eureka [Enter]
You entered Eure
```

In this program `cin` only reads 4 characters into the word array. Without the field width, `cin` would have written the entire word "Eureka" into the array, overflowing it.

Using Formatting Member Functions with File Streams

The same formatting member functions used with `cout` and `cin` may also be called from file stream objects. For example, the `precision` member function may be used to establish the number of digits of precision for floating point numbers that are written to a file. Program K-3 demonstrates this.

Program K-3

```
1  // This program uses the precision member function of a
2  // file stream object to format file output.
3  #include <iostream>
4  #include <fstream>
5  using namespace std;
6
7  int main()
8  {
9      fstream dataFile;
10     double num = 123.456;
11
12     // Open the file.
13     dataFile.open("numfile.txt", ios::out);
14
15     // Write a value at various digits of precision
16     dataFile << num << endl;
17     dataFile.precision(5);
18     dataFile << num << endl;
19     dataFile.precision(4);
20     dataFile << num << endl;
21     dataFile.precision(3);
22     dataFile << num << endl;
23
24     // Close the file.
25     dataFile.close();
26     return 0;
27 }
```

Contents of File numfile.txt

```
123.456
123.46
123.5
124
```

Notice the file output is formatted just as `cout` would format screen output. In addition, you may use the `width`, `setf`, and `unsetf` functions with file stream objects.