

File Processing in C

By: Atiya Jokhio

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
- However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

The Data Hierarchy

- 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: your name
 - Record – group of related fields
 - Represented by a struct or a class
 - Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

The Data Hierarchy (Cont'd)

- Data Hierarchy (continued):
 - File – group of related records
 - Example: payroll file
 - Database – group of related files

The Data Hierarchy (Cont'd)

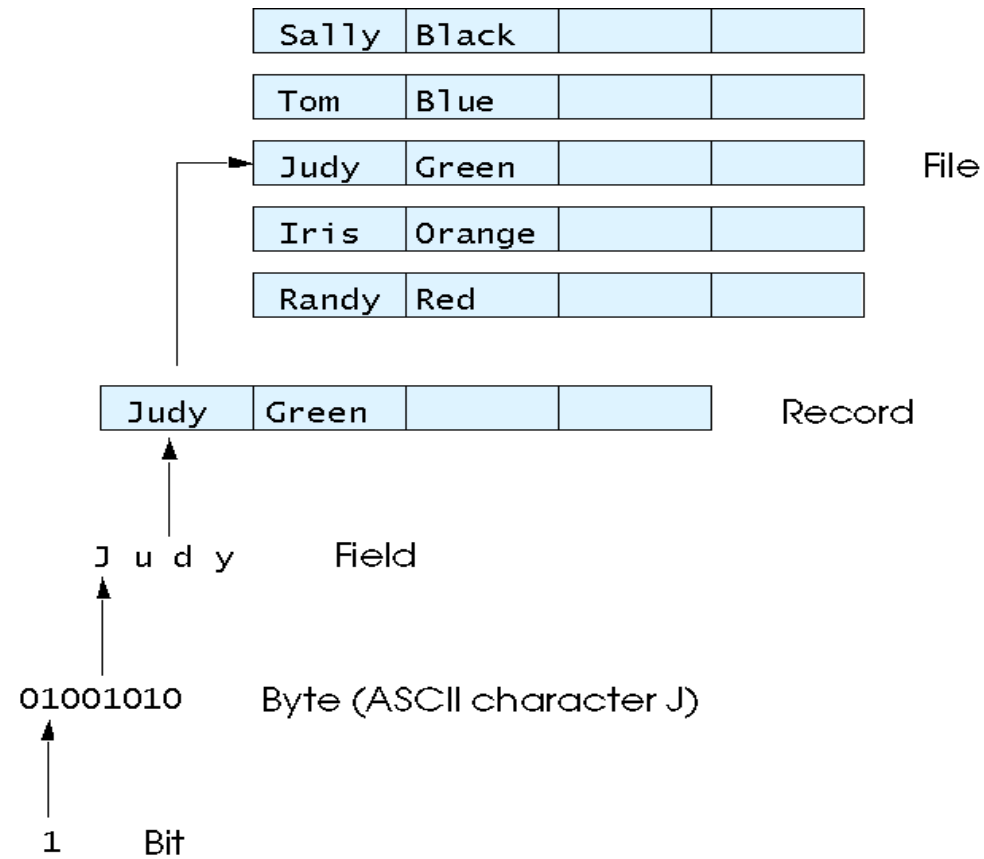


Fig. 11.1 The data hierarchy.

The Data Hierarchy (Cont'd)

- Data files
 - Record key
 - Identifies a record to facilitate the retrieval of specific records from a file
 - Sequential file
 - Records typically sorted by key

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
- 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)

Files and Streams

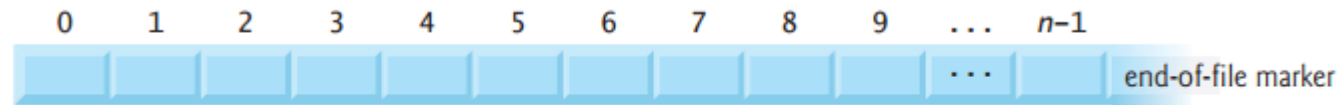


Fig. 11.1 | C's view of a file of n bytes.

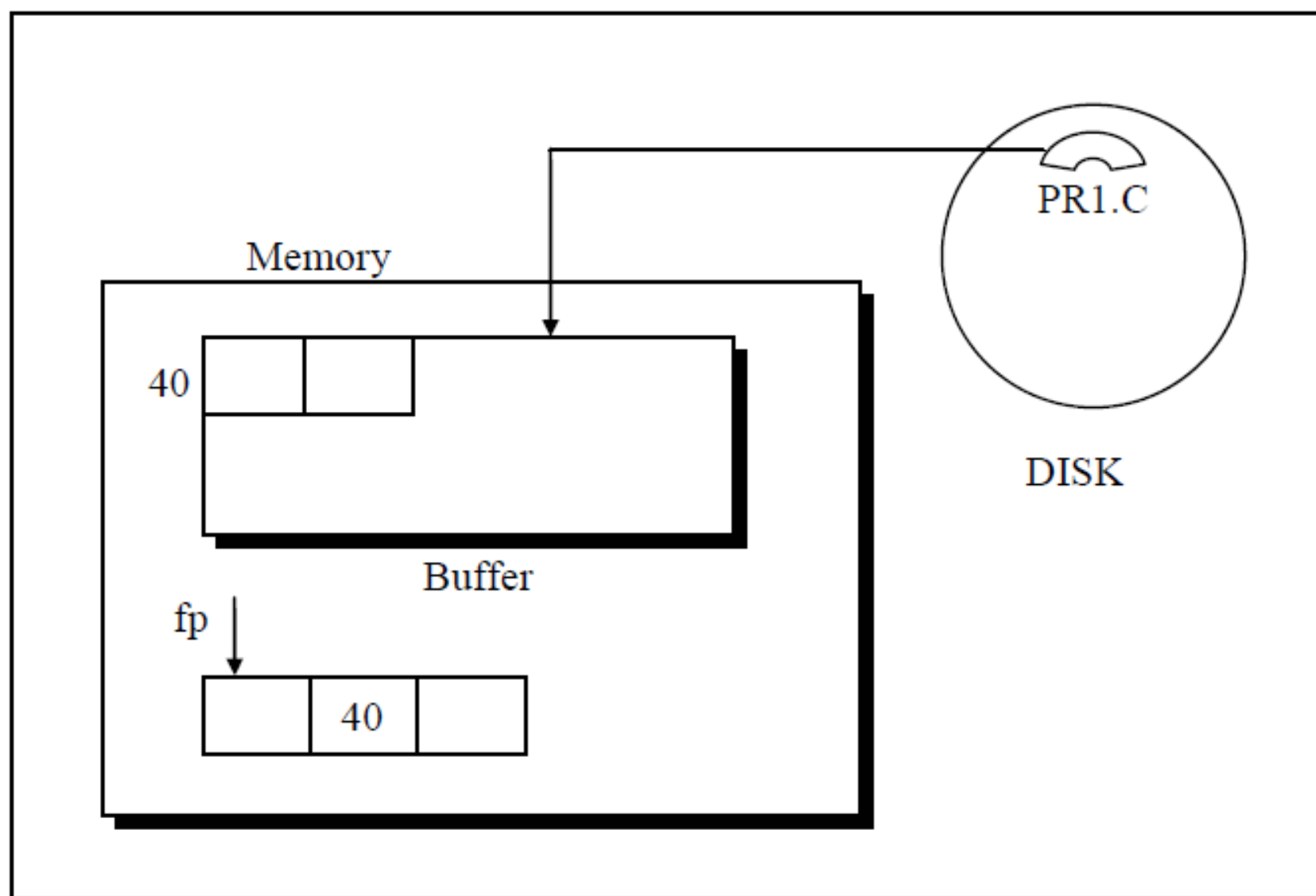
- Opening a file returns a pointer to a FILE structure (defined in `<stdio.h>`) that contains information used to process the file.
- In some operating systems, this structure includes a file descriptor, i.e., an index into an operating system array called the open file table.
- Each array element contains a file control block (FCB)—information that the operating system uses to administer a particular file.

Files and Streams (Cont'd)

- Read/Write functions in standard library
 - 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

Buffer Memory

- Reading and writing to and from the files stored in the disk is relatively slow process when compared to reading and writing data stored in the RAM. As a result, all standard input/output functions uses something called buffer memory to store the data temporarily.
- The buffer memory is a memory where data is temporarily stored before it is written to the file. When the buffer memory becomes full , it finally transfer the contents from the buffer to the disk.



Types of Files

- Text files
 - Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.
 - They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.
- Binary files
 - Binary files are mostly the .bin files in your computer.
 - Instead of storing data in plain text, they store it in the binary form (0's and 1's).
 - They can hold higher amount of data, are not readable easily and provides a better security than text files.

File Operations

There are different operations that can be carried out on a file. These are:

1. Creation of a new file
2. Opening an existing file
3. Reading from a file
4. Writing to a file
5. Closing a file

Working with files

- When working with files, you need to declare a pointer of type file.
 - `FILE *fptr;`
- Opening a file is performed using the library function in the "stdio.h" header file
 - `fptr = fopen("E:\\cprogram\\newprogram.txt","w");`
 - `fptr = fopen("E:\\cprogram\\oldprogram.bin","rb");`
- Perform Read/Write operation.
- The file (both text and binary) should be closed after reading/writing.
 - `fclose(fptr);`

Reading and writing to a text file

- For reading and writing to a text file, we use the functions
 - `fprintf()` // Writing a text file
 - `fscanf()` // Reading from text file
- They are just the file versions of `printf()` and `scanf()`. The only difference is that, `fprintf` and `fscanf` expects a pointer to the structure `FILE`.

Writing to a text file

```
#include <stdio.h>
int main()
{
    int num;
    FILE *fptr;
    fptr = fopen("program.txt","w");
    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }
    printf("Enter number: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);
    return 0;
}
```

Reading from text file

```
#include <stdio.h>
int main()
{
    int num;
    FILE *fptr;
    if ((fptr = fopen("program.txt","r")) == NULL){
        printf("Error! opening file");
        // Program exits if the file pointer returns NULL.
        exit(1);
    }
    fscanf(fptr,"%d", &num);
    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```


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;`
 - Creates 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

Creating a Sequential Access File

Computer system	Key combination
UNIX systems	<i><return> <ctrl> d</i>
IBM PC and compatibles	<i><ctrl> z</i>
Macintosh	<i><ctrl> d</i>
Fig. 11.4 End-of-file key combinations for various popular computer systems.	

Creating a Sequential Access File

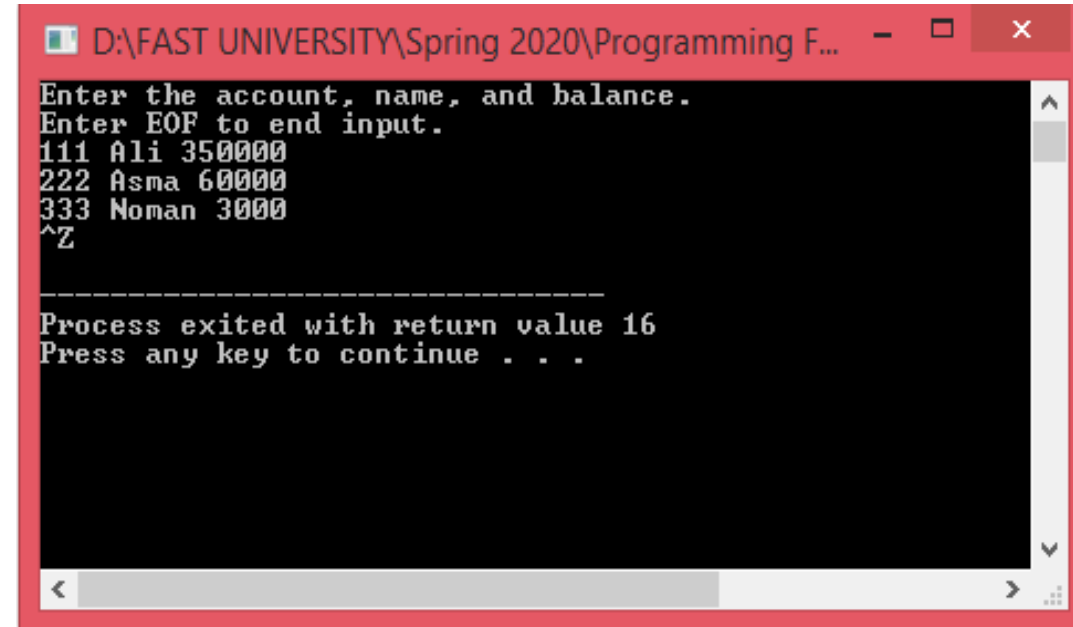
- `fprintf`
 - Used to print to a file
 - Like `printf`, except first argument is a `FILE` pointer (pointer to the file you want to print in)
- `feof(FILE pointer)`
 - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- `fclose(FILE pointer)`
 - Closes specified 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

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	Append: open or create a file for writing at the end of the file.
r+	Open an existing 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.
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.

Creating/ writing to a Sequential Access File

```
#include<stdio.h>
int main()
{
    unsigned int account; // account number
    char name[ 30]; // account name
    double balance; // account balance
    FILE *cfPtr;
    // fopen opens file. Exit program if unable to create file
    if( (cfPtr=fopen("clients.dat","w")) == NULL) {
        puts( "File could not be opened");
    } // end if
    else{
        puts( "Enter the account, name, and balance.");
        puts( "Enter EOF to end input.");
        //printf( "%s", "? ");
        scanf( "%d%29s%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( "%s", "? ");
            scanf( "%d%29s%lf", &account, name, &balance );
        } // end while
    } // end else
} // end main
```



```
D:\FAST UNIVERSITY\Spring 2020\Programming F...
Enter the account, name, and balance.
Enter EOF to end input.
111 Ali 350000
222 Asma 600000
333 Noman 3000
^Z

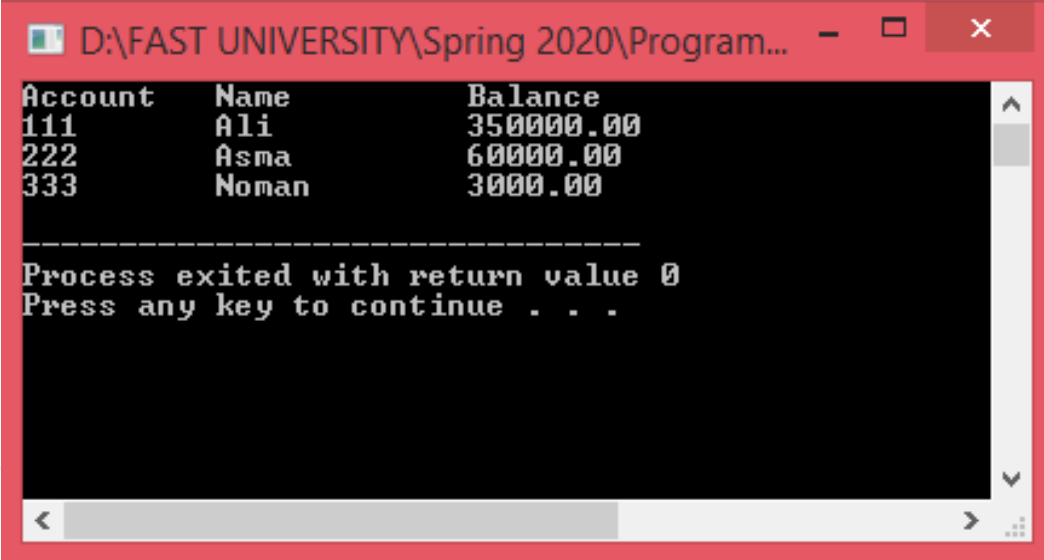
-----
Process exited with return value 16
Press any key to continue . . .
```

Reading from a Sequential Access File

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

    FILE *cfPtr;
    // fopen opens file; exits program if file cannot be opened
    if ( (cfPtr=fopen("clients.dat","r")) == NULL ) {
        puts( "File could not be opened");
    } // end if
    else{
        // read account, name and balance from file
        printf( "%-10s%-13s%\n", "Account", "Name","Balance");
        fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );

        while( !feof(cfPtr)) {
            printf( "%-10d%-13s%7.2f\n", account, name, balance );
            fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );
        }
        fclose( cfPtr );
    }
}
```



```
D:\FAST UNIVERSITY\Spring 2020\Program...
Account      Name          Balance
111          Ali           350000.00
222          Asma          60000.00
333          Noman         3000.00

-----
Process exited with return value 0
Press any key to continue . . .
```

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.

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.

Credit Inquiry Program- Example

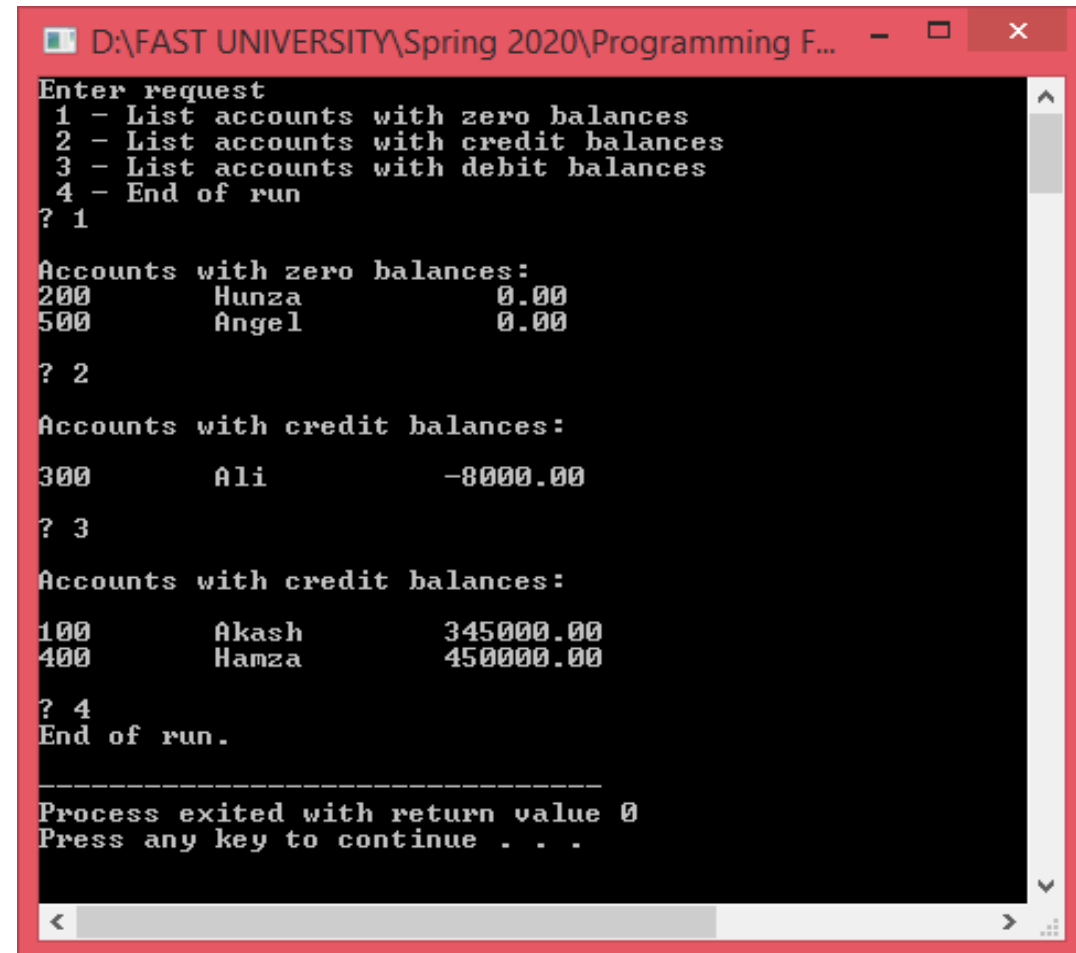
```
// Credit inquiry program
#include<stdio.h>
// function main begins program execution
int main( void)
{
    unsigned int request; // request number
    unsigned int account; // account number
    double balance; // account balance
    char name[ 30]; // account name
    FILE *cfPtr;
    // fopen 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 );

        // 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;
                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
                    }
                }
            }
        }
    }
}
```


Credit Inquiry Program- Example

```
fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );
} // end while
break;
case 3:
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;
} // end switch
rewind( cfPtr );
printf( "%s", "\n? ");
scanf( "%d", &request );
} // end while

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



```
D:\FAST UNIVERSITY\Spring 2020\Programming F...
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:
200      Hunza          0.00
500      Angel          0.00
? 2
Accounts with credit balances:
300      Ali           -8000.00
? 3
Accounts with credit balances:
100      Akash         345000.00
400      Hamza         450000.00
? 4
End of run.

Process exited with return value 0
Press any key to continue . . .
```

- Sequential access file

- Cannot be modified without the risk of destroying other data
- Fields can vary in size
 - Different representation in files and screen than internal representation
 - 1, 34, -890 are all ints, but have different sizes on disk

300 White 0.00 400 Jones 32.87 (old data in file)

If we want to change White's name to Worthington,

300 Worthington 0.00

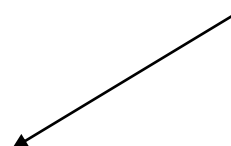


300 White 0.00 400 Jones 32.87



300 Worthington 0.00ones 32.87

Data gets overwritten



Examples

```
//reading the content of a file.
#include<stdio.h>
int main( )
{
FILE *fp;
char ch;
fp = fopen ( "para.txt", "r" );
while( 1 )
{
ch = fgetc ( fp );
if ( ch == EOF )
break;
printf ( "%c", ch );
}
fclose ( fp );
return 0;
}
```

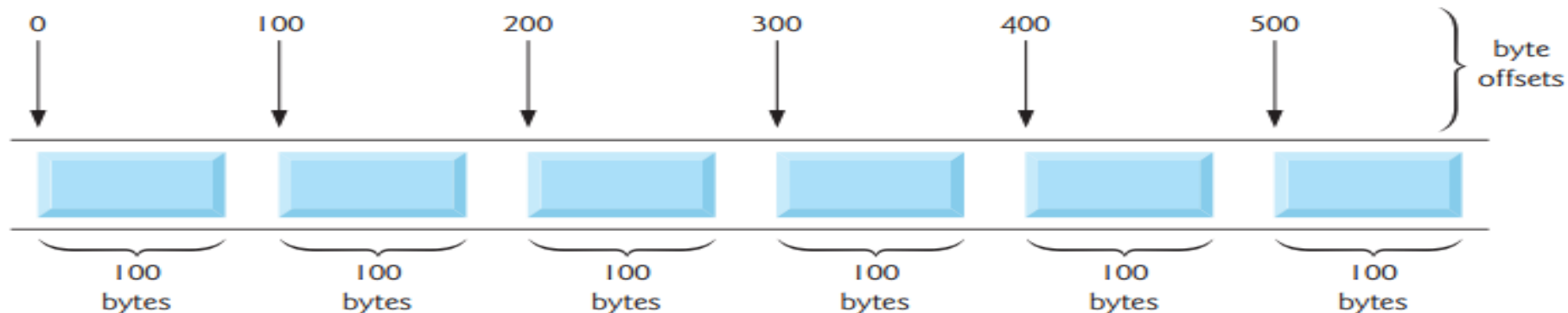
```
//Program: Copying contents of a File to another file
#include<stdio.h>
int main( )
{
FILE *fs,*ft;// fs=> source file and ft => target file
char ch;
fs = fopen("para.txt", "r");
if ( fs == NULL )
{
puts ("cannot open file");
}
ft = fopen ("TargetFile.txt", "w");
while( 1 )
{
ch = fgetc ( fs );
if ( ch == EOF )
break;
else
fputc(ch,ft);
}
fclose(fs);
fclose(ft);
return 0;
}
```

Examples

```
/* Receives strings from keyboard and writes them to file */
#include <stdio.h>
main( )
{
    FILE *fp ;
    char s[100] ;
    fp = fopen ( "Stringiput.txt", "w" ) ;
    if ( fp == NULL )
    {
        puts ( "Cannot open file" ) ;
    }
    printf ( "\nEnter a few lines of text:\n" ) ;
    while ( ( gets ( s ) ) > 0 )
    {
        fputs ( s, fp ) ;
        fputs ( "\n", fp ) ;
    }
    fclose ( fp ) ;
}
```

Random-Access Files

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



Creating a Randomly Accessed File

- Data in random access files
 - Unformatted (stored as "raw bytes")
 - All data of the same type (`ints`, for example) uses the same amount of memory
 - All records of the same type have a fixed length
 - Data not human readable

Reading and writing to a binary file

- To write into a binary file, you need to use the function `fwrite()`. The function takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.
- `// Write on binary file`
- `fwrite(address_data,size_data,numbers_data,pointer_to_file);`
- `// Read from binary file`
- `fread(address_data,size_data,numbers_data,pointer_to_file);`

Creating a Randomly Accessed File

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

```
fwrite( &number, sizeof( int ), 1, myPtr );  
fread( &number, sizeof( int ), 1, myPtr );
```

 - `&number` – 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 to transfer to or from

Creating a Randomly Accessed File

- Writing structs

- `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

Writing to a binary file using fwrite()

```
#include <stdio.h>
struct threeNum
{
    int n1, n2, n3;
};
int main ()
{
    int n;
    struct threeNum num;
    FILE *fptr;
    if ((fptr= fopen("bfile.txt", "wb"))== NULL)
    {
        printf("Error! Opening File");
    }
    for (n=1; n<5; n++)
    {
        num.n1 = n;
        num.n2 = 5*n;
        num.n3= 5*n+1;
        fwrite(&num, sizeof (struct threeNum), 1, fptr);
    }
    fclose(fptr);
    return 0;
}
```

Reading from a binary file using fread()

```
#include <stdio.h>
struct threeNum
{
    int n1, n2, n3;
};
int main ()
{
    int n;
    struct threeNum num;
    FILE *fptr;

    if ((fptr= fopen("bfile.txt", "rb"))== NULL)
    {
        printf("Error! Opening File");
    }
    for (n=1; n<5; n++)
    {
        fread(&num, sizeof (struct threeNum), 1, fptr);
        printf("n1: %d\t n2: %d\t n3: %d", num.n1, num.n2, num.n3);
        printf("\n");
    }
    fclose(fptr);
    return 0;
}
```

Writing Data Randomly to a Randomly Accessed File

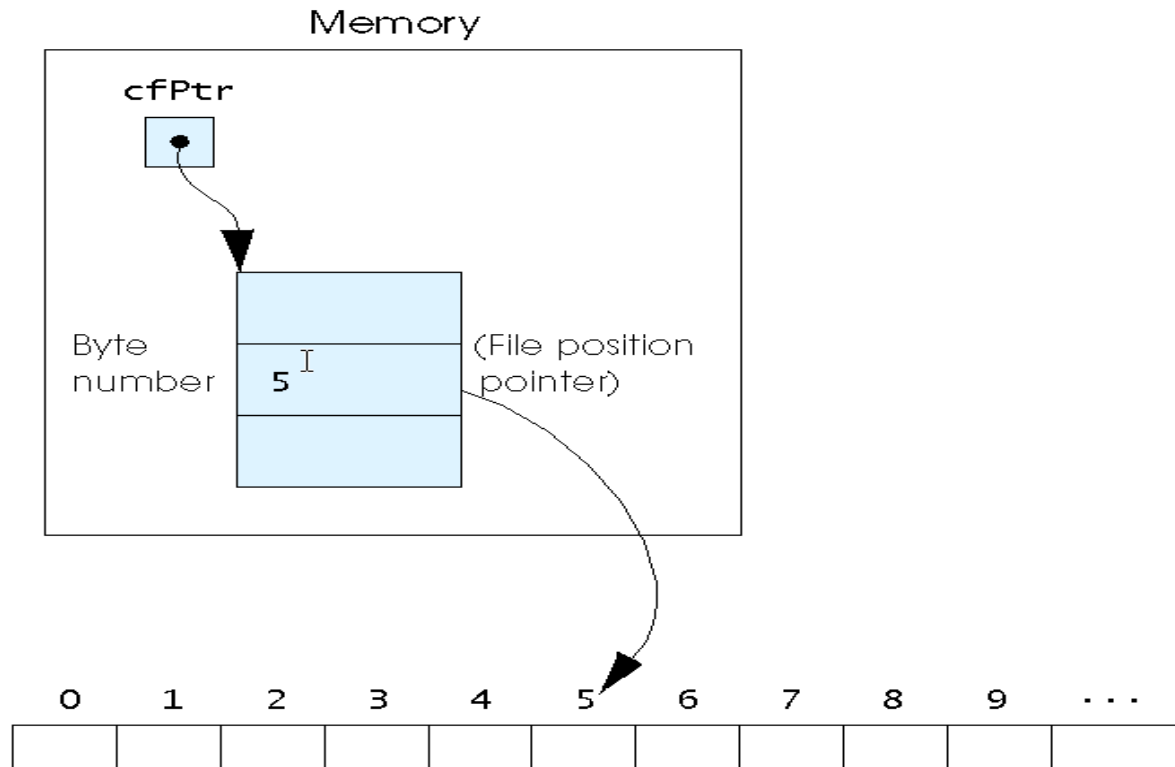


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

Writing Data Randomly to a Randomly Accessed File

- **fseek**
 - Sets file position pointer to a specific position

`int fseek(FILE *stream, int offset, int whence)`

number of bytes to offset from position

pointer to a FILE object

the position from where offset is added

whence defines the point with respect to where the file pointer needs to be moved. It is specified by one of the following constants:

- **SEEK_END**: End of the file.
- **SEEK_SET**: Beginning of the file.
- **SEEK_CUR**: Current position of the file pointer.

- **fseek Example:**

```
/* fseek example */
#include <stdio.h>

int main ()
{
    FILE * pFile;
    pFile = fopen ( "example.txt" , "wb" );
    fputs ( "This is an apple." , pFile );
    fseek ( pFile , 9 , SEEK_SET );
    fputs ( " sam" , pFile );
    fclose ( pFile );
    return 0;
}
```

• Writing Data Randomly to a Randomly Accessed File

```
// 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 );

        // 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
```

```
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
```

• Reading Data from a Randomly Accessed File

```
// 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
}; // end structure clientData
```

```
int main( void )
{
    FILE *cfPtr; // credit.dat file pointer
    int result; // used to test whether fread read any bytes

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

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

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

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

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