11

C File Processing

I read part of it all the way through.

—Samuel Goldwyn

Hats off!

The flag is passing by.

—Henry Holcomb Bennett

Consciousness … does not appear to itself chopped up in bits. … A "river" or a "stream" are the metaphors by which it is most naturally described.

—William James

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

—Senator Frank Church

OBJECTIVES

In this chapter you will learn:

- To create, read, write and update files.
- Sequential access file processing.
- Random-access file processing.

Outline

11.1 Introduction

- **Data files**
  - **Can be created, updated, and processed by C programs**
  - **Are used for permanent storage of large amounts of data**
    - **Storage of data in variables and arrays is only temporary**

11.2 Data Hierarchy

- **Data Hierarchy:**
  - **Bit – smallest data item**
    - **Value of `0` or `1`**
  - **Byte – 8 bits**
    - **Used to store a character**

      Decimal digits, letters, and special symbols
  - **Field – group of characters conveying meaning**
    - **Example: your name**
  - **Record – group of related fields**
    - **Represented by a `struct` or a `class`**
    - **Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.**

11.2 Data Hierarchy

- **Data Hierarchy (continued):**
  - **File – group of related records**
    - **Example: payroll file**
  - **Database – group of related files**

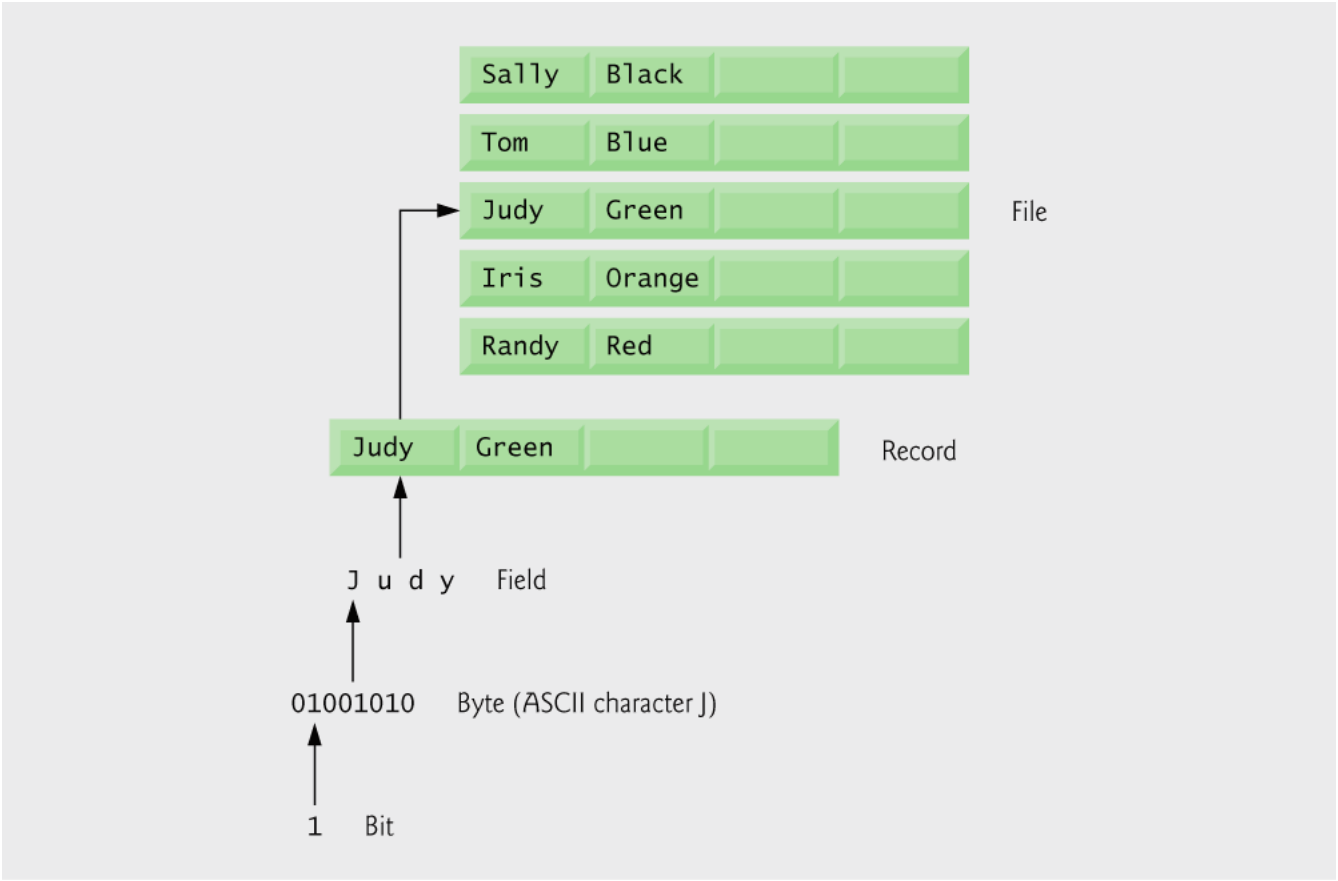


Fig. 11.1 | Data hierarchy.

11.2 Data Hierarchy

- **Data files**
  - **Record key**
    - **Identifies a record to facilitate the retrieval of specific records from a file**
  - **Sequential file**
    - **Records typically sorted by key**

11.3 Files and Streams

- C views each file as a sequence of bytes
  - File ends with the *end-of-file marker*
    - Or, file ends at a specified byte
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a `FILE` structure
    - Example file pointers:
    - `stdin` - standard input (keyboard)
    - `stdout` - standard output (screen)
    - `stderr` - standard error (screen)

11.3 Files and Streams

- **`FILE` structure**
  - **File descriptor**
    - **Index into operating system array called the open file table**
  - **File Control Block (FCB)**
    - **Found in every array element, system uses it to administer the file**

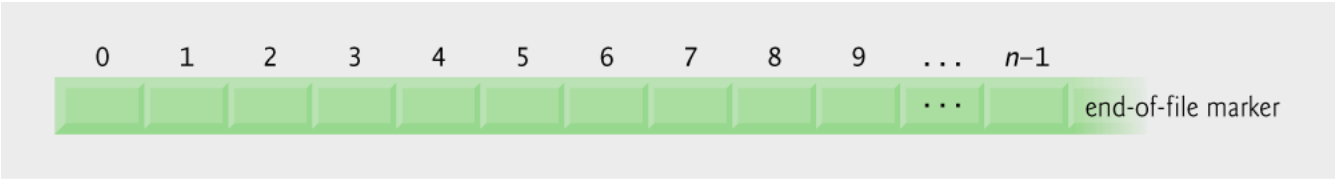


Fig. 11.2 | C's view of a file of *n* bytes.

11.3 Files and Streams

- **Read/Write functions in standard library**
  - `fgetc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc( stdin )` equivalent to `getchar()`
  - `fputc`
    - Writes one character to a file
    - Takes a `FILE` pointer and a character to write as an argument
    - `fputc( 'a', stdout )` equivalent to `putchar( 'a' )`
  - `fgets`
    - Reads a line from a file
  - `fputs`
    - Writes a line to a file
  - `fscanf` / `fprintf`
    - File processing equivalents of `scanf` and `printf`

11.4 Creating a Sequential-Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - `FILE *cfPtr;`
    - Creates a `FILE` pointer called `cfPtr`
  - `cfPtr = fopen("clients.dat", "w");`
    - Function `fopen` returns a `FILE` pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, `NULL` returned

11.4 Creating a Sequential-Access File

- `fprintf`
  - Used to print to a file
  - Like `printf`, except first argument is a `FILE` pointer (pointer to the file you want to print in)
- `feof( FILE pointer )`
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- `fclose( FILE pointer )`
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

- **Details**
  - Programs may process no files, one file, or many files
  - Each file must have a unique name and should have its own pointer

Outline

fig11_03.c

(1 of 2)

```
1  /* Fig. 11.3: fig11_03.c
2     Create a sequential file */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;     /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file. Exit program if unable to create file  */
14    if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else {
18       printf( "Enter the account, name, and balance.\n" );
19       printf( "Enter EOF to end input.\n" );
20       printf( "? " );
21       scanf( "%d%s%lf", &account, name, &balance );
22
```

FILE pointer definition creates new file pointer

fopen function opens a file; **w** argument means the file is opened for writing

Outline

```
23       /* write account, name and balance into file with fprintf */
24       while ( !feof( stdin ) ) {
25          fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26          printf( "? " );
27          scanf( "%d%s%lf", &account, name, &balance );
28       } /* end while */
29
30       fclose( cfPtr ); /* fclose closes file */
31    } /* end else */
32
33    return 0; /* indicates successful termination */
34
35 } /* end main */
```

feof returns true when end of file is reached

fprintf writes a string to a file

fclose closes a file

fig11_03.c

(2 of 2)

```
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
```

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.

Common Programming Error 11.3

Using the wrong file pointer to refer to a file is a logic error.

Error-Prevention Tip 11.1

Be sure that calls to file processing functions in a program contain the correct file pointers.

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

Fig. 11.4 | End-of-file key combinations for various popular operating systems.

Good Programming Practice 11.1

Explicitly close each file as soon as it is known that the program will not reference the file again.

Performance Tip 11.1

Closing a file can free resources for which other users or programs may be waiting.

Common Programming Error 11.4

Opening a nonexistent file for reading is an error.

Common Programming Error 11.5

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 11.6

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

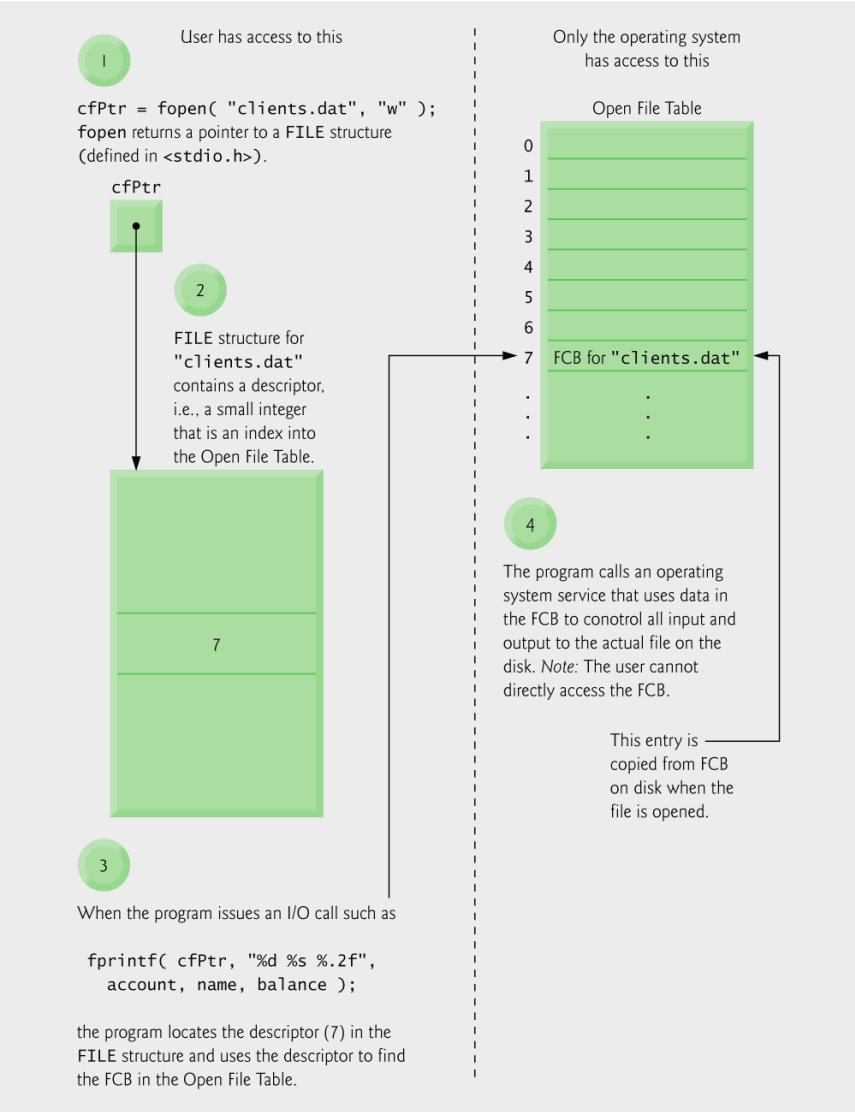


Fig. 11.5 | Relationship between **`FILE`** pointers, **`FILE`** structures and FCBs.

Common Programming Error 11.7

Opening a file with the incorrect file mode is a logic error. For example, 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.

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`	Append; open or create a file for writing at the end of the file.
`r+`	Open an existing file for update (reading and writing).
`w+`	Create a file for update. If the file already exists, discard the current contents.
`a+`	Append: open or create a file for update; writing is done at the end of 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.

Fig. 11.6 | File opening modes.

Error-Prevention Tip 11.2

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

11.5 Reading Data from a Sequential-Access File

- **Reading a sequential access file**
  - Create a `FILE` pointer, link it to the file to read
    ```
    cfPtr = fopen( “clients.dat", "r" );
    ```
  - Use `fscanf` to read from the file
    - Like `scanf`, except first argument is a `FILE` pointer
    ```
    fscanf( cfPtr, "%d%s%f", &accounnt, name, &balance );
    ```
  - Data read from beginning to end
  - File position pointer
    - Indicates number of next byte to be read / written
    - Not really a pointer, but an integer value (specifies byte location)
    - Also called byte offset
  - `rewind( cfPtr )`
    - Repositions file position pointer to beginning of file (byte `0`)

Outline

fig11_07.c

(1 of 2)

```
1  /* Fig. 11.7: fig11_07.c
2     Reading and printing a sequential file */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int account;     /* account number */
8     char name[ 30 ]; /* account name */
9     double balance;  /* account balance */
10
11    FILE *cfPtr;     /* cfPtr = clients.dat file pointer */
12
13    /* fopen opens file; exits program if file cannot be opened */
14    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15       printf( "File could not be opened\n" );
16    } /* end if */
17    else { /* read account, name and balance from file */
18       printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
19       fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
20
```

fopen function opens a file; **r** argument means the file is opened for reading

Outline

```
21       /* while not end of file */
22       while ( !feof( cfPtr ) ) {
23          printf( "%-10d%-13s%7.2f\n", account, name, balance );
24          fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
25       } /* end while */
26
27       fclose( cfPtr ); /* fclose closes the file */
28    } /* end else */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

fscanf function reads a string from a file

fig11_07.c

(2 of 2)

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

Outline

fig11_08.c

(1 of 4)

```c
/* Fig. 11.8: fig11_08.c
   Credit inquiry program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int request;     /* request number */
   int account;     /* account number */
   double balance;  /* account balance */
   char name[ 30 ]; /* account name */
   FILE *cfPtr;     /* clients.dat file pointer */

   /* fopen opens the file; exits program if file cannot be opened */
   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else {

      /* display request options */
      printf( "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? " );
      scanf( "%d", &request );

```

Outline

fig11_08.c

(2 of 4)

```
28       /* process user's request */
29       while ( request != 4 ) {
30
31          /* read account, name and balance from file */
32          fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
33
34          switch ( request ) {
35
36             case 1:
37                printf( "\nAccounts with zero balances:\n" );
38
39                /* read file contents (until eof) */
40                while ( !feof( cfPtr ) ) {
41
42                   if ( balance == 0 ) {
43                      printf( "%-10d%-13s%7.2f\n",
44                         account, name, balance );
45                   } /* end if */
46
47                   /* read account, name and balance from file */
48                   fscanf( cfPtr, "%d%s%lf",
49                      &account, name, &balance );
50                } /* end while */
51
52                break;
53
```

Outline

fig11_08.c

(3 of 4)

```
54          case 2:
55             printf( "\nAccounts with credit balances:\n" );
56
57             /* read file contents (until eof) */
58             while ( !feof( cfPtr ) ) {
59
60                if ( balance < 0 ) {
61                   printf( "%-10d%-13s%7.2f\n",
62                      account, name, balance );
63                } /* end if */
64
65                /* read account, name and balance from file */
66                fscanf( cfPtr, "%d%s%lf",
67                   &account, name, &balance );
68             } /* end while */
69
70             break;
71
72          case 3:
73             printf( "\nAccounts with debit balances:\n" );
74
75             /* read file contents (until eof) */
76             while ( !feof( cfPtr ) ) {
77
78                if ( balance > 0 ) {
79                   printf( "%-10d%-13s%7.2f\n",
80                      account, name, balance );
81                } /* end if */
82
```

Outline

fig11_08.c

(4 of 4)

```
83                /* read account, name and balance from file */
84                fscanf( cfPtr, "%d%s%lf",
85                   &account, name, &balance );
86             } /* end while */
87
88             break;
89
90          } /* end switch */
91
92          rewind( cfPtr ); /* return cfPtr to beginning of file */
93
94          printf( "\n? " );
95          scanf( "%d", &request );
96       } /* end while */
97
98       printf( "End of run.\n" );
99       fclose( cfPtr ); /* fclose closes the file */
100   } /* end else */
101
102   return 0; /* indicates successful termination */
103
104 } /* end main */
```

rewind function moves the file pointer back to the beginning of the file

Outline

```
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.
```

11.5 Reading Data from a Sequential-Access File

- **Sequential access file**
  - **Cannot be modified without the risk of destroying other data**
  - **Fields can vary in size**
    - **Different representation in files and screen than internal representation**
    - **`1`, `34`, `-890` are all `ints`, but have different sizes on disk**

11.6 Random-Access Files

- **Random access files**
  - **Access individual records without searching through other records**
  - **Instant access to records in a file**
  - **Data can be inserted without destroying other data**
  - **Data previously stored can be updated or deleted without overwriting**
- **Implemented using fixed length records**
  - **Sequential files do not have fixed length records**

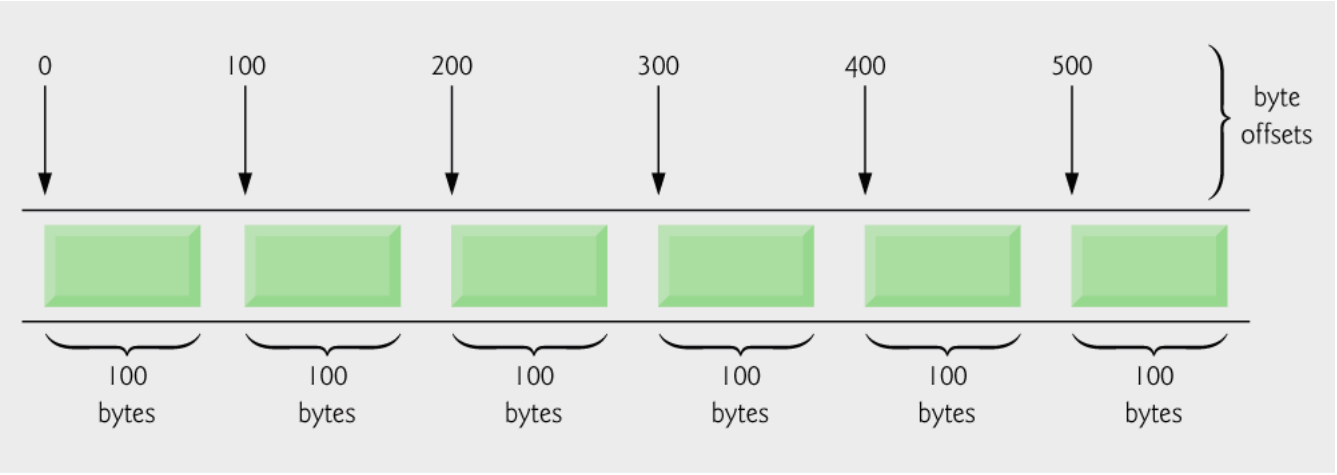


Fig. 11.10 | C's view of a random-access file.

11.7 Creating a Random-Access File

- **Data in random access files**
  - **Unformatted (stored as "raw bytes")**
    - **All data of the same type (`ints`, for example) uses the same amount of memory**
    - **All records of the same type have a fixed length**
    - **Data not human readable**

11.7 Creating a Random-Access File

- **Unformatted I/O functions**
  - `fwrite`
    - Transfer bytes from a location in memory to a file
  - `fread`
    - Transfer bytes from a file to a location in memory
  - Example:

```
fwrite( &number, sizeof( int ), 1, myPtr );
```

    - `&number` – Location to transfer bytes from
    - `sizeof( int )` – Number of bytes to transfer
    - `1` – For arrays, number of elements to transfer
      In this case, "one element" of an array is being transferred
    - `myPtr` – File to transfer to or from

11.7 Creating a Random-Access File

- **Writing `structs`**

```
fwrite( &myObject, sizeof (struct myStruct), 1, myPtr
  );
```

  - **`sizeof` – returns size in bytes of object in parentheses**
- **To write several array elements**
  - **Pointer to array as first argument**
  - **Number of elements to write as third argument**

Outline

fig11_11.c

(1 of 2)

```
1  /* Fig. 11.11: fig11_11.c
2     Creating a random-access file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;          /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;       /* account balance */
11 }; /* end structure clientData */
12
13 int main( void )
14 {
15    int i; /* counter used to count from 1-100 */
16
17    /* create clientData with default information */
18    struct clientData blankClient = { 0, "", "", 0.0 };
19
```

Outline

fig11_11.c

(2 of 2)

```
20    FILE *cfPtr; /* credit.dat file pointer */
21
22    /* fopen opens the file; exits if file cannot be opened */
23    if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
24       printf( "File could not be opened.\n" );
25    } /* end if */
26    else {
27
28       /* output 100 blank records to file */
29       for ( i = 1; i <= 100; i++ ) {
30          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
31       } /* end for */
32
33       fclose ( cfPtr ); /* fclose closes the file */
34    } /* end else */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

fopen function opens a file; **wb** argument means the file is opened for writing in binary mode

fwrite transfers bytes into a random-access file

11.8 Writing Data Randomly to a Random-Access File

- `fseek`
  - Sets file position pointer to a specific position
  - `fseek(` *pointer, offset, symbolic_constant* `);`
    - *pointer* – pointer to file
    - *offset* – file position pointer (0 is first location)
    - *symbolic_constant* – specifies where in file we are reading from
    - `SEEK_SET` – seek starts at beginning of file
    - `SEEK_CUR` – seek starts at current location in file
    - `SEEK_END` – seek starts at end of file

Outline

fig11_12.c

(1 of 2)

```
1  /* Fig. 11.12: fig11_12.c
2     Writing to a random access file */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;          /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;       /* account balance */
11 }; /* end structure clientData */
12
13 int main( void )
14 {
15    FILE *cfPtr; /* credit.dat file pointer */
16
17    /* create clientData with default information */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen opens the file; exits if file cannot be opened */
21    if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
22       printf( "File could not be opened.\n" );
23    } /* end if */
24    else {
25
26       /* require user to specify account number */
27       printf( "Enter account number"
28          " ( 1 to 100, 0 to end input )\n? " );
29       scanf( "%d", &client.acctNum );
30
```

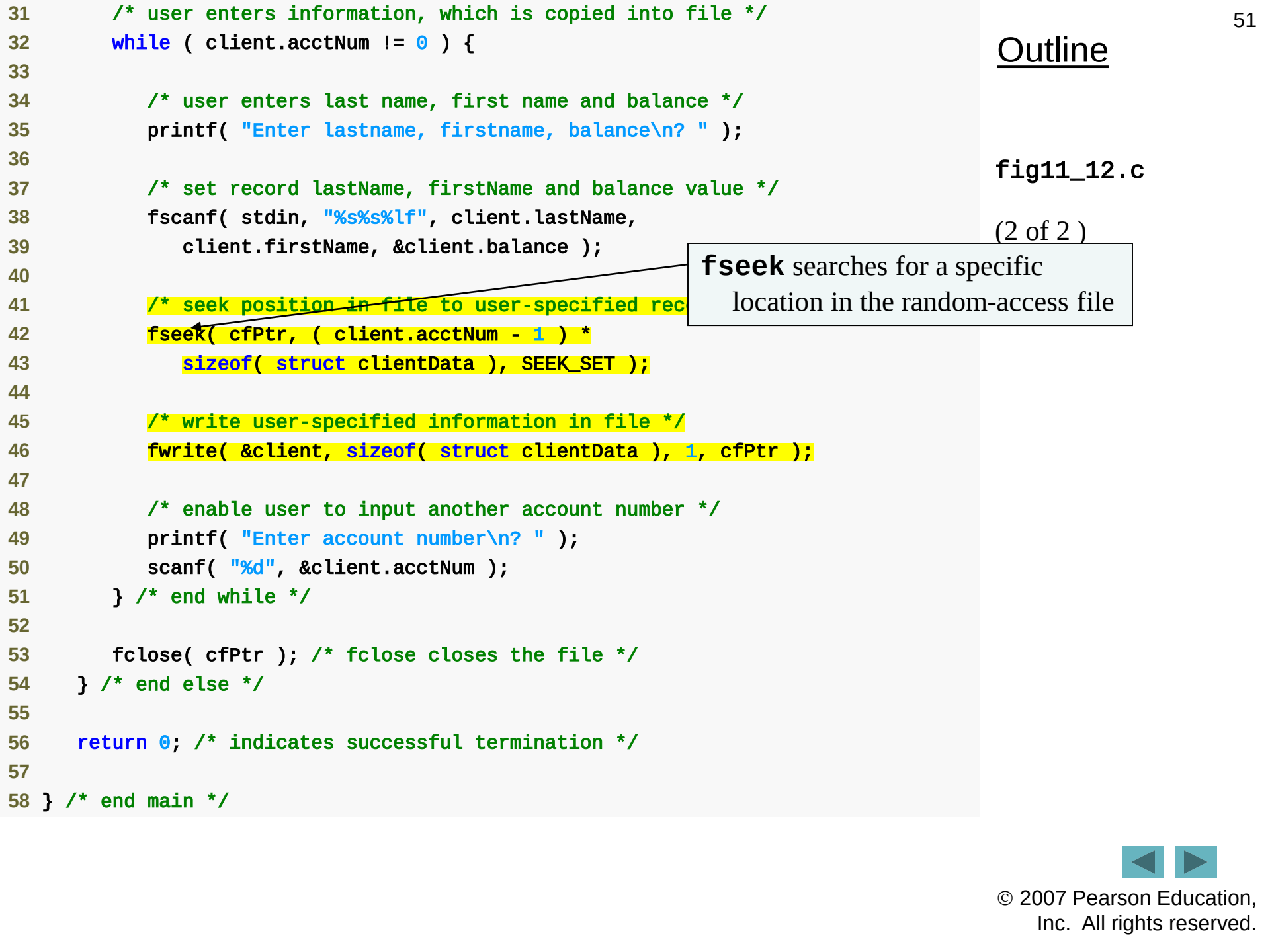
Outline

fig11_12.c

(2 of 2)

```
31       /* user enters information, which is copied into file */
32       while ( client.acctNum != 0 ) {
33
34          /* user enters last name, first name and balance */
35          printf( "Enter lastname, firstname, balance\n? " );
36
37          /* set record lastName, firstName and balance value */
38          fscanf( stdin, "%s%s%lf", client.lastName,
39             client.firstName, &client.balance );
40
41          /* seek position in file to user-specified rec
42          fseek( cfPtr, ( client.acctNum - 1 ) *
43             sizeof( struct clientData ), SEEK_SET );
44
45          /* write user-specified information in file */
46          fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
47
48          /* enable user to input another account number */
49          printf( "Enter account number\n? " );
50          scanf( "%d", &client.acctNum );
51       } /* end while */
52
53       fclose( cfPtr ); /* fclose closes the file */
54    } /* end else */
55
56    return 0; /* indicates successful termination */
57
58 } /* end main */
```

fseek searches for a specific location in the random-access file



```
Enter account number ( 1 to 100, 0 to end input )
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

Outline

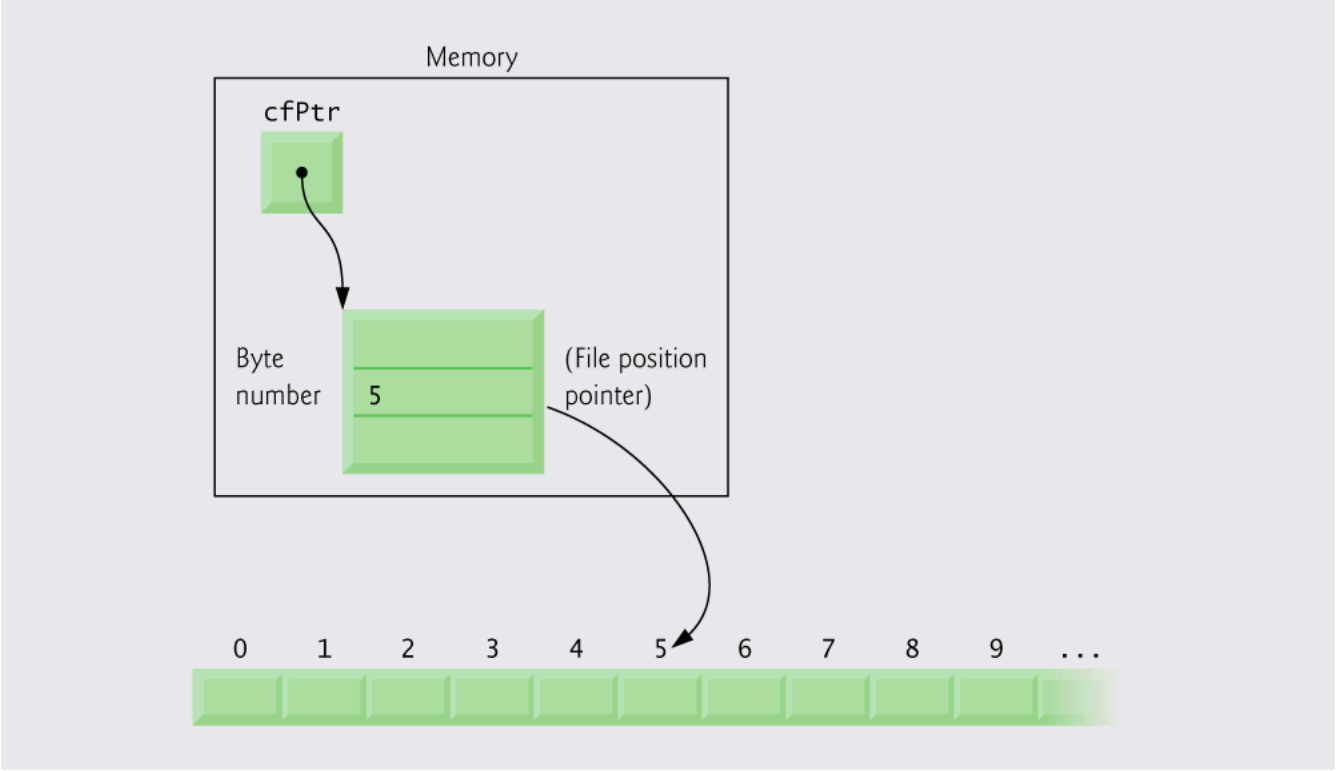


Fig. 11.14 | File position pointer indicating an offset of 5 bytes from the beginning of the file.

11.9 Reading Data from a Random-Access File

- `fread`
  - Reads a specified number of bytes from a file into memory

```
fread( &client, sizeof (struct clientData), 1,
  myPtr );
```

  - Can read several fixed-size array elements
    - Provide pointer to array
    - Indicate number of elements to read
  - To read multiple elements, specify in third argument

Outline

fig11_15.c

(1 of 2)

```
1  /* Fig. 11.15: fig11_15.c
2     Reading a random access file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;          /* account number */
8     char lastName[ 15 ];  /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;       /* account balance */
11 }; /* end structure clientData */
12
13 int main( void )
14 {
15    FILE *cfPtr; /* credit.dat file pointer */
16
17    /* create clientData with default information */
18    struct clientData client = { 0, "", "", 0.0 };
19
20    /* fopen opens the file; exits if file cannot be opened */
21    if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
22       printf( "File could not be opened.\n" );
23    } /* end if */
```

Outline

fig11_15.c

(2 of 2)

```
24    else {
25       printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
26          "First Name", "Balance" );
27
28       /* read all records from file (until eof) */
29       while ( !feof( cfPtr ) ) {
30          fread( &client, sizeof( struct clientData ), 1, cfPtr );
31
32          /* display record */
33          if ( client.acctNum != 0 ) {
34             printf( "%-6d%-16s%-11s%10.2f\n",
35                client.acctNum, client.lastName,
36                client.firstName, client.balance );
37          } /* end if */
38
39       } /* end while */
40
41       fclose( cfPtr ); /* fclose closes the file */
42    } /* end else */
43
44    return 0; /* indicates successful termination */
45
46 } /* end main */
```

fread reads bytes from a random-access file to a location in memory

```
Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

Outline

fig11_16.c

(1 of 10)

```
1  /* Fig. 11.16: fig11_16.c
2     This program reads a random access file sequentially, updates data
3     already written to the file, creates new data to be placed in the
4     file, and deletes data previously in the file. */
5  #include <stdio.h>
6
7  /* clientData structure definition */
8  struct clientData {
9     int acctNum;          /* account number */
10    char lastName[ 15 ];  /* account last name */
11    char firstName[ 10 ]; /* account first name */
12    double balance;       /* account balance */
13 }; /* end structure clientData */
14
15 /* prototypes */
16 int enterChoice( void );
17 void textFile( FILE *readPtr );
18 void updateRecord( FILE *fPtr );
19 void newRecord( FILE *fPtr );
20 void deleteRecord( FILE *fPtr );
21
22 int main( void )
23 {
24    FILE *cfPtr; /* credit.dat file pointer */
25    int choice;  /* user's choice */
26
27    /* fopen opens the file; exits if file cannot be opened */
28    if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
29       printf( "File could not be opened.\n" );
30    } /* end if */
```

Outline

fig11_16.c

(2 of 10)

```
31    else {
32
33       /* enable user to specify action */
34       while ( ( choice = enterChoice() ) != 5 ) {
35
36          switch ( choice ) {
37
38             /* create text file from record file */
39             case 1:
40                textFile( cfPtr );
41                break;
42
43             /* update record */
44             case 2:
45                updateRecord( cfPtr );
46                break;
47
48             /* create record */
49             case 3:
50                newRecord( cfPtr );
51                break;
52
53             /* delete existing record */
54             case 4:
55                deleteRecord( cfPtr );
56                break;
57
```

Outline

fig11_16.c

(3 of 10)

```
58            /* display message if user does not select valid choice */
59            default:
60               printf( "Incorrect choice\n" );
61               break;
62
63         } /* end switch */
64
65      } /* end while */
66
67      fclose( cfPtr ); /* fclose closes the file */
68   } /* end else */
69
70   return 0; /* indicates successful termination */
71
72 } /* end main */
73
74 /* create formatted text file for printing */
75 void textFile( FILE *readPtr )
76 {
77    FILE *writePtr; /* accounts.txt file pointer */
78
79    /* create clientData with default information */
80    struct clientData client = { 0, "", "", 0.0 };
81
82    /* fopen opens the file; exits if file cannot be opened */
83    if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
84       printf( "File could not be opened.\n" );
85    } /* end if */
```

Function **`textFile`** creates a text file containing all account data

Outline

fig11_16.c

(4 of 10)

```
86    else {
87       rewind( readPtr ); /* sets pointer to beginning of file */
88       fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
89          "Acct", "Last Name", "First Name","Balance" );
90
91       /* copy all records from random-access file into text file */
92       while ( !feof( readPtr ) ) {
93          fread( &client, sizeof( struct clientData ), 1, readPtr );
94
95          /* write single record to text file */
96          if ( client.acctNum != 0 ) {
97             fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
98                client.acctNum, client.lastName,
99                client.firstName, client.balance );
100         } /* end if */
101
102      } /* end while */
103
104      fclose( writePtr ); /* fclose closes the file */
105   } /* end else */
106
107 } /* end function textFile */
108
109  /* update balance in record */
110  void updateRecord( FILE *fPtr )
111 {
112    int account;        /* account number */
113    double transaction; /* transaction amount */
114
```

Function **updateRecord** changes the balance of a specified account

Outline

fig11_16.c

(5 of 10)

```
115   /* create clientData with no information */
116   struct clientData client = { 0, "", "", 0.0 };
117
118   /* obtain number of account to update */
119   printf( "Enter account to update ( 1 - 100 ): " );
120   scanf( "%d", &account );
121
122   /* move file pointer to correct record in file */
123   fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
124      SEEK_SET );
125
126   /* read record from file */
127   fread( &client, sizeof( struct clientData ), 1, fPtr );
128
129   /* display error if account does not exist */
130   if ( client.acctNum == 0 ) {
131      printf( "Acount #%d has no information.\n", account );
132   } /* end if */
133   else { /* update record */
134      printf( "%-6d%-16s%-11s%10.2f\n\n",
135         client.acctNum, client.lastName,
136         client.firstName, client.balance );
137
138      /* request transaction amount from user */
139      printf( "Enter charge ( + ) or payment ( - ): " );
140      scanf( "%lf", &transaction );
141      client.balance += transaction; /* update record balance */
142
```

Outline

fig11_16.c

(6 of 10)

```
143      printf( "%-6d%-16s%-11s%10.2f\n",
144         client.acctNum, client.lastName,
145         client.firstName, client.balance );
146
147      /* move file pointer to correct record in file */
148      fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
149         SEEK_SET );
150
151      /* write updated record over old record in file */
152      fwrite( &client, sizeof( struct clientData ), 1, fPtr );
153   } /* end else */
154
155 } /* end function updateRecord */
156
157  /* delete an existing record */
158  void deleteRecord( FILE *fPtr )
159 {
160
161   struct clientData client; /* stores record read from file */
162   struct clientData blankClient = { 0, "", "", 0 }; /* blank client */
163
164   int accountNum; /* account number */
165
166   /* obtain number of account to delete */
167   printf( "Enter account number to delete ( 1 - 100 ): " );
168   scanf( "%d", &accountNum );
169
```

Function **deleteRecord** removes an existing account from the file

Outline

fig11_16.c

(7 of 10)

```
170   /* move file pointer to correct record in file */
171   fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
172      SEEK_SET );
173
174   /* read record from file */
175   fread( &client, sizeof( struct clientData ), 1, fPtr );
176
177   /* display error if record does not exist */
178   if ( client.acctNum == 0 ) {
179      printf( "Account %d does not exist.\n", accountNum );
180   } /* end if */
181   else { /* delete record */
182
183      /* move file pointer to correct record in file */
184      fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
185         SEEK_SET );
186
187      /* replace existing record with blank record */
188      fwrite( &blankClient,
189         sizeof( struct clientData ), 1, fPtr );
190   } /* end else */
191
192 } /* end function deleteRecord */
193
```

Outline

```
194 /* create and insert record */
195 void newRecord( FILE *fPtr )
196 {
197   /* create clientData with default information */
198   struct clientData client = { 0, "", "", 0.0 };
199
200   int accountNum; /* account number */
201
202   /* obtain number of account to create */
203   printf( "Enter new account number ( 1 - 100 ): " );
204   scanf( "%d", &accountNum );
205
206   /* move file pointer to correct record in file */
207   fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
208      SEEK_SET );
209
210   /* read record from file */
211   fread( &client, sizeof( struct clientData ), 1, fPtr );
212
213   /* display error if account already exists */
214   if ( client.acctNum != 0 ) {
215      printf( "Account #%d already contains information.\n",
216         client.acctNum );
217   } /* end if */
```

Function **newRecord** adds a new account to the file

fig11_16.c

(8 of 10)

Outline

fig11_16.c

(9 of 10)

```
218   else { /* create record */
219
220      /* user enters last name, first name and balance */
221      printf( "Enter lastname, firstname, balance\n? " );
222      scanf( "%s%s%lf", &client.lastName, &client.firstName,
223         &client.balance );
224
225      client.acctNum = accountNum;
226
227      /* move file pointer to correct record in file */
228      fseek( fPtr, ( client.acctNum - 1 ) *
229         sizeof( struct clientData ), SEEK_SET );
230
231      /* insert record in file */
232      fwrite( &client,
233         sizeof( struct clientData ), 1, fPtr );
234   } /* end else */
235
236 } /* end function newRecord */
237
```

Outline

fig11_16.c

(10 of 10)

```
238 /* enable user to input menu choice */
239 int enterChoice( void )
240 {
241   int menuChoice; /* variable to store user's choice */
242
243   /* display available options */
244   printf( "\nEnter your choice\n"
245      "1 - store a formatted text file of acounts called\n"
246      "    \"accounts.txt\" for printing\n"
247      "2 - update an account\n"
248      "3 - add a new account\n"
249      "4 - delete an account\n"
250      "5 - end program\n? " );
251
252   scanf( "%d", &menuChoice ); /* receive choice from user */
253
254   return menuChoice;
255
256 } /* end function enterChoice */
```