Sams Teach Yourself C in 24 Hours

Hour 21 - Disk File Input and Output: Part I

I can only assume that a "Do Not File" document is filed in a "Do Not File" file.

—F. Church

In Hour 5, "Reading from and Writing to Standard I/O," you learned how to read or write characters through standard input or output. In this lesson you'll learn to read data from or write data to disk files. The following topics are discussed in this lesson:

- Files and streams
- Opening a file with fopen()
- Closing a file with fclose()
- The fgetc() and fputc() functions
- The fgets() and fputs() functions
- The fread() and fwrite() functions
- The feof() function

Files Versus Streams

The C language provides a set of rich library functions to perform input and output (I/O) operation. Those functions can read or write any type of data to files. Before we go any further in discussing the C I/O functions, let's first understand the definitions of files and streams in C.

What Is a File?

In C, a file can refer to a disk file, a terminal, a printer, or a tape drive. In other words, a file represents a concrete device with which you want to exchange information. Before you perform any communication to a file, you have to open the file. Then you need to close the opened file after you finish exchanging information with it.

What Is a Stream?

The data flow you transfer from your program to a file, or vice versa, is called a stream, which is a series of bytes. Not like a file, a stream is device-independent. All streams have the same behavior. To perform I/O operations, you can read from or write to any type of files by simply associating a stream to the file.

There are two formats of streams. The first one is called the text stream, which consists of a sequence of characters (that is, ASCII data). Depending on the compilers, each character line in a text stream may be terminated by a newline character. Text streams are used for textual data, which has a consistent appearance from one environment to another, or from one machine to another.

The second format of streams is called the binary stream, which is a series of bytes. The content of an .exe file would be one example. Binary streams are primarily used for nontextual data, which is required to keep the exact contents of the file.

Buffered I/O

In C, a memory area, which is temporarily used to store data before it is sent to its destination, is called a buffer. With the help of buffers, the operating system can improve efficiency by reducing the number of accesses to I/O devices (that is, files).

By default, all I/O streams are buffered. The buffered I/O is also called the high-level I/O. Accordingly, the low-level I/O refers to the unbuffered I/O.

The Basics of Disk File I/O

Now let's focus on how to open and close a disk data file and how to interpret error messages returned by I/O functions.

Pointers of FILE

The FILE structure is the file control structure defined in the header file stdio.h. A pointer of type FILE is called a file pointer, which references a disk file. A file pointer is used by a stream to conduct the operation of the I/O functions. For instance, the following defines a file pointer called fptr:

```
FILE *fptr;
```

In the FILE structure there is a member, called the file position indicator, that points to the position in a file where data will be read from or written to. You'll learn how to move the file position indicator in the next lesson.

Opening a File

The C I/O function fopen() gives you the ability to open a file and associate a stream to the opened file. You need to specify the way to open a file and the filename with the fopen() function.

The syntax for the fopen() function is

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

Here filename is a char pointer that references a string of a filename. The filename is given to the file that is about to be opened by the fopen() function. mode points to another string that specifies the way to open the file. The fopen() function returns a pointer of type FILE. If an error occurs during the procedure to open a file, the fopen() function returns a null pointer.

The following list shows the possible ways to open a file by various strings of modes:

- "r" opens an existing text file for reading.
- "w" creates a text file for writing.
- "a" opens an existing text file for appending.
- "r+" opens an existing text file for reading or writing.
- "w+" creates a text file for reading or writing.
- "a+" opens or creates a text file for appending.
- "rb" opens an existing binary file for reading.
- "wb" creates a binary file for writing.
- "ab" opens an existing binary file for appending.
- "r+b" opens an existing binary file for reading or writing.
- "w+b" creates a binary file for reading or writing.
- "a+b" opens or creates a binary file for appending.

Note that you might see people use the mode "rb+" instead of "r+b". These two strings are equivalent. Similarly, "wb+" is the same as "w+b"; "ab+" is equivalent to "a+b".

The following statements try to open a file called test.txt:

```
FILE *fptr;
if ((fptr = fopen("test.txt", "r")) == NULL){
```

```
  printf("Cannot open test.txt file.\n");
  exit(1);
}
```

Here "r" is used to indicate that a text file is about to be opened for reading only. If an error occurs when the fopen() function tries to open the file, the function returns a null pointer. Then an error message is printed out by the printf() function and the program is aborted by calling the exit() function with a nonzero value.

Closing a File

After a disk file is read, written, or appended with some new data, you have to disassociate the file from a specified stream by calling the fclose() function.

The syntax for the fclose() function is

```
#include <stdio.h>
int fclose(FILE *stream);
```

Here stream is a file pointer that is associated with a stream to the opened file. If fclose() closes a file successfully, it returns 0. Otherwise, the function returns EOF. Normally, the fclose() function fails only when the disk is removed before the function is called or there is no more space left on the disk.

Since all high-level I/O operations are buffered, the fclose() function flushes data left in the buffer to ensure that no data will be lost before it disassociates a specified stream with the opened file.

Note that a file that is opened and associated with a stream has to be closed after the I/O operation. Otherwise, the data saved in the file may be lost; some unpredictable errors might occur during the execution of your program.

The program in listing 21.1 shows you how to open and close a text file and how to check the returned file pointer value as well.

TYPE
Listing 21.1. Opening and closing a text file.

```
1:  /* 21L01.c: Opening and closing a file */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL};
5:
6:  main(void)
7:  {
8:    FILE *fptr;
9:    char filename[]= "haiku.txt";
10:   int reval = SUCCESS;
11:
12:   if ((fptr = fopen(filename, "r")) == NULL){
13:     printf("Cannot open %s.\n", filename);
14:     reval = FAIL;
15:   } else {
16:     printf("The value of fptr: 0x%p\n", fptr);
17:     printf("Ready to close the file.");
18:     fclose(fptr);
19:   }
20:
```

```
21:   return reval;
22: }
```

OUTPUT

The following output shows on my screen after running the executable 21L01.exe of the program in Listing 21.1. (Note that the value of the fptr is likely to be different on your machine. That's okay.)

ANALYSIS

```
C:\app>21L01
The value of fptr: 0x013E
Ready to close the file.
C:\app>
```

The purpose of the program in Listing 21.1 is to show you how to open a text file. From the expression in line 12, you can see that the fopen() function tries to open a text file with the name contained by the character array filename for reading. The filename array is defined and initialized with the name haiku.txt in line 9.

If an error occurs when you try to open the text file, the fopen() function returns a null pointer. Line 13 then prints a warning message, and line 14 assigns the value represented by the enum name FAIL to the int variable reval. From the declaration of the enum data type in line 4, we know that the value of FAIL is 1.

If, however, the fopen() function opens the text file successfully, the statement in line 16 prints the value contained by the file pointer fptr. Line 17 tells the user that the program is about to close the file, and line 18 then closes the file by calling the fclose() file.

In line 21, the return statement returns the value of reval that contains 0 if the text file has been opened successfully, or 1 otherwise.

From the output shown on my screen, I see that the value held by the file pointer fptr is 0x013E after the text file is opened.

Reading and Writing Disk Files

The program in Listing 21.1 does not do anything with the text file, haiku.txt, except open and close it. In fact, there are two pieces of Japanese haiku, written by Sodo and Chora, saved in the haiku.txt file. So how can you read them from the file?

In C, you can perform I/O operations in the following ways:

- Read or write one character at a time.
- Read or write one line of text (that is, one character line) at a time.
- Read or write one block of characters at a time.

The following three sections explain the three ways to read and write to disk files.

One Character at a Time

Among the C I/O functions, there is a pair of functions, fgetc() and fputc(), that can be used to read from or write to a disk file one character at a time.

The syntax for the fgetc() function is

```
#include <stdio.h>
int fgetc(FILE *stream);
```

Here stream is the file pointer that is associated with a stream. The fgetc() function fetches the next character from the stream specified by stream. The function then returns the value of an int that is converted from the character.

The syntax for the fputc() function is

```
#include <stdio.h>
int fputc(int c , FILE *stream);
```

Here c is an int value that represents a character. In fact, the int value is converted to an unsigned char before being outputted. stream is the file pointer that is associated with a stream. The fputc() function returns the character written if the function is successful; otherwise, it returns EOF. After a character is written, the fputc() function advances the associated file pointer.

To learn how to use the fgetc() and fputc() functions, let's have a look at Listing 21.2, which contains a program that opens a text file, and then reads and writes one character at a time.

TYPE
Listing 21.2. Reading and writing one character at a time.

```
1:  /* 21L02.c: Reading and writing one character at a time */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL};
5:
6:  void CharReadWrite(FILE *fin, FILE *fout);
7:
8:  main(void)
9:  {
10:   FILE *fptr1, *fptr2;
11:   char filename1[]= "outhaiku.txt";
12:   char filename2[]= "haiku.txt";
13:   int reval = SUCCESS;
14:
15:   if ((fptr1 = fopen(filename1, "w")) == NULL){
16:     printf("Cannot open %s.\n", filename1);
17:     reval = FAIL;
18:   } else if ((fptr2 = fopen(filename2, "r")) == NULL){
19:     printf("Cannot open %s.\n", filename2);
20:     reval = FAIL;
21:   } else {
22:     CharReadWrite(fptr2, fptr1);
23:     fclose(fptr1);
24:     fclose(fptr2);
25:   }
26:
27:   return reval;
28: }
29: /* function definition */
30: void CharReadWrite(FILE *fin, FILE *fout)
31: {
32:   int c;
33:
34:   while ((c=fgetc(fin)) != EOF){
35:     fputc(c, fout);  /* write to a file */
36:     putchar(c);     /* put the character on the screen */
```

```
37:   }
38: }
```

OUTPUT

After running the executable 21L02.exe, I get the following output:

```
C:\app>21L02
Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
  (1641-1716)
```

ANALYSIS

```
A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
  (1729-1781)
C:\app>
```

The purpose of the program in Listing 21.2 is to read one character from a file, write the character to another file, and then display the character on the screen. (You need to copy the file, haiku.txt, from the CD-ROM in the book, and put it in the same directory where you save the executable file 21L02.exe. haiku.txt is the text file that is going to be read by 21L02.exe.)

In Listing 21.2 there is a function called CharReadWrite(), which has two file pointers as its arguments. (See the declaration of the CharReadWrite() function in line 6.)

The statement in line 10 defines two file pointers, fptr1 and fptr2, which are used later in the program. Lines 11 and 12 define two character arrays, filename1 and filename2, and initialize the two arrays with two strings containing filenames, outhaiku.txt and haiku.txt.

In line 15, a text file with the name outhaiku.txt is opened for writing. outhaiku.txt is contained by the filename1 array. The file pointer fptr1 is associated with the file. If the fopen() function returns NULL, which means an error occurs, a warning message is printed out in line 16. Also, in line 17, the reval variable is assigned 1 and is represented by the enum name FAIL.

If the file outhaiku.txt is opened successfully, another text file, called haiku.txt, is opened for reading in line 18. The file pointer fptr2 is associated with the opened text file.

If no error occurs, the CharReadWrite() function is invoked in line 22 with two file pointers, fptr1 and fptr2, passed to the function as arguments. From the definition of the CharReadWrite() function in lines 30 and 38, we see that there is a while loop that keeps calling the fgetc() function to read the next character from the haiku.txt text file until the function reaches the end of the file. (See line 34.)

Within the while loop, the fputc() function in line 35 writes each character read from the haiku.txt file to another text file, outhaiku.txt, which is pointed to by fout. In addition, putchar() is called in line 36 in order to put the character returned by the fgetc() function on the screen.

After the CharReadWrite() function finishes its job, the two opened files, which are associated with fptr1 and fptr2, are closed with a call to the fclose() function respectively in lines 23 and 24.

As mentioned earlier, the haiku.txt file contains two pieces of Japanese haiku written by Sodo and Chora. If the program in Listing 21.2 is run successfully, we see the two pieces of haiku shown on the screen, and they

are written into the outhaiku.txt file as well. You can view outhaiku.txt in a text editor to confirm that the content of haiku.txt has been correctly copied to outhaiku.txt.

One Line at a Time

Besides reading or writing one character at a time, you can also read or write one character line at time. There is a pair of C I/O functions, fgets() and fputs(), that allows you to do so.

The syntax for the fgets() function is

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

Here s references a character array that is used to store characters read from the opened file pointed to by the file pointer stream. n specifies the maximum number of array elements. If it is successful, the fgets() function returns the char pointer s. If EOF is encountered, the fgets() function returns a null pointer and leaves the array untouched. If an error occurs, the function returns a null pointer, and the contents of the array are unknown.

The fgets() function can read up to n-1 characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered. Note that if a newline is encountered during the reading, the fgets() function includes the newline in the array. This is different from what the gets() function does. The gets() function just replaces the newline character with a null character. (The gets() function was introduced in Hour 13, "Manipulating Strings.")

The syntax for the fputs() function is

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

Here s points to the array that contains the characters to be written to a file associated with the file pointer stream. The const modifier indicates that the content of the array pointed to by s cannot be changed. (You learned about the const modifier in Hour 14, "Scope and Storage Classes in C.") If it fails, the fputs() function returns a nonzero value; otherwise, it returns zero.

Note that the character array must include a null character at the end as the terminator to the fputs() function. Also, unlike the puts() function, the fputs() function does not insert a newline character to the string written to a file. (The puts() function was introduced in Hour 13.)

We can modify the program in Listing 21.2 to read or write one character line at a time by calling the fgets() and fputs() functions. The modified version is shown in Listing 21.3.

TYPE
Listing 21.3. Reading and writing one character line at a time.

```
1:  /* 21L03.c: Reading and writing one line at a time */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL, MAX_LEN = 81};
5:
6:  void LineReadWrite(FILE *fin, FILE *fout);
7:
8:  main(void)
9:  {
10:   FILE *fptr1, *fptr2;
11:   char filename1[]= "outhaiku.txt";
12:   char filename2[]= "haiku.txt";
```

```
13:   int reval = SUCCESS;
14:
15:   if ((fptr1 = fopen(filename1, "w")) == NULL){
16:     printf("Cannot open %s for writing.\n", filename1);
17:     reval = FAIL;
18:   } else if ((fptr2 = fopen(filename2, "r")) == NULL){
19:     printf("Cannot open %s for reading.\n", filename2);
20:     reval = FAIL;
21:   } else {
22:     LineReadWrite(fptr2, fptr1);
23:     fclose(fptr1);
24:     fclose(fptr2);
25:   }
26:
27:   return reval;
28: }
29: /* function definition */
30: void LineReadWrite(FILE *fin, FILE *fout)
31: {
32:   char buff[MAX_LEN];
33:
34:   while (fgets(buff, MAX_LEN, fin) != NULL){
35:     fputs(buff, fout);
36:     printf("%s", buff);
37:   }
38: }
```

OUTPUT

Because the program in Listing 21.3 reads the same text file, haiku.txt, as the program in Listing 21.2 did, I get the same output on the screen:

```
C:\app>21L03
Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
  (1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
  (1729-1781)
C:\app>
```

ANALYSIS

From the program in Listing 21.3, you can see that a function called LineReadWrite() has replaced the CharReadWrite() function.

The definition of the LineReadWrite() function is shown in lines 30_38. The fgets() function is called repeatedly in a while loop to read one character line at a time from the haiku.txt text file, until it reaches the end of the text file. In line 34, the array name buff and the maximum number of the array elements MAX_LEN are passed to the fgets() function, along with the file pointer fin that is associated with the opened haiku.txt file.

Meanwhile, each line read by the fgets() function is written to another opened text file called outhaiku.txt that is associated with the file pointer fout. This is done by invoking the fputs() function in line 35.

The statement in line 36 prints the contents of each string on the screen so that you see the two pieces of Japanese verse after running the program in Listing 21.3. Also, you can view the outhaiku.txt file in a text editor to make sure that the contents of the haiku.txt file have been copied to the outhaiku.txt file.

One Block at a Time

SYNTAX
If you like, you can also read or write a block of data at a time. In C, there are two I/O functions, fread() and fwrite(), that can be used to perform block I/O operations. The fread() and fwrite() functions are mirror images of each other.

The syntax for the fread() function is

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

Here ptr is a pointer to an array in which the data is stored. size indicates the size of each array element. n specifies the number of elements to read. stream is a file pointer that is associated with the opened file for reading. size_t is an integral type defined in the header file stdio.h. The fread() function returns the number of elements actually read.

SYNTAX
The number of elements read by the fread() function should be equal to the value specified by the third argument to the function, unless an error occurs or an EOF (end-of-file) is encountered. The fread() function returns the number of elements that are actually read, if an error occurs or an EOF is encountered.

The syntax for the fwrite() function is

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

Here ptr references the array that contains the data to be written to an opened file pointed to by the file pointer stream. size indicates the size of each element in the array. n specifies the number of elements to be written. The fwrite() function returns the number of elements actually written.

If there is no error occurring, the number returned by fwrite() should be the same as the third argument in the function. The return value may be less than the specified value if an error occurs.

Note that it's the programmer's responsibility to ensure that the array is large enough to hold data for either the fread() function or the fwrite() function.

SYNTAX
In C, a function called feof() can be used to determine when the end of a file is encountered. This function is more useful when you're reading a binary file because the values of some bytes may be equal to the value of EOF. If you determine the end of a binary file by checking the value returned by fread(), you may end up at the wrong position. Using the feof() function helps you to avoid mistakes in determining the end of a file.

The syntax for the feof() function is

```
#include <stdio.h>
int feof(FILE *stream);
```

Here stream is the file pointer that is associated with an opened file. The feof() function returns 0 if the end of the file has not been reached; otherwise, it returns a nonzero integer.

TYPE

The program in Listing 21.4 demonstrates how to read and write one block of characters at a time by calling the fread() and fwrite() functions. In fact, the program in Listing 21.4 is another modified version of the program from Listing 21.2.

Listing 21.4. Reading and writing one block of characters at a time.

```
1:  /* 21L04.c: Reading and writing one block at a time */
2:  #include <stdio.h>
3:
4:  enum {SUCCESS, FAIL, MAX_LEN = 80};
5:
6:  void BlockReadWrite(FILE *fin, FILE *fout);
7:  int ErrorMsg(char *str);
8:
9:  main(void)
10: {
11:    FILE *fptr1, *fptr2;
12:    char filename1[]= "outhaiku.txt";
13:    char filename2[]= "haiku.txt";
14:    int reval = SUCCESS;
15:
16:    if ((fptr1 = fopen(filename1, "w")) == NULL){
17:       reval = ErrorMsg(filename1);
18:    } else if ((fptr2 = fopen(filename2, "r")) == NULL){
19:       reval = ErrorMsg(filename2);
20:    } else {
21:       BlockReadWrite(fptr2, fptr1);
22:       fclose(fptr1);
23:       fclose(fptr2);
24:    }
25:
26:    return reval;
27: }
28: /* function definition */
29: void BlockReadWrite(FILE *fin, FILE *fout)
30: {
31:    int num;
32:    char buff[MAX_LEN + 1];
33:
34:    while (!feof(fin)){
35:       num = fread(buff, sizeof(char), MAX_LEN, fin);
36:       buff[num * sizeof(char)] = `\0';  /* append a null character */
37:       printf("%s", buff);
38:       fwrite(buff, sizeof(char), num, fout);
39:    }
40: }
41: /* function definition */
42: int ErrorMsg(char *str)
43: {
44:    printf("Cannot open %s.\n", str);
45:    return FAIL;
46: }
```

OUTPUT

Again, I get the same output on the screen because the program in Listing 21.4 also reads the same text file, haiku.txt:

```
C:\app>21L04
Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
  (1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
  (1729-1781)
C:\app>
```

ANALYSIS

The purpose of the program in Listing 21.4 is to show you how to invoke the fread() and fwrite() functions in your program to perform block I/O operations. In List-ing 21.4, the haiku.txt file is read by the fread() function, and then the fwrite() function is used to write the contents read from haiku.txt to another file called outhaiku.txt. We call the two C I/O functions from our own function, BlockReadWrite().

From the definition of the BlockReadWrite() function in lines 29_40, you can see that a character array called buff is defined with the number of elements of MAX_LEN + 1 in line 32, although we only read MAX_LEN number of characters by calling the fread() function in line 35. The reason is that we append a null character in line 36 after the last character read so that we ensure the block of characters saved in buff is treated as a string and can be printed out on the screen properly by the printf() function in line 37.

The while loop, shown in lines 34_39, keeps calling the fread() function to read a character block with MAX_LEN elements, until the feof() function in line 34 returns 0, which means that the end of the text file has been reached. As shown in lines 35 and 38, we use the sizof operator to measure the size of the char data type because the elements in the buff array are all characters.

If everything goes smoothly, you should see the Japanese verses again on the screen or in the outhaiku.txt file after running the program in Listing 21.4.

Summary

In this lesson you've learned the following:

- In C, a file can refer to a disk file, a terminal, a printer, or a tape drive.
- The data flow you transfer from your program to a file, or vice versa, is called a stream.
- A stream is a series of ordered bytes.
- Not like a file, a stream is device-independent.
- There are two stream formats: text stream and binary stream.
- The file position indicator in the FILE structure points to the position in a file where data will be read from or written to.
- The fopen() function is used to open a file and associate a stream to the opened file.
- You can specify different modes for opening a file.
- The fclose() function is responsible for closing an opened file and disassociating a stream with the file.
- The fgetc() and fputc() functions read or write one character at a time.
- The fgets() and fputs() functions read or write one line at a time.
- The fread() and fwrite() functions read or write one block of data at a time.
- The feof() function can determine when the end of a file has been reached.
- In a binary file, the feof() function should be used to detect EOF.

In the next lesson you'll learn more about disk file I/O in C.

Q&A

Q What are the differences between a text stream and a binary stream?

A A text stream is a sequence of characters that may not have a one-to-one relationship with the data on the device. Text streams are normally used for textual data, which have a consistent appearance from one environment to another, or from one machine to another. A binary stream, on the other hand, is a sequence of bytes that has a one-to-one correspondence to those on the device. Binary streams are primarily used for nontextual data that is needed to keep the exact contents on the device.

Q Why do you need a file pointer?

A A file pointer is used to associate a stream with an opened file for reading or writing purposes. A pointer of the type FILE is called a file pointer. FILE is a typedef for a structure that contains overhead information about a disk file. A file pointer plays an important role in the communication between programs and disk files.

Q What does the fclose() function do before it closes an opened file?

A As you know, all high-level I/O operations are buffered. One of the jobs of the fclose() function is to flush data left in the buffer to ensure that no data is lost. For instance, when you finish writing several blocks of characters to an opened text file, you call the fclose() function to disassociate a specified stream and close the text file. The fclose() function will first flush all characters left in the buffer and write them into the text file before it closes the file. In this way, all characters you write to the file will be saved properly.

Q What is the difference between fgets() and gets()?

A The major difference between the fgets() and gets() functions is that the fgets() function includes a newline character in the array if the newline is encountered during the reading, whereas the gets() function just replaces the newline character with a null character.

Workshop

To help solidify your understanding of this 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. What do the following expressions do?

```
fopen("test.bin", "r+b")
fopen("test.txt" "a")
fopen("test.ini", "w+")
```

2. What's wrong with the following code segment?

```
FILE *fptr;
int c;
if ((fptr = fopen("test1.txt", "r")) == NULL){
  while ((c=fgetc(fptr)) != EOF){
    putchar(c);
  }
```

```
}
fclose(fptr);
```

3. What's wrong with the following code segment?

```
FILE *fptr;
int c;
if ((fptr = fopen("test2.txt", "r")) != NULL){
  while ((c=fgetc(fptr)) != EOF){
    fputc(c, fptr);
  }
  fclose(fptr);
}
```

4. What's wrong with the following code segment?

```
FILE *fptr1, *fptr2;
int c;
if ((fptr1 = fopen("test1.txt", "r")) != NULL){
  while ((c=fgetc(fptr1)) != EOF){
    putchar(c);
  }
}
fclose(fptr1);
if ((fptr2 = fopen("test2.txt", "w")) != NULL){
  while ((c=fgetc(fptr1)) != EOF){
    fputc(c, fptr2);
  }
}
fclose(fptr2);
```

Exercises

1. Write a program to read the text file haiku.txt and count the number of characters in the file. Also, print out the contents of the file and the total character number on the screen.
2. Write a program to receive a string entered by the user, and then save the string into a file with the name also given by the user.
3. Given the string "Disk file I/O is tricky.", write a program to write the string into a file called test_21.txt by writing one character at a time. Meanwhile, print out the string on the screen.
4. Rewrite exercise 3. This time, try to write one block of characters (that is, one string) at a time.

Previous | Table of Contents | Next