11

C File Processing

11.2 Data Hierarchy

- **Data files**
  - **Are used for permanent storage of large amounts of data**
    - **Storage of data in variables and arrays is only temporary**
- **Data Hierarchy:**
  - **Byte – 8 bits**
    - **Could be 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.**
  - **File – group of related records**
    - **Example: payroll file**
  - **Database – group of related files**

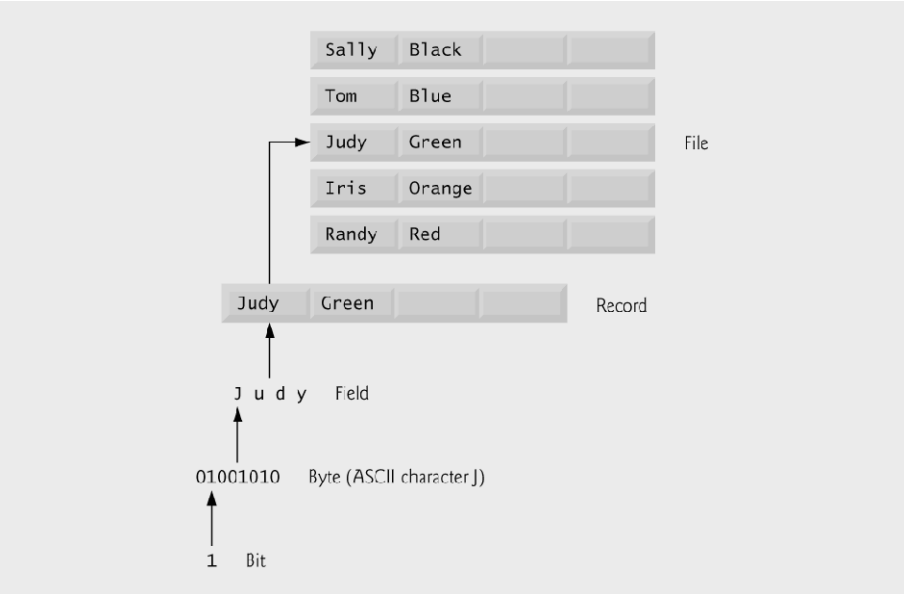
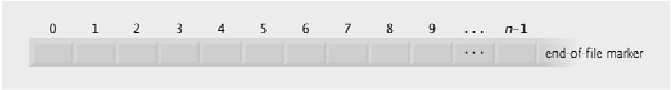


Fig. 11.1 | Data hierarchy.

11.3 Files and Streams

- **C views each file as a sequence of bytes**
  - **File ends with the *end-of-file marker***
    - **Ctrl+z in Windows, Ctrl+d in UNIX/Linux**



- **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)**

11.3 Files and Streams

- **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`**

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;`
    - **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**

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
- `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**

Outline

fig11_03.c

(1 of 2)

```
1  /* Fig. 11.3: fig11_03.c
2     Create a sequential file */
3  #include <stdio.h>
4
5  int main( void )
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
```

FILE pointer definition creates new file pointer

fopen function opens a file; **w** argument means the file is opened for writing

```
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%lf", &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
```

feof returns true when end of file is reached

fprintf writes a string to a file

fclose closes a file

Outline

fig11_03.c

(2 of 2)

Common Programming Error 11.1

Opening an existing file for writing (`"w"`) when, in fact, the user wants to preserve the file, discards the contents of the file without warning.

Good Programming Practice 11.1

Explicitly close each file as soon as it is known that the program will not reference the file again.

→ Closing a file can free resources for which other users or programs may be waiting.

Common Programming Error 11.4

Opening a nonexistent file for reading is an error.

→ A NULL pointer will be returned

Common Programming Error 11.6

Opening a file for writing when no disk space is available is an error.

→ The typical error when opening a file for write

No writing permission is another possibility to get an error while opening a file for write.

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.

Fig. 11.6 | File opening modes.

Common Programming Error 11.7

Opening a file with the incorrect file mode is a logic error. For example, opening a file in write mode (`"w"`) when it should be opened in update mode (`"r+"`) causes the contents of the file to be discarded.

→ Open a file only for reading (and not update) if the contents of the file should not be modified.

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", &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( cfPtr )`**
    - **Repositions file position pointer to beginning of file (byte `0`)**

Outline

fig11_08.c

(1 of 4)

```
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( void )
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? " );
26       scanf( "%d", &request );
27
```

fopen function opens a file; **r** argument means the file is opened for reading

Outline

fig11_08.c

(2 of 4)

```
28       /* process user's request */
29       while ( request != 4 ) {
30
31          /* read account, name and balance from file */
32          fscanf( cfPtr, "%d%s%lf", &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%s%lf",
49                      &account, name, &balance );
50                } /* end while */
51
52                break;
53
```

fscanf function reads a string from a file

```
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%lf",
67                      &account, name, &balance );
68                } /* end while */
69
70                break;
71
72             case 3:
73                printf( "\nAccounts with debit balances:\n" );
74
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
```

Outline

fig11_08.c

(3 of 4)


```
83                   /* read account, name and balance from file */
84                   fscanf( cfPtr, "%d%s%lf",
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
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 */
```

rewind function moves the file pointer back to the beginning of the file

Outline

fig11_08.c

(4 of 4)

```
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.
```

Outline

11.5 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 `int`s, but have different sizes on disk**
  - **Example: change name "White" to "Worthington"**
    - **Old data**

```
300 White 0.00 400 Jones 32.87
```

    - **Insert new data**

```
300 Worthington 0.00
```

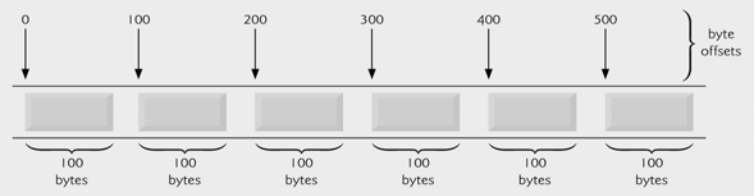
```
300 White 0.00 400 Jones 32.87
```

Data gets overwritten

```
300 Worthington 0.00ones 32.87
```

11.6 Random-Access Files

- **Random access files**
  - **Access individual records without searching entire file**
  - **Instant access to records in a file**
  - **Data can be inserted without destroying other data**
  - **Previous data can be updated or deleted without overwriting**



- **Implemented using fixed length records**
  - **Sequential files do not have fixed length records**
    - **Must be specified by yourself, ex: `fprintf(fp,"%8s",name);`**

11.8 Writing Data Randomly to a Random-Access 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**

11.7 Creating a Random-Access File

- **Data in random access files (binary mode)**
  - **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 (binary data)**
  - **The opening modes with "b" (binary) are used for random-access files**
  - **`fwrite`**
    - **Transfer bytes from a location in memory to a file**
  - **`fread`**
    - **Transfer bytes from a file to a location in memory**
- **Use it only when you are an experienced user !!**

Chapter 14 - File Processing

- To perform file processing in C++
  - Include **`<iostream>`** and **`<fstream>`**
- To open file, create objects
  - Creates "line of communication" from object to file
  - Classes
    - **`ifstream`** (input only)
    - **`ofstream`** (output only)
    - **`fstream`** (I/O)
  - Constructors take *file name* and *file-open mode*

```
ofstream outClientFile( "filename", fileOpenMode );
```

  - To attach a file later

```
Ofstream outClientFile;
outClientFile.open( "filename", fileOpenMode);
```


Source: H. M. Deitel and P. J. Deitel, "C++ How to Program", 4th Ed., Prentice Hall Inc., 2003.

14.4 Creating a Sequential-Access File

- File-open modes

Mode	Description
ios::app	Write all output to the end of the file.
ios::ate	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written anywhere in the file.
ios::in	Open a file for input.
ios::out	Open a file for output.
ios::trunc	Discard the file's contents if it exists (this is also the default action for **ios::out**)
ios::binary	Open a file for binary (i.e., non-text) input or output.

  - **ofstream** opened for output by default
    - `ofstream outClientFile( "clients.dat", ios::out );`
    - `ofstream outClientFile( "clients.dat");`

14.4 Creating a Sequential-Access File for Write

- Overloaded **operator!**
  - **!outClientFile**
  - Returns nonzero (true) if **badbit** or **failbit** set
    - Opened non-existent file for reading, wrong permissions
- Overloaded **operator void***
  - Converts stream object to pointer
  - **0** when when **failbit** or **badbit** set, otherwise nonzero
    - **failbit** set when EOF found
  - **while (cin >> myVariable)**
    - Implicitly converts **cin** to pointer; loops until EOF
- Writing to file (just like **cout**)
  - **outClientFile << myVariable**
- Closing file
  - **outClientFile.close()**
  - Automatically closed when destructor called

Outline

fig14_04.cpp (1 of 2)

```
1  // Fig. 14.4: fig14_04.cpp
2  // Create a sequential file.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::ios;
8  using std::cerr;
9  using std::endl;
10
11 #include <fstream>
12
13 using std::ofstream;
14
15 #include <cstdlib>  // exit prototype
16
17 int main()
18 {
19    // ofstream constructor opens file
20    ofstream outClientFile( "clients.dat", ios::out );
21
22    // exit program if unable to create file
23    if ( !outClientFile ) {  // overloaded ! operator
24       cerr << "File could not be opened" << endl;
25       exit( 1 );
26
27    } // end if
```

Notice the the header files required for file I/O.

`ofstream` object created and used to open file **`"clients.dat"`**. If the file does not exist, it is created.

`!` operator used to test if the file opened properly.

Outline

fig14_04.cpp (2 of 2)

```
28
29    cout << "Enter the account, name, and balance." << endl
30         << "Enter end-of-file t
31
32    int account;
33    char name[ 30 ];
34    double balance;
35
36    // read account, name and balance from cin, then place in file
37    while ( cin >> account >> name >> balance ) {
38       outClientFile << account << ' ' << name << ' ' << balance
39                     << endl;
40       cout << "? ";
41
42    } // end while
43
44    return 0;  // ofstream destructor closes file
45
46 } // end main
```

`cin` is implicitly converted to a pointer. When EOF is encountered, it returns 0 and the loop stops.

Write data to file like a regular stream.

File closed when destructor called for object. Can be explicitly closed with **`close()`**.

```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Outline

fig14_04.cpp output (1 of 1)

14.5 Reading Data from a Sequential-Access File

- Reading files
  - `ifstream inClientFile( "filename", ios::in );`
  - Overloaded `!`
    - `!inClientFile` tests if file was opened properly
  - `operator void*` converts to pointer
    - `while (inClientFile >> myVariable)`
    - Stops when EOF found (gets value `0`)

Outline

fig14_07.cpp
(1 of 3)

```
1  // Fig. 14.7: fig14_07.cpp
2  // Reading and printing a sequential file.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::ios;
8  using std::cerr;
9  using std::endl;
10 using std::left;
11 using std::right;
12 using std::fixed;
13 using std::showpoint;
14
15 #include <fstream>
16
17 using std::ifstream;
18
19 #include <iomanip>
20
21 using std::setw;
22 using std::setprecision;
23
24 #include <cstdlib> // exit prototype
25
26 void outputLine( int, const char * const, double );
27
```


Outline

fig14_07.cpp
(2 of 3)

```
28 int main()
29 {
30    // ifstream constructor opens the file
31    ifstream inClientFile( "clients.dat", ios::in );
32
33    // exit program if ifstream could not open file
34    if ( !inClientFile ) {
35       cerr << "File could not be opened" << endl;
36       exit( 1 );
37
38    } // end if
39
40    int account;
41    char name[ 30 ];
42    double balance;
43
44    cout << left << setw( 10 ) << "Acco
45         << "Name" << "Balance" << endl
46
47    // display each record in file
48    while ( inClientFile >> account >> name >> balance )
49       outputLine( account, name, balance );
50
51    return 0; // ifstream destructor closes the file
52
53 } // end main
```

Open and test file for input.

Read from file until EOF found.

```
54
55  // display single record from file
56  void outputLine( int account, const char * const name,
57     double balance )
58  {
59     cout << left << setw( 10 ) << account << setw( 13 ) << name
60          << setw( 7 ) << setprecision( 2 ) << right << balance
61          << endl;
62
63  } // end function outputLine
```

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

Outline

fig14_07.cpp (3 of 3)

fig14_07.cpp output (1 of 1)

14.5 Reading Data from a Sequential-Access File

- File position pointers
  - Number of next byte to read/write
  - Functions to reposition pointer
    - **`seekg`** (seek get for **`istream`** class)
    - **`seekp`** (seek put for **`ostream`** class)
    - Classes have "get" and "put" pointers
  - **`seekg`** and **`seekp`** take *offset* and *direction*
    - Offset: number of bytes relative to direction
    - Direction (**`ios::beg`** default)
      - **`ios::beg`** - relative to beginning of stream
      - **`ios::cur`** - relative to current position
      - **`ios::end`** - relative to end

14.5 Reading Data from a Sequential-Access File

- Examples
  - **`fileObject.seekg(0)`**
    - Goes to front of file (location **`0`**) because **`ios::beg`** is default
  - **`fileObject.seekg(n)`**
    - Goes to nth byte from beginning
  - **`fileObject.seekg(n, ios::cur)`**
    - Goes n bytes forward
  - **`fileObject.seekg(y, ios::end)`**
    - Goes y bytes back from end
  - **`fileObject.seekg(0, ios::cur)`**
    - Goes to last byte
  - **`seekp`** similar

14.5 Reading Data from a Sequential-Access File

- To find pointer location
  - **`tellg`** and **`tellp`**
  - **`location = fileObject.tellg()`**
- To test if EOF is encountered
  - **`fileObject.eof()`**
  - **`while ( !inClientFile.eof() )`**
    - If the file is not finished, execute the loop again
- To reset EOF flag and restart searching file
  - **`fileObject.clear()`**