hapter 4

File Input/Output

Specific Learning Outcomes

The students must have the ability to:

1. Understand the concepts of files and streams;
2. Create and read data using sequential-access file processing;
3. Understand and use file handling techniques in the C programming language.

Learning Content

1. Declaring a file type variable;
2. Opening or Creating a File;
3. Closing a File;
4. Reading from a File;
5. Writing to a File;
6. Finding the EOF.

Files and Streams

C views each file simply as a sequential stream of bytes (Figure 5.1). Each file ends either with an end-of-file marker or at a specific byte number recorded in a system-maintained, administrative data structure—this is determined by each platform and is hidden from you.

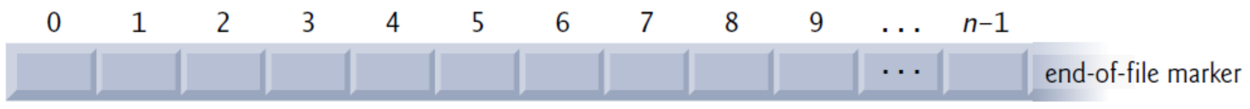


Figure 5.1

String-manipulation functions of the string-handling library

Standard Streams in Every Program

When a file is opened, a stream is associated with it. Three streams are automatically opened when program execution begins:

- the standard input (which receives input from the keyboard),
- the standard output (which displays output on the screen) and
- the standard error (which displays error messages on the screen).

Communication Channels

Streams provide communication channels between files and programs. For example, the standard input stream enables a program to read data from the keyboard, and the standard output stream enables a program to print data on the screen.

FILE Structure

Opening a file returns a pointer to a FILE structure (defined in <stdio.h>) that contains information used to process the file. In some operating systems, this structure includes a file descriptor, i.e., an integer index into an operating-system array called the open file table. Each array element contains a file control block (FCB)—information that the operating system uses to administer a particular file. The standard input, standard output and standard error are manipulated using stdin, stdout and stderr.

File-Processing Function fgetc

The standard library provides many functions for reading data from files and for writing data to files. Function fgetc, like getchar, reads one character from a file. Function fgetc receives as an argument a FILE pointer for the file from which a character will be read. The call fgetc(stdin) reads one character from stdin—the standard input. This call is equivalent to the call getchar( ).

File-Processing Function fputc

Function fputc, like putchar, writes one character to a file. Function fputc receives as arguments a character to be written and a pointer for the file to which the character will be written. The function call fputc('a', stdout) writes the character 'a' to stdout— the standard output. This call is equivalent to putchar('a').

Other File-Processing Functions

Several other functions used to read data from standard input and write data to standard output have similarly named file-processing functions. The fgets and fputs functions, for example, can be used to read a line from a file and write a line to a file, respectively. In the next several sections, we introduce the file-processing equivalents of functions scanf and printf—fscanf and fprintf. Later in the chapter we discuss functions fread and fwrite.

Creating a Sequential-Access File

C imposes no structure on a file. Thus, notions such as a record of a file are not part of the C language. The following example shows how you can impose your own record structure on a file. Program 32 creates a simple sequential-access file that might be used in an accounts receivable system to keep track of the amounts owed by a company's credit clients. For each client, the program obtains an account number, the client's name and the client's balance (i.e., the amount the client owes the company for goods and services received in the past). The data obtained for each client constitutes a "record" for that client. The account number is used as the record key in this application—the file will be created and maintained in account-number order. This program assumes the user enters the records in accountnumber order. In a comprehensive accounts receivable system, a sorting capability would be provided so the user could enter the records in any order. The records would then be sorted and written to the file. [Note: Program 33 and Program 34 use the data file created in Program 32, so you must run the program in Program 32 before programs 33 and 34]

```
// Program_34.c
// Creating a sequential file
#include <stdio.h>

int main(void)
{
    FILE *cfPtr; // cfPtr = clients.txt file pointer

    // fopen opens file. Exit program if unable to create file
    if (( ) == NULL) {
        puts("File could not be opened");
    }
    else {
```

```
14              puts("Enter the account, name, and balance.");
15              puts("Enter EOF to end input.");
16              printf("%s", "? ");
17
18              unsigned int account; // account number
19              char name[30]; // account name
20              double balance; // account balance
21
22              scanf("%d%29s%lf", &account, name, &balance);
23
24              // write account, name and balance into file with fprintf
25              while ( ) {
26                      fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
27                      printf("%s", "? ");
28                      scanf("%d%29s%lf", &account, name, &balance);
29              }
30
31              fclose(cfPtr); // fclose closes file
32      }
33 }
```

output

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Program 34 Creating a sequential file

Pointer to a FILE

Now let's examine this program. Line 7 states that cfPtr is a pointer to a FILE structure. A C program administers each file with a separate FILE structure. Each open file must have a separately declared pointer of type FILE that's used to refer to the file. You need not know the specifics of the FILE structure to use files, but you can study the declaration in stdio.h if you like. We'll soon see precisely how the FILE structure leads indirectly to the operating system's file control block (FCB) for a file.

Using fopen to Open the File

Line 10 names the file—"clients.txt"—to be used by the program and establishes a “line of communication” with the file. The file pointer cfPtr is assigned a pointer to the FILE structure for the file opened with fopen. Function fopen takes two arguments:

- a filename (which can include path information leading to the file’s location) and
- a file open mode.

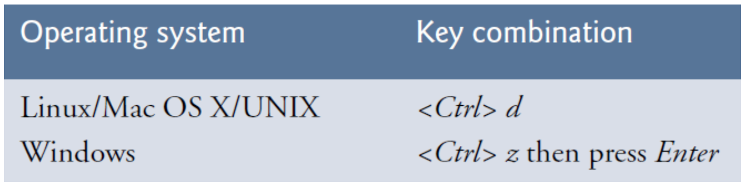
The file open mode "w" indicates that the file is to be opened for writing. If a file does not exist and it’s opened for writing, fopen creates the file. If an existing file is opened for writing, the contents of the file are discarded without warning. In the program, the if statement is used to determine whether the file pointer cfPtr is NULL (i.e., the file is not opened because it does not exist or the user does not have permission to open the file). If it’s NULL, the program prints an error message and terminates. Otherwise, the program processes the input and writes it to the file.

Common Programming Error. Opening an existing file for writing ("w") when, in fact, the user wants to preserve the file, discards the contents of the file without warning.

Common Programming Error. Forgetting to open a file before attempting to reference it in a program is a logic error.

Using feof to Check for the End-of-File Indicator

The program prompts the user to enter the various fields for each record or to enter endof-file when data entry is complete. Figure 5.2 lists the key combinations for entering end-of-file for various computer systems.



Operating system	Key combination
Linux/Mac OS X/UNIX	*<Ctrl> d*
Windows	*<Ctrl> z* then press *Enter*

Figure 5.2

End-of-file key combinations for various popular operating systems

Line 25 uses function feof to determine whether the end-of-file indicator is set for stdin. The end-of-file indicator informs the program that there’s no more data to be processed. In Program 32, the end-of-file indicator is set for the standard input when the user enters the end-of-file key combination. The argument to function feof is a pointer to the file being tested for the end-of-file indicator (stdin in this case).

The function returns a nonzero (true) value when the end-of-file indicator has been set; otherwise, the function returns zero. The while statement that includes the feof call in this program continues executing while the end-of-file indicator is not set.

Using fprintf to Write to the File

Line 26 writes data to the file clients.txt. The data may be retrieved later by a program designed to read the file. Function fprintf is equivalent to printf except that fprintf also receives as an argument a file pointer for the file to which the data will be written. Function fprintf can output data to the standard output by using stdout as the file pointer, as in:

```
fprintf(stdout, "%d %s %.2f\n", account, name, balance);
```

Using fclose to Close the File

After the user enters end-of-file, the program closes the clients.txt file with fclose (line 31) and terminates. Function fclose also receives the file pointer (rather than the filename) as an argument. If function fclose is not called explicitly, the operating system normally will close the file when program execution terminates. This is an example of operating system “housekeeping.”

Closing a file can free resources for which other users or programs may be waiting, so you should close each file as soon as it’s no longer needed rather than waiting for the operating system to close it at program termination.

In the sample execution for the program 34, the user enters information for five accounts, then enters end-of-file to signal that data entry is complete. The sample execution does not show how the data records actually appear in the file. To verify that the file has been created successfully, in the next section we present a program that reads the file and prints its contents.

Relationship Betweeen FILE Pointers, FILE Structures and FCBs

Figure 5.3 illustrates the relationship between FILE pointers, FILE structures and FCBs. When the file "clients.txt" is opened, an FCB for the file is copied into memory. The figure shows the connection between the file pointer returned by fopen and the FCB used by the operating system to administer the file. Programs may process no files, one file or several files. Each file used in a program will have a different file pointer returned by fopen. All subsequent file-processing functions after the file is opened must refer to the file with the appropriate file pointer.

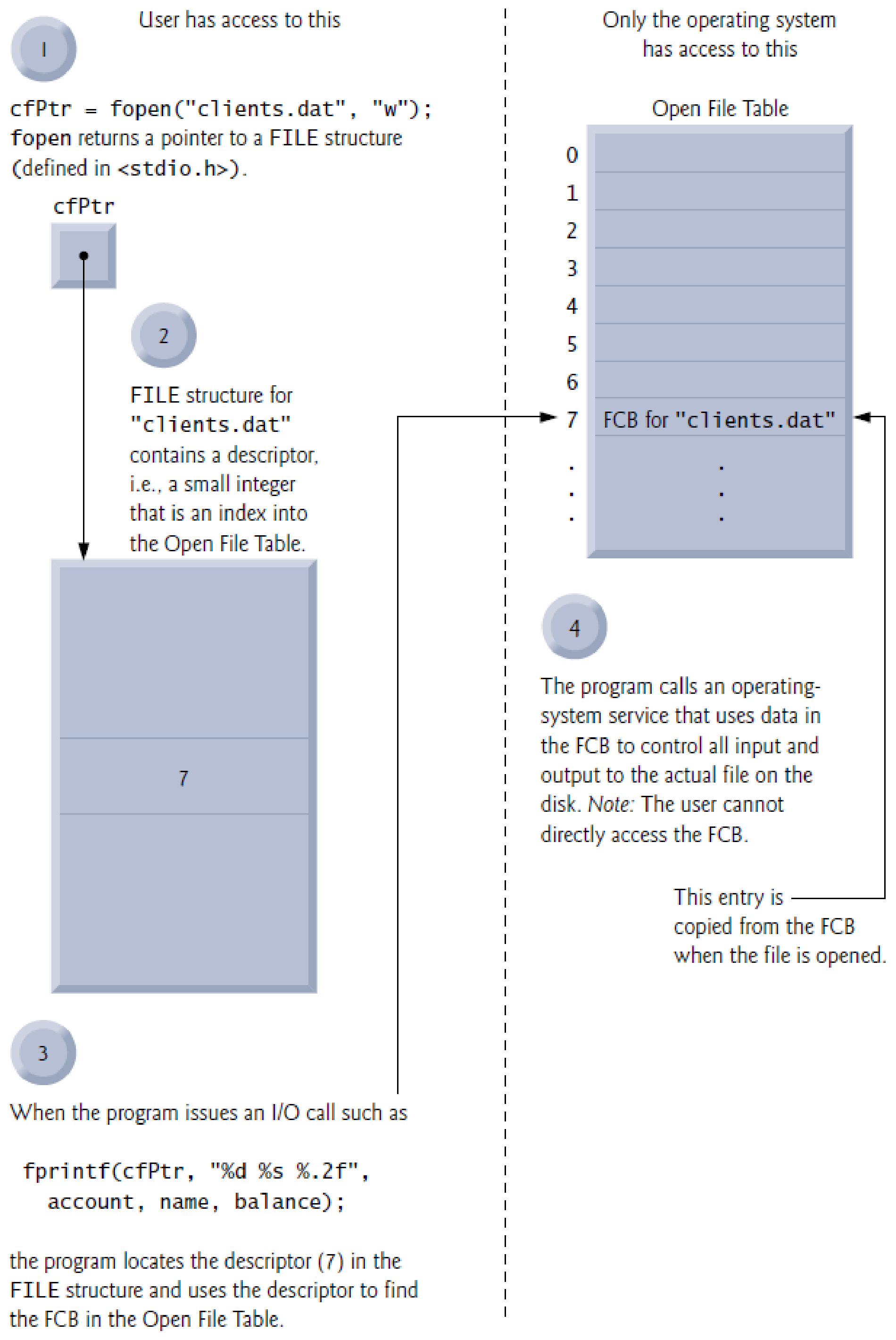
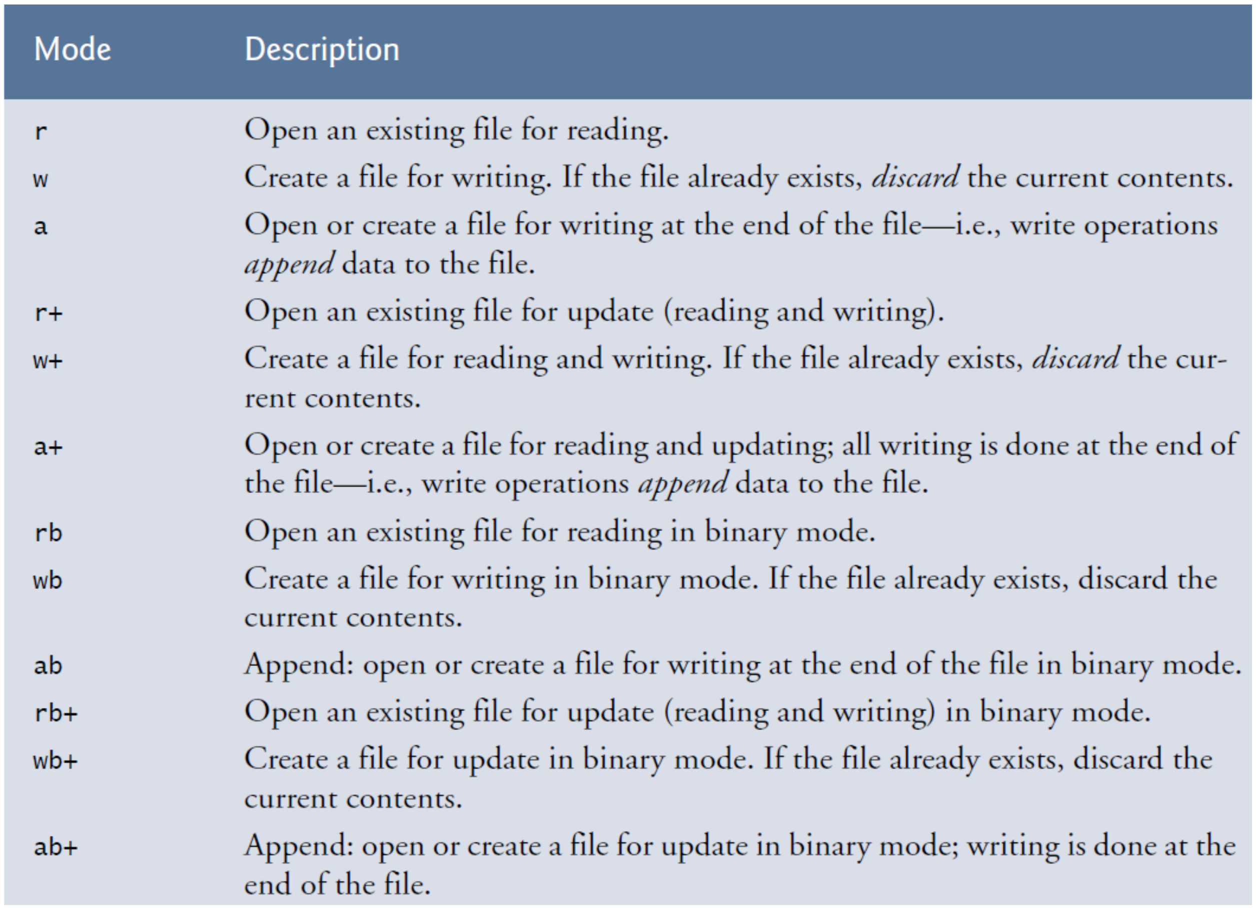

User has access to this
Only the operating system has access to this
1
cfPtr = fopen("clients.dat", "w");
fopen returns a pointer to a FILE structure (defined in <stdio.h>).
cfPtr
2
FILE structure for "clients.dat" contains a descriptor, i.e., a small integer that is an index into the Open File Table.
7
Open File Table
0
1
2
3
4
5
6
7
FCB for "clients.dat"
3
When the program issues an I/O call such as
fprintf(cfPtr, "%d %s %.2f", account, name, balance);
the program locates the descriptor (7) in the FILE structure and uses the descriptor to find the FCB in the Open File Table.
4
The program calls an operating-system service that uses data in the FCB to control all input and output to the actual file on the disk. Note: The user cannot directly access the FCB.
This entry is copied from the FCB when the file is opened.

Figure 5.3

Relationship between FILE pointers, FILE structures and FCBs

File Open Modes

Files may be opened in one of several modes, which are summarized in Figure 5.4. Each file open mode in the first half of the table has a corresponding binary mode (containing the letter b) for manipulating binary files.



Mode	Description
r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, *discard* the current contents.
a	Open or create a file for writing at the end of the file—i.e., write operations *append* data to the file.
r+	Open an existing file for update (reading and writing).
w+	Create a file for reading and writing. If the file already exists, *discard* the current contents.
a+	Open or create a file for reading and updating; all writing is done at the end of the file—i.e., write operations *append* data to the file.
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append: open or create a file for writing at the end of the file in binary mode.
rb+	Open an existing file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append: open or create a file for update in binary mode; writing is done at the end of the file.

Figure 5.4

File opening modes

C11 Exclusive Write Mode

In addition, C11 provides exclusive write mode, which you indicate by adding an x to the end of the w, w+, wb or wb+ modes. In exclusive write mode, fopen will fail if the file already exists or cannot be created. If opening a file in exclusive write mode is successful and the underlying system supports exclusive file access, then only your program can access the file while it's open. (Some compilers and platforms do not support exclusive write mode.) If an error occurs while opening a file in any mode, fopen returns NULL.

Common Programming Error. Opening a nonexistent file for reading is an error.

Common Programming Error. Opening a file for reading or writing without having been granted the appropriate access rights to the file (this is operating-system dependent) is an error.

Common Programming Error. Opening a file for writing when no space is available is a runtime error.

Common Programming Error. Opening a file in write mode ("w") when it should be opened in update mode ("r+") causes the contents of the file to be discarded.

Error-Prevention Tip. Open a file only for reading (and not updating) if its contents should not be modified. This prevents unintentional modification of the file's contents. This is another example of the principle of least privilege.

Reading Data from a Sequential-Access File

Data is stored in files so that it can be retrieved for processing when needed. The previous section demonstrated how to create a file for sequential access. This section shows how to read data sequentially from a file. Program 35 reads records from the file "clients.txt" created by the program 34 and prints their contents. Line 7 indicates that cfPtr is a pointer to a FILE. Line 10 attempts to open the file "clients.txt" for reading ("r") and determines whether it opened successfully (i.e., fopen does not return NULL). Line 19 reads a "record" from the file. Function fscanf is equivalent to function scanf, except fscanf receives as an argument a file pointer for the file from which the data is read. After this statement executes the first time, account will have the value 100, name will have the value "Jones" and balance will have the value 24.98. Each time the second fscanf statement (line 24) executes, the program reads another record from the file and account, name and balance take on new values. When the program reaches the end of the file, the file is closed (line 27) and the program terminates. Function feof returns true only after the program attempts to read the nonexistent data following the last line.

```
1 // Program_35.c
2 // Reading and printing a sequential file
3 #include <stdio.h>
4
5 int main(void)
6 {
7         FILE *cfPtr; // cfPtr = clients.txt file pointer
8
9         // fopen opens file; exits program if file cannot be opened
10        if (( cfPtr = fopen("clients.txt", "r")) == NULL) {
        puts("File could not be opened");
    }
    else { // read account, name and balance from file
        unsigned int account; // account number
        char name[30]; // account name
        double balance; // account balance

        printf("%-10s%-13s%s\n", "Account", "Name", "Balance");
        fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

        // while not end of file
        while ( ) {
            printf("%-10d%-13s%7.2f\n", account, name, balance);
            fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
        }

        fclose(cfPtr); // fclose closes the file
    }
}
```

output

```
Account     Name        Balance
100         Jones       24.98
200         Doe         345.67
300         White       0.00
400         Stone       -42.16
500         Rich        224.62
```

Program 35 Reading and printing a sequential file

Resetting the File Position Pointer

To retrieve data sequentially from a file, a program normally starts reading from the beginning of the file and reads all data consecutively until the desired data is found. It may be desirable to process the data sequentially in a file several times (from the beginning of the file) during the execution of a program. The statement

```
rewind(cfPtr);
```

causes a program's file position pointer—which indicates the number of the next byte in the file to be read or written—to be repositioned to the beginning of the file (i.e., byte 0) pointed to by cfPtr.

The file position pointer is not really a pointer. Rather it's an integer value that specifies the byte in the file at which the next read or write is to occur. This is sometimes referred to as the file offset. The file position pointer is a member of the FILE structure associated with each file.

Credit Inquiry Program

Program 36 allows a credit manager to obtain lists of customers with zero balances (i.e., customers who do not owe any money), customers with credit balances (i.e., customers to whom the company owes money) and customers with debit balances (i.e., customers who owe the company money for goods and services received). A credit balance is a negative amount; a debit balance is a positive amount. The program displays a menu and allows the credit manager to enter one of four options:

- Option 1 produces a list of accounts with zero balances.
- Option 2 produces a list of accounts with credit balances.
- Option 3 produces a list of accounts with debit balances.
- Option 4 terminates program execution.

```
// Program_36.c
// Credit inquiry program
#include <stdio.h>

// function main begins program execution
int main(void)
{
    FILE *cfPtr; // clients.txt file pointer

    // fopen opens the file; exits program if file cannot be opened
    if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
        puts("File could not be opened");
    }
    else {

        // display request options
        printf("%s", "Enter request\n"
            " 1 - List accounts with zero balances\n"
            " 2 - List accounts with credit balances\n"
            " 3 - List accounts with debit balances\n"
            " 4 - End of run\n? ");
        unsigned int request; // request number
        scanf("%u", &request);

        // process user's request
        while (request != 4) {
            unsigned int account; // account number
            double balance; // account balance
            char name[30]; // account name

            // read account, name and balance from file
            fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

            switch (request) {
                case 1:
                    puts("\nAccounts with zero balances:");

                    // read file contents (until eof)
                     while (!feof(cfPtr)) {
                        // output only if balance is 0
                        if (balance == 0) {
                            printf("%-10d%-13s%7.2f\n",
                                account, name, balance);
                        }

                        // read account, name and balance from file
                        fscanf(cfPtr, "%d%29s%lf",
                            &account, name, &balance);
                    }

                    break;
                case 2:
                    puts("\nAccounts with credit balances:\n");

                    // read file contents (until eof)
                    while (!feof(cfPtr)) {
                        // output only if balance is less than 0
                        if (balance < 0) {
                            printf("%-10d%-13s%7.2f\n",
                                account, name, balance);
                        }

                        // read account, name and balance from file
                        fscanf(cfPtr, "%d%29s%lf",
                            &account, name, &balance);
                    }

                    break;
                case 3:
                    puts("\nAccounts with debit balances:\n");

                    // read file contents (until eof)
                    while (!feof(cfPtr)) {
                        // output only if balance is greater than 0
                        if (balance > 0) {
                            printf("%-10d%-13s%7.2f\n",
                                account, name, balance);
                        }

                    // read account, name and balance from file
                    fscanf(cfPtr, "%d%29s%lf",
                        &account, name, &balance);
                }

                break;
            }
            rewind(cfPtr); // return cfPtr to beginning of file

            printf("%s", "\n? ");
            scanf("%d", &request);
        }

        puts("End of run.");
        fclose(cfPtr); // fclose closes the file
    }
}
```

Program 3 Credit inquiry program

output

```
Enter request
  1 - List accounts with zero balances
  2 - List accounts with credit balances
  3 - List accounts with debit balances
  4 - End of run
? 1
Accounts with zero balances:
300      White    0.00
? 2
Accounts with credit balances:
400      Stone    -42.16
? 3
Accounts with debit balances:
100      Jones    24.98
200      Doe      345.67
500      Rich     224.62
? 4
End of run.
```

Updating a Sequential File

Data in this type of sequential file cannot be modified without the risk of destroying other data. For example, if the name “White” needs to be changed to “Worthington,” the old name cannot simply be overwritten. The record for White was written to the file as

300 White 0.00

If the record is rewritten beginning at the same location in the file using the new name, the record will be

300 Worthington 0.00

The new record is larger (has more characters) than the original record. The characters beyond the second “o” in “Worthington” will overwrite the beginning of the next sequential record in the file. The problem here is that in the formatted input/output model using fprintf and fscanf, fields—and hence records—can vary in size. For example, the values 7, 14, –117, 2074 and 27383 are all ints stored in the same number of bytes internally, but they’re different-sized fields when displayed on the screen or written to a file as text.

Therefore, sequential access with fprintf and fscanf is not usually used to update records in place. Instead, the entire file is usually rewritten. To make the preceding name change, the records before 300 White 0.00 in such a sequential-access file would be copied to a new file, the new record would be written and the records after 300 White 0.00 would be copied to the new file. This requires processing every record in the file to update one record.

References

Paul Deitel, and Harvey Deitel, “C How to Program with an introduction to C++” Eight Edition

Kimberly Nelson King, “C Programming: A Modern Approach” Second Edition

Brian W. Kernighan and Dennis M. Ritchie, “C Programming Language” 2nd Edition

Jeri R. Hanly and Elliot B. Koffman, “Problem Solving and Program Design in C”, 8th Edition

https://www.thecrazyprogrammer.com

https://www.geeksforgeeks.org

Additional Resources and Links

Learn File Handling in C from
https://nesoacademy.page.link/watch?id=cs002_14

Chapter Assessment

Answer the following.

I. Fill in the blanks in each of the following:

a. Function _____ closes a file.
b. The ______ function reads data from a file in a manner similar to how scanf reads from stdin.
c. Function ______ reads a character from a specified file.
d. Function ______ reads a line from a specified file.
e. Function ______ opens a file.
f. Function ______ is normally used when reading data from a file in random-access applications.
g. Function ______ repositions the file position pointer to a specific location in the file.

II. State which of the following are true and which are false. If false, explain why.

a. Function fscanf cannot be used to read data from the standard input.
b. You must explicitly use fopen to open the standard input, standard output and standard error streams.
c. A program must explicitly call function fclose to close a file.
d. If the file position pointer points to a location in a sequential file other than the beginning of the file, the file must be closed and reopened to read from the beginning of the file.
e. Function fprintf can write to the standard output.
f. Data in sequential-access files is always updated without overwriting other data.

III. Write a single statement to accomplish each of the following. Assume that each of these statements applies to the same program.

a. Write a statement that opens the file "oldmast.dat" for reading and assigns the returned file pointer to ofPtr.
b. Write a statement that opens the file "trans.dat" for reading and assigns the returned file pointer to tfPtr.
c. Write a statement that opens the file "newmast.dat" for writing (and creation) and assigns the returned file pointer to nfPtr.
d. Write a statement that reads a record from the file "oldmast.dat". The record consists of integer accountNum, string name and floating-point currentBalance.
e. Write a statement that reads a record from the file "trans.dat". The record consists of the integer accountNum and floating-point dollarAmount.

f. Write a statement that writes a record to the file "newmast.dat". The record consists of the integer accountNum, string name and floating-point currentBalance.

IV. Find the error in each of the following program segments and explain how to correct it:

a. The file referred to by fPtr ("payables.dat") has not been opened.
```
printf(fPtr, "%d%s%d\n", account, company, amount);
```
b. open("receive.dat", "r+");
c. The following statement should read a record from the file "payables.dat". File pointer payPtr refers to this file, and file pointer recPtr refers to the file "receive.dat":
```
scanf(recPtr, "%d%s%d\n", &account, company, &amount);
```
d. The file "tools.dat" should be opened to add data to the file without discarding the current data.
```
if ((tfPtr = fopen("tools.dat", "w")) != NULL)
```
e. The file "courses.dat" should be opened for appending without modifying the current contents of the file.
```
if ((cfPtr = fopen("courses.dat", "w+")) != NULL)
```

Notes

Notes

Notes