Lecture#6

File Processing – Sequential Access Files

CENG 102- Algorithms and Programming II,

2024-2025, Spring

Contains materials from:
P. Deitel, H. Deitel, ''C How to Program with an Introduction to C++'', 8th edition, Pearson

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* storage of data.
- Computers store files on secondary storage devices, such as hard drives, CDs, DVDs, flash drives and so on.
- In this chapter, we explain how data files are created, updated and processed by C programs.
- We consider both **sequential-access** and **random-access** file processing.

11.2 Files and Streams

- C views each file simply as a sequential stream of bytes (Fig. 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 (Cont.)

- 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`.

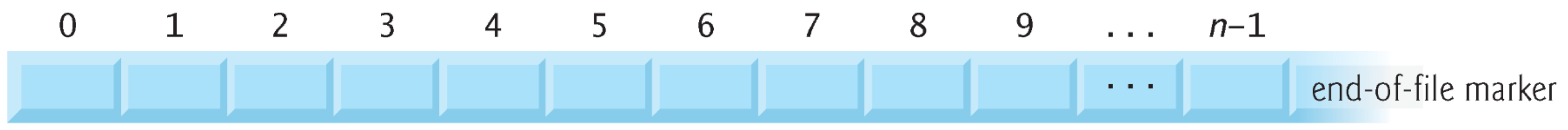


Fig. 11.1 | C's view of a file of *n* bytes.

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`, 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()`.

```
// C program to illustrate fgetc() function
#include <stdio.h>
int main ()
{
        // open the file
        FILE *fp = fopen("test.txt","r");
        // Return if could not open file
        if (fp == NULL)
                return 0;
        do
        {
                // Taking input single character at a time
                char c = fgetc(fp);
                // Checking for end of file
                if (feof(fp))
                        break ;
                printf("%c", c);
        } while(1);
        fclose(fp);
        return(0);
}
```

11.2 Files and Streams (Cont.)

- If an error occurs while opening a file in any mode, `fopen` returns **`NULL`**.
- 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')`.

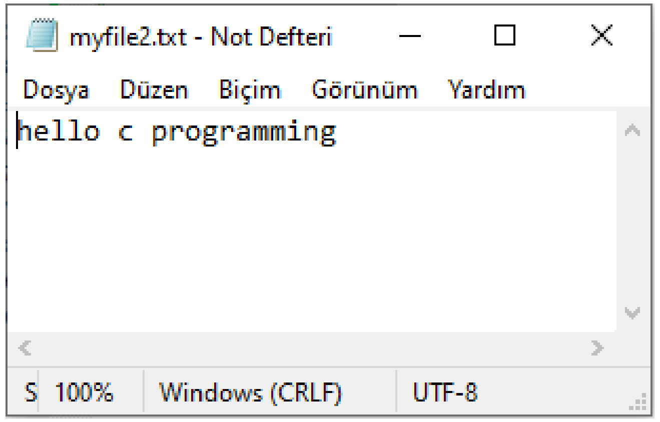
```
// C program to illustrate fputc() function
#include<stdio.h>
int main()
{
        FILE *fp = fopen("output.txt","w");
        // Return if could not open file
        if (fp == NULL)
                return 0;
        char string[] = "good bye", received_string[20];
        for (int i = 0; string[i]!='\0'; i++)
                // Input string into the file
                // single character at a time
                fputc(string[i], fp);
        fclose(fp);
        fp = fopen("output.txt","r");
        // Reading the string from file
        fgets(received_string,20,fp);
        printf("%s", received_string);
        fclose(fp);
        return 0;
}
```

11.2 Files and Streams (Cont.)

- The `fgets` and `fputs` functions 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`.

11.2 Files and Streams (Cont.)

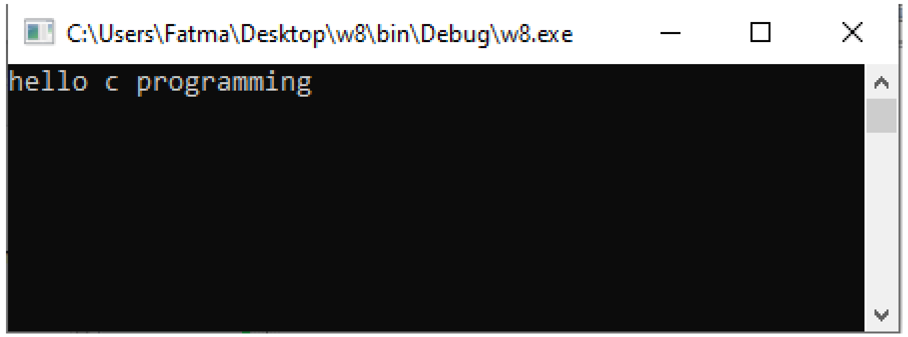
```
#include<stdio.h>
#include<conio.h>
void main(){
        FILE *fp
        fp=fopen("myfile2.txt","w");
        fputs("hello c programming",fp);
        fclose(fp);
        getch();
}
```


myfile2.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
S 100% Windows (CRLF) UTF-8

```
hello c programming
```

11.2 Files and Streams (Cont.)

```
#include<stdio.h>
#include<conio.h>
void main(){
        FILE *fp;
        char text[300];
        fp=fopen("myfile2.txt","r");
        printf("%s",fgets(text,200,fp));
        fclose(fp);
        getch();
}
```



```
hello c programming
```

11.3 Creating a Sequential-Access 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.
- 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.

11.3 Creating a Sequential-Access File (Cont.)

- The **account number** is used as the **record key** in this application—the file will be created and maintained in account-number order.
- 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
// 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);
```

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

```c

      // write account, name and balance into file with fprintf
      while (!feof(stdin)  ) {
         fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
         printf("%s", "? ");
         scanf("%d%29s%lf", &account, name, &balance);
      }

      fclose(cfPtr); // fclose closes file
   }
}
```

```
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.)

Hit Control+Z then ENTER keys on keyboard

11.3 Creating a Sequential-Access File (Cont.)

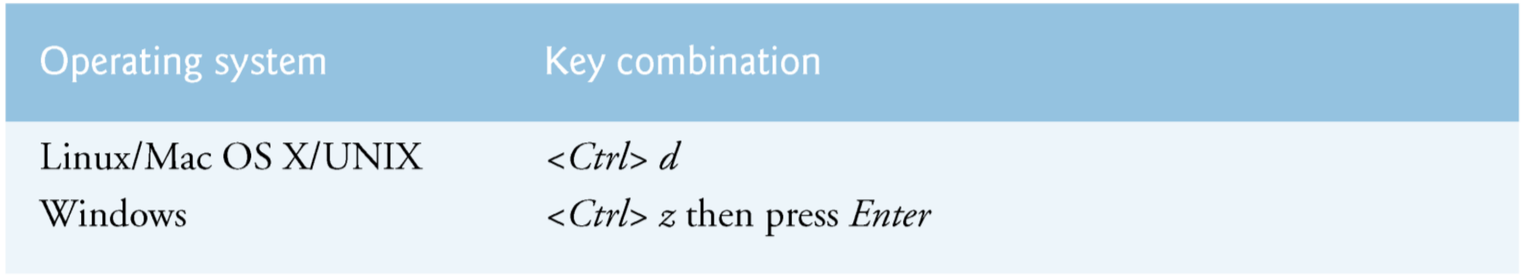
- The file open mode `"w"` indicates that the file is to be opened for writing.
- If a file *does not* exist, `fopen` creates the file.
- If a file does exist, `fopen` deletes the contents of the file.
  - *without warning*

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 (Cont.)

- 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 is no more data to be processed.



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 `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 (Cont.)

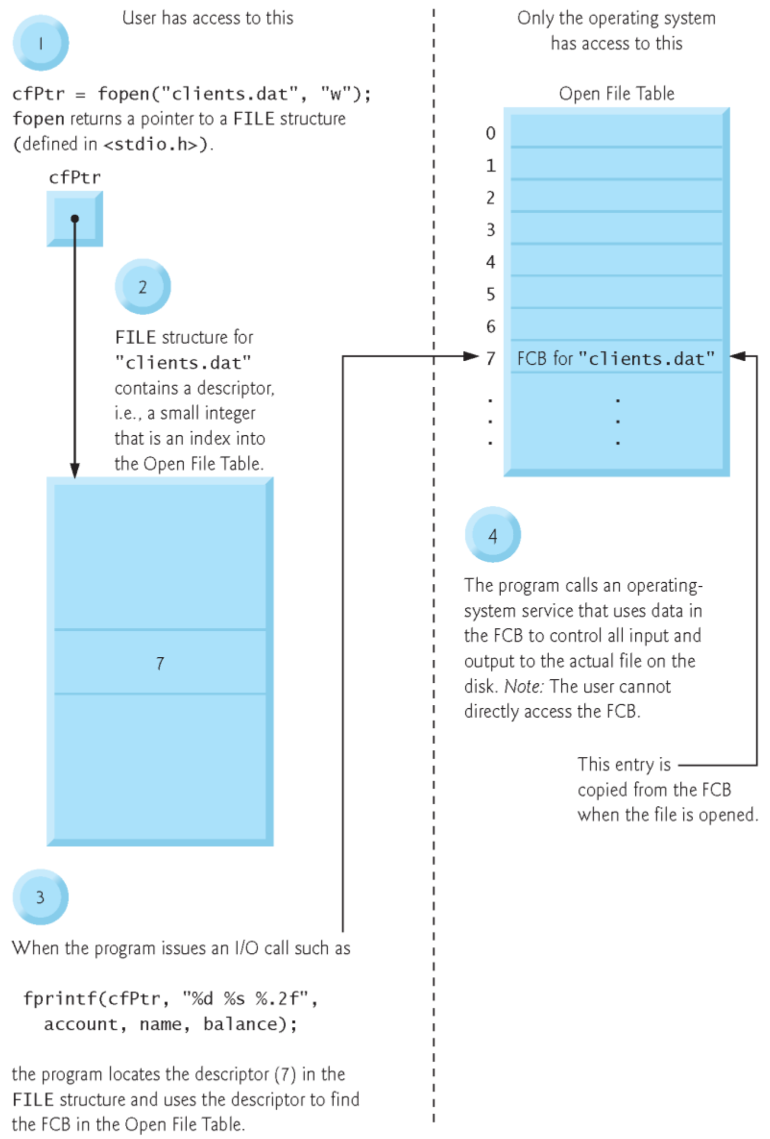
- If function `fclose` is not called explicitly, the operating system normally will close the file when program execution terminates.

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 (Cont.)

- 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.


User has access to this
Only the operating system has access to this
1
cfPtr
2
FILE structure for "clients.dat" contains a descriptor, i.e., a small integer that is an index into the Open File Table.
7
3
Open File Table
0
1
2
3
4
5
6
7
FCB for "clients.dat"
4
This entry is copied from the FCB when the file is opened.

```
cfPtr = fopen("clients.dat", "w");
```

`fopen` returns a pointer to a `FILE` structure (defined in `<stdio.h>`).

When the program issues an I/O call such as

```
fprintf(cfPtr, "%d %s %.2f",
   account, name, balance);
```

the program locates the descriptor (7) in the `FILE` structure and uses the descriptor to find the FCB in the Open File Table.

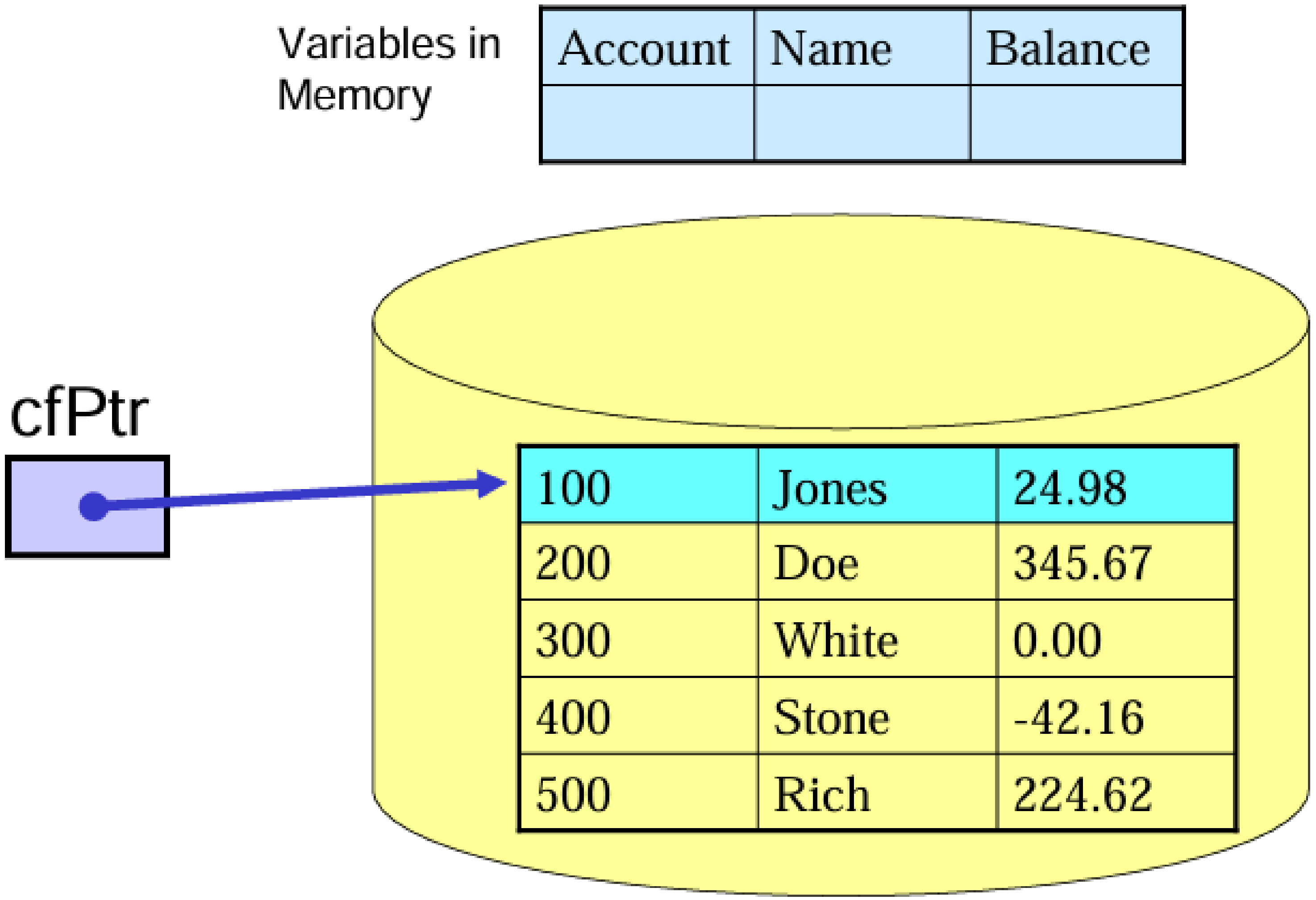
The program calls an operating-system service that uses data in the FCB to control all input and output to the actual file on the disk. *Note:* The user cannot directly access the FCB.

Fig. 11.4 | Relationship between `FILE` pointers, `FILE` structures and FCBs.

11.3 Creating a Sequential-Access File (Cont.)

- 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.*

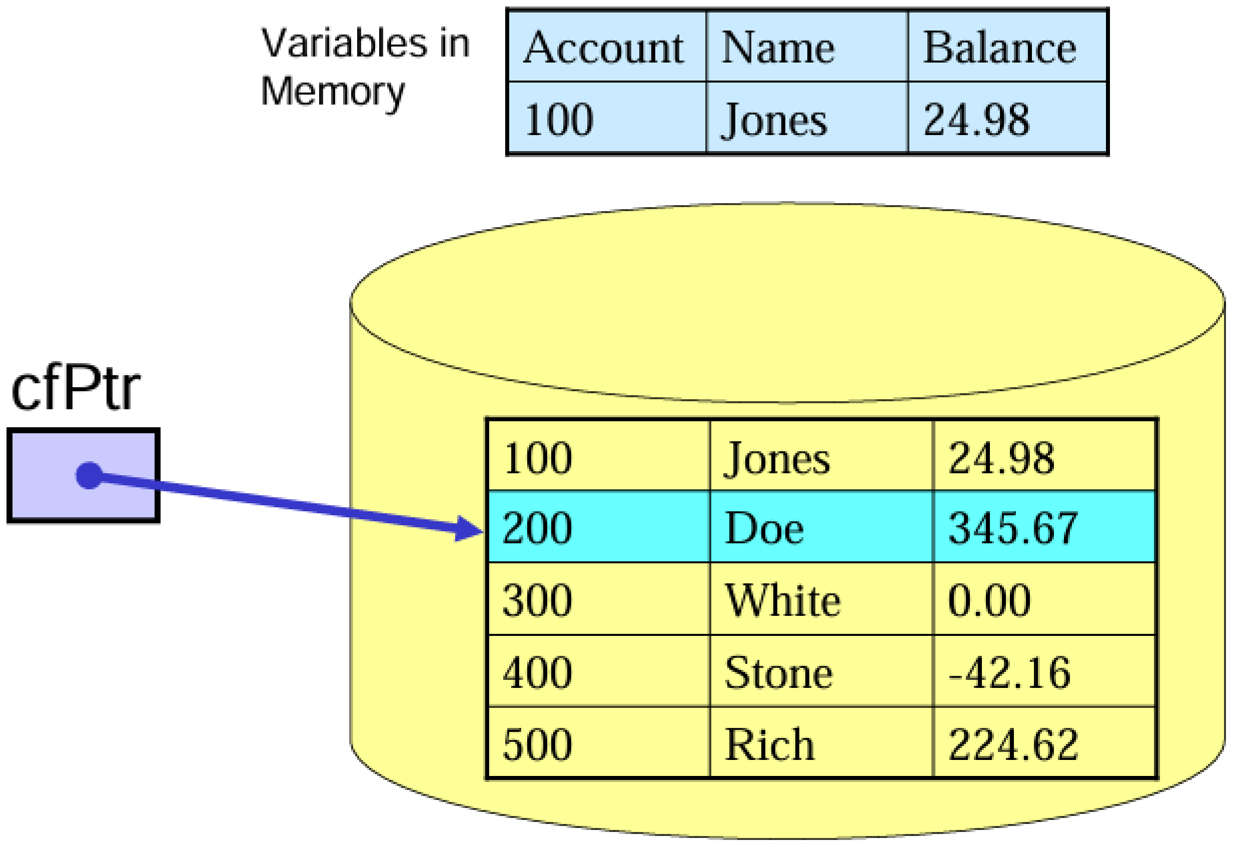
File pointer after fopen()



Account	Name	Balance

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

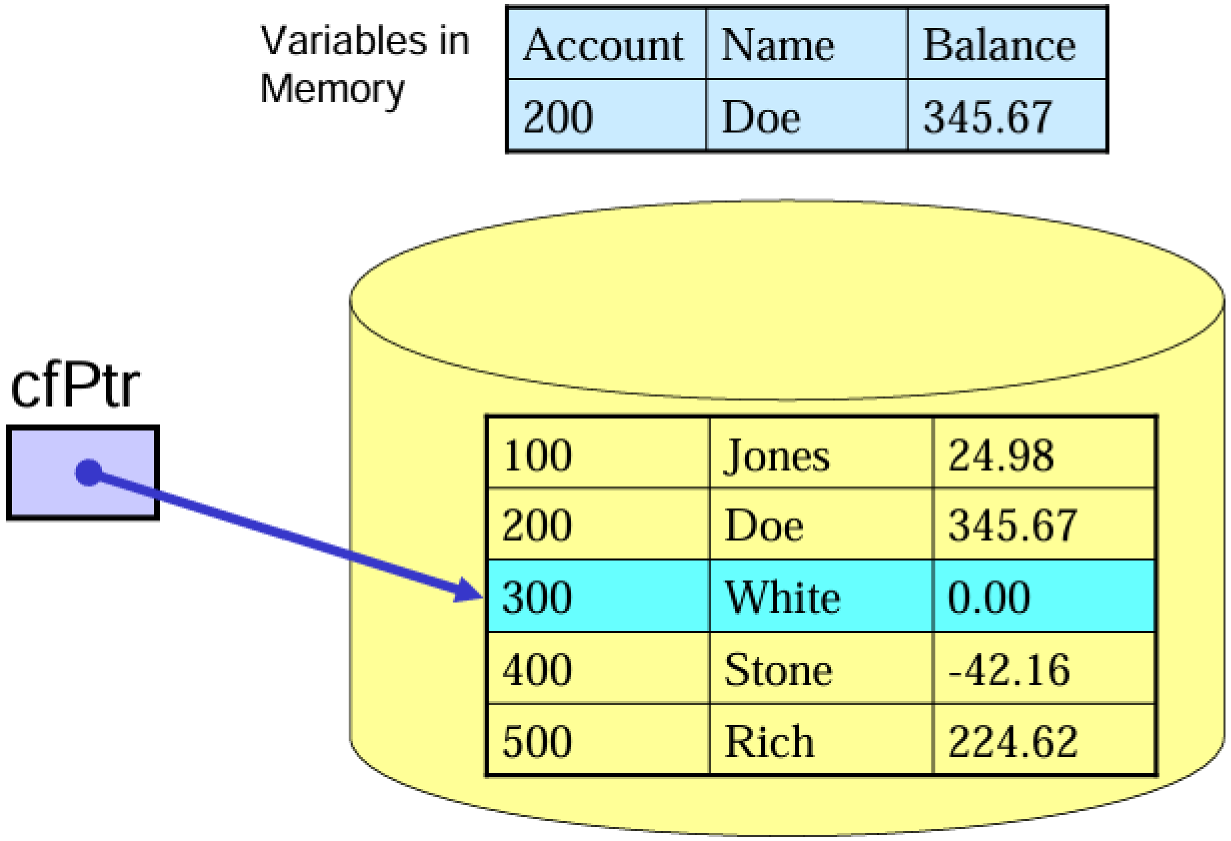
File pointer after 1st fscanf()



Account	Name	Balance
100	Jones	24.98

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

File pointer after 2nd fscanf()



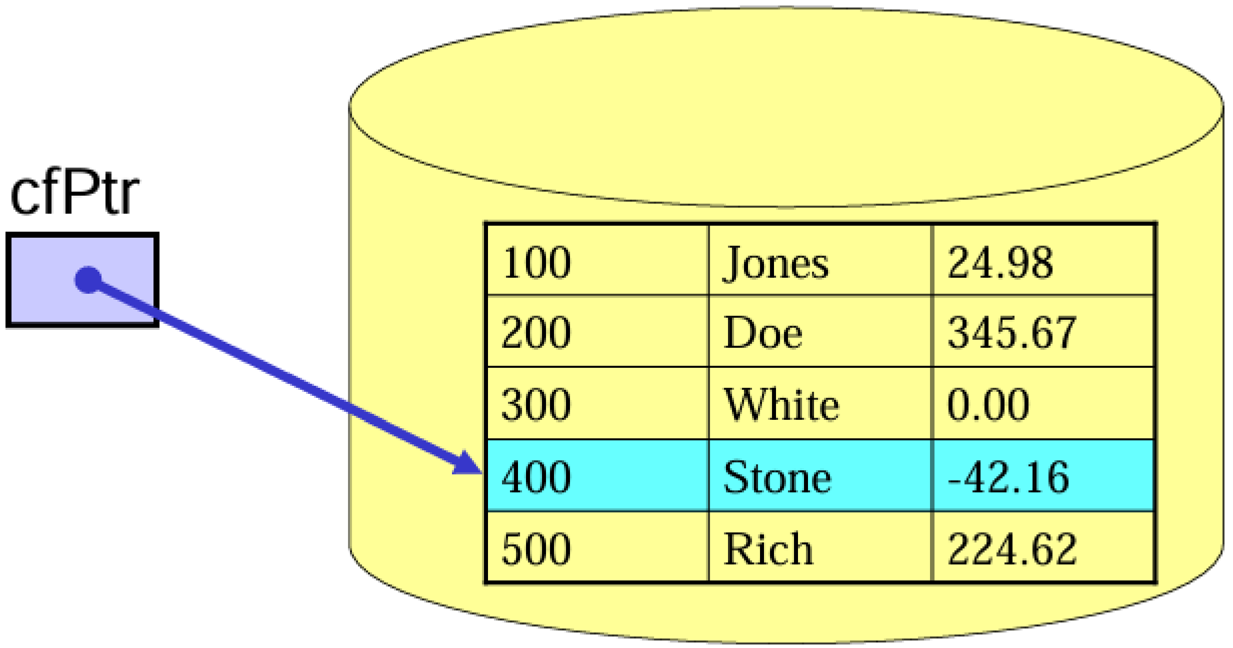
Account	Name	Balance
200	Doe	345.67

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

File pointer after 3rd fscanf()

Variables in Memory

Account	Name	Balance
300	White	0.00

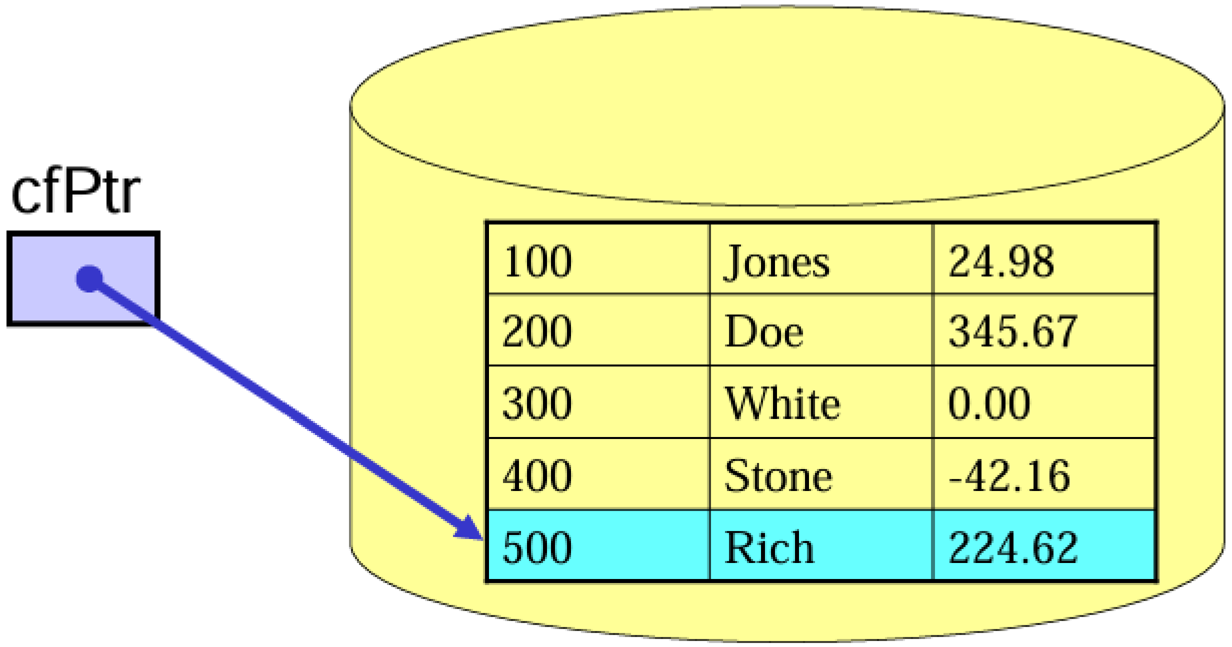


100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

File pointer after 4th fscanf()

Variables in Memory

Account	Name	Balance
400	Stone	-42.16

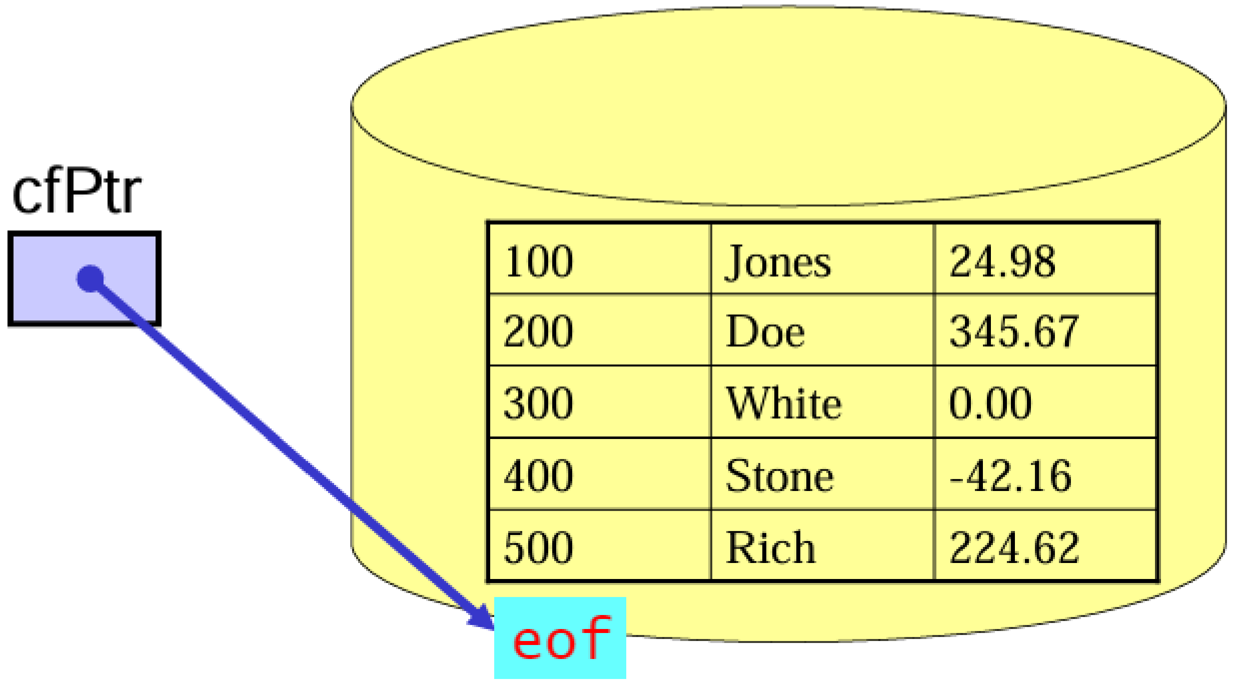


100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

File pointer after 5th fscanf()

Variables in Memory

Account	Name	Balance
500	Rich	224.62



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

11.3 Creating a Sequential-Access File (Cont.)

- Files may be opened in one of several modes (Fig. 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 (Cont.)

- 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 (Cont.)

- 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 is 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.

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.)

Common Programming Error 11.3

Opening a nonexistent file for reading 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

- 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.txt"` created by the program of Fig. 11.2 and prints their contents.

- [*Note:* Figures 11.6–11.7 use the data file created in Fig. 11.2, so you must run Fig. 11.2 before Figs. 11.6–11.7.]

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

- 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 `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
// Fig. 11.6: fig11_06.c
// Reading and printing a sequential file
#include <stdio.h>

int main(void)
{
   FILE *cfPtr; // cfPtr = clients.txt file pointer

   // fopen opens file; exits program if file cannot be opened
   if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
      puts("File could not be opened");
   }
   else { // read account, name and balance from file
      unsigned int account; // account number
      char name[30]; // account name
      double balance; // account balance

      printf("%-10s%-13s%s\n", "Account", "Name", "Balance");
      fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

```

Fig. 11.6 | Reading and printing a sequential file. (Part 1 of 2.)

```
        // while not end of file
        while (!feof(cfPtr)  ) {
           printf("%-10d%-13s%7.2f\n", account, name, balance);
           fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);
        }

        fclose(cfPtr); // fclose closes the file
     }
  }
```

```
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.)

Resetting the File Position Pointer

- 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.

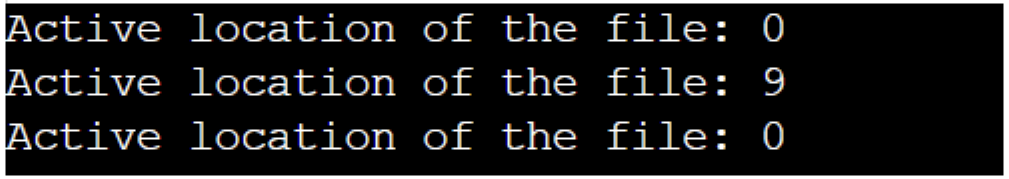
11.4 Reading Data from a Sequential-Access File (Cont.)

- 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.

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

```
#include <stdio.h>
int main(void)
{
   FILE *fp;
   if ((fp = fopen ("test.txt", "w+")) == NULL) {
       printf("Error!");
       return(0);
   }
   printf("Active location of the file: %ld\n", ftell(fp));
   fputs("algorithm", fp);
   printf("Active location of the file: %ld\n", ftell(fp));
   rewind(fp);
   printf("Active location of the file: %ld", ftell(fp));
   fclose(fp);
   return 0;
}
```



```
Active location of the file: 0
Active location of the file: 9
Active location of the file: 0
```

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

Credit Inquiry Program

- The program of Fig. 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)
  - 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.

```c
// Fig. 11.7: fig11_07.c
// Credit inquiry program
#include <stdio.h>

// function main begins program execution
int main(void)
{
   FILE *cfPtr; // clients.txt file pointer

   // fopen opens the file; exits program if file cannot be opened
   if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
      puts("File could not be opened");
   }
   else {

      // display request options
      printf("%s", "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? ");
      unsigned int request; // request number
      scanf("%u", &request);

```

Fig. 11.7 | Credit inquiry program. (Part 1 of 6.)

```
      // process user's request
      while (request != 4) {
         unsigned int account; // account number
         double balance; // account balance
         char name[30]; // account name

         // read account, name and balance from file
         fscanf(cfPtr, "%d%29s%lf", &account, name, &balance);

```

Fig. 11.7 | Credit inquiry program. (Part 2 of 6.)

```c
      switch (request) {
         case 1:
            puts("\nAccounts with zero balances:");

            // read file contents (until eof)
            while (!feof(cfPtr)) {
               // output only if balance is 0
               if (balance == 0) {
                  printf("%-10d%-13s%7.2f\n",
                     account, name, balance);
               }

               // read account, name and balance from file
               fscanf(cfPtr, "%d%29s%lf",
                  &account, name, &balance);
            }

            break;
```

Fig. 11.7 | Credit inquiry program. (Part 3 of 6.)

```
         case 2:
            puts("\nAccounts with credit balances:\n");

            // read file contents (until eof)
            while (!feof(cfPtr)) {
               // output only if balance is less than 0
               if (balance < 0) {
                  printf("%-10d%-13s%7.2f\n",
                     account, name, balance);
               }

               // read account, name and balance from file
               fscanf(cfPtr, "%d%29s%lf",
                  &account, name, &balance);
            }

            break;
```

Fig. 11.7 | Credit inquiry program. (Part 4 of 6.)

```
            case 3:
               puts("\nAccounts with debit balances:\n");

               // read file contents (until eof)
               while (!feof(cfPtr)) {
                  // output only if balance is greater than 0
                  if (balance > 0) {
                     printf("%-10d%-13s%7.2f\n",
                        account, name, balance);
                  }

                  // read account, name and balance from file
                  fscanf(cfPtr, "%d%29s%lf",
                     &account, name, &balance);
               }

               break;
         }

         rewind(cfPtr); // return cfPtr to beginning of file

         printf("%s", "\n? ");
         scanf("%d", &request);
      }
```

Fig. 11.7 | Credit inquiry program. (Part 5 of 6.)

```

        puts("End of run.");
        fclose(cfPtr); // fclose closes the file
    }
}
```

Fig. 11.7 | Credit inquiry program. (Part 6 of 6.)

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

- 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 Fig. 11.8.

```
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.
```

Fig. 11.8 | Sample output of the credit inquiry program of Fig. 11.7.

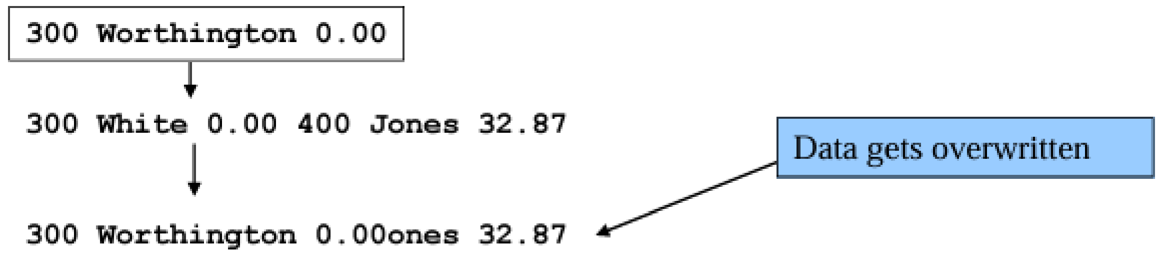
11.4 Reading Data from a Sequential-Access File (Cont.)

- 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

```
300 White 0.00
```

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

- 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.


300 Worthington 0.00
300 White 0.00 400 Jones 32.87
300 Worthington 0.00ones 32.87
Data gets overwritten

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

- The problem here is that in the formatted input/output model using `fprintf` and `fscanf`, fields—and hence records—can vary in size.
- For example, the values 7, 14, –117, 2074 and 27383 are all `int`s stored in the same number of bytes internally, but they are different-sized fields when displayed on the screen or written to a file as text.

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

- 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 (Cont.)

- 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.

Example

```
#include <stdio.h>
int main(){
      char str[5];
      fgets(str, 5, stdin);                       // read from stdin
      fputs(str, stdout);                         // print read content out to stdout

      FILE *f = fopen("test.txt" , "r");          // open the file
      if(f == NULL){                              // if there was an error
            puts("Error opening file");
            return(0);
       }
       else{                                      // if there was no error
            fgets(str, 5, f);                     // read from file
            fputs(str, stdout);                   // print read content out to stdout
       }
       fclose(f);                                 // close file
       return(0);
}
```

Example

```
#include <stdio.h>
#include <string.h>
int main()
{
      FILE* filePointer;
      char dataToBeWritten[50] = "Algorithms and C Programming ";
      filePointer = fopen("Test.txt", "w");
      if (filePointer == NULL) {
            printf("Test.c file failed to open.");
      }
      else {
            printf("The file is now opened.\n");
            if (strlen(dataToBeWritten) > 0) {
                  fputs(dataToBeWritten, filePointer);
                  fputs("\n", filePointer);
             }
             fclose(filePointer);
             printf("Data successfully written in file Test.txt and the file is not closed.\n");
      }
      return 0;
}
```

Exercise

Write a C program to get name and marks of n number of students from and store them in a file. If the file previously exits, add the information to the file.

Solution

```
#include <stdio.h>
int main()
{
    char name[50];
    int marks, i, num;
    printf("Enter number of students: ");
    scanf("%d", &num);
    FILE *fptr = (fopen("student.txt", "a"));
    if(fptr == NULL) {
        printf("Error!");
        return(0);
    }
    for(i = 0; i < num; ++i)
    {
        printf("For student %d\nEnter name: ", i+1);
        scanf("%s", name);
        printf("Enter marks: ");
        scanf("%d", &marks);
        fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);
    }
    fclose(fptr);
    return 0;
}
```