Chapter 11

File Processing

Chapter 11 – File Processing

Outline

11.1 Introduction
11.2 The Data Hierarchy
11.3 Files and Streams
11.4 Creating a Sequential Access File
11.5 Reading Data from a Sequential Access File
11.6 Random Access Files
11.7 Creating a Randomly Accessed File
11.8 Writing Data to a Randomly Accessed File
11.9 Reading Data from a Randomly Accessed File

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 because these are implemented in main memory (RAM)
    - Data files are used to store the data permanently
  - Examples of permanent storage devices:
    - Hard Disk, CD / DVD, Flash Memory, etc.

11.2 The 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: Student name

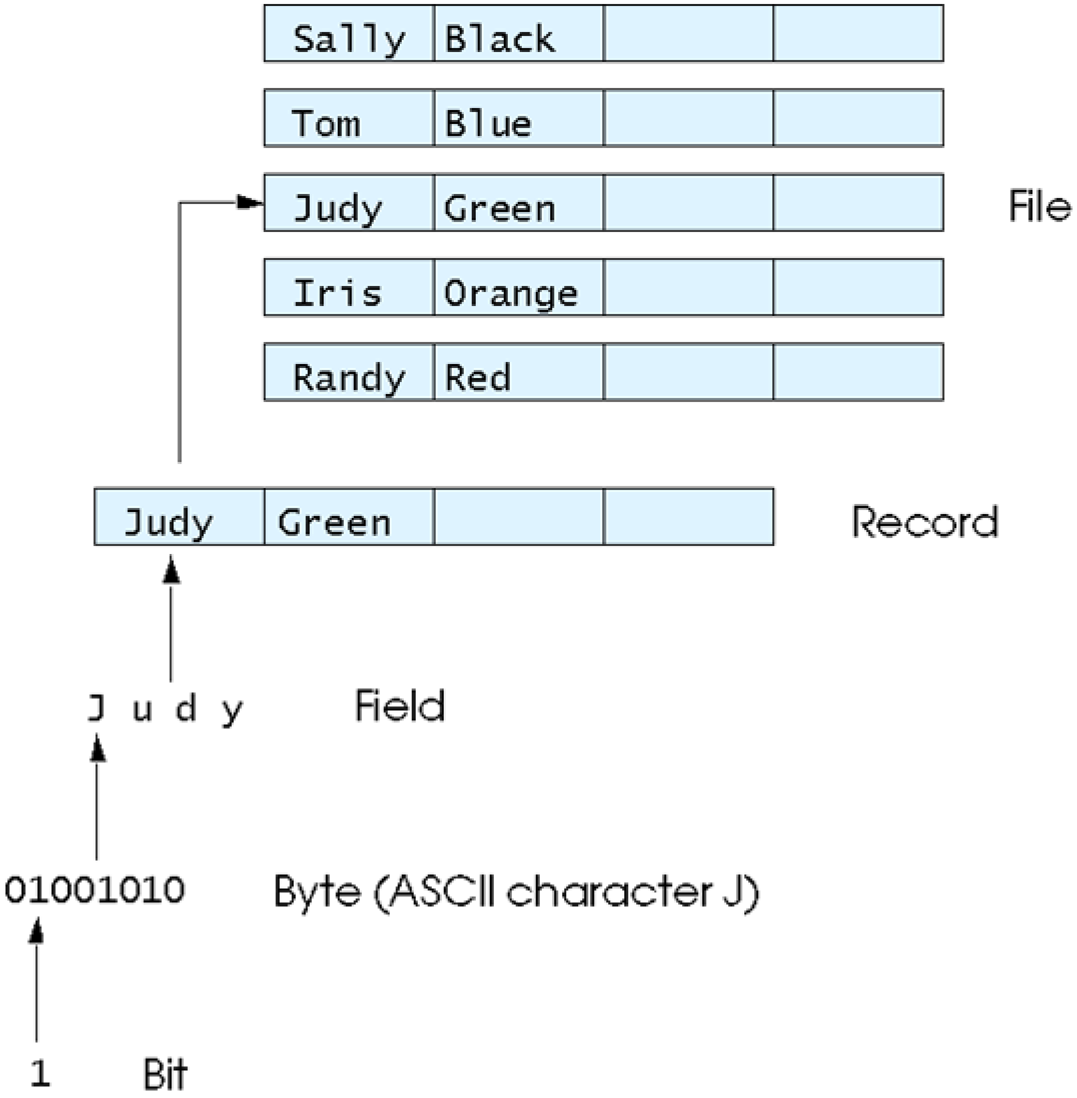
- **Record :** group of related fields
  - Can be represented by a `struct` (in C) or a `class` (in C++)
  - Example: In a school system, a record for a particular student that contained student identification number, name, address, etc.

11.2 The Data Hierarchy

- **File :** group of related records
  - Example: Student file

- **Database :** group of related files
  - Example: School database
  - Files in a database are called tables such as followings:
    - Students, Courses, Classrooms, Teachers

11.2 The Data Hierarchy



Sally	Black		
Tom	Blue		
Judy	Green		
Iris	Orange		
Randy	Red		

File

Judy	Green		

Record

J u d y Field

01001010 Byte (ASCII character J)

1 Bit

Record Key

- Records typically sorted by a record key.

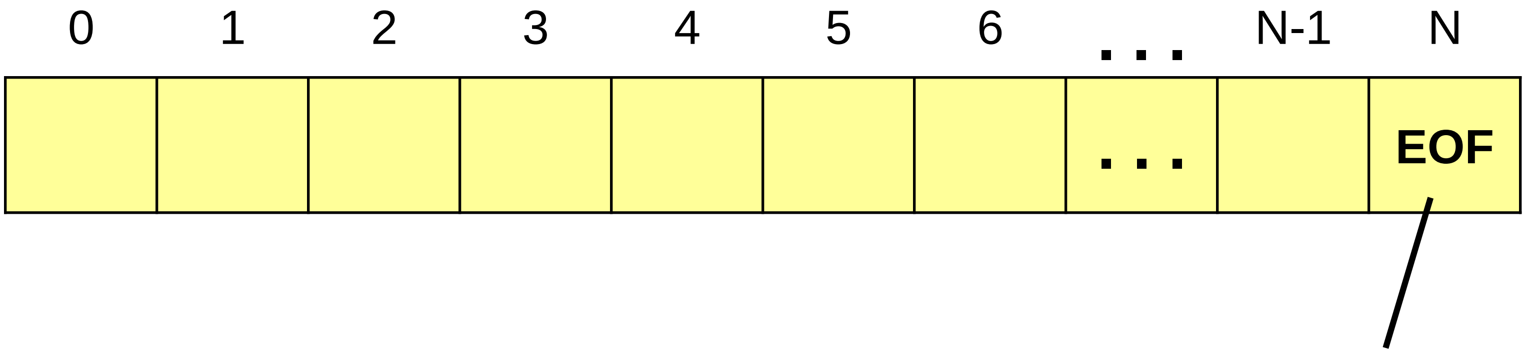
- Record key identifies a record to facilitate the retrieval of specific records from a file.

  Example: Student number is a key field, because it is unique for a student.

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

C's view of a file of N bytes:



0 1 2 3 4 5 6 . . . N-1 N

. . . EOF

End-Of-File marker
such as CTL+Z

11.3 Files and Streams

- 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)

Example: stdin and stdout

- This program uses keyboard and screen as if they were files.

```
#include <stdio.h>

int main()
{
  int sayi;

  fprintf(stdout, "Enter a number : ");
  fscanf(stdin, "%d", &sayi);
  printf("Square = %d \n", sayi * sayi);
}
```

Types of Data Files
(Based on Access Method)

	Record Length	Updating	File I/O Functions
Sequential Access Files	Variable	Not suitable	fgetc, fputc, fscanf, fprintf, fgets, fputs
Random Access Files	Fixed	Always suitable	fread, fwrite

File Open Modes (Text Files)

Mode	Description
r	Open a 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 end of file.
r+	Open a 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.

File Open Modes
(Binary Files)

Mode	Description
rb	Open a 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 end of file in binary mode.
rb+	Open a 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.

Sequential Access Files

Functions for Sequential Access Files

- Read/Write functions in standard library <stdio.h>
  - `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`

Example: Sequential Data File

- The following is a sequential data file which can be edited with a text editor such as Notepad.
- The first line contains the count of numbers.
- The second line contains the numbers (grades).
- We want to write a C program to calculate and display the grade average.

veriler.txt file

```
5

73     44     100     28   92
```

Example: fscanf

Part 1 of 2

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int N; //Count of numbers
  int i; // Loop counter
  int sayi;
  int Total=0;

  FILE * fPtr;
  fPtr = fopen("veriler.txt", "r");

  if (fPtr == NULL)
  {
    printf("Dosya acilamadi\n");
    system("pause");
    return 0;
  }
```

Example: fscanf (cont.)

Part 2 of 2

```
  // First, read the count of numbers:
  fscanf(fPtr, "%d", &N);

  // Now, read the numbers:
  for (i=1; i <= N; i++)
  {
     fscanf(fPtr, "%d", &sayi);
     printf("%d\n", sayi);
     Total += sayi;
  }

  fclose(fPtr);
  printf("Ortalama = %d \n", Total / N);

} // end main
```

Example : Creating a Sequential Access File

```
// Create a sequential file
#include <stdio.h>
int main() {
   int account;     // account number
   char name[ 30 ]; // account name
   double balance;  // account balance

   FILE *cfPtr;     // cfPtr = clients.dat file pointer
   // fopen opens file. Exit program if unable to create file
   if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } // end if
   else {
      printf( "Enter the account, name, and balance.\n" );
      printf( "Enter EOF to end input.\n" );
      printf( "? " );
      scanf( "%d%s%lf", &account, name, &balance );

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

Program
Output

```
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
```

**Control+Z then Enter Key
on keyboard
(like sentinel)**

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;`
    - Defines 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
    - We can also specify a device name and a directory path:

```
fopen(“E:\\Accounting\\clients.dat", “w”);
```

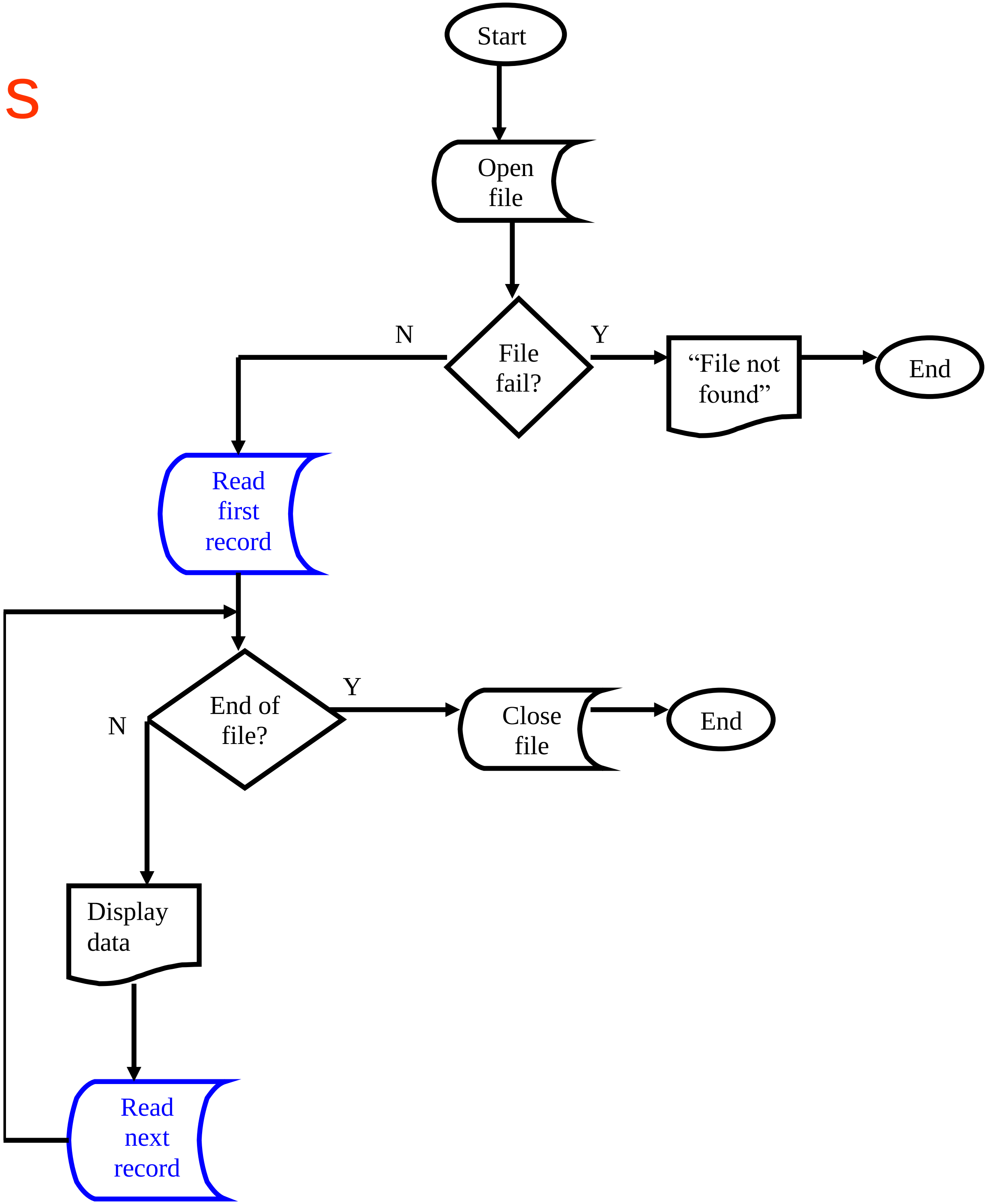
11.4 Creating a Sequential Access File

- `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
  - Its role is similar to a sentinel
- `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

Reading Records from a File

(with EOF checking)



Example : Reading Data from a Sequential Access File

```
// Read a sequential file
#include <stdio.h>
int main() {
   int account;     // account number
   char name[ 30 ]; // account name
   double balance;  // account balance

   FILE *cfPtr;     // cfPtr = clients.dat file pointer

   // fopen opens 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 { // read account, name and balance from file
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

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

      fclose( cfPtr ); // fclose closes the file
   } // end else
} // end main
```

Program Output

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

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

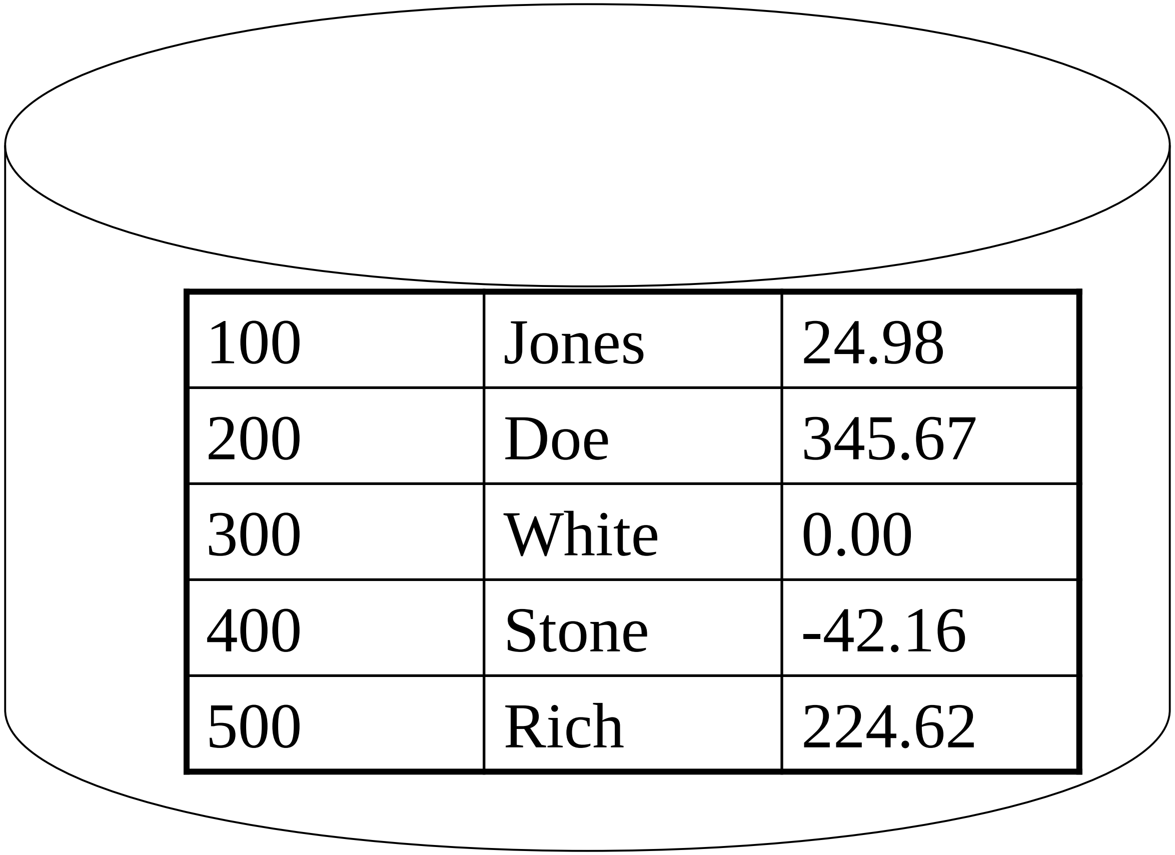
    ```
    cfPtr = fopen( “clients.dat", "r" );
    ```

  - Use fscanf to read from the file
    - Like scanf, except first argument is a FILE pointer

    ```
    fscanf( cfPtr, "%d%s%f", &account, 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)
    - It is the same as using fclose followed by fopen.

Data File on Hard Disk

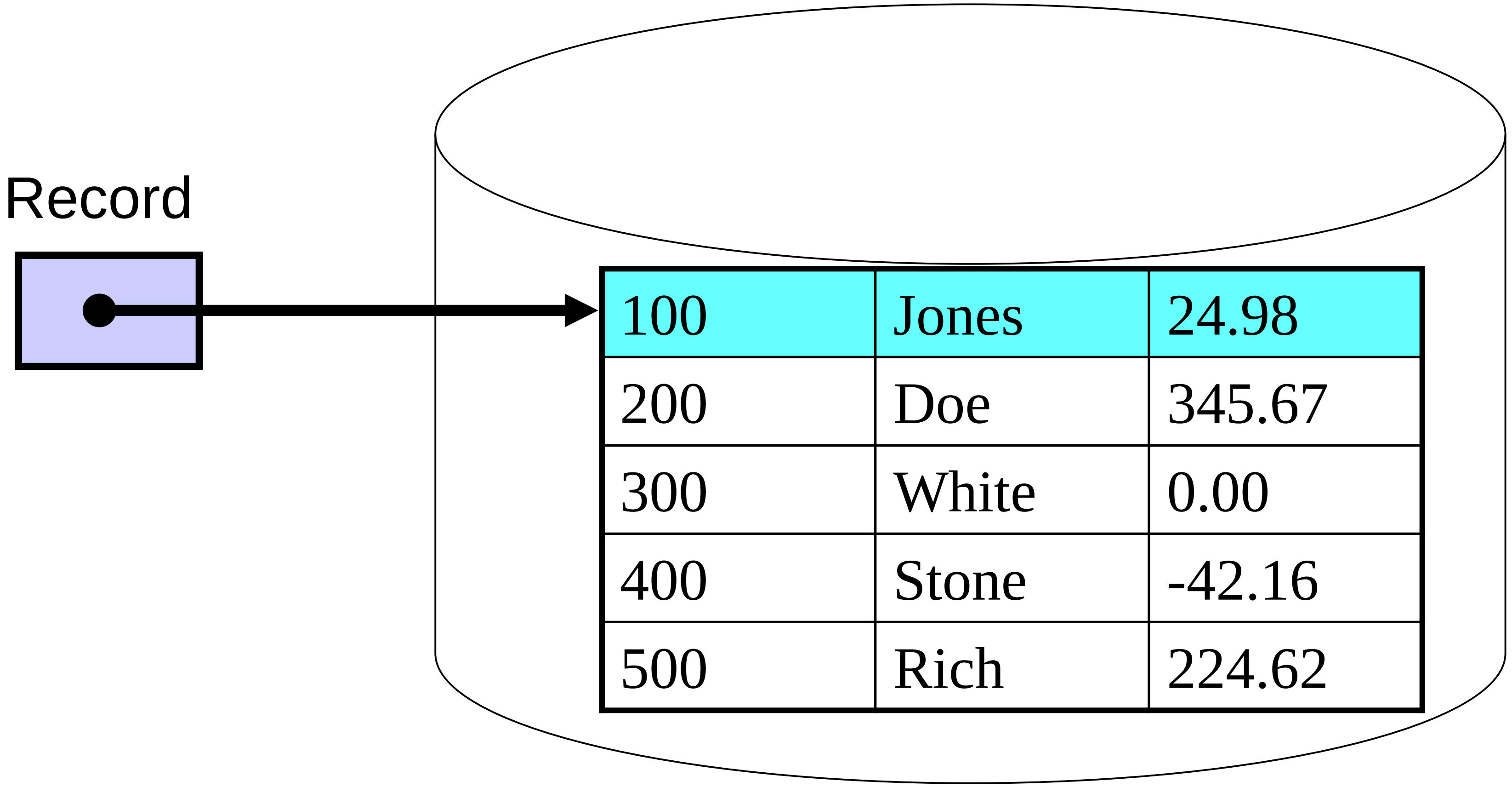


100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

Record location after fopen()

Variables in Memory

Account	Name	Balance



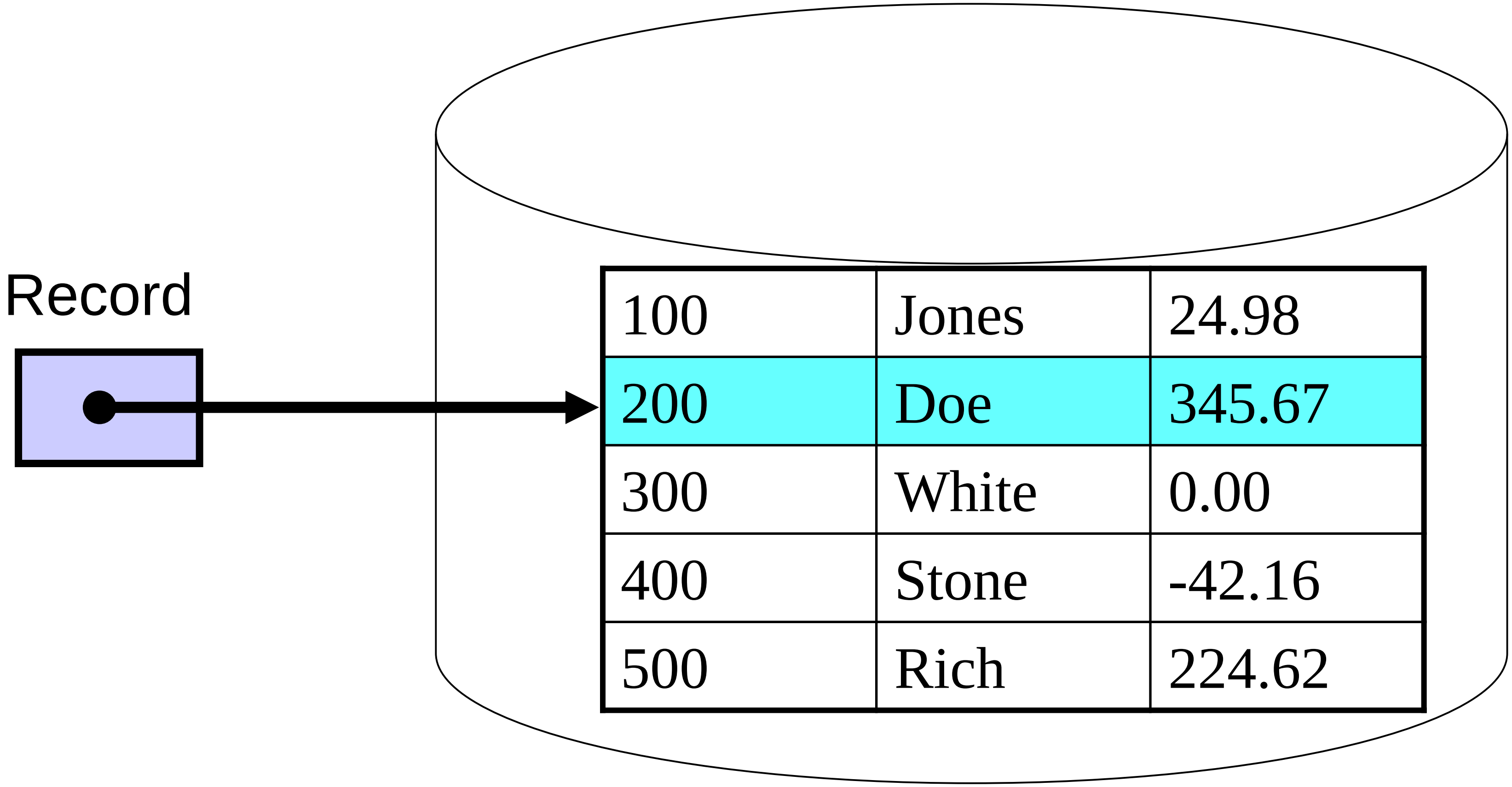
Record

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

Record location after 1st fscanf()

Variables in Memory

Account	Name	Balance
100	Jones	24.98



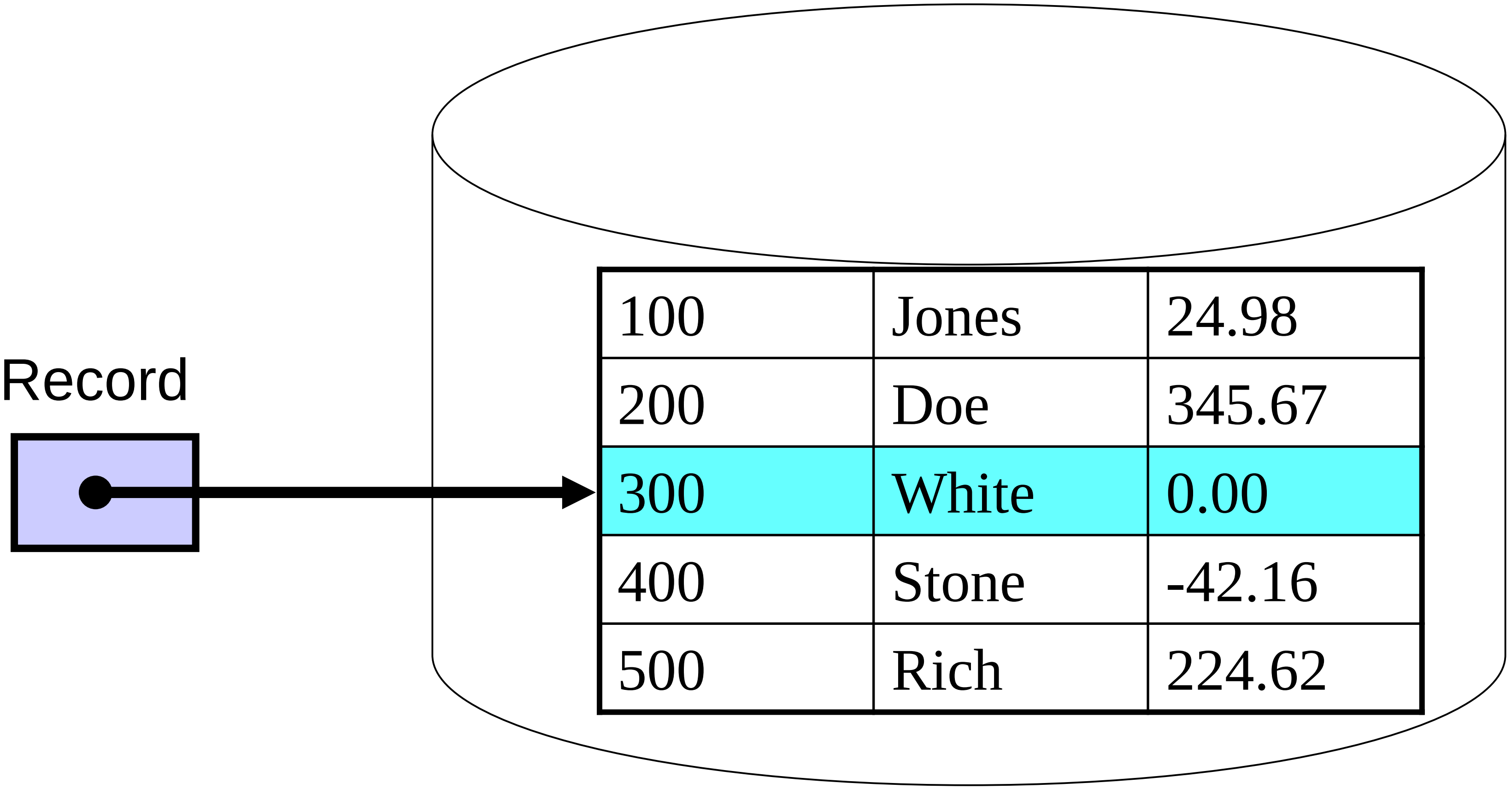
Record

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

Record location after 2nd fscanf()

Variables in Memory

Account	Name	Balance
200	Doe	345.67

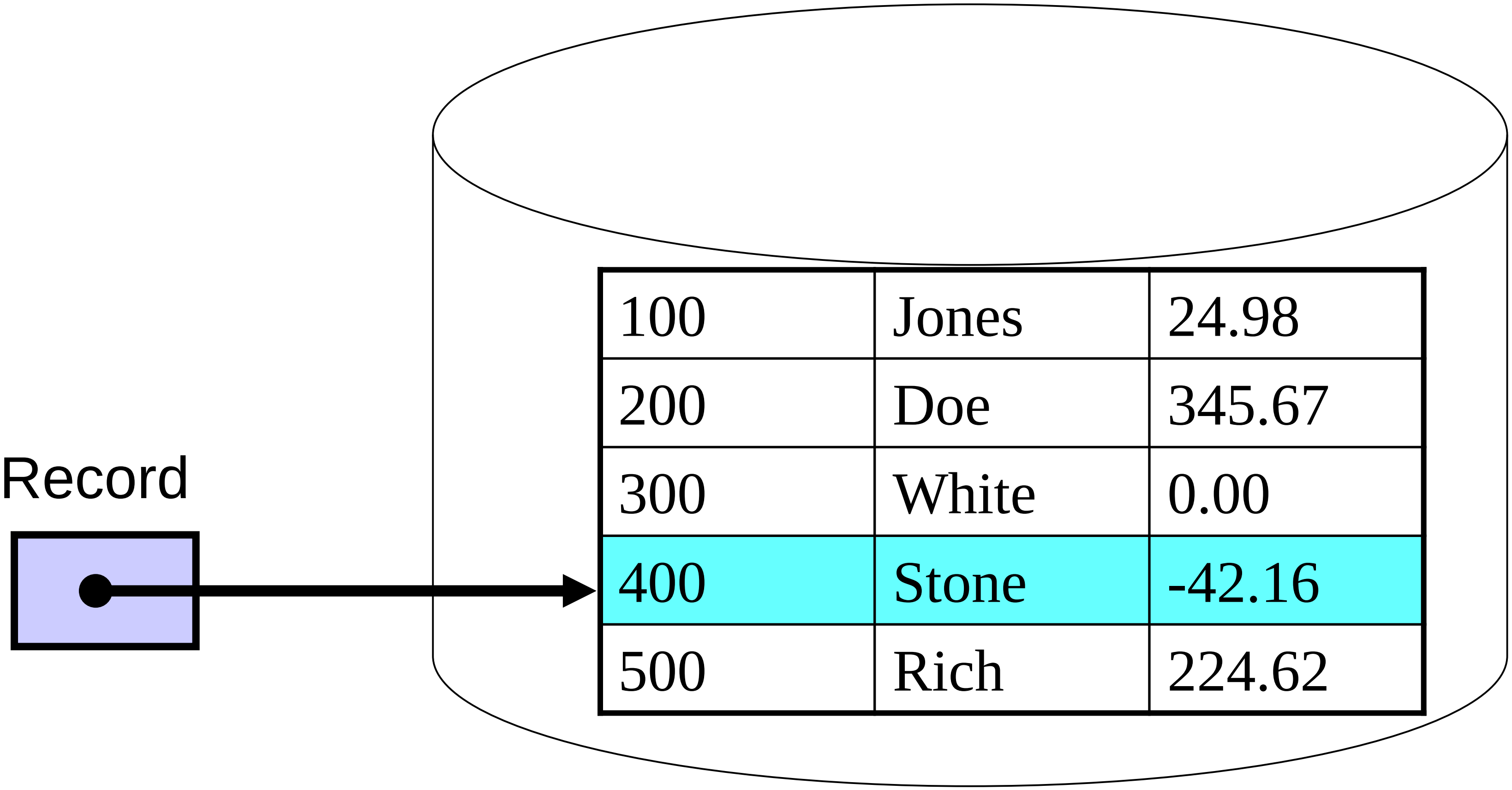


100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

Record location after 3rd fscanf()

Variables in Memory

Account	Name	Balance
300	White	0.00



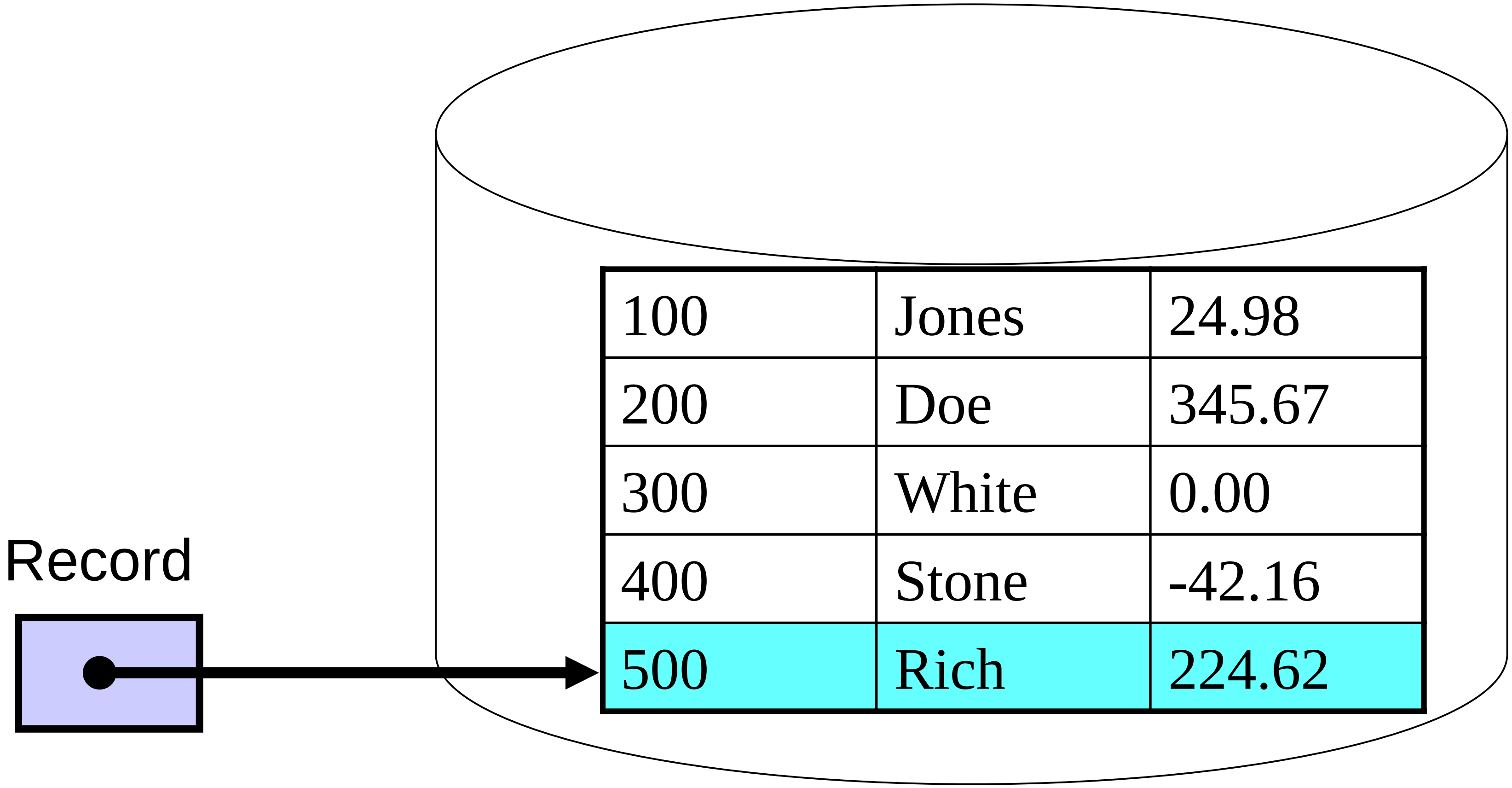
Record

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

Record location after 4th fscanf()

Variables in Memory

Account	Name	Balance
400	Stone	-42.16

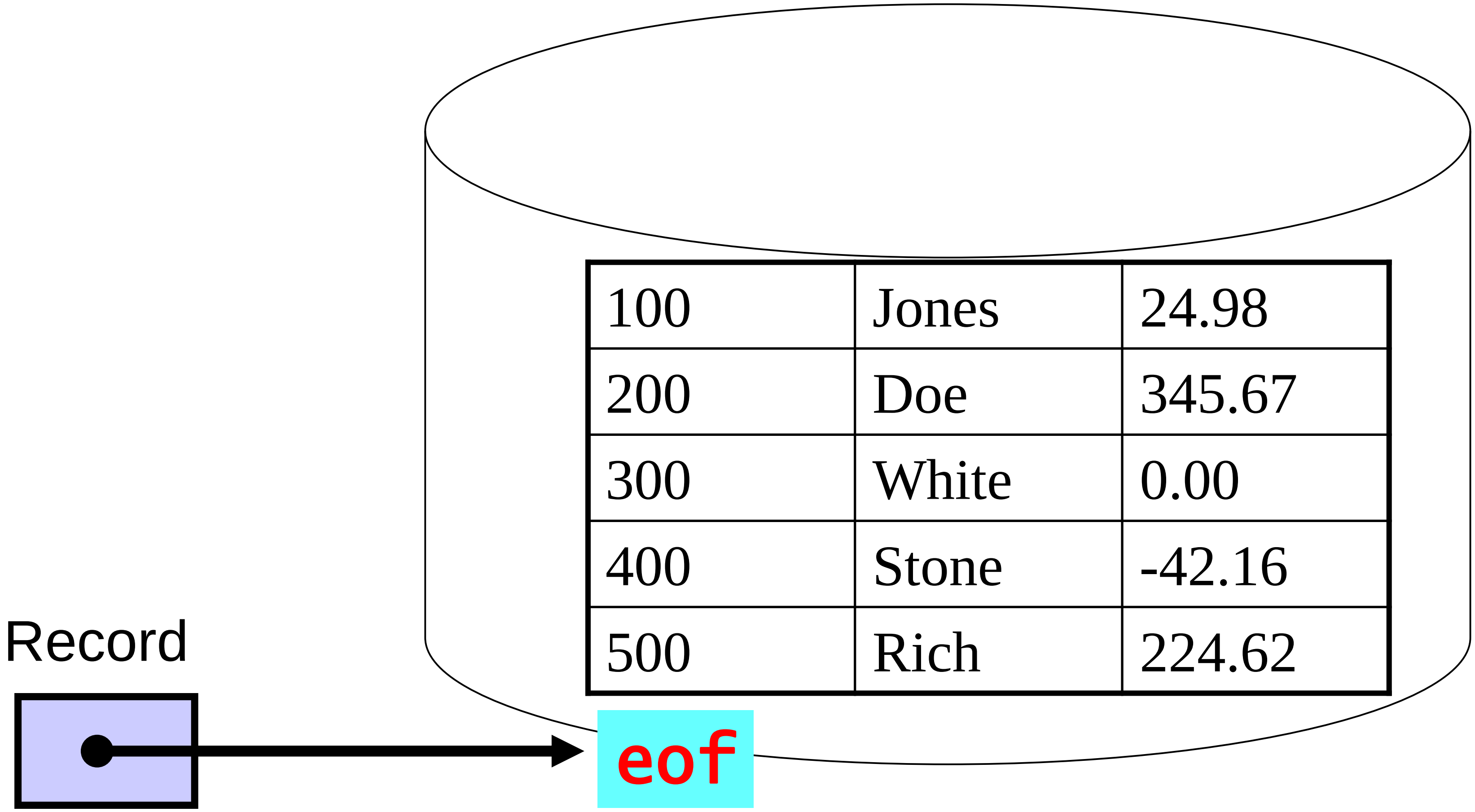


100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	-42.16
500	Rich	224.62

Record location 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

Example : Menu-driven Credit Inquiry Program

Part 1 of 4

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

int main()
{
   unsigned int request; // request number
   unsigned 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 ) {
      puts( "File could not be opened" );
   } // end if
   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? " );
      scanf( "%u", &request );
```

Part 2 of 4

```
// process user's request
     while ( request != 4 ) {

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

        switch ( request ) {

           case 1:
              puts( "\nAccounts with zero balances:" );

              // read file contents (until eof)
              while ( !feof( cfPtr ) ) {

                 if ( balance == 0 ) {
                    printf( "%-10d%-13s%7.2f\n",
                       account, name, balance );
                 } // end if

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

              break;
```

Part 3 of 4

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

             // read file contents (until eof)
             while ( !feof( cfPtr ) ) {

                if ( balance < 0 ) {
                   printf( "%-10d%-13s%7.2f\n",
                      account, name, balance );
                } // end if

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

             break;
```

Part 4 of 4

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

               // read file contents (until eof)
               while ( !feof( cfPtr ) ) {

                  if ( balance > 0 ) {
                     printf( "%-10d%-13s%7.2f\n",
                        account, name, balance );
                  } // end if

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

               break;
         } // end switch

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

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

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

Program Output

```
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.
```

Trying to Modify 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 sequential files and screen than internal main memory (RAM) representation
    - Example:
    - `1`, `34`, `-890` are all `int`s, and each takes 4 bytes in memory
    - But they have different sizes on disk

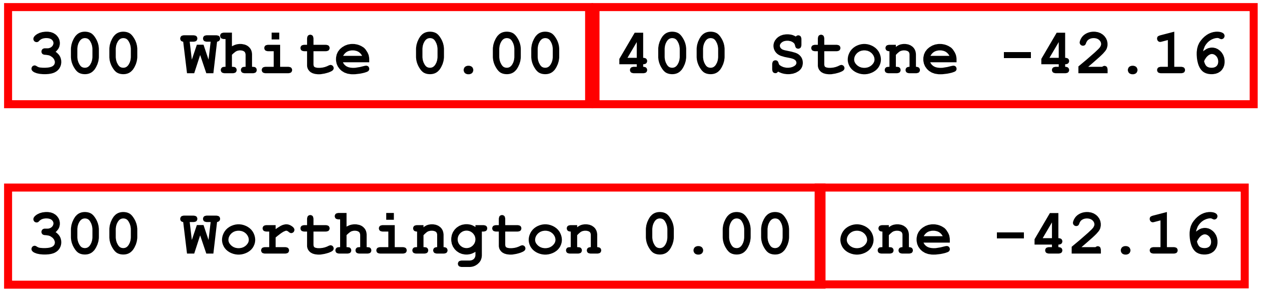
      1 is one byte

      34 is two bytes

      -890 is four bytes

Trying to Modify a Sequential Access File

- Suppose user wants to change customer name from “White” to “Worthington”.



```
300 White 0.00 | 400 Stone -42.16
```

(Old data in file)

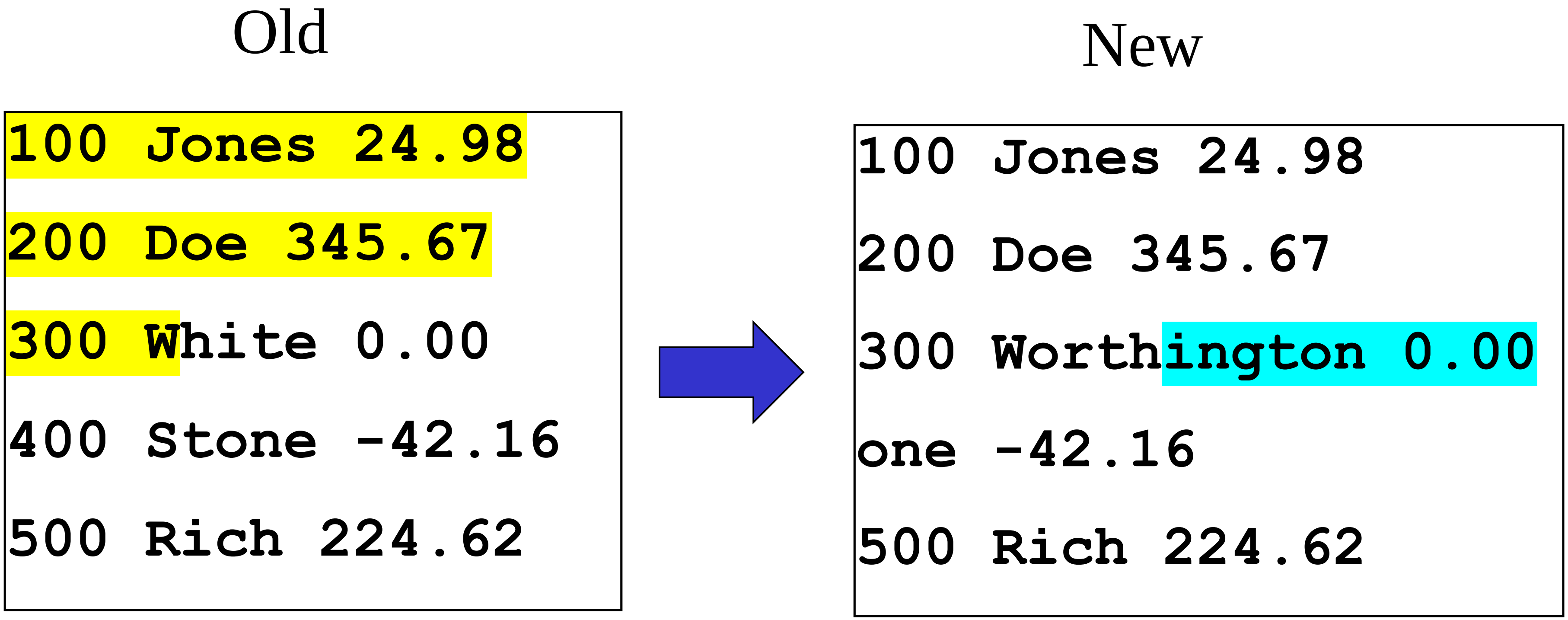
```
300 Worthington 0.00 | one -42.16
```

Data gets overwritten onto the next person’s record!

Trying to Modify a Sequential Access File

```
#include <stdio.h>

int main()
{
   FILE *cfPtr = fopen( "clients.dat", "r+" );;

   // Skip 33 bytes until beginning of customer White’s record:
   fseek( cfPtr, 33, SEEK_SET );

   // Write the new data values:
   fprintf( cfPtr, "%d %s %.2lf\n", 300, "Worthington", 0.0 );

   fclose( cfPtr );
} // end main
```

clients.dat File



Old

```
100 Jones 24.98

200 Doe 345.67

300 White 0.00

400 Stone -42.16

500 Rich 224.62
```

New

```
100 Jones 24.98

200 Doe 345.67

300 Worthington 0.00

one -42.16

500 Rich 224.62
```

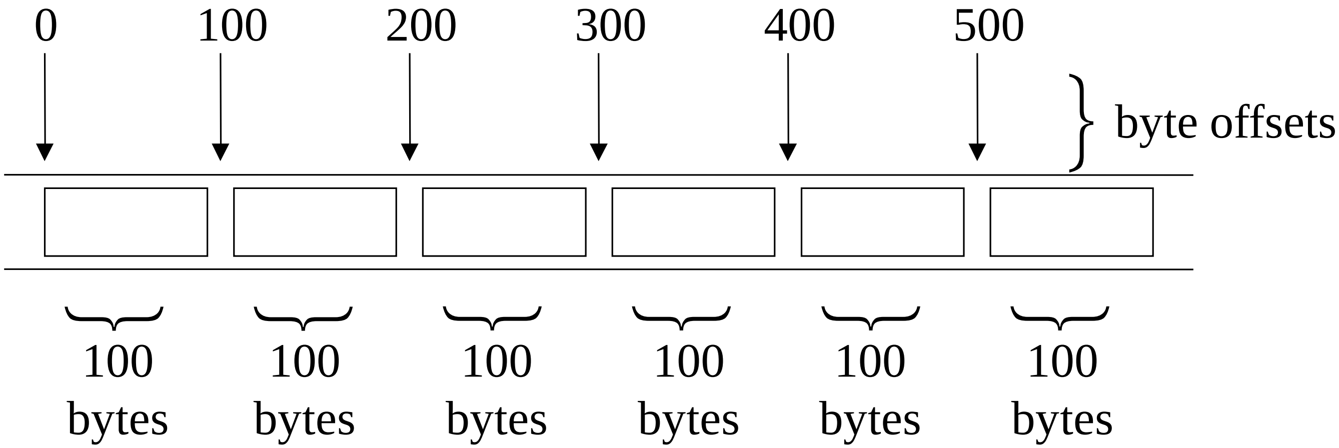
Random
Access Files

11.6 Random-Access Files

- Random access files (Direct access)
  - 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 next record

11.6 Random-Access Files

- Random access files are implemented using fixed length records
  - Sequential files do not have fixed length records



11.7 Creating a Randomly Accessed File

- Data in random access files
  - Unformatted (stored as "raw bytes")
    - All data of the same type (`int`s, for example) uses the same amount of memory
    - Example: `1`, `34`, `-890` are all `int`s, and each takes 4 bytes both in memory and in random access file.
    - All records of the same type have a fixed length
    - Data not human readable (especially the numerical data)

Functions for Random Access Files

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

fwrite and fread Functions

```
fwrite( &variable_name,
       sizeof(variable),
       number_of_blocks,
       filePtr );
```

```
fread( &variable_name,
       sizeof(variable),
       number_of_blocks,
       filePtr );
```

- **`&variable_name`** : Location to transfer bytes from.

- **`sizeof(variable)`** : Number of bytes to transfer.

- **`number_of_ blocks`** : For arrays, number of elements to transfer. Usually 1 element at a time is transferred.

- **`filePtr`** : File pointer

11.7 Creating a Randomly Accessed File

- Example:

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

  - `&studentID` : 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 pointer

11.7 Creating a Randomly Accessed File

- Writing `struct`s

```
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

Example: fwrite an array

- The following program writes (without a loop) an array to a file.

```
#include <stdio.h>
#define N 5
int main()
{
 int dizi[N] = {10,20,30,40,50};
 int M;
 FILE *dosya;
 dosya = fopen("veriler.txt", "w");
 if (!dosya)  {
    printf("Dosya acilamadi\n");
    return 0;
 }
 M = fwrite(dizi, sizeof(int), N, dosya);
 printf( "%d  adet veri yazildi\n",  M);

} //end main
```

Example: fread an array

- The following program reads (without a loop) an array from a file.

```c
#include <stdio.h>
#define  N  5
int main()
{
 int dizi[N];
 int  M, i;
 FILE *dosya;
 dosya = fopen("veriler.txt", "r");
 if (!dosya)  {
    printf("Dosya acilamadi\n");
    return 0;
 }
 M = fread(dizi, sizeof(int), N, dosya);
 printf("%d  adet veri okundu\n",  M);

 for (i=0; i < M; i++)
     printf("%d \n",  dizi[i] );

} //end main
```

Example : Creating an Empty Random Access File

Part 1 of 2

```
// 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; // credit.dat file pointer

   // create clientData with default information
   struct clientData client = { 0, "", "", 0.0 };

   // fopen opens the file; exits if file cannot be opened
   if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
      puts( "File could not be opened." );
   } // end if
   else {
      // require user to specify account number
      printf( "%s", "Enter account number"
         " ( 1 to 100, 0 to end input )\n? " );
      scanf( "%d", &client.acctNum );
```

Part 2 of 2

```
      // user enters information, which is copied into file
      while ( client.acctNum != 0 ) {
         // user enters last name, first name and balance
         printf( "%s", "Enter lastname, firstname, balance\n? " );

         // set record lastName, firstName and balance value
         fscanf( stdin, "%14s%9s%lf", client.lastName,
            client.firstName, &client.balance );

         // seek position in file to user-specified record
         fseek( cfPtr, ( client.acctNum - 1 ) *
            sizeof( struct clientData ), SEEK_SET );

         // write user-specified information in file
         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );

         // enable user to input another account number
         printf( "%s", "Enter account number\n? " );
         scanf( "%d", &client.acctNum );
      } // end while

      fclose( cfPtr ); // fclose closes the file
   } // end else
} // end main
```

PROGRAM OUTPUT :

File with 100 blank records

Acct	Last Name	First Name	Balance
0			0
0			0
0			0
0			0
0			0
…			…
…			…
…			…
0			0
0			0
0			0
0			0

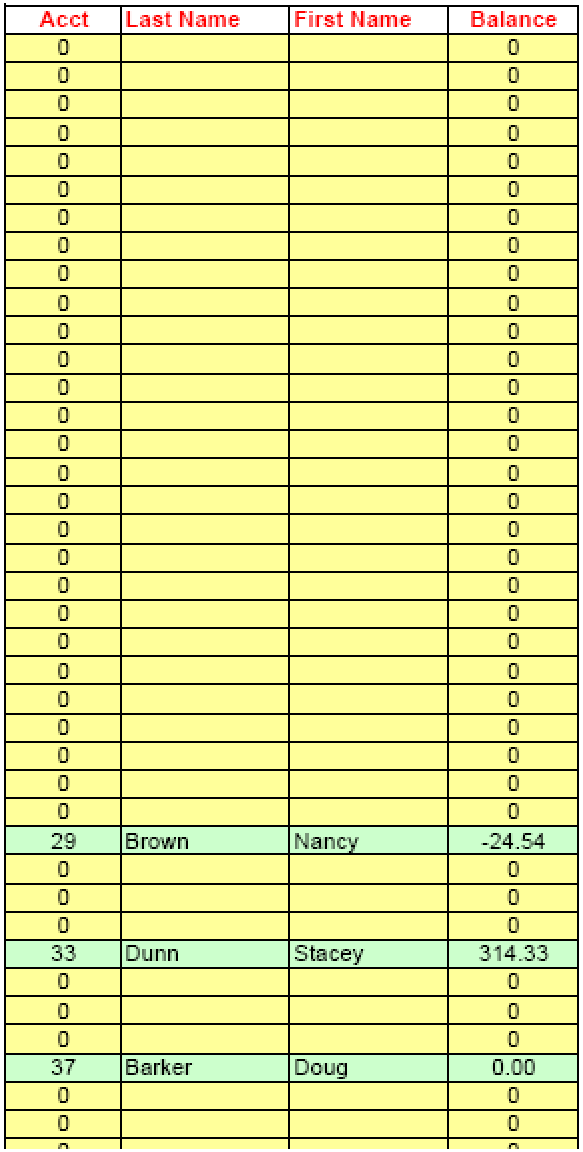
Internal Representation of credit.dat File

- The (credit.dat) file is a random access file, which is not human readable.

- The records of this data file is similar to an array of structs.

- Initially, all records will be blank.

- Add / Delete / Update operations can be implemented as a MODIFY (OVERWRITE) operation.



Acct	Last Name	First Name	Balance
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
0			0
29	Brown	Nancy	-24.54
0			0
0			0
0			0
33	Dunn	Stacey	314.33
0			0
0			0
0			0
37	Barker	Doug	0.00
0			0
0			0
0			0

11.8 Writing Data to a Randomly Accessed 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

11.8 Writing Data to a Randomly Accessed File

```
fseek(cfPtr, 5, SEEK_SET);
```

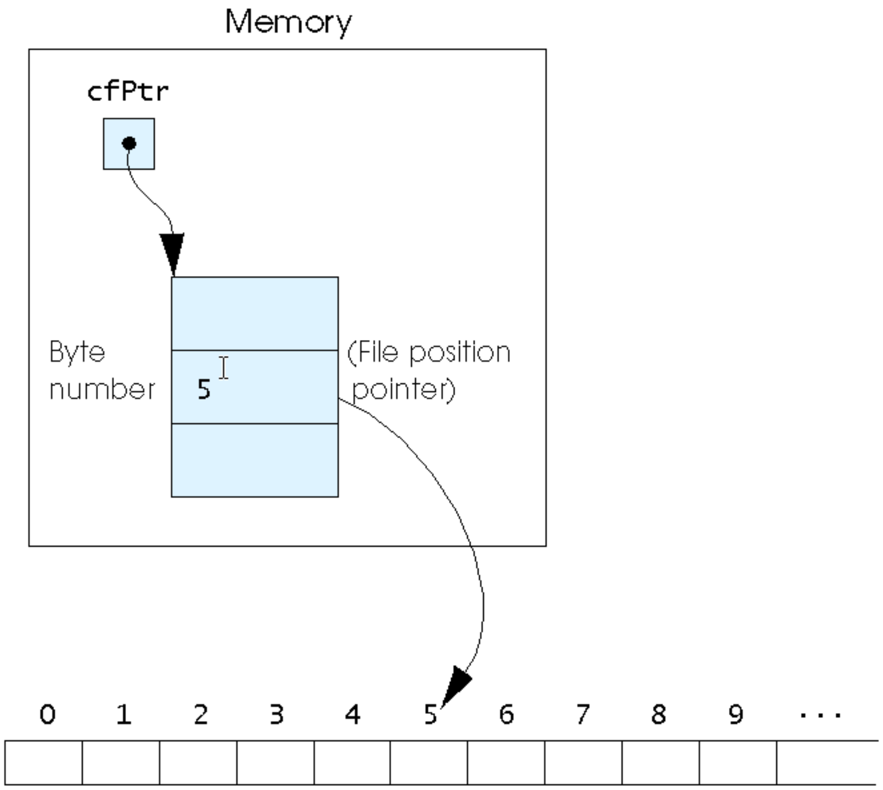


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

Example : Writing Data to a Random Access File

Part 1 of 2

```
/* Writing to a random access file */
#include <stdio.h>

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

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

   // create clientData with default information
   struct clientData client = { 0, "", "", 0.0 };

   // fopen opens the file; exits if file cannot be opened
   if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
  return 0;
   } // end if

      // require user to specify account number
      printf( "Enter account number"
         " ( 1 to 100, 0 to end input )\n? " );
      scanf( "%d", &client.acctNum );
```

Part 2 of 2

```
      // user enters information, which is copied into file
      while ( client.acctNum != 0 ) {

         // user enters last name, first name and balance
         printf( "Enter lastname, firstname, balance\n? " );

         /* set record lastName, firstName and balance value */
         fscanf( stdin, "%s%s%lf", client.lastName,
            client.firstName, &client.balance );

         // seek position in file to user-specified record
         fseek( cfPtr, ( client.acctNum - 1 ) *
            sizeof( struct clientData ), SEEK_SET );

         // write user-specified information in file
         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );

         // enable user to input another account number
         printf( "Enter account number\n? " );
         scanf( "%d", &client.acctNum );
      } // end while

      fclose( cfPtr );

} // end main
```

Program Output

```
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
```

11.9 Reading Data from a Randomly Accessed 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

Example : Reading Data from a Random Access File

Part 1 of 2

```
/* Reading a random access file sequentially */
#include <stdio.h>

struct clientData {
   int acctNum;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
};

int main()
{
   FILE *cfPtr;

   // fopen opens the file; exits if file cannot be opened
   if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
      return 0;
   } // end if

      printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
         "First Name", "Balance" );
```

Part 2 of 2

```
      // read all records from file (until eof)
      while ( !feof( cfPtr ) ) {
         fread( &client, sizeof( struct clientData ),
                1, cfPtr );

         // display record
         if ( client.acctNum != 0 ) {
            printf( "%-6d%-16s%-11s%10.2f\n",
               client.acctNum, client.lastName,
               client.firstName, client.balance );
         } // end if

      } // end while

      fclose( cfPtr );
   } // end main
```

Program Output

```
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
```