FILE INPUT/OUTPUT

Erkut ERDEM

Hacettepe University

December 2010

Content

- In this chapter, you will learn:
  - To be able to create, read, write and update files.
  - To become familiar with sequential access file processing.
  - To become familiar with random-access file processing.

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

- When you use a file to store data for use by a program, that file usually consists of text (alphanumeric data) and is therefore called a **text file.**

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

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

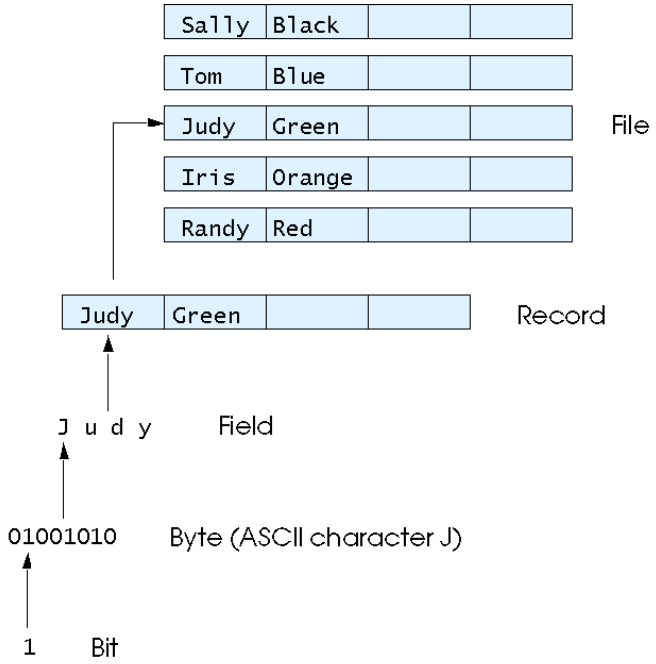


Fig. 11.1 The data hierarchy.

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

- `FILE` structure
  - File descriptor
    - Index into operating system array called the open file table
  - File Control Block (FCB)
    - Found in every array element, system uses it to administer the file

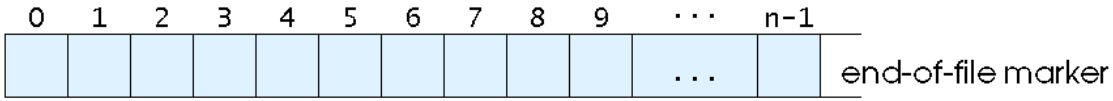


Fig. 11.2 C's view of a file of *n* bytes.

Files and Streams

- Read/Write functions in standard library
  - `fscanf` / `fprintf`
    - File processing equivalents of `scanf` and `printf`
  - `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

```c
/*
   Create a sequential file */
#include <stdio.h>

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

   if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      printf( "Enter the account, name, and balance.\n" );
      printf( "Enter EOF to end input.\n" );
      printf( "? " );
      scanf( "%d%s%lf", &account, name, &balance );

      while ( !feof( stdin ) ) {
         fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );

         printf( "? " );
         scanf( "%d%s%lf", &account, name, &balance );
      }

      fclose( cfPtr );
   }

   return 0;
}
```

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
```

Creating a Sequential Access File

- Creating a File
  - `FILE *myPtr;`
    - Creates a `FILE` pointer called `myPtr`
  - `myPtr = fopen(`*`filename`*`, `*`openmode`*`);`
    - Function `fopen` returns a `FILE` pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, `NULL` 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.

Creating a Sequential Access File

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.
`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.

Fig. 11.6 File open modes.

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

Reading Data from a File

- Reading a sequential access file
  - Create a `FILE` pointer, link it to the file to read

    ```
    myPtr = fopen( “myfile.dat", "r" );
    ```
  - Use `fscanf` to read from the file
    - Like `scanf`, except first argument is a `FILE` pointer

    ```
    fscanf( myPtr, "%d%s%f", &accounnt, 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( myPtr )`
    - Repositions file position pointer to beginning of file (byte 0)

```c

/* Reading and printing a sequential file */
#include <stdio.h>

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

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

      fclose( cfPtr );
   }

   return 0;
}
```

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

Example: Merge two files

```
#include <stdio.h>
int main()
{   FILE *fileA,  /* first input file */
         *fileB,  /* second input file */
         *fileC; /*  output file to be created */
   int num1,  /* number to be read from first file */
       num2;  /* number to be read from second file */
   int f1, f2;

   /* Open files for processing */
   fileA = fopen("class1.txt","r");
   fileB = fopen("class2.txt","r");
   fileC = fopen("class.txt","w");
```

```
 /* As long as there are numbers in both files, read and compare numbers one
by one. Write the smaller number to the output file and read the next number
in the file from which the smaller number is read. */

   f1 = fscanf(fileA, "%d", &num1);
   f2 = fscanf(fileB, "%d", &num2);

   while ((f1!=EOF) && (f2!=EOF)){
      if (num1 < num2){
         fprintf(fileC,"%d\n", num1);
         f1 = fscanf(fileA, "%d", &num1);
      }
      else if (num2 < num1) {
         fprintf(fileC,"%d\n", num2);
         f2 = fscanf(fileB, "%d", &num2);
      }
      else {  /* numbs are equal:read from both files */
         fprintf(fileC,"%d\n", num1);
         f1 = fscanf(fileA, "%d", &num1);
         f2 = fscanf(fileB, "%d", &num2);
      }
   }
```

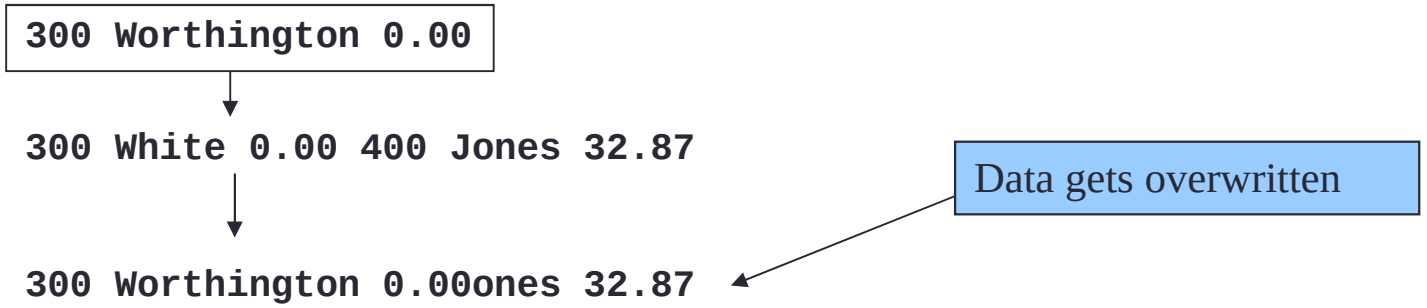
```
   while (f1!=EOF){/* if reached end of second file, read
          the remaining numbers from first file and write to
          output file */
      fprintf(fileC,"%d\n", num1);
      f1 = fscanf(fileA, "%d", &num1);
   }
   while (f2!=EOF){ if reached the end of first file, read
          the remaining numbers from second file and write
          to output file */
      fprintf(fileC,"%d\n", num2);
      f2 = fscanf(fileB, "%d", &num2);
   }

   /* close files */
   fclose(fileA);
   fclose(fileB);
   fclose(fileC);
   return 0;
}  /* end of main */
```

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 `ints`, but have different sizes on disk

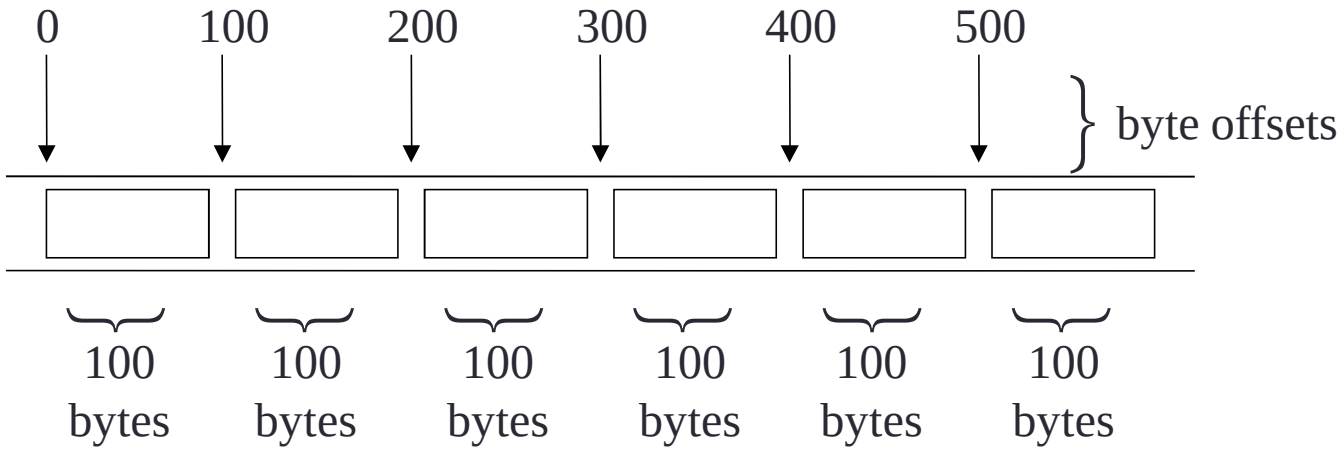
300 White 0.00 400 Jones 32.87 (old data in file)

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



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 (**`int`**s, for example) uses the same amount of memory
    - All records of the same type have a fixed length
    - Data not human readable

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 );
    ```

    - `&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

```c
/* Fig. 11.11: fig11_11.c
   Creating a randomly accessed file sequentially */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   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()
{
   int i; /* counter */

   /* create clientData with no information */
   struct clientData blankClient = { 0, "sevil", "sen", 5000.0 };

   FILE *cfPtr; /* credit.dat file pointer */
```

```
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 */
```

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

```
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
```

Writing Data Randomly to a Randomly Accessed File

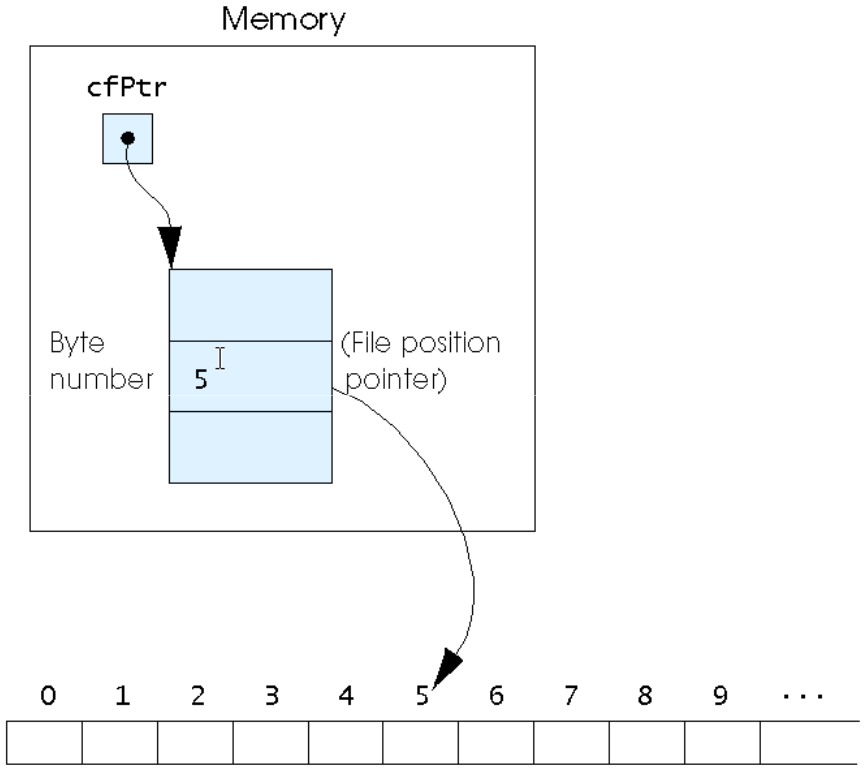


Fig. 11.14 The file position pointer indicating an offset of 5 bytes from the beginning of the file.

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

```c
/* Fig. 11.15: fig11_15.c
   Reading a random access file sequentially */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   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()
{
   FILE *cfPtr; /* credit.dat file pointer */

   /* create clientData with no 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" );
   } /* end if */
```

```
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;
45
46 } /* 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
```