
C Programming for Engineers

File Handling



UNIVERSITY
AT ALBANY
State University of New York

ICEN 360– Spring 2017

Prof. Dola Saha

Files in C

- 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.
- Objective: how data files are created, updated and processed by C programs.
- We both consider **sequential-access** and **random-access** file processing.

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 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.

Text file vs Binary files

- Text file is a term used for a file that is essentially a sequence of character codes.
- Binary file is a term used for a file in which most bytes are not intended to be interpreted as character codes.
Here are a few common binary file formats:
 - PDF, for documents
 - JPEG, GIF, and PNG, for images
 - MP3, for audio tracks

Steps in processing a file

- Create the stream via a pointer variable using the **FILE** structure:
FILE *p;
- Open the file, associating the stream name with the file name.
- Read or write the data.
- Close the file.

Open the file: `fopen()`

➤ `FILE *fopen(const char *filename, const char *mode);`

Mode	Purpose	Stream Position
r	Read File exists	Beginning of file
r+	Read and write File exists	Beginning of file
w	Write If file exists, it is truncated to NULL, otherwise new created.	Beginning of file
w+	Write and read If file exists, it is truncated to NULL, otherwise new created.	Beginning of file
a	Append (write at end) File exists	End of file
a+	Read and append File exists	End of file

Opening Binary Files

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.

Functions to read and write data to file

➤ Function `fgetc`

- like `getchar`, reads one character from a file.
- 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.

➤ Function `fputc`,

- like `putchar`, writes one character to a file.
- receives as arguments a character to be written and a pointer for the file to which the character will be written.

Functions to read and write data to file

➤ Function `fgets`

- Reads one line from a file.
- `char *fgets(char *str, int n, FILE *stream)`

➤ Function `fputs`

- Writes one line to a file.
- `int fputs(const char *str, FILE *stream)`

Functions to read and write data to file

➤ Function `fprintf`

- Like `printf`
- Takes first argument as file pointer

➤ Function `fscanf`

- Like `scanf`
- Takes first argument as file pointer

Close the File: `fclose()`

- `int fclose(FILE * stream)`
- Returns 0 if successfully closed
- *If function `fclose` is not called explicitly, the operating system normally will close the file when program execution terminates.*

Create a sequential file ... (1)

```
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);
```



Create a sequential file ... (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



Read a record from 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    }
```



```

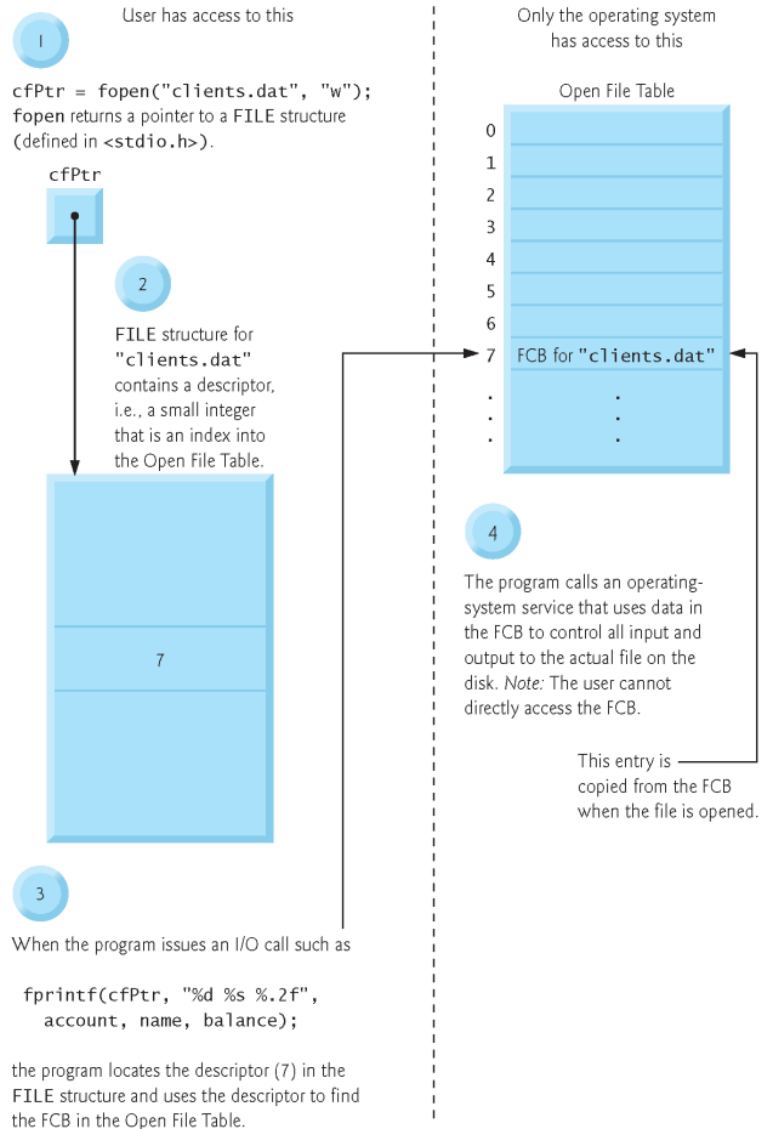
21 // while not end of file
22 while (!feof(cfPtr) ) {
23     printf("%-10d%-13s%7.2f\n", account, name, balance);
24     fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
25 }
26
27 fclose(cfPtr); // fclose closes the file
28 }
29 }

```

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



FILE Pointer



Operating System's File Control Block

Read a record from file.... (1)

```
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     }
```



Read a record from file.... (1)

```
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

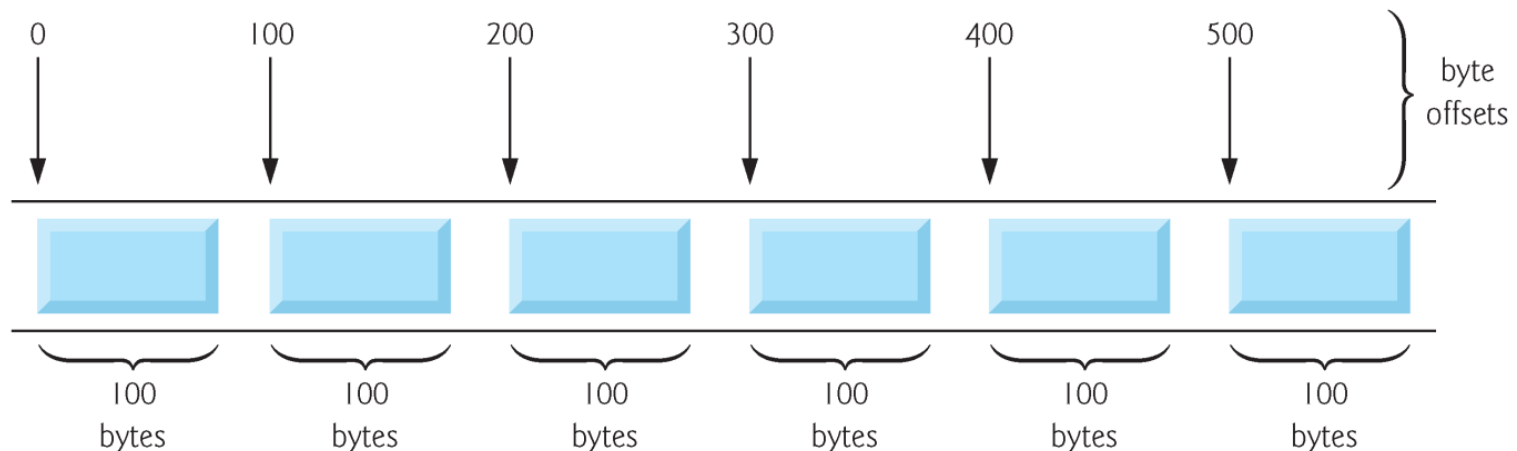


Reset a file position pointer

- 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.

Random Access File

- 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.
- Random-access files are appropriate for
 - airline reservation systems, banking systems, point-of-sale systems, and other kinds of transaction-processing systems that require rapid access to specific data.



Random Access File

- Fixed-length records enable data to be inserted in a random-access file *without destroying other data in the file*.
- Data stored previously can also be updated or deleted without rewriting the entire file.

fwrite()

➤ Example use

- `fprintf(fPtr, "%d", number);`

could print a single digit or as many as 11 digits (10 digits plus a sign, each of which requires 1 byte of storage)

➤ For a four-byte integer, we can use

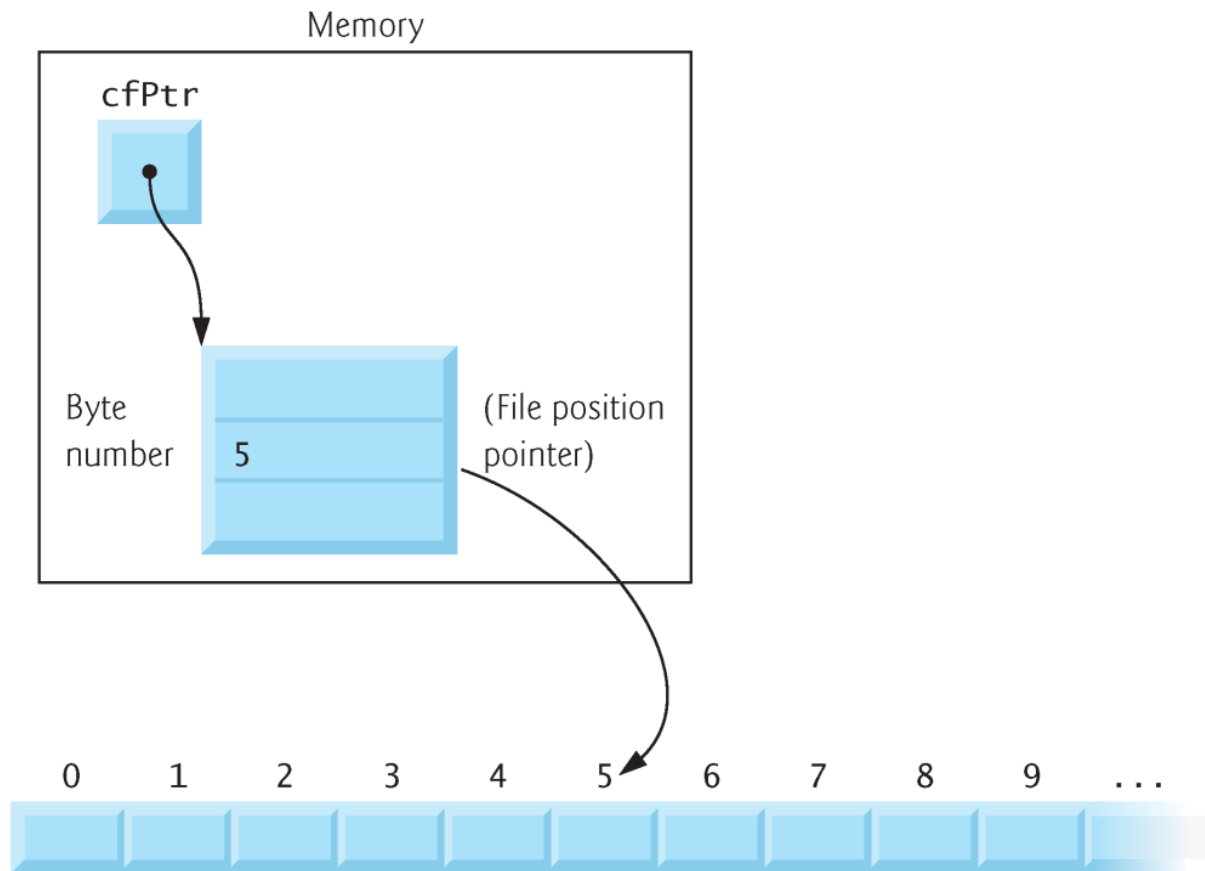
- `fwrite(&number, sizeof(int), 1, fPtr);`

which *always* writes four bytes on a system with four-byte integers from a variable number to the file represented by `fPtr`. 1 denotes one integer will be written.

fread()

- Function `fread` reads a specified number of bytes from a file into memory.
- For example,
 - `fread(&client, sizeof(struct clientData), 1, cfPtr);`
reads the number of bytes determined by `sizeof(struct clientData)` from the file referenced by `cfPtr`, stores the data in `client` and returns the number of bytes read.
- The bytes are read from the location specified by the file position pointer.

Random Access View



Moving to a location

➤ `fseek`

- `int fseek(FILE *stream, long int offset, int whence);`

- `offset` is the number of bytes to seek from
- `whence` in the file pointed to by `stream`—a positive `offset` seeks forward and a negative one seeks backward.

➤ Argument `whence` is one of the values

- `SEEK_SET`: Value 0, beginning of file.
- `SEEK_CUR`: Value 1, current position.
- `SEEK_END`: Value 2, end of file.

Random Access File Code

```
1  // Fig. 11.10: fig11_10.c
2  // Creating a random-access file sequentially
3  #include <stdio.h>
4
5  // clientData structure definition
6  struct clientData {
7      unsigned int acctNum; // account number
8      char lastName[15]; // account last name
9      char firstName[10]; // account first name
10     double balance; // account balance
11 };
12
13 int main(void)
14 {
15     FILE *cfPtr; // accounts.dat file pointer
16
17     // fopen opens the file; exits if file cannot be opened
18     if ((cfPtr = fopen("accounts.dat", "wb")) == NULL) {
19         puts("File could not be opened.");
20     }
```

Random Access File Code

```
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 }
```
