Chapter 11
C File Processing

C How to Program, 8/e

11.2 Files and Streams

- C views each file simply as a sequential stream of bytes.
- 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 the file.

- Three files are automatically opened when program execution begins—the standard input, the standard output and the standard error.

```c
// Fig. 11.2: fig11_02.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 ((cfPtr = fopen("clients.txt", "w")) == NULL) {
      puts("File could not be opened");
   }
   else {
      puts("Enter the account, name, and balance.");
      puts("Enter EOF to end input.");
      printf("%s", "? ");

      unsigned int account; // account number
      char name[30]; // account name
      double balance; // account balance

      scanf("%d%29s%lf", &account, name, &balance);
```

宣告一檔案pointer

fopen開啟檔案

檔案讀寫性質
參見Fig. 11.5

若fopen傳
回NULL代
表開檔失敗

Fig. 11.2 | Creating a sequential file. (Part 1 of 2.)

EOF: End-Of-File

```
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    }
```

使用完關閉檔案

```
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
```

Fig. 11.2 | Creating a sequential file. (Part 2 of 2.)

11.4 Creating a Sequential-Access File

- `FILE *cfPtr;`
  - `cfPtr` is a pointer to a `FILE` structure.

- `cfPtr = fopen("client.dat", "w");`
  - Function `fopen` takes two arguments: a file name and a file open mode. (See Fig. 11.6)
  - `fopen` is defined in `<stdio.h>`

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.

Fig. 11.5 | File opening modes. (Part 1 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.

Fig. 11.5 | File opening modes. (Part 2 of 2.)

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

Fig. 11.3 | End-of-file key combinations for various popular operating systems.

11.3 Creating a Sequential-Access File (Cont.)

- Function `fprintf` is equivalent to `printf` except that the first argument of `fprintf` is a file pointer.

- 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 (Cont.)

- After the user enters end-of-file, the program closes the `clients.dat` file with `fclose` .
  - `fclose(cfptr);`

- If you do not close the file, OS will close it when the program terminates.

11.2 Files and Streams (Cont.)

- The standard library provides many functions for reading data from files and for writing data to files.

- Function `fgetc` receives as an argument a `FILE` pointer for the file from which a character will be read.
  - `fgetc( stdin )` reads one character from `stdin`—the standard input.
  - This call is equivalent to the call `getchar()`.

11.2 Files and Streams (Cont.)

- Function `fputc`, like `putchar`, writes one character to a file.
  - `fputc('a', stdout)`
  - This call is equivalent to `putchar('a')`.

- The `fgets` and `fputs` functions can be used to read a line from a file and write a line to a file, respectively.

11.4 Reading Data from a Sequential-Access File

```
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
```

Open clients.txt for reading

從檔案讀取

Fig. 11.6 | Reading and printing a sequential file. (Part 1 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
```

Fig. 11.6 | Reading and printing a sequential file. (Part 2 of 2.)

11.4 Reading Data from a Sequential-Access File (Cont.)

- A statement such as
  - `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`.

11.4 Reading Data from a Sequential-Access File (Cont.)

- Modifying a record in a sequential access file must move all the records following

- For example: `300 White 0.00` =>
  `300 Worthington 0.00`
  - Copy records before `300 White 0.00` to a new file.
  - Write the new record
  - Copy records after `300 White 0.00` to the new file.

11.5 Random-Access Files

- Individual records of a random-access file are normally fixed in length and may be accessed directly (and thus quickly) without searching through other records.

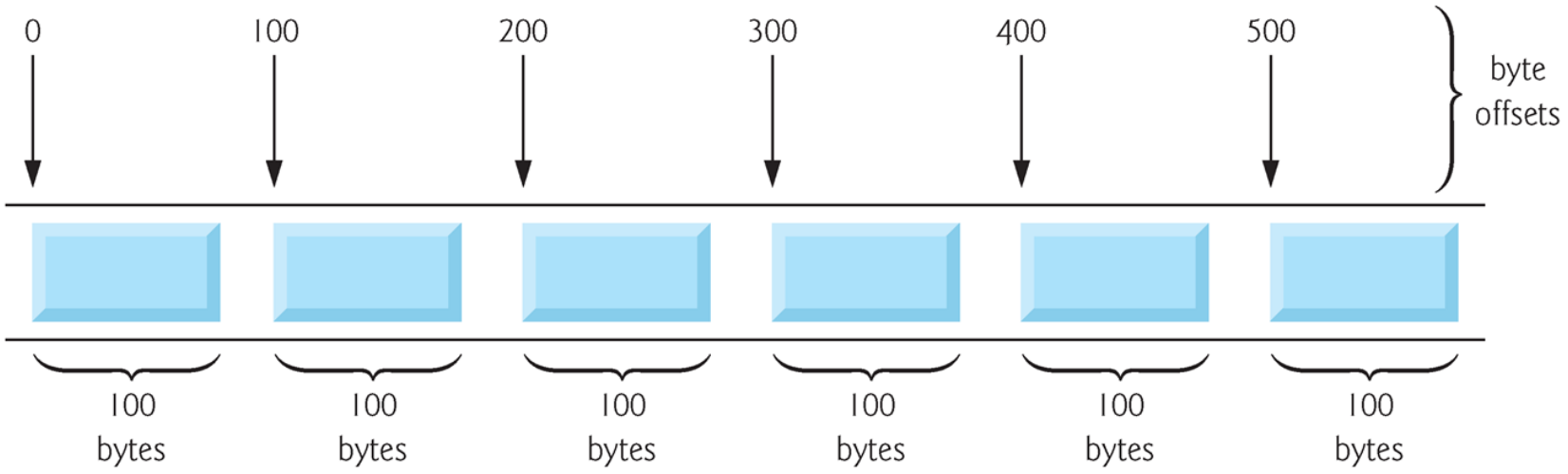

Fig. 11.9 | C's view of a random-access file.

11.6 Creating a Random-Access File

- Function `fwrite` transfers a specified number of bytes beginning at a specified location in memory to a file.
  - `int number;`
  - `fwrite( &number, sizeof( int ), 1, fPtr );`
    always writes 4 bytes from a variable `number` to the file represented by `fPtr`
  - `fprintf( fPtr, "%d", number );`
    prints a single digit or as many as 11 digits (10 digits plus a sign, each of which requires 1 byte of storage) for a 4-byte integer

11.6 Creating a Random-Access File

- Function `fread` transfers a specified number of bytes from the location in the file specified by the file position pointer to an area in memory beginning with a specified address.
  - `fread( &client, sizeof( struct int ), 1, cfPtr );`
    - The bytes are read from the location in the file specified by the file position pointer.

11.6 Creating a Random-Access File (Cont.)

- Consider the following problem statement:
  - Create a credit processing system capable of storing up to 100 fixed-length records. Each record should consist of an account number that will be used as the record key, a last name, a first name and a balance. The resulting program should be able to update an account, insert a new account record, delete an account and list all the account records in a formatted text file for printing. Use a random-access file.

```c
// Fig. 11.10: fig11_10.c
// Creating a random-access file sequentially
#include <stdio.h>

// clientData structure definition
struct clientData {
   unsigned int acctNum; // account number
   char lastName[15]; // account last name
   char firstName[10]; // account first name
   double balance; // account balance
};

int main(void)
{
   FILE *cfPtr; // accounts.dat file pointer

   // fopen opens the file; exits if file cannot be opened
   if ((cfPtr = fopen("accounts.dat", "wb")) == NULL) {
      puts("File could not be opened.");
   }
```

Fig. 11.10 | Creating a random-access file sequentially. (Part 1 of 2.)

```
21      else {
22         // create clientData with default information
23         struct clientData blankClient = {0, "", "", 0.0};
24
25         // output 100 blank records to file
26         for (unsigned int i = 1; i <= 100; ++i) {
27            fwrite(&blankClient, sizeof(struct clientData), 1, cfPtr);
28         }
29
30         fclose (cfPtr); // fclose closes the file
31      }
32   }
```

Fig. 11.10 | Creating a random-access file sequentially. (Part 2 of 2.)

11.7 Writing Data Randomly to a Random-Access File

- Figure 11.12 writes data to the file `"credit.dat"`.
- It uses the combination of `fseek` and `fwrite` to store data at specific locations in the file.
- Function `fseek` sets the file position pointer to a specific position in the file, then `fwrite` writes the data.

```c
// Fig. 11.11: fig11_11.c
// Writing data randomly to a random-access file
#include <stdio.h>

// clientData structure definition
struct clientData {
   unsigned int acctNum; // account number
   char lastName[15]; // account last name
   char firstName[10]; // account first name
   double balance; // account balance
}; // end structure clientData

int main(void)
{
   FILE *cfPtr; // accounts.dat file pointer

   // fopen opens the file; exits if file cannot be opened
   if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL) {
      puts("File could not be opened.");
   }
   else {
      // create clientData with default information
      struct clientData client = {0, "", "", 0.0};

```

Fig. 11.11 | Writing data randomly to a random-access file. (Part 1 of 3.)

```
25          // require user to specify account number
26          printf("%s", "Enter account number"
27             " (1 to 100, 0 to end input): ");
28          scanf("%d", &client.acctNum);
29
30          // user enters information, which is copied into file
31          while (client.acctNum != 0) {
32             // user enters last name, first name and balance
33             printf("%s", "\nEnter lastname, firstname, balance: ");
34
35             // set record lastName, firstName and balance value
36             fscanf(stdin, "%14s%9s%lf", client.lastName,
37                client.firstName, &client.balance);
38
39             // seek position in file to user-specified record
40             fseek(cfPtr, (client.acctNum - 1) *
41                sizeof(struct clientData), SEEK_SET);
42
43             // write user-specified information in file
44             fwrite(&client, sizeof(struct clientData), 1, cfPtr);
45
```

Fig. 11.11 | Writing data randomly to a random-access file. (Part 2 of 3.)

```
46            // enable user to input another account number
47            printf("%s", "\nEnter account number: ");
48            scanf("%d", &client.acctNum);
49         }
50
51         fclose(cfPtr); // fclose closes the file
52      }
53   }
```

Fig. 11.11 | Writing data randomly to a random-access file. (Part 3 of 3.)

```
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
```

Fig. 11.12 | Sample execution of the program in Fig. 11.11.

11.7 Writing Data Randomly to a Random-Access File (Cont.)

- `int fseek( FILE *stream, long int offset,int whence );`
  - `offset` is the number of bytes to seek from location
  - `whence` in the file pointed to by `stream`.
    - `SEEK_SET`, `SEEK_CUR` or `SEEK_END` (all defined in `<stdio.h>`)
    - Indicating the location in the file from which the seek begins.

11.7 Writing Data Randomly to a Random-Access File (Cont.)

- SEEK_SET indicates that the seek starts at the beginning of the file;
- SEEK_CUR indicates that the seek starts at the current location in the file;
- SEEK_END indicates that the seek starts at the end of the file.

11.7 Writing Data Randomly to a Random-Access File (Cont.)

- Function `fseek` returns a nonzero value if the seek operation cannot be performed.
- Function `fwrite` returns the number of items it successfully output.
  - If this number is less than the third argument in the function call, then a write error occurred.
- Function `fread` returns the number of items it successfully input.
  - If this number is less than the third argument in the function call, then a read error occurred

11.8 Reading Data from a Random-Access File (Cont.)

- Figure 11.14 reads sequentially every record in the `"credit.dat"` file, determines whether each record contains data and displays the formatted data for records containing data.

```c
// Fig. 11.14: fig11_14.c
// Reading a random-access file sequentially
#include <stdio.h>

// clientData structure definition
struct clientData {
   unsigned int acctNum; // account number
   char lastName[15]; // account last name
   char firstName[10]; // account first name
   double balance; // account balance
};

int main(void)
{
   FILE *cfPtr; // accounts.dat file pointer

   // fopen opens the file; exits if file cannot be opened
   if ((cfPtr = fopen("credit.txt", "rb")) == NULL) {
      puts("File could not be opened.");
   }
```

Fig. 11.14 | Reading a random-access file sequentially. (Part 1 of 3.)

```
21      else {
22         printf("%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
23            "First Name", "Balance");
24
25         // read all records from file (until eof)
26         while (!feof(cfPtr)) {
27            // create clientData with default information
28            struct clientData client = {0, "", "", 0.0};
29
30            int result = fread(&client, sizeof(struct clientData), 1, cfPtr);
31
32            // display record
33            if (result != 0 && client.acctNum != 0) {
34               printf("%-6d%-16s%-11s%10.2f\n",
35                  client.acctNum, client.lastName,
36                  client.firstName, client.balance);
37            }
38         }
39
40         fclose(cfPtr); // fclose closes the file
41      }
42   }
```

Fig. 11.14 | Reading a random-access file sequentially. (Part 2 of 3.)

```
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
```

Fig. 11.14 | Reading a random-access file sequentially. (Part 3 of 3.)