FILE PROCESSING

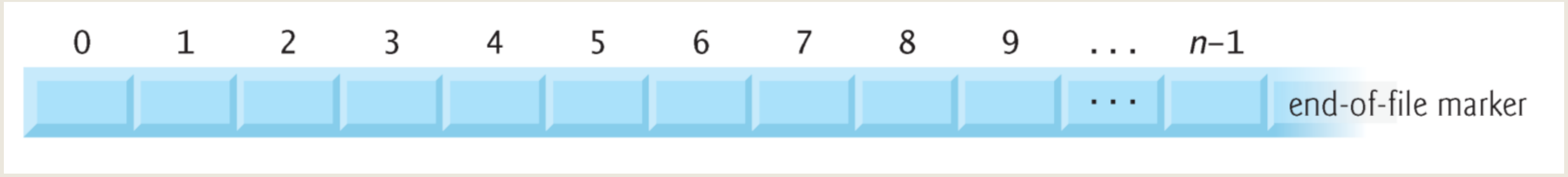
CEN116 Algorithms and Programming II

C File Processing

- 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

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.

Files and Streams

- Streams provide communication channels between files and programs.
- 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.

Files and Streams

- 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().**

Files and Streams

- 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')**.

Files and Streams

- 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**.

Creating a Sequential-Access File

- 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.
- Next figure 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.

Creating a Sequential-Access File

- 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.

Creating a Sequential-Access File

- 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.

```c
#include <stdio.h>

int main(void)
{
  int account;
  char name[30];
  float balance;
  FILE * fPtr;
  if ((fPtr = 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("? ");
    scanf("%d %29s %f", &account, name, &balance);
    while (!feof(stdin)) {
      fprintf(fPtr, "%d %s %.2f\n", account, name, balance);
      printf("? ");
      scanf("%d %29s %f", &account, name, &balance);
    }
    fclose(fPtr);
  }
}
```

Creating a Sequential-Access File

- fptr 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.

Creating a Sequential-Access File

- The file name—"clients.txt"—is used by the program and establishes a “line of communication” with the file.
- The file pointer fPtr 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.

Creating a Sequential-Access 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 fPtr 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.

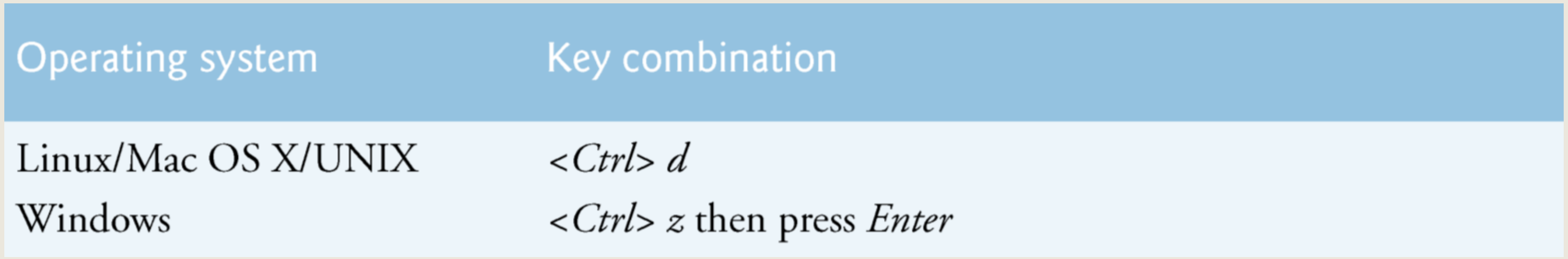
Creating a Sequential-Access File

- The program prompts the user to enter the fields for each record or to enter end-of-file when data entry is complete.
- 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.
- 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).

Creating a Sequential-Access File

- The program prompts the user to enter the fields for each record or to enter end-of-file when data entry is complete.
- 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.
- 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).

Creating a Sequential-Access File



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

Creating a Sequential-Access File

- 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.

Creating a Sequential-Access 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);*

Creating a Sequential-Access File

- After the user enters end-of-file, the program closes the clients.dat 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.

Creating a Sequential-Access 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.
- To create a file, or to discard the contents of a file before writing data, open the file for writing ("w").

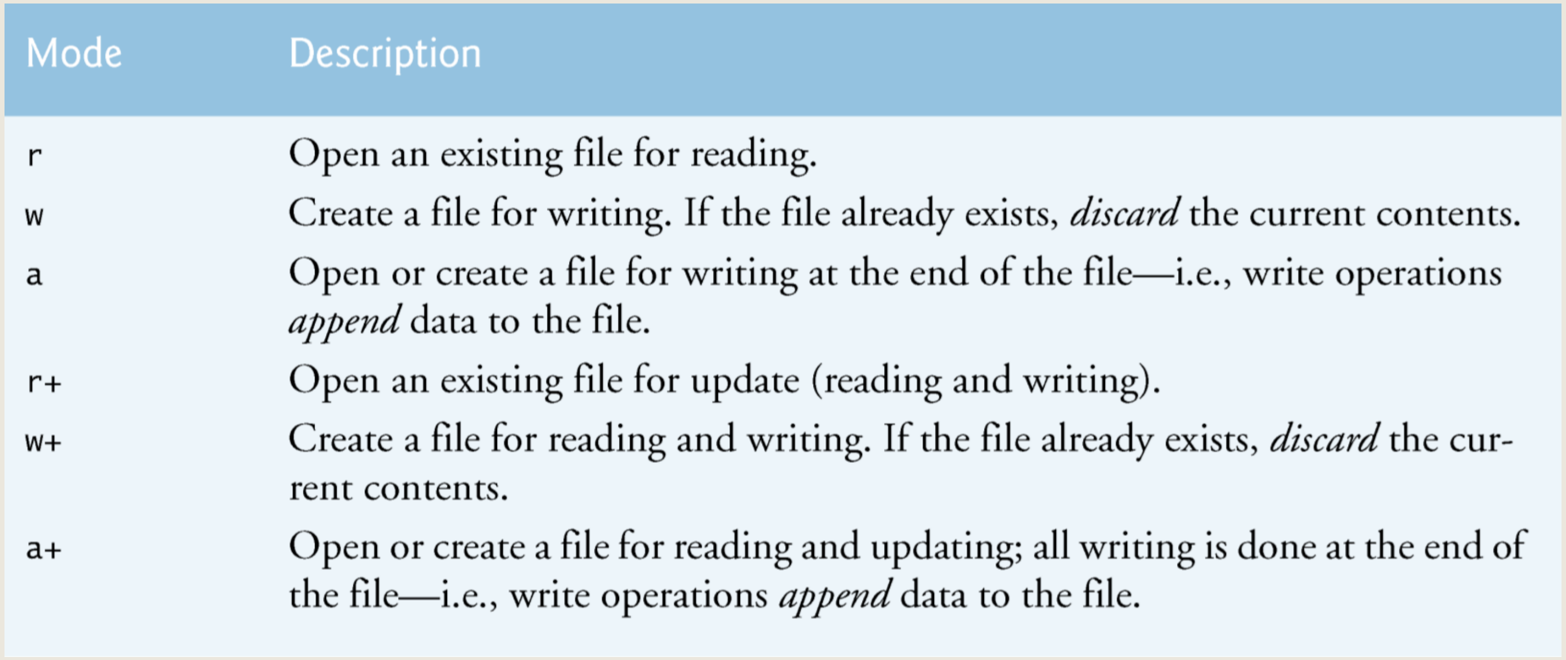
Creating a Sequential-Access File

- 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.

Creating a Sequential-Access File

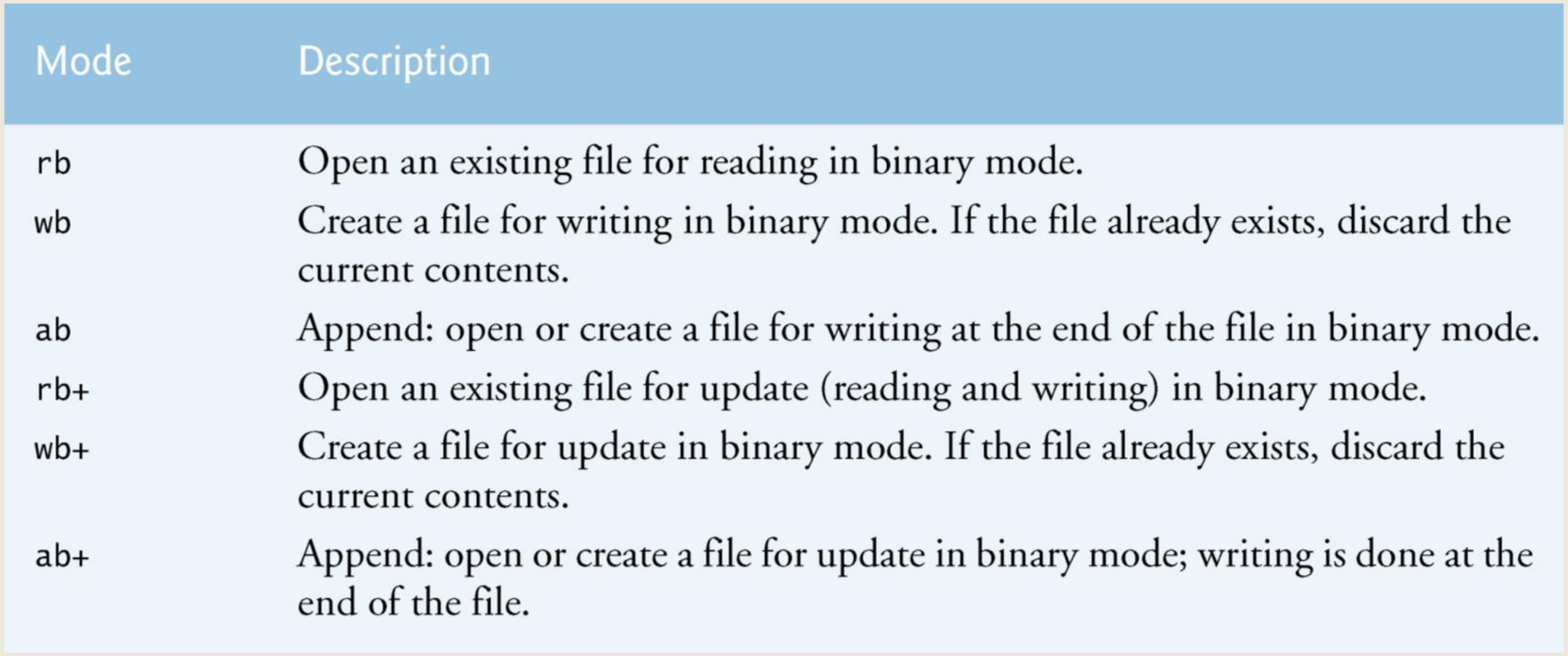
- Mode "a+" opens a file for reading and writing—all writing is done at the end of the file.
- If the file does not exist, it's created.
- Each file open mode has a corresponding binary mode (containing the letter b) for manipulating binary files.
- The binary modes are used in random access when we introduce random-access files.

Creating a Sequential-Access File



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.

Creating a Sequential-Access File



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.

Reading Data from a Sequential-Access File

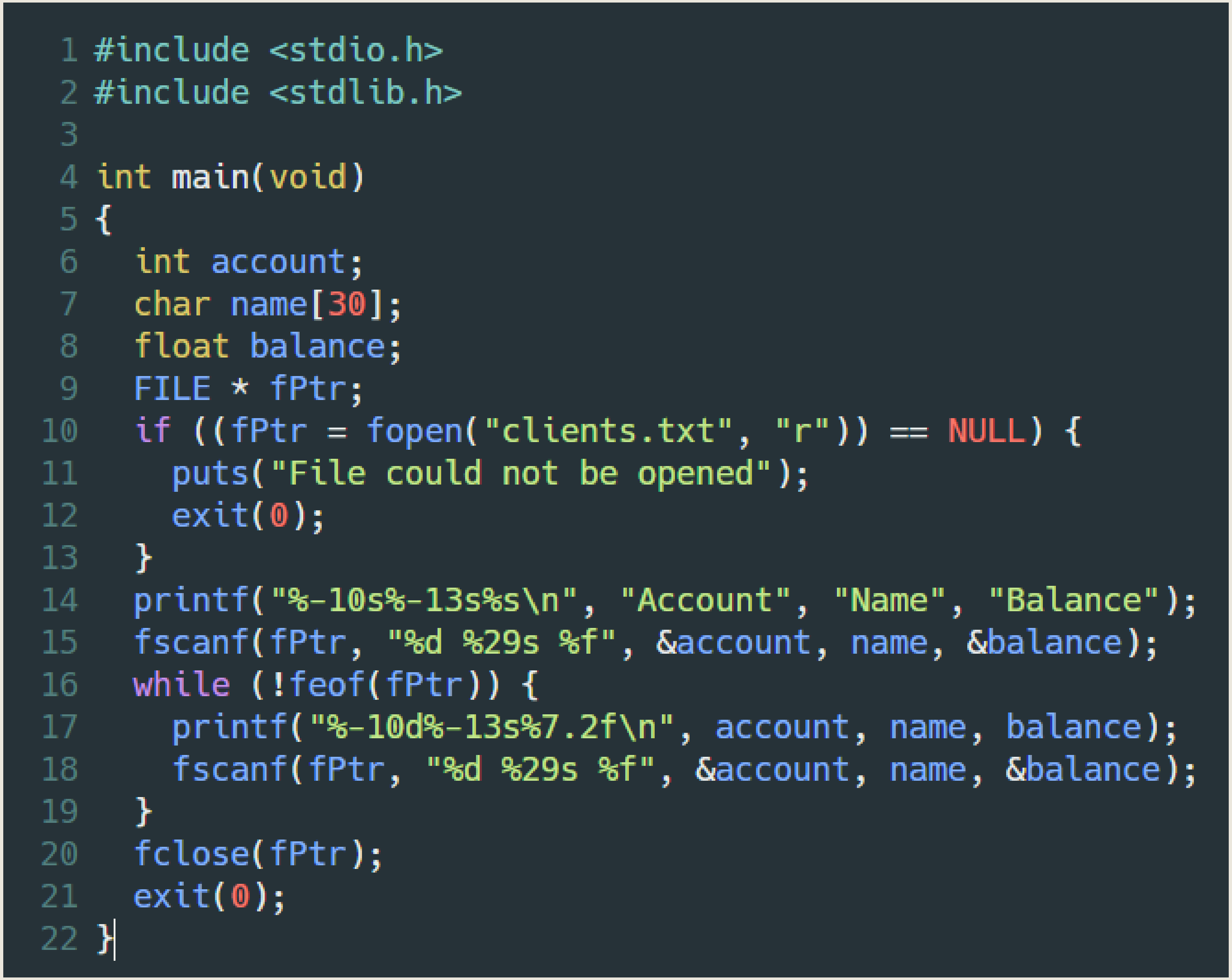
- Data is stored in files so that the data can be retrieved for processing when needed.
- Next figure reads records from the file "clients.txt" created by the program of previous example and prints their contents.
- fPtr is a pointer to a FILE.
- We attempt to open the file "clients.txt" for reading ("r") and determine whether it opened successfully (i.e., fopen does not return NULL).

Reading Data from a Sequential-Access File

- 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.

Reading Data from a Sequential-Access File

- 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.

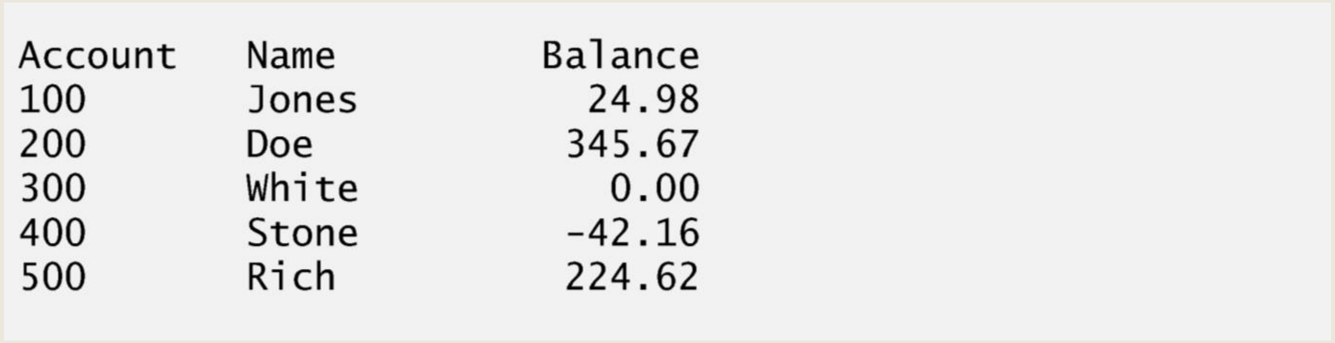


```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int account;
  char name[30];
  float balance;
  FILE * fPtr;
  if ((fPtr = fopen("clients.txt", "r")) == NULL) {
    puts("File could not be opened");
    exit(0);
  }
  printf("%-10s%-13s%s\n", "Account", "Name", "Balance");
  fscanf(fPtr, "%d %29s %f", &account, name, &balance);
  while (!feof(fPtr)) {
    printf("%-10d%-13s%7.2f\n", account, name, balance);
    fscanf(fPtr, "%d %29s %f", &account, name, &balance);
  }
  fclose(fPtr);
  exit(0);
}
```

Reading Data from a Sequential-Access File

- Example Output



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

Reading Data from a Sequential-Access 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.

Reading Data from a Sequential-Access File

- The statement rewind(fPtr); 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 fPtr.
- 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.

References

- C How to Program, 8ed, by Paul Deitel and Harvey Deitel, Pearson, 2016.