

# 9. Struct, Enum and File Processing

DR. LABIB ARAFEH  
YACOUB SABATIN  
MUNTASER ABULAFI

1

## Introduction

- Structure: A **data type** suitable for grouping data elements (several pieces of related information) together →
  - Can contain variables of **different data types** (Arrays allow for a named collection of **identical objects**).
  - Can use other structures, arrays or pointers as some of its members, though this can get **complicated!**
  - Commonly used to define records to be stored in files;
  - Combined with pointers, can create queues and trees
  - **structure definition**: general format →

```
struct tag_name
{
    datatype member1;
    datatype member2;
    ...
}
```

- Lets create a new data structure proper for storing the date

```
struct date {
    int month;
    int day;
    int year;
};
```

Declares a NEW data type (*date*). This date structure consists of three integer elements.

This is a **definition to the compiler**. It does not create any storage space and cannot be used as a variable.

Defines a variable called ***todays\_date*** to be of the same data type as that of the **struct *date***.

```
struct date todays_date;
```

- **ASSIGNING VALUES TO STRUCTURE ELEMENTS:**

- Assign today's date to the individual elements of the structure *todays\_date* as follow:

```
todays_date.day = 30;
todays_date.month = 04;
todays_date.year = 2011;
```

C\_Prog. Spring 2012

3

```
/* Program to illustrate a structure */
#include <stdio.h>
struct date { // global definition of type date
    int month;
    int day;
    int year;
};
main() {
    struct date today;
    today.month = 04;
    today.day = 30;
    today.year = 2011;
    printf("Todays date is %d / %d / %d.\n",
today.month, today.day, today.year );
return 0; }
```

A structure is usually defined before main along with macro definitions

**NOTE** the use of the **.** (DOT) element to reference the individual elements within ***todays\_date***

C\_Prog. Spring 2012

4

```

struct time {
    int hour, minutes, seconds;
};
struct time current_time;
printf("Enter the time (hh:mm:ss):\n");
scanf("%d:%d:%d",&current_time.hour,
&current_time.minutes,&current_time.seconds);
    struct x_struct {
        int a;
        int b;
        int c;
    };
    struct x_struct z;
    z.a = 10;
    z.b = 20;
    z.c = 30;
    z.a++;
    printf("first member is %d \n", z.a);

```

```

struct {
    double real;
    double imag;
} complex;

```

- **INITIALIZING STRUCTURES:** Similar to the initialization of arrays; the elements are simply listed inside a pair of braces, with each element separated by a comma.  
*struct date today = { 04, 23, 2011};*
- **ARRAYS OF STRUCTURES:** Consider the following:  
*struct date {*  
*int month, day, year;*  
*};*  
*struct date birthdays[5]; // Builds an array of 5 cells*  
*birthdays[1].month = 12;*  
*birthdays[1].day = 04;*  
*birthdays[1].year = 2011;*  
*--birthdays[1].year;*

Constructs an array (*birthdays*) of the same data type as the structure *date*

```

struct month {
    int number_of_days;
    char name[4];
};
struct month this_month = { 31, "Jan" };
this_month.number_of_days = 31;
strcpy( this_month.name, "Jan" );
printf("The month is %s\n", this_month.name );

```

Structures can also contain arrays

- VARIATIONS IN DECLARING STRUCTURES: Consider:

```

struct date {
    int month, day, year;
} today's_date, purchase_date;      OR
struct date {
    int month, day, year;
} today's_date = { 9, 30, 2011 };      OR

```

C\_Prog. Spring 2012

7

```

struct date {
    int month, day, year;
} dates[100];

```

An array of structures similar to date

- A program to enter in 5 dates using an array of structures:

```

#include <stdio.h>
struct date { /* Global definition of date */
    int day, month, year;
};
main()
{
    struct date dates[5];
    int i;
    for( i = 0; i < 5; ++i ) {
        printf("Please enter the date (dd:mm:yy)");
        scanf("%d:%d:%d",&dates[i].day,
&dates[i].month, &dates[i].year ); } return 0;}

```

C\_Prog. Spring 2012

8

- **STRUCTURES THAT CONTAIN STRUCTURES:** Consider *date\_time* structure that combines **date** & **time** structures:

```
struct date {
    int month, day, year;
};
```

```
struct time {
    int hours, mins, secs;
};
```

```
struct date_time {
    struct date sdate;
    struct time stime;
};
```

Declares a structure whose elements consist of two other previously declared structures.

Initialization could be done as follows:

```
struct date_time today = { { 2, 11, 2011 }, { 3, 4, 55 } };
```

That sets the *sdate* element of the structure *today* to: 11 / 2 / 2011. *stime* element is initialized to 3 hours, 4 minutes, 55 seconds.

Each item can be referenced if desired:

```
++today.stime.secs;
```

```
if( today.stime.secs == 60) ++today.stime.mins;
```

C\_Prog. Spring 2012

9

- For a typesetting package, individual characters have their character values, other attributes like **font** (normal, *italics*, **bold**) and point size. If our characters have 3 independent attributes, how can they be represented in a single object?
- Let us have a **char** and **two shorts** that will be treated as a single entity:

```
struct wp_char{
    char wp_cval;
    short int wp_font;
    short int wp_psize;
};
```

The tag (wp\_char) only serves the purpose of giving a name to this type of structure and allows us to refer to the type later on.

- After a declaration, the tag can be used like this:

```
struct wp_char x, y;
```

Defines 2 variables (x and y)

- It is quite common to see a structured object being defined with the same name as its structure tag.

```
struct wp_char wp_char;
```

Defines a variable **wp\_char** of type struct **wp\_char**

C\_Prog. Spring 2012

10

- Variables can be defined directly following a structure declaration:

```
struct wp_char{
    char wp_cval;
    short wp_font;
    short wp_psize;
} v1;
struct wp_char v2;
```

The tag (**wp\_char**) only serves the purpose of giving a name to this type of structure and allows us to refer to the type later on.

- We now have two variables, **v1** and **v2**. They are structured objects, each containing three separate *members*: **wp\_cval**, **wp\_font** and **wp\_psize**.
- To access the individual members of the structures:

```
v1.wp_cval = 'x';
v1.wp_font = 1;
v1.wp_psize = 10;
v2 = v1;
```

Individual members of v1 are initialized to suitable values, then the whole of **v1** is copied into **v2** in an assignment.

- **Example: Consider the following structure:**

```
struct lib_books
{
    char title[20];
    char author[15];
    int pages;
    float price;
};
```

Declares a structure to hold the details of 4 fields (title, authors, pages and price). Each field / member belongs to different data type. The tag name(lib\_books) can be used to define objects that have the tag names structure.

Declares book1, book2 and book3 as variables of type struct lib\_books. Each declaration has four elements of the structure lib\_books.

```
struct lib_books book1,book2,book3;
```

```
struct lib_books
{
    char title[20];
    char author[15];
    int pages;
    float price;
} book1,book2,book3;
```

Combines definition and **Declarations** in one statement is valid.

```
/* Example program for using a structure*/
#include <stdio.h>
int main()
{
    struct {
        int id_no;
        char name[20];
        int combination;
        char address[20];
        int age;
    } newstudent;
    printf("Enter the student information\n");
    printf("Now Enter the student id_no\t");
    scanf("%d",&newstudent.id_no);
    printf("Enter the name of the student\t");
    scanf("%s",&newstudent.name);
    printf("Enter the combination of the student\t");
    scanf("%d",&newstudent.combination);
    printf("Enter the address of the student\t");
    scanf("%s",&newstudent.address);
    printf("Enter the age of the student\t");
    scanf("%d",&newstudent.age);
    printf("Student information\n");
    printf("student id_number=%d\n",newstudent.id_no);
    printf("student name=%s\n",newstudent.name);
    printf("student Address=%s\n",newstudent.address);
    printf("students combination=%d\n",newstudent.combination);
    printf("Age of student=%d\n",newstudent.age);
    return 0;}

```

```
Enter the student information
Now Enter the student id_no    12345
Enter the name of the student   Fadi
Enter the combination of the student    678
Enter the address of the student    Jerusalem
Enter the age of the student    18
Student information
student id_number=12345
student name=Fadi
student Address=Jerusalem
students combination=678
Age of student=18
Press any key to continue . . .

```

```

#include <stdio.h>
{
struct info {
    int id_no;
    char name[20];
    char address[20];
    char combination[3];
    int age;
}
main()
{
struct info std[100];
int l,n;
printf("Enter the number of students");
scanf("%d",&n);
printf(" Enter Id_no, name, address,
combination, age\n");

```

C Prog. Spring 2012

15

```

for(l = 0; l < n; l++)
scanf("%d%s%s%s%d",&std[l].id_no,std[l].name,
std[l].address,std[l].combination,&std[l].age);
printf("\n Student information");
for (l=0;l< n;l++)
printf("%d%s%s%s%d\n", std[l].id_no,
std[l].name, std[l].address, std[l].combination,
std[l].age);
return 0; }

```

- Example: Consider a student's record:

```

struct {
    char name[64];
    char course[128];
    int age;
    int year;
} student;
student st_rec;

```

Defines a new type **student** variables of type **student** can be declared as follows.

The variable name is **st\_rec** has members: name, course, age and year.

C Prog. Spring 2012

16

## Enumeration Constants

- Enumeration: Set of integer constants represented by identifiers.
  - Enumeration constants are like symbolic constants whose values are automatically set:
    - Values start at 0 and are incremented by 1
    - Values can be set explicitly with =
    - Need unique constant names

```
enum identifier { enumerator-list }  
enum Months {JAN = 1, FEB, MAR, APR, MAY,  
JUN, JUL, AUG, SEP, OCT, NOV, DEC};
```

Creates a new type **enum Months** in which the identifiers are set to integers 1 to 12

- Enumeration variables can only assume their enumeration constant values (not the integer representations)

C\_Prog. Spring 2012

17

- Example:

```
enum DAY  
{ Saturday,  
  Sunday = 0,  
  Monday,  
  Tuesday, // Tuesday is associated with 2  
  Wednesday,  
  Thursday,  
  Friday } weekday;
```

Defines an enumeration type **day** and declares a variable **weekday** with that type

The value 0 is associated with Saturday by default. The identifier Sunday is explicitly set to 0. The remaining identifiers are given the values 1 through 5 by default.

- **enum DAY today = Wednesday;**
- **enum BOOLEAN**  
**{ false,**  
 **true };**
- **enum BOOLEAN end\_flag, match\_flag;**

A value from the set **DAY** is assigned to the variable **today**

Declares an enumeration data type **BOOLEAN**: false = 0, true = 1

Two variables of type **BOOLEAN**

C\_Prog. Spring 2012

18

```

enum
{
    Mon, Tue, Wed, Thu, Fri, Sat, Sun
} day;
day = Wed;
if(day==Sat || day==Sun) printf("\n day is a weekend day");
else if(day == Wed) printf(" Day is hump day");
enum
{
    Saturday = 0, Sunday = 0, Monday, Tuesday, Wednesday,
    Thursday, Friday
} DAY;
DAY day = Sunday;
if(day == 0) printf("Day is a weekend day");
else if(day == Wednesday) printf("Day is middle of the
work week" );

```

C\_Prog. Spring 2012

19

```

enum score
{
    poor, average, good
};
enum rating
{
    below, average, above
};
enum colors
{
    BLACK, BROWN, RED, ORANGE, YELLOW,
    GREEN, BLUE, VIOLET, GREY, WHITE
};

```

This is equivalent to:  

```

#define poor 0
#define average 1
#define good 2

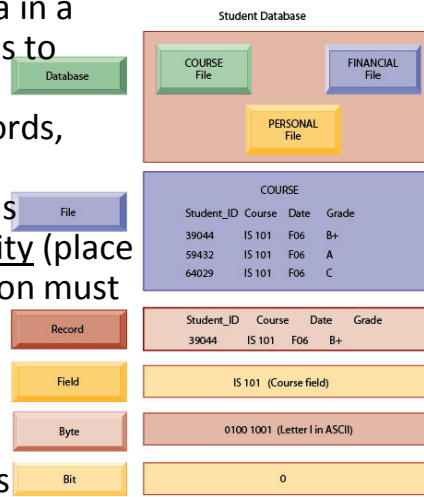
```

C\_Prog. Spring 2012

20

# File Processing

- Data files: Created, updated, & processed by C programs.
  - Used for **permanent** storage of large amounts of data: Storage of data in variables & arrays is only **temporary**.
- Computer systems organize data in a hierarchy: **bits, bytes**, progresses to complex groupings of data:
  - **Fields**: Group of characters, words, or a complete number.
  - **Records**: Group of related fields (struct/class), describes an **entity** (place person) about which information must be kept. Each characteristic of an entity is an **attribute**.
  - **File**: Group of records of the same type.
  - **Database**: Group of related files



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

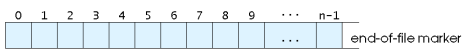
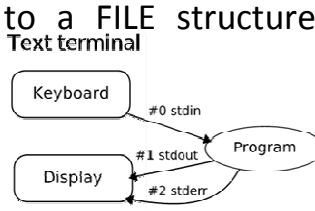


Fig. 11.2 C's view of a file of n bytes.

- Example file pointers:
- **Stdin**: Standard input (keyboard)
  - **Stdout**: Standard output (screen)
  - **Stderr**: Standard error (screen)



- **FILE** structure:
  - File descriptor: Index into O/S array (the open file table).
  - File Control Block (FCB): Found in every array element, system uses it to administer the file.
- 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 equals **scanf** & **printf**

C\_Prog. Spring 2012

23

```

1  /* Fig. 11.3: fig11_03.c
2     Create a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;    /* account number */
8     char name[ 30 ]; /* account name */
9     double balance; /* account balance */
10
11     FILE *cfPtr;    /* cfPtr = clients.dat file pointer */
12
13     /* fopen opens file. Exit program if unable to create file */
14     if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15         printf( "File could not be opened\n" );
16     } /* end if */
17     else {
18         printf( "Enter the account, name, and balance.\n" );
19         printf( "Enter EOF to end input.\n" );
20         printf( "? " );
21         scanf( "%d%s%lf", &account, name, &balance );
22

```

24

```

23  /* write account, name and balance into file with fprintf */
24  while ( !feof( stdin ) ) {
25      fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26      printf( "? " );
27      scanf( "%d%s%f", &account, name, &balance );
28  } /* end while */
29
30      fclose( cfPtr ); /* fclose closes file */
31  } /* end else */
32
33  return 0; /* indicates successful termination */
34
35 } /* end main */

```

```

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

```

25

## Constructing a Sequential Access File

- C imposes no file structure: Programmer must provide file structure
- Constructing a File
  - FILE \*cfPtr;** // Creates a FILE pointer called cfPtr
  - cfPtr = fopen("clients.dat", "w");**
    - **fopen** function returns a FILE pointer to file specified
    - Takes two arguments: file to open and file open mode
    - If open fails, NULL is returned.


Computer system	Key combination
UNIX systems	<return> <ctrl> d
IBM PC and compatibles	<ctrl> z
Macintosh	<ctrl> d

Fig. 11.4 End-of-file key combinations for various popular computer systems.

- **fprintf**: Used to print to a file: Like printf, except first argument is a FILE pointer (pointer to the file we 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 process no files, one file, or many files.

- Each file  
must have a unique name  
and should have its own pointer.

• File Open Modes: 

Mode	Description
<b>r</b>	Open a file for reading.
<b>w</b>	Create a file for writing. If the file already exists, discard the current contents.
<b>a</b>	Append; open or create a file for writing at end of file.
<b>r+</b>	Open a file for update (reading and writing).
<b>w+</b>	Create a file for update. If the file already exists, discard the current contents.
<b>a+</b>	Append; open or create a file for update; writing is done at the end of the file.
<b>rb</b>	Open a file for reading in binary mode.
<b>wb</b>	Create a file for writing in binary mode. If the file already exists, discard the current contents.
<b>ab</b>	Append; open or create a file for writing at end of file in binary mode.
<b>rb+</b>	Open a file for update (reading and writing) in binary mode.
<b>wb+</b>	Create a file for update in binary mode. If the file already exists, discard the current contents.
<b>ab+</b>	Append; open or create a file for update in binary mode; writing is done at the end of the file.

Fig. 11.6 File open modes.

- Reading a sequential access file
  - Construct 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 is read from beginning to end
  - **File position pointer** (An integer value that specifies byte location). Also called byte offset:
    - Indicates number of next byte to be read / written
  - **rewind( cfPtr )**: Repositions file position pointer to beginning of file (byte 0).

```

1  /* Fig. 11.7: fig11_07.c
2     Reading and printing a sequential file */
3  #include <stdio.h>
4
5  int main()
6  {
7     int account;    /* account number */
8     char name[ 30 ]; /* account name */
9     double balance; /* account balance */
10
11     FILE *cfPtr;    /* cfPtr = clients.dat file pointer */
12
13     /* fopen opens file; exits program if file cannot be opened */
14     if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
15         printf( "File could not be opened\n" );
16     } /* end if */
17     else { /* read account, name and balance from file */
18         printf( "%-10s%-13s\n", "Account", "Name", "Balance" );
19         fscanf( cfPtr, "%d%s%f", &account, name, &balance );
20
21         /* while not end of file */
22         while ( !feof( cfPtr ) ) {
23             printf( "%-10d%-13s%7.2f\n", account, name, balance );
24             fscanf( cfPtr, "%d%s%f", &account, name, &balance );
25         } /* end while */
26

```

29

```

27     fclose( cfPtr ); /* fclose closes the file */
28 } /* end else */
29
30 return 0; /* indicates successful termination */
31
32 } /* end main */

```

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

30

```

1  /* Fig. 11.8: fig11_08.c
2     Credit inquiry program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     int request;    /* request number */
9     int account;   /* account number */
10    double balance; /* account balance */
11    char name[ 30 ]; /* account name */
12    FILE *cfPtr;   /* clients.dat file pointer */
13
14    /* fopen opens the file; exits program if file cannot be opened */
15    if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
16        printf( "File could not be opened\n" );
17    } /* end if */
18    else {
19
20        /* display request options */
21        printf( "Enter request\n"
22              " 1 - List accounts with zero balances\n"
23              " 2 - List accounts with credit balances\n"
24              " 3 - List accounts with debit balances\n"
25              " 4 - End of run?\n? " );

```

31

```

26    scanf( "%d", &request );
27
28    /* process user's request */
29    while ( request != 4 ) {
30
31        /* read account, name and balance from file */
32        fscanf( cfPtr, "%d%1f", &account, name, &balance );
33
34        switch ( request ) {
35
36            case 1:
37                printf( "\nAccounts with zero balances:\n" );
38
39                /* read file contents (until eof) */
40                while ( !feof( cfPtr ) ) {
41
42                    if ( balance == 0 ) {
43                        printf( "%-10d%-13s%7.2f\n",
44                              account, name, balance );
45                    } /* end if */
46
47                    /* read account, name and balance from file */
48                    fscanf( cfPtr, "%d%1f",
49                          &account, name, &balance );
50                } /* end while */
51

```

32

```

52         break;
53
54     case 2:
55         printf( "\nAccounts with credit balances:\n" );
56
57         /* read file contents (until eof) */
58         while ( !feof( cfPtr ) ) {
59
60             if ( balance < 0 ) {
61                 printf( "%-10d%-13s%7.2f\n",
62                     account, name, balance );
63             } /* end if */
64
65             /* read account, name and balance from file */
66             fscanf( cfPtr, "%d%s%1f",
67                 &account, name, &balance );
68         } /* end while */
69
70         break;
71
72     case 3:
73         printf( "\nAccounts with debit balances:\n" );
74

```

33

```

75         /* read file contents (until eof) */
76         while ( !feof( cfPtr ) ) {
77
78             if ( balance > 0 ) {
79                 printf( "%-10d%-13s%7.2f\n",
80                     account, name, balance );
81             } /* end if */
82
83             /* read account, name and balance from file */
84             fscanf( cfPtr, "%d%s%1f",
85                 &account, name, &balance );
86         } /* end while */
87
88         break;
89
90     } /* end switch */
91
92     rewind( cfPtr ); /* return cfPtr to beginning of file */
93
94     printf( "\n? " );
95     scanf( "%d", &request );
96 } /* end while */
97

```

34

```

98     printf( "End of run.\n" );
99     fclose( cfPtr ); /* fclose closes the file */
100 } /* end else */
101
102     return 0; /* indicates successful termination */
103
104 } /* end main */

```

```

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.

```

- **Reading Data from a Sequential Access File:**

- Sequential access file cannot be modified without the risk of destroying other data.

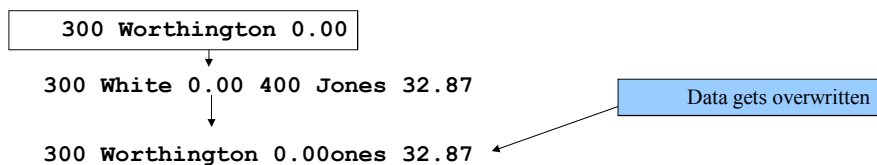
- Fields can vary in size

- Different representation in files and screen than internal representation

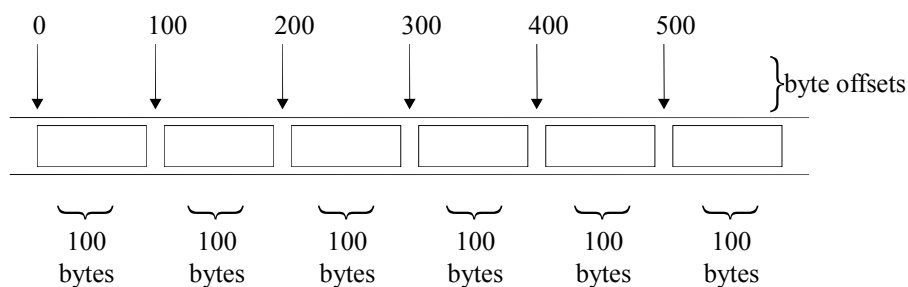
- 1, 34, -890 are all integers, but have different sizes on disk

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

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



- **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



C\_Prog. Spring 2012

37

- **Constructing a Randomly Accessed File:** Data in random access files: Unformatted (stored as "raw bytes") →
  - All data of the same type uses same amount of memory
  - All records of the same type have a fixed length
  - Data not human readable
- Unformatted I/O functions
  - **fwrite:** Transfer bytes from a location in memory to a file  
**`fwrite( &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
  - **fread:** Transfer bytes from a file to a location in memory

C\_Prog. Spring 2012

38

- 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

```

1 /* Fig. 11.11: fig11_11.c
2    Creating a randomly accessed file sequentially */
3 #include <stdio.h>
4
5 /* clientData structure definition */
6 struct clientData {
7     int acctNum; /* account number */
8     char lastName[ 15 ]; /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15     int i; /* counter */
16
17     /* create clientData with no information */
18     struct clientData blankClient = { 0, "", "", 0.0 };
19
20     FILE *cfPtr; /* credit.dat file pointer */
21
22     /* fopen opens the file; exits if file cannot be opened */
23     if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
24         printf( "File could not be opened.\n" );
25     } /* end if */

```

```

26  else {
27
28      /* output 100 blank records to file */
29      for ( i = 1; i <= 100; i++ ) {
30          fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
31      } /* end for */
32
33      fclose ( cfPtr ); /* fclose closes the file */
34  } /* end else */
35
36  return 0; /* indicates successful termination */
37
38 } /* end main */

```

41

### Writing Data Randomly 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

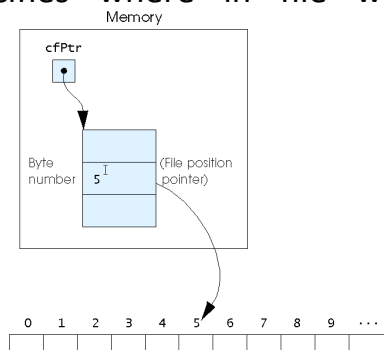


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

```

1  /* Fig. 11.12: fig11_12.c
2     Writing to a random access file */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum;      /* account number */
8     char lastName[ 15 ]; /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance;  /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15     FILE *cfPtr; /* credit.dat file pointer */
16
17     /* create clientData with no information */
18     struct clientData client = { 0, "", "", 0.0 };
19
20     /* fopen opens the file; exits if file cannot be opened */
21     if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
22         printf( "File could not be opened.\n" );
23     } /* end if */
24     else {
25

```

```

26     /* require user to specify account number */
27     printf( "Enter account number"
28            " ( 1 to 100, 0 to end input )\n? " );
29     scanf( "%d", &client.acctNum );
30
31     /* user enters information, which is copied into file */
32     while ( client.acctNum != 0 ) {
33
34         /* user enters last name, first name and balance */
35         printf( "Enter lastname, firstname, balance\n? " );
36
37         /* set record lastName, firstName and balance value */
38         fscanf( stdin, "%s%s%lf", client.lastName,
39                client.firstName, &client.balance );
40
41         /* seek position in file of user-specified record */
42         fseek( cfPtr, ( client.acctNum - 1 ) *
43                sizeof( struct clientData ), SEEK_SET );
44
45         /* write user-specified information in file */
46         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );
47
48         /* enable user to specify another account number */
49         printf( "Enter account number\n? " );
50         scanf( "%d", &client.acctNum );

```

```

51     } /* end while */
52
53     fclose( cfPtr ); /* fclose closes the file */
54 } /* end else */
55
56 return 0; /* indicates successful termination */
57
58 } /* 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

```

45

### Reading Data Randomly 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.

```

1  /* Fig. 11.15: fig11_15.c
2     Reading a random access file sequentially */
3  #include <stdio.h>
4
5  /* clientData structure definition */
6  struct clientData {
7     int acctNum; /* account number */
8     char lastName[ 15 ]; /* account last name */
9     char firstName[ 10 ]; /* account first name */
10    double balance; /* account balance */
11 }; /* end structure clientData */
12
13 int main()
14 {
15     FILE *cfPtr; /* credit.dat file pointer */
16
17     /* create clientData with no information */
18     struct clientData client = { 0, "", "", 0.0 };
19
20     /* fopen opens the file; exits if file cannot be opened */
21     if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
22         printf( "File could not be opened.\n" );
23     } /* end if */

```

47

```

24     else {
25         printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
26             "First Name", "Balance" );
27
28         /* read all records from file (until eof) */
29         while ( !feof( cfPtr ) ) {
30             fread( &client, sizeof( struct clientData ), 1, cfPtr );
31
32             /* display record */
33             if ( client.acctNum != 0 ) {
34                 printf( "%-6d%-16s%-11s%10.2f\n",
35                     client.acctNum, client.lastName,
36                     client.firstName, client.balance );
37             } /* end if */
38
39         } /* end while */
40
41         fclose( cfPtr ); /* fclose closes the file */
42     } /* end else */
43
44     return 0; /* indicates successful termination */
45
46 } /* end main */

```

48

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

49

## A Transaction Processing Project

- In three-student groups develop an interactive Transaction Processing System to demonstrate using random access files to achieve instant access processing of a bank's account information.
- Your system should:
  - Update existing accounts
  - Add new accounts
  - Delete accounts
  - Store a formatted listing of all accounts in a text file.
- Presentations and submissions of soft and hard copies on \_\_\_\_/\_\_\_\_/\_\_\_\_

C\_Prog. Spring 2012

50