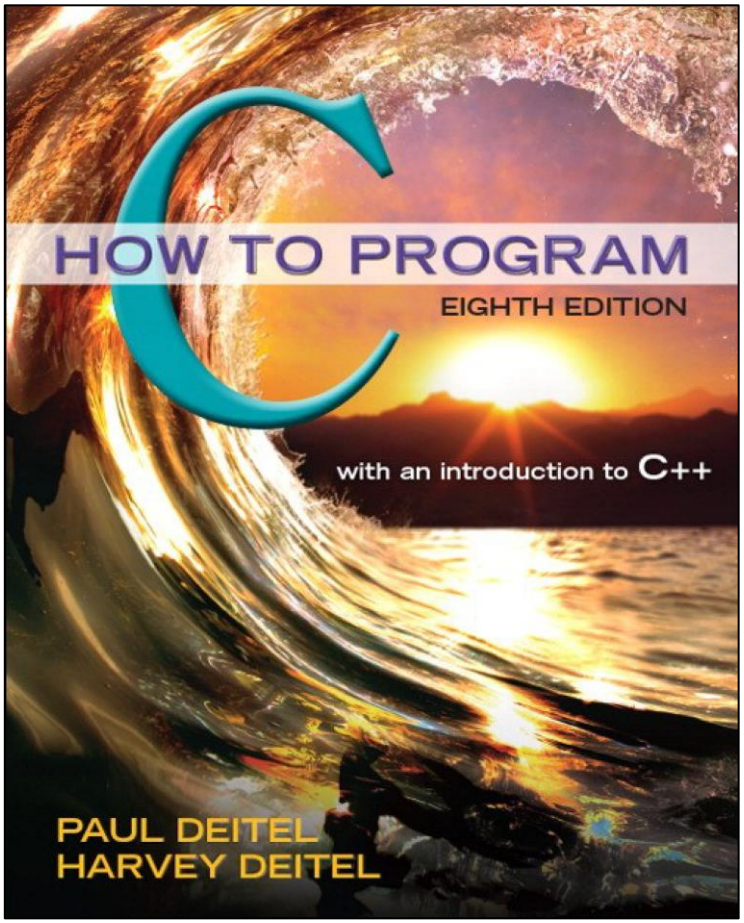
C How to Program

Eighth Edition



Chapter 11

C File Processing

Learning Objectives

- Understand the concepts of files and streams.
- Create and read data using sequential-access file processing.

Outline (1 of 2)

11.1 Introduction

- Storage of data in variables and arrays is temporary—such data is lost when a program terminates.
- **Files** are used for **permanent** retention of data.
- Computers store files on secondary storage devices, such as hard drives, CDs, DVDs and flash drives.
- In this chapter, we explain how data files are created, updated and processed by C programs.
- We both consider sequential-access and random-access file processing.

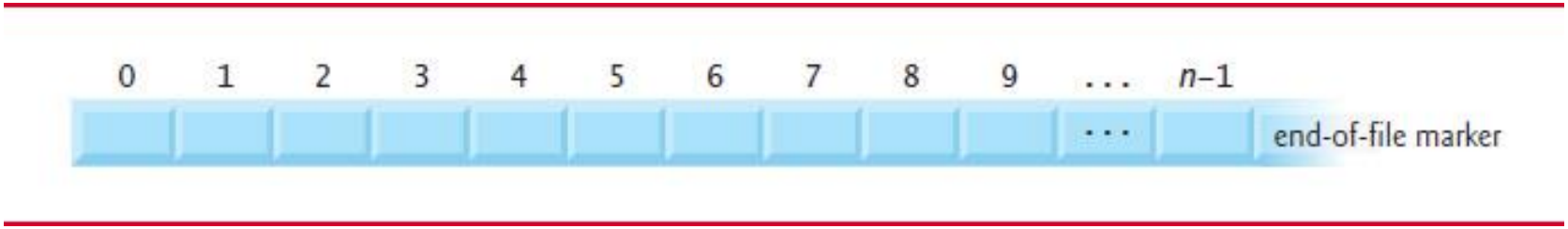
11.2 Files and Streams (1 of 4)

- C views each file simply as a sequential stream of bytes (Figure 11.1).
- Each file ends either with an **end-of-file marker** or at a specific byte number recorded in a system-maintained, administrative data structure.
- When a file is opened, a **stream** is associated with it.
- Three files and their associated streams are automatically opened when program execution begins—the **standard input**, the **standard output** and the **standard error**.
- Streams provide communication channels between files and programs.

11.2 Files and Streams (2 of 4)

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

Figure 11.1 C's View of a File of *n* Bytes



11.2 Files and Streams (3 of 4)

- 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()`**.
- 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.

11.2 Files and Streams (4 of 4)

- The function call `fputc('a', stdout)` writes the character `'a'` to `stdout`—the standard output.
- This call is equivalent to `putchar('a')`.
- 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`**.

11.3 Creating a Sequential-Access File (1 of 15)

- C imposes no structure on a file.
- Thus, notions such as a record of a file do not exist as part of the C language.
- The following example shows how you can impose your own record structure on a file.
- Figure 11.2 creates a simple sequential-access file that might be used in an accounts receivable system to help keep track of the amounts owed by a company's credit clients.

11.3 Creating a Sequential-Access File (2 of 15)

- 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.

11.3 Creating a Sequential-Access File (3 of 15)

- This program assumes the user enters the records in account-number 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:** Figures 11.6–11.7 use the data file created in Figure 11.2, so you must run Figure 11.2 before Figures 11.6–11.7.]

Figure 11.2 Creating a Sequential File (1 of 2)

```
1  // Fig. 11.2: fig11_02.c
2  // Creating 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. Exit program if unable to create file
10    if ((cfPtr = fopen("clients.txt", "w")) == NULL) {
11       puts("File could not be opened");
12    }
13    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);
```

Figure 11.2 Creating a Sequential File (2 of 2)

```
23
24          // write account, name and balance into file with fprintf
25          while (!feof(stdin)  ) {
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    }
```

```
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
```

11.3 Creating a Sequential-Access File (4 of 15)

- Now let's examine this program.
- `cfptr` is a **pointer to a `FILE` structure**.
- A C program administers each file with a separate `FILE` structure.
- 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.
- Each open file must have a separately declared pointer of type `FILE` that's used to refer to the file.

11.3 Creating a Sequential-Access File (5 of 15)

- The file name—`"clients.txt"`—is 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.

11.3 Creating a Sequential-Access File (6 of 15)

- 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).
- 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 11.1

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 11.2

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

11.3 Creating a Sequential-Access File (7 of 15)

- The program prompts the user to enter the fields for each record or to enter **end-of-file** when data entry is complete.
- Figure 11.3 lists the key combinations for entering end-of-file for various computer systems.
- Function **`feof`** to determine whether the end-of-file indicator is set for the file to which `stdin` refers.
- The **end-of-file** indicator informs the program that there's no more data to be processed.
- In Figure 11.2, 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).

Figure 11.3 End-Of-File Key Combinations for Various Popular Operating Systems

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

11.3 Creating a Sequential-Access File (8 of 15)

- 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.
- The data may be retrieved later by a program designed to read the file (see Section 11.4).

11.3 Creating a Sequential-Access File (9 of 15)

- 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);`

11.3 Creating a Sequential-Access File (10 of 15)

- After the user enters end-of-file, the program closes the `clients.txt` file with **`fclose`** 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."

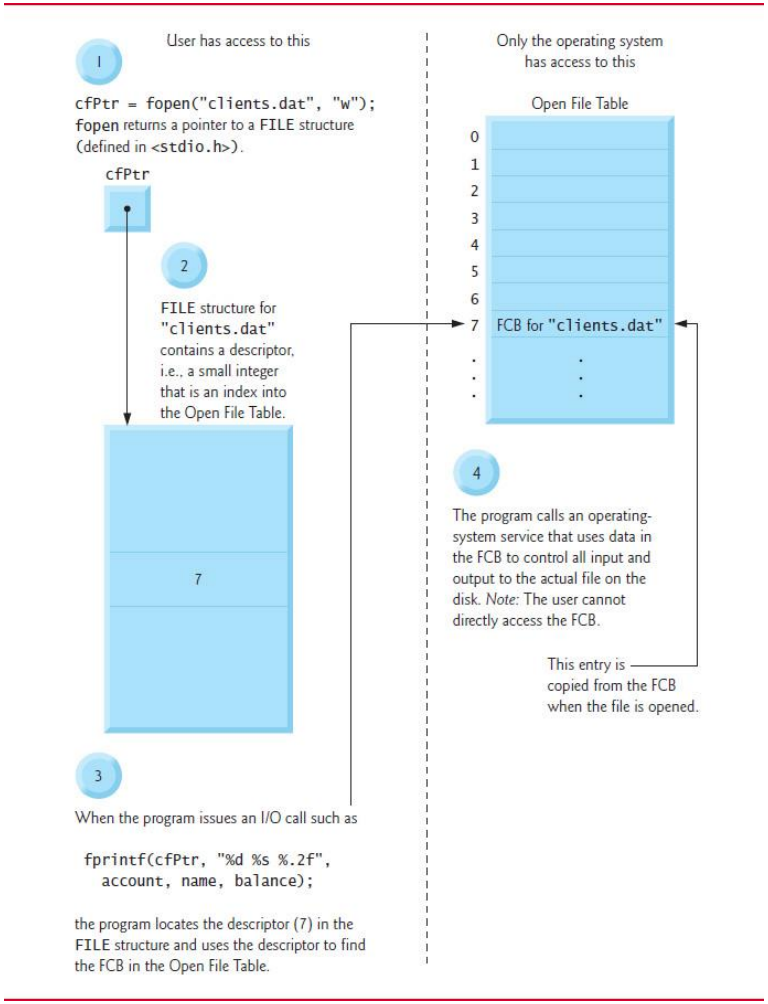
Performance Tip 11.1

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.

11.3 Creating a Sequential-Access File (11 of 15)

- In the sample execution for the program of Figure 11.3, 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.
- Figure 11.4 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.

Figure 11.4 Relationship Between `FILE` Pointers, `FILE` Structures and FCBs



11.3 Creating a Sequential-Access File (12 of 15)

- 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.
- Files may be opened in one of several modes (Figure 11.5)).
- To create a file, or to discard the contents of a file before writing data, open the file for writing (`"w"`).

11.3 Creating a Sequential-Access File (13 of 15)

- To read an existing file, open it for reading (`"r"`).
- To add records to the end of an existing file, open the file for appending (`"a"`).
- To open a file so that it may be written to and read from, open the file for updating in one of the three update modes—`"r+"`, `"w+"` or `"a+"`.
- Mode `"r+"` opens an existing file for reading and writing.
- Mode `"w+"` creates a file for reading and writing.
- If the file already exists, it's opened and its current contents are discarded.

11.3 Creating a Sequential-Access File (14 of 15)

- Mode `"a+"` opens a file for reading and writing—all writing is done at the end of the file.
- If the file does not exist, it's created.
- Each file open mode has a corresponding binary mode (containing the letter b) for manipulating binary files.
- The binary modes are used in Sections 11.5–11.9 when we introduce random-access files.
- In addition, C provides **exclusive** write mode, which you indicate by adding an x to the end of the `w`, `w+`, `wb` or `wb+` modes.

11.3 Creating a Sequential-Access File (15 of 15)

- 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`.

Figure 11.5 File Opening Modes (1 of 2)

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.

Figure 11.5 File Opening Modes (2 of 2)

Mode	Description
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.

Common Programming Error 11.3

Opening a nonexistent file for reading is an error.

Common Programming Error 11.4

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 11.5

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

Common Programming Error 11.6

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 11.1

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.

11.4 Reading Data from a Sequential-Access File (1 of 10)

- Data is stored in files so that the data 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.
- Figure 11.6 reads records from the file `"clients.dat"` created by the program of Figure 11.2 and prints their contents.
- `cfPtr` is a pointer to a `FILE`.
- We attempt to open the file `"clients.dat"` for reading `("r")` and determine whether it opened successfully (i.e., `fopen` does **not** return `NULL`).

11.4 Reading Data from a Sequential-Access File (2 of 10)

- Read a “record” from the file.
  - Function `fscanf` is equivalent to `scanf`, except `fscanf` receives a file pointer for the file being 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 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 and the program terminates.
- Function `feof` returns true only **after** the program attempts to read the nonexistent data following the last line.

Figure 11.6 Reading and Printing a Sequential File (1 of 2)

```
1  // Fig. 11.6: fig11_06.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) {
11       puts("File could not be opened");
12    }
13    else { // read account, name and balance from file
14       unsigned int account; // account number
15       char name[30]; // account name
16       double balance; // account balance
17
18       printf("%-10s%-13s%s\n", "Account", "Name", "Balance");
19       fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
20
```

Figure 11.6 Reading and Printing a Sequential File (2 of 2)

```
21          // while not end of file
22          while (!feof(cfPtr)  ) {
23             printf("%-10d%-13s%7.2f\n", account, name, balance);
24             fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
25          }
26
27          fclose(cfPtr); // fclose closes the file
28       }
29    }
```

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

11.4 Reading Data from a Sequential-Access File (3 of 10)

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.

11.4 Reading Data from a Sequential-Access File (4 of 10)

- 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.

11.4 Reading Data from a Sequential-Access File (5 of 10)

Credit Inquiry Program

- The program of Figure 11.7 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.

Figure 11.7 Credit Inquiry Program (1 of 6)

```
1  // Fig. 11.7: fig11_07.c
2  // Credit inquiry program
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     FILE *cfPtr; // clients.txt file pointer
9
10    // fopen opens the file; exits program if file cannot be opened
11    if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
12       puts("File could not be opened");
13    }
14    else {
15
16       // display request options
17       printf("%s", "Enter request\n"
18          " 1 - List accounts with zero balances\n"
19          " 2 - List accounts with credit balances\n"
20          " 3 - List accounts with debit balances\n"
21          " 4 - End of run\n? ");
22       unsigned int request; // request number
23       scanf("%u", &request);
24
```

Figure 11.7 Credit Inquiry Program (2 of 6)

```
25         // process user's request
26         while (request != 4) {
27            unsigned int account; // account number
28            double balance; // account balance
29            char name[30]; // account name
30
31            // read account, name and balance from file
32            fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
33
```

Figure 11.7 Credit Inquiry Program (3 of 6)

```
34          switch (request) {
35             case 1:
36                puts("\nAccounts with zero balances:");
37
38                // read file contents (until eof)
39                while (!feof(cfPtr)) {
40                   // output only if balance is 0
41                   if (balance == 0) {
42                      printf("%-10d%-13s%7.2f\n",
43                         account, name, balance);
44                   }
45
46                   // read account, name and balance from file
47                   fscanf(cfPtr, "%d%29s%lf",
48                      &account, name, &balance);
49                }
50
51                break;
```

Figure 11.7 Credit Inquiry Program (4 of 6)

```
52              case 2:
53                 puts("\nAccounts with credit balances:\n");
54
55                 // read file contents (until eof)
56                 while (!feof(cfPtr)) {
57                    // output only if balance is less than 0
58                    if (balance < 0) {
59                       printf("%-10d%-13s%7.2f\n",
60                          account, name, balance);
61                    }
62
63                    // read account, name and balance from file
64                    fscanf(cfPtr, "%d%29s%lf",
65                       &account, name, &balance);
66                 }
67
68                 break;
```

Figure 11.7 Credit Inquiry Program (5 of 6)

```
69              case 3:
70                 puts("\nAccounts with debit balances:\n");
71
72                 // read file contents (until eof)
73                 while (!feof(cfPtr)) {
74                    // output only if balance is greater than 0
75                    if (balance > 0) {
76                       printf("%-10d%-13s%7.2f\n",
77                          account, name, balance);
78                    }
79
80                    // read account, name and balance from file
81                    fscanf(cfPtr, "%d%29s%lf",
82                       &account, name, &balance);
83                 }
84
85                 break;
86           }
87
88           rewind(cfPtr); // return cfPtr to beginning of file
89
90           printf("%s", "\n? ");
91           scanf("%d", &request);
92        }
```

Figure 11.7 Credit Inquiry Program (6 of 6)

```
93
94          puts("End of run.");
95          fclose(cfPtr); // fclose closes the file
96       }
97    }
```

11.4 Reading Data from a Sequential-Access File (6 of 10)

- The program displays a menu and allows the credit manager to enter one of three options to obtain credit information.
- 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.
- A sample output is shown in Figure 11.8.

Figure 11.8 Sample Output of the Credit Inquiry Program of Figure 11.7 (see slides 47-52)

```
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.
```

11.4 Reading Data from a Sequential-Access File (7 of 10)

- 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

11.4 Reading Data from a Sequential-Access File (8 of 10)

- 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.

11.4 Reading Data from a Sequential-Access File (9 of 10)

- 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**.

11.4 Reading Data from a Sequential-Access File (10 of 10)

- 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.

Copyright