Sams Teach Yourself C in 24 Hours

Hour 22 - Disk File Input and Output: Part II

Disk space: the final frontier.

—Captain Kirk's younger brother

In last hour's lesson you learned the basics of reading and writing disk data files. In this lesson you'll learn more about communication with disk data files. The main topics discussed in this hour are

- Random access to files
- Reading or writing binary data
- Redirecting the standard streams

In addition, the following C I/O functions are introduced in this lesson:

- The fseek(), ftell(), and rewind() functions
- The fscanf() and fprintf() functions
- The freopen() function

Random Access to Disk Files

So far you've learned how to read or write data sequentially to an opened disk file, known as sequential access. In other words, you start with the first byte and keep reading or writing each successive byte in order. In many cases, however, you need to access particular data somewhere in the middle of a disk file. One way to do this is to keep reading data from the file until the particular data is fetched. Obviously, this is not an efficient way, especially when the file contains many data items.

Random access is another way to read or write data to disk files. In random access, specific file elements can be accessed in random order (that is, without reading through all the preceding data).

In C there are two I/O functions, fseek() and ftell(), that are designed to deal with random access.

The fseek() and ftell() Functions

As just mentioned, we need functions that enable us to access files randomly. The fseek() and ftell() functions provide us with such a capability.

In the previous lesson you learned that one of the members in the FILE structure is called the file position indicator. The file position indicator has to point to the desired position in a file before data can be read from or written to there. You can use the fseek() function to move the file position indicator to the spot you want to access in a file.

The syntax for the fseek() function is

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

Here stream is the file pointer associated with an opened file. offset indicates the number of bytes from a fixed position, specified by whence, that can have one of the following integral values represented by SEEK_SET, SEEK_CUR, and SEEK_END. If it is successful, the fseek() function returns 0; otherwise, the function returns a nonzero value.

You can find the values represented by SEEK_SET, SEEK_CUR, and SEEK_END in the header file stdio.h.

If SEEK_SET is chosen as the third argument to the fseek() function, the offset is counted from the beginning of the file and the value of the offset is greater than or equal to zero. If, however, SEEK_END is picked up, then the offset starts from the end of the file; the value of the offset should be negative. When SEEK_CUR is passed to the fseek() function, the offset is calculated from the current value of the file position indicator.

You can obtain the value of the current file position indicator by calling the ftell() function.

The syntax for the ftell() function is

```
#include <stdio.h>
long ftell(FILE *stream);
```

Here stream is the file pointer associated with an opened file. The ftell() function returns the current value of the file position indicator.

The value returned by the ftell() function represents the number of bytes from the beginning of the file to the current position pointed to by the file position indicator.

If the ftell() function fails, it returns -1L (that is, a long value of minus 1). One thing that can cause the failure of the ftell() function is the file being a terminal or some other type for which the file position indicator becomes meaningless.

The program in Listing 22.1 shows how to randomly access a disk file by using the fseek() and ftell() functions.

TYPE
Listing 22.1. Random access to a file.

```
1:  /* 22L01.c: Random access to a file */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL, MAX_LEN = 80};
5:
6:  void PtrSeek(FILE *fptr);
7:  long PtrTell(FILE *fptr);
8:  void DataRead(FILE *fptr);
9:  int ErrorMsg(char *str);
10:
11: main(void)
12: {
13:    FILE *fptr;
14:    char filename[]= "haiku.txt";
15:    int reval = SUCCESS;
16:
17:    if ((fptr = fopen(filename, "r")) == NULL){
18:       reval = ErrorMsg(filename);
19:    } else {
20:       PtrSeek(fptr);
21:       fclose(fptr);
22:    }
23:
24:    return reval;
25: }
26: /* function definition */
```

```
27: void PtrSeek(FILE *fptr)
28: {
29:    long offset1, offset2, offset3;
30:
31:    offset1 = PtrTell(fptr);
32:    DataRead(fptr);
33:    offset2 = PtrTell(fptr);
34:    DataRead(fptr);
35:    offset3 = PtrTell(fptr);
36:    DataRead(fptr);
37:
38:    printf("\nRe-read the haiku:\n");
39:    /* re-read the third verse of the haiku */
40:    fseek(fptr, offset3, SEEK_SET);
41:    DataRead(fptr);
42:    /* re-read the second verse of the haiku */
43:    fseek(fptr, offset2, SEEK_SET);
44:    DataRead(fptr);
45:    /* re-read the first verse of the haiku */
46:    fseek(fptr, offset1, SEEK_SET);
47:    DataRead(fptr);
48: }
49: /* function definition */
50: long PtrTell(FILE *fptr)
51: {
52:    long reval;
53:
54:    reval = ftell(fptr);
55:    printf("The fptr is at %ld\n", reval);
56:
57:    return reval;
58: }
59: /* function definition */
60: void DataRead(FILE *fptr)
61: {
62:    char buff[MAX_LEN];
63:
64:    fgets(buff, MAX_LEN, fptr);
65:    printf("---%s", buff);
66: }
67: /* function definition */
68: int ErrorMsg(char *str)
69: {
70:    printf("Cannot open %s.\n", str);
71:    return FAIL;
72: }
```

OUTPUT

I have the following output shown on my screen after running the executable 22L01.exe of the program in Listing 22.1:

```
C:\app>22L01
The fptr is at 0
---Leading me along
The fptr is at 18
```

```
---my shadow goes back home
The fptr is at 44
---from looking at the moon.
Re-read the haiku:
---from looking at the moon.
---my shadow goes back home
---Leading me along
C:\app>
```

ANALYSIS

The purpose of the program in Listing 22.1 is to move the file position indicator around in order to read different verses from the haiku.txt file.

Inside the main() function, a file pointer fptr is defined in line 13, and the name of the haiku.txt file is assigned to the array called filename in line 14. Then, in line 17, we try to open the haiku.txt file for reading by calling the fopen() function. If successful, we invoke the PtrSeek() function with the fptr file pointer as the argument in line 20.

The definition of our first function PtrSeek() is shown in lines 27_48. The statement in line 31 obtains the original value of the fptr file pointer by calling another function, PtrTell(), which is defined in lines 50_58. The PtrTell() function can find and print out the value of the file position indicator with the help of the C ftell() function. The original value of the file position indicator contained by fptr is assigned to the long variable offset1 in line 31.

In line 32, the third function, DataRead(), is called to read one line of characters from the opened file and print out the line of characters on the screen. Line 33 gets the value of the fptr file position indicator right after the reading and assigns the value to another long variable, offset2.

Then the DataRead() function in line 34 reads the second line of characters from the opened file. Line 35 obtains the value of the file position indicator that points to the first byte of the third verse and assigns the value to the third long variable offset3. Line 36 calls the DataRead() function to read the third verse and print it out on the screen.

Therefore, from the first portion of the output, you can see the three different values of the file position indicator at three different positions, and the three verses of the haiku written by Sodo. The three values of the file position indicator are saved respectively by offset1, offset2, and offset3.

Now, starting from line 40 to line 47, we read Sodo's haiku backward, one verse at a time. That is, we read the third verse first, then the second verse, and finally the first verse. To do so, we first call the fseek() function to move the file position indicator to the beginning of the third verse by passing the value contained by offset3 to the function. Then we call fseek() again and pass the value of offset2 to the function so that the file position indicator is set to point to the first byte of the second verse. Finally, we move the file position indicator to the beginning of the first verse by passing the value of offset1 to the fseek() function. Therefore, in the second portion of the output, you see the three verses of the haiku in reverse order.

The rewind() Function

Sometimes you might want to reset the file position indicator and put it at the beginning of a file. There is a handy C function, called rewind(), that can be used to rewind the file position indicator.

The syntax for the rewind() function is

```
#include <stdio.h>
void rewind(FILE *stream);
```

Here stream is the file pointer associated with an opened file. No value is returned by the rewind() function.

In fact, the following statement of the rewind() function

```
rewind(fptr);
```

is equivalent to this:

```
(void)fseek(fptr, 0L, SEEK_SET);
```

Here the void data type is cast to the fseek() function because the rewind() function does not return a value. Listing 22.2 contains an example that calls the rewind() function to move the file position indicator to the beginning of an opened file.

More Examples of Disk File I/O

The following sections show several more examples of disk file I/O, such as reading and writing binary data and redirecting the standard streams. Three more I/O functions, fscanf(), fprintf(), and freopen(), are introduced, too.

Reading and Writing Binary Data

As you learned in Hour 21, "Disk File Input and Output: Part I," you can indicate to the compiler that you're going to open a binary file by setting a proper mode when calling the fopen() function. For instance, the following statement tries to open an existing binary file for reading:

```
fptr = fopen("test.bin", "rb");
```

Note that the "rb" mode is used to indicate that the file we're going to open for reading is a binary file.

Listing 22.2 contains an example of reading and writing binary data.

TYPE
Listing 22.2. Reading and writing binary data.

```
1:  /* 22L02.c: Reading and writing binary data */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL, MAX_NUM = 3};
5:
6:  void DataWrite(FILE *fout);
7:  void DataRead(FILE *fin);
8:  int ErrorMsg(char *str);
9:
10: main(void)
11: {
12:    FILE *fptr;
13:    char filename[]= "double.bin";
14:    int reval = SUCCESS;
15:
16:    if ((fptr = fopen(filename, "wb+")) == NULL){
17:       reval = ErrorMsg(filename);
18:    } else {
19:       DataWrite(fptr);
20:       rewind(fptr);  /* reset fptr */
21:       DataRead(fptr);
22:       fclose(fptr);
23:    }
24:
25:    return reval;
```

```
26: }
27: /* function definition */
28: void DataWrite(FILE *fout)
29: {
30:    int i;
31:    double buff[MAX_NUM] = {
32:              123.45,
33:              567.89,
34:              100.11};
35:
36:    printf("The size of buff: %d-byte\n", sizeof(buff));
37:    for (i=0; i<MAX_NUM; i++){
38:       printf("%5.2f\n", buff[i]);
39:       fwrite(&buff[i], sizeof(double), 1, fout);
40:    }
41: }
42: /* function definition */
43: void DataRead(FILE *fin)
44: {
45:    int i;
46:    double x;
47:
48:    printf("\nRead back from the binary file:\n");
49:    for (i=0; i<MAX_NUM; i++){
50:       fread(&x, sizeof(double), (size_t)1, fin);
51:       printf("%5.2f\n", x);
52:    }
53: }
54: /* function definition */
55: int ErrorMsg(char *str)
56: {
57:    printf("Cannot open %s.\n", str);
58:    return FAIL;
59: }
```

OUTPUT

After running the executable 22L02.exe, I have the following output on the screen:

```
C:\app>22L02
The size of buff: 24-byte
123.45
567.89
100.11

Read back from the binary file:
123.45
567.89
100.11
C:\app>
```

ANALYSIS

The purpose of the program in Listing 22.2 is to write three values of the double data type into a binary file and then rewind the file position indicator and read back the three double values from the binary file. The two functions, DataWrite() and DataRead(), that perform the writing and reading, are declared in lines 6 and 7.

The enum names, SUCCESS, FAIL, and MAX_NUM, are defined in line 4 with values of 0, 1, and 3, respectively.

Inside the main() function, the statement in line 16 tries to create and open a binary file called double.bin for both reading and writing. Note that the "wb+" mode is used in the fopen() function in line 16.

If the fopen() function is successful, the DataWrite() function is called in line 19 to write three double data items, 123.45, 567.89, and 100.11, into the opened binary file, according to the definition of the DataWrite() function in lines 28_41. The fwrite() function in line 39 does the writing. Because the three double data items are saved in an array named buff, we also measure and print out the size of the buff array in line 36. On my machine, the size of the buff array is 24 bytes because each double data item is 8 bytes.

Right after the execution of the DataWrite() function, the file position indicator is reset to the beginning of the binary file by calling the rewind() function in line 20 because we want to read back all three double data items written to the file.

Then in line 21, the DataRead() function reads the three double data items from the opened binary file double.bin. From the definition of the DataRead() function in lines 43_53, you can see that the fread() function is used to perform the reading operation (see line 50).

The output from running the program in Listing 22.2 shows the three double data items before the writing and after the reading as well.

The fscanf() and fprintf() Functions

As you learned, the two C library functions scanf() and printf() can be used to read or write formatted data through the standard I/O (that is, stdin and stdout). Among the C disk file I/O functions, there are two equivalent functions, fscanf() and fprintf(), that can do the same jobs as the scanf() and printf() functions. In addition, the fscanf() and fprintf() functions allow the programmer to specify I/O streams.

The syntax for the fscanf() function is

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, …);
```

Here stream is the file pointer associated with an opened file. format, whose usage is the same as in the scanf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fscanf() function returns the number of data items read. Otherwise, the function returns EOF.

The syntax for the fprintf() function is

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, …);
```

Here stream is the file pointer associated with an opened file. format, whose usage is the same as in the printf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fprintf() function returns the number of formatted expressions. Otherwise, the function returns a negative value.

To know more about the fprintf() and fscanf() functions, you can review the explanations on the printf() and scanf() functions in Hour 5, "Reading from and Writing to Standard I/O," and Hour 13, "Manipulating Strings."

The program in Listing 22.3 demonstrates how to use the fscanf() and fprintf() functions to read and write differently typed data items.

TYPE
Listing 22.3. Using the fscanf() and fprintf() functions.

```
1:  /* 22L03.c: Using the fscanf() and fprintf() functions */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL,
5:        MAX_NUM = 3,
6:        STR_LEN = 23};
7:
8:  void DataWrite(FILE *fout);
9:  void DataRead(FILE *fin);
10: int ErrorMsg(char *str);
11:
12: main(void)
13: {
14:    FILE *fptr;
15:    char filename[]= "strnum.mix";
16:    int reval = SUCCESS;
17:
18:    if ((fptr = fopen(filename, "w+")) == NULL){
19:       reval = ErrorMsg(filename);
20:    } else {
21:       DataWrite(fptr);
22:       rewind(fptr);
23:       DataRead(fptr);
24:       fclose(fptr);
25:    }
26:
27:    return reval;
28: }
29: /* function definition */
30: void DataWrite(FILE *fout)
31: {
32:    int i;
33:    char cities[MAX_NUM][STR_LEN] = {
34:             "St.Louis->Houston:",
35:             "Houston->Dallas:",
36:             "Dallas->Philadelphia:"};
37:    int miles[MAX_NUM] = {
38:             845,
39:             243,
40:             1459};
41:
42:    printf("The data written:\n");
43:    for (i=0; i<MAX_NUM; i++){
44:       printf("%-23s  %d miles\n", cities[i], miles[i]);
45:       fprintf(fout, "%s  %d", cities[i], miles[i]);
46:    }
47: }
48: /* function definition */
49: void DataRead(FILE *fin)
50: {
51:    int i;
52:    int miles;
53:    char cities[STR_LEN];
54:
55:    printf("\nThe data read:\n");
```

```
56:   for (i=0; i<MAX_NUM; i++){
57:     fscanf(fin, "%s%d", cities, &miles);
58:     printf("%-23s  %d miles\n", cities, miles);
59:   }
60: }
61: /* function definition */
62: int ErrorMsg(char *str)
63: {
64:   printf("Cannot open %s.\n", str);
65:   return FAIL;
66: }
```

OUTPUT
The following output is shown on the screen after the executable 22L03.exe is created and run:

```
C:\app>22L03
The data written:
St.Louis->Houston:    845 miles
Houston->Dallas:      243 miles
Dallas->Philadelphia:  1459 miles

The data read:
St.Louis->Houston:    845 miles
Houston->Dallas:      243 miles
Dallas->Philadelphia:  1459 miles
C:\app>
```

ANALYSIS
The purpose of the program in Listing 22.3 is to write data items of different types into a file with the help of the fprintf() function and read back the data items in the same format by calling the fscanf() functions. The two functions declared in lines 8 and 9, DataWrite() and DataRead(), actually perform the writing and reading.

The statement of the main() function in line 18 tries to create and open a text file called strnum.mix for both reading and writing by specifying the second argument to the fopen() function as "w+". If fopen() does not return a null pointer, the DataWrite() function is called in line 21 to write strings and int data items into the strnum.mix file. Note that the fprintf() function is invoked inside the DataWrite() function in line 45 to write the formatted data into the text file.

From the definition of the DataWrite() function in lines 30_47, you can see that there are two arrays, cities and miles. The cities array contains three strings that indicate three pairs of cities, and the miles array has three int values representing the corresponding distances between the cities shown in the cities array. For instance, 845 in the miles array is the distance (in miles) between the two cities expressed by the string St.Louis->Houston: in the cities array.

In line 22, the rewind() function is called to rewind the file position indicator and reset it to the beginning of the strnum.mix file. Then the DataRead() function in line 23 reads back what has been saved in strnum.mix with the help of the fscanf() function. The definition of the DataRead() function is shown in lines 49_60.

From this example, you see that it is convenient to use the fprintf() and fscanf() functions together to perform formatted disk file I/O operations.

Redirecting the Standard Streams with freopen()

In Hour 5 you learned how to read from or write to standard I/O. Also, you were told that the C functions, such as getc(), gets(), putc, and printf(), direct their I/O operations automatically to either stdin or stdout.

In this section you're going to learn how to redirect the standard streams, such as stdin and stdout, to disk files. A new C function you're going to use is called freopen(), which can associate a standard stream with a disk file.

The syntax for the freopen() function is

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Here filename is a char pointer referencing the name of a file that you want to associate with the standard stream represented by stream. mode is another char pointer pointing to a string that defines the way to open a file. The values that mode can have in freopen() are the same as the mode values in the fopen() function. (The definition of all mode values is given in Hour 21.)

The freopen() function returns a null pointer if an error occurs. Otherwise, the function returns the standard stream that has been associated with a disk file identified by filename.

Listing 22.4 demonstrates an example of redirecting the standard output, stdout, with the help of the freopen() function.

TYPE
Listing 22.4. Redirecting the standard stream stdout.

```
1:  /* 22L04.c: Redirecting a standard stream */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL,
5:        STR_NUM = 4};
6:
7:  void StrPrint(char **str);
8:  int ErrorMsg(char *str);
9:
10: main(void)
11: {
12:    char *str[STR_NUM] = {
13:          "Be bent, and you will remain straight.",
14:          "Be vacant, and you will remain full.",
15:          "Be worn, and you will remain new.",
16:          "---- by Lao Tzu"};
17:    char filename[]= "LaoTzu.txt";
18:    int reval = SUCCESS;
19:
20:    StrPrint(str);
21:    if (freopen(filename, "w", stdout) == NULL){
22:       reval = ErrorMsg(filename);
23:    } else {
24:       StrPrint(str);
25:       fclose(stdout);
26:    }
27:    return reval;
28: }
29: /* function definition */
30: void StrPrint(char **str)
31: {
32:    int i;
33:
```

```
34:   for (i=0; i<STR_NUM; i++)
35:     printf("%s\n", str[i]);
36: }
37: /* function definition */
38: int ErrorMsg(char *str)
39: {
40:   printf("Cannot open %s.\n", str);
41:   return FAIL;
42: }
```

OUTPUT

Note that if you're using Microsoft Visual C++ on your machine, you need to make sure the project type is set to MS-DOS application (.EXE). Don't set the project type to QickWin application (.EXE) or other Windows application types, because there might be some conflicts between a Windows environment and the freopen() function. After the executable 22L04.exe is created and run, the following output is printed on the screen:

```
C:\app>22L04
Be bent, and you will remain straight.
Be vacant, and you will remain full.
Be worn, and you will remain new.
---- by Lao Tzu
C:\app>
```

ANALYSIS

The purpose of the program in Listing 22.4 is to save a paragraph of Tao Te Ching written by a Chinese philosopher, Lao Tzu, into a text file, LaoTzu.txt. To do so, we call the printf() function instead of the fprintf() function or other disk I/O functions after we redirect the default stream, stdout, of the printf() function to point to the text file.

The function that actually does the writing is called StrPrint(), which invokes the C function printf() to send out formatted character strings to the output stream. (See the definition of the StrPrint() function in lines 30_36.)

Inside the main() function, we call the StrPrint() function in line 20 before we redirect stdout to the LaoTzu.txt file. It's not surprising to see that the paragraph adopted from Tao Te Ching is printed on the screen because the printf() function automatically sends out the paragraph to stdout that directs to the screen by default.

Then, in line 21, we redirect stdout to the LaoTzu.txt text file by calling the freopen() function. There the "w" is used as the mode that indicates to open the text file for writing. If freopen() is successful, we then call the StrPrint() function in line 24. However, this time, the StrPrint() function writes the paragraph into the opened text file, LaoTzu.txt. The reason is that stdout is now associated with the text file, not the screen, so that strings sent out by the printf() function inside StrPrint() are directed to the text file.

After the execution of the program in Listing 22.4, you can open the LaoTzu.txt file in a text editor and see that the paragraph of Tao Te Ching has been saved in the file.

NOTE

As mentioned previously, the I/O streams are buffered by default. Occasionally, you may want to turn off the buffering so that you can process the input immediately. In C there are two functions, setbuf() and setvbuf(), that can be used to turn off the buffering. Appendix B contains the syntax of the two functions, although unbuffering I/O is beyond the scope of this book.

Also, there is a set of low-level I/O functions, such as open(), create(), close(), read(), write(), lseek(), and tell(), which are not sup-ported by the ANSI C standard. You may still see them in

some platform-dependent C programs. To use them, you need to read your C compiler's reference manual to make sure they're available.

Summary

In this lesson you've learned the following:

- The file position indicator can be reset by the fseek() function.
- The ftell() function can tell you the value of the current file position indicator.
- The rewind() function can set the file position indicator to the beginning of a file.
- After you specify the mode of the fopen() function for the binary file, you can use the fread() or fwrite() functions to perform I/O operations on binary data.
- Besides the fact that the fscanf() and fprintf() functions can do the same jobs as the scanf() and printf() functions, the fscanf() and fprintf() functions also allow the programmer to specify I/O streams.
- You can redirect the standard streams, such as stdin and stdout, to a disk file with the help of the freopen() function.

In the next lesson you'll learn about the C preprocessor.

Q&A

Q Why is random access to a disk file necessary?

A When you want to fetch a piece of information from a large file that contains a huge amount of data, random access to the file is a more efficient way than sequential access at the file. The functions that perform random access can put the file position indicator directly at the right place in the file, and then you can simply start to fetch the required information from there. In C, the fseek() and ftell() functions are two handy functions that help you to carry out the random access operation.

Q How do you specify the format of a new disk file you're going to create by calling fopen()?

A We have to add b to the mode argument to the fopen() function to specify that the file we're going to create is a binary file. We can use "wb" to create a new file for writing and "wb+" to create a new file for writing and reading. If, however, the file to be created is a text file, no b is needed in the mode argument.

Q What is the difference between the printf() and fprintf() functions?

A Basically, the printf() and fprintf() functions can do a similar job: send the formatted data items to the output streams. However, the printf() function automatically sends formatted data to stdout, whereas the fprintf() function can be assigned a file pointer that is associated with a specified output stream.

Q Can you redirect a standard stream to a disk file?

A Yes. With the help of the freopen() function, you can redirect a standard stream and associate the stream with a disk file.

Workshop

To help solidify your understanding of this hour's lesson, you are encouraged to answer the quiz questions and finish the exercises provided in the Workshop before you move to the next lesson. The answers and hints to the questions and exercises are given in Appendix E, "Answers to Quiz Questions and Exercises."

Quiz

1. Are the following two statements equivalent?

   ```
   rewind(fptr);
   (void)fseek(fptr, 0L, SEEK_SET);
   ```

2. Are the following two statements equivalent?

   ```
   rewind(fptr);
   (void)fseek(fptr, 0L, SEEK_CUR);
   ```

3. After the statement

   ```
   freopen("test.txt", "r", stdin);

   scanf("%s%d", str, &num);
   ```

4. Given that the size of the double data type is 8 bytes long and includes four double data items, if you write the four double data items into a binary file, how many bytes do the four data items take in the file?

Exercises

1. Assume that the following paragraph of Tao Te Ching is saved in a text file called LaoTzu.txt:

   ```
   Be bent, and you will remain straight.
   Be vacant, and you will remain full.
   Be worn, and you will remain new.
   ```

   Write a program to use ftell() to find the positions of the three strings in the file, and then call fseek() to set the file position indicator in such a way that the three strings are printed out in reverse order.
2. Rewrite the program you made in exercise 1 by calling the rewind() function to reset the file position indicator at the beginning of the LaoTzu.txt file.
3. Given a double value of 123.45 and an int value of 10000, write a program to save them into a binary file, called data.bin, and then read them back from the binary file. Also, print out what you're writing or reading. What do you think the size of the binary file will be?
4. Read the text file strnum.mix, which is created by the program in Listing 22.3. Redirect the input stream so that you can use the scanf() function to perform the reading operation. (Note that if you're using Microsoft Visual C++ on your machine, make sure the project type is set to MS-DOS application (.EXE).)