C++ Programming
Chapter 8
Sequential-Access Files

Yih-Peng Chiou

Room 617, BL Building

(02) 3366-3603

ypchiou@cc.ee.ntu.edu.tw

Photonic Modeling and Design Lab.
Graduate Institute of Photonics and Optoelectronics &
Department of Electrical Engineering
National Taiwan University

OBJECTIVES

In this chapter you'll learn:

- The data hierarchy from bits, to files to databases.
- To create, read, write and update sequential files.
- Some of the key streams that are associated with file processing.

Contents

8.1 Introduction

- Storage:
  - Volatile: temporary, data is lost as power off
    - RAM (DRAM, SRAM)
  - Nonvolatile: “permanent”, data is still there even power off
    - Hard disks (HD), CDs, DVDS, flash drives and tapes
    - Files are used in most cases
- This chapter: how to build C++ programs that create, update and process sequential files
  - techniques for input of data from, and output of data to, `string` streams rather than files in Chapter 18, Class `string` and String Stream Processing.

8.2 Data Hierarchy

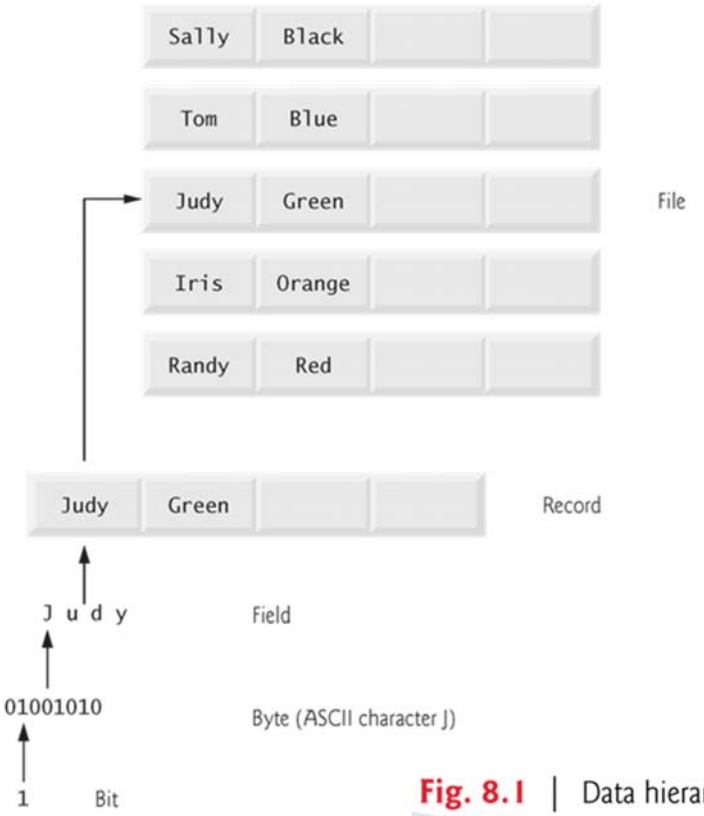

Fig. 8.1 | Data hierarchy.

8.2 Data Hierarchy

- Ultimately, all data items that digital computers process are reduced to combinations of zeros and ones.
  - simple and economical to build electronic devices with two stable states — one state represents 0 and the other represents 1.
- Bit, short for "binary digit"
  - the smallest data item that computers support—
  - Each data item, or bit, can assume either the value 0 or the value 1.
  - Computer circuitry performs various simple bit manipulations
- It is preferable to program with data in forms such as decimal digits (0–9), letters (A–Z and a–z) and special symbols (e.g., $, @, %, &, * and many others) – referred to as characters.
  - The set of all characters used to write programs and represent data items on a particular computer is called that computer's character set.
  - Every character in a computer's character set is represented as a pattern of 1s and 0s.
- Bytes are composed of eight bits.

8.2 Data Hierarchy

- You create programs and data items with characters; computers manipulate and process these characters as patterns of bits.
- Each char typically occupies one byte.
  - C++ also provides data type wchar_t to occupy more than one byte)
  - to support larger character sets, such as the Unicode® character set; for more information on Unicode®, visit www.unicode.org
- A **field** is a group of characters that conveys some meaning.
- Typically, a **record** (which can be represented as a class in C++) is composed of several fields (called data members in C++).
  - Thus, a record is a group of related fields.
- A **file** is a group of related records.
- To facilitate retrieving specific records from a file, at least one field in each record is chosen as a record key.
- A record key identifies a record as belonging to a particular person or entity and distinguishes that record from all others.

8.2 Data Hierarchy

- There are many ways of organizing records in a file.
- A common type of organization is called a **sequential file**, in which records typically are stored in order by a record-key field.
- Most businesses use many different files to store data.
- A group of related files often are stored in a **database**.
- A collection of programs designed to create and manage databases is called a database management system (DBMS).

8.3 Files and Streams

- C++ views each file as a sequence of bytes (Fig. 8.2).
- Each file ends either with an end-of-file marker or at a specific byte number recorded in an operating-system-maintained, administrative data structure.
- When a file is opened, an object is created, and a stream is associated with the object.
  - In Chapter 15, we saw that objects cin, cout, cerr and clog are created when <iostream> is included.
- The streams associated with these objects provide communication channels between a program and a particular file or device.
- To perform file processing in C++, header files <iostream> and <fstream> must be included.

Fig. 8.2 | C++'s view of a file of *n* bytes.

8.4 Creating a Sequential File

- C++ imposes no structure on a file.
- A concept like that of a “record” does not exist in a C++ file.
- You must structure files to meet the application’s requirements.
- Figure 8.3 creates a sequential file that might be used in an accounts-receivable system to help manage the money owed by a company’s credit clients.
- For each client, the program obtains the client’s account number, name and balance (i.e., the amount the client owes the company for goods and services received in the past).
- The data obtained for each client constitutes a record for that client.
- The account number serves as the record key.
- This program assumes the user enters the records in account number order.
- In a comprehensive accounts receivable system, a sorting capability would be provided to eliminate this restriction.


```
1  // Fig. 8.3: Fig08_03.cpp
2  // Create a sequential file.
3  #include <iostream>
4  #include <string>
5  #include <fstream> // file stream
6  #include <cstdlib>
7  using namespace std;
8
9  int main()
10 {
11    // ofstream constructor opens file
12    ofstream outClientFile( "clients.txt", ios::out );
13
14    // exit program if unable to create file
15    if ( !outClientFile ) // overloaded ! operator
16    {
17       cerr << "File could not be opened" << endl;
18       exit( 1 );
19    } // end if
20
21    cout << "Enter the account, name, and balance." << endl
22       << "Enter end-of-file to end input.\n? ";
23
24    int account; // customer's account number
25    string name; //customer's name
26    double balance; // amount of money customer owes company
27
28    // read account, name and balance from cin, then place in file
29    while ( cin >> account >> name >> balance )
30    {
31       outClientFile << account << ' ' << name << ' ' << balance << endl;
32       cout << "? ";
33    } // end while
34 } // end main
```

outClientFile.is_open()

outClientFile.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
```

Fig. 8.3 | Creating a sequential file.

- The file is to be opened for output, so an ofstream object is created.
- Filename and file-open mode are passed to the object's constructor

Mode	Description
ios::app	Append 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 (this also is the default action for ios::out).
ios::binary	Open a file for binary (i.e., nontext) input or output.

8.4 Creating a Sequential File

- Existing files opened with mode ios::out are truncated
  - all data in the file is discarded
- If the specified file does not yet exist, the ofstream object creates the file, using that filename.
- The ofstream constructor opens the file—this establishes a "line of communication" with the file.
- By default, ofstream objects are opened for output, so the open mode is not required in the constructor call.
- An ofstream object can be created without opening a specific file—a file can be attached to the object later. For example, the statements

```
ofstream outClientFile;
outClientFile.open("clients.txt", ios::out);
```

  - The ofstream member function open opens a file and attaches it to an existing ofstream object.

Common Programming Error 8.2

Not opening a file before attempting to reference it in a program will result in an error.

8.4 Creating a Sequential File

Common Programming Error 8.1

Use caution when opening an existing file for output (`ios::out`), especially when you want to preserve the file's contents, which will be discarded without warning.

- Avoid it by changing the properties to **Read Only**

- The condition in the if statement in lines 15–19 returns true if the open operation failed. Some possible errors are
  - attempting to open a nonexistent file for reading,
  - attempting to open a file for reading or writing without permission, and
  - opening a file for writing when no disk space is available.
- Function exit terminates a program.
  - The argument to exit is returned to the environment from which the program was invoked.
  - Argument 0 indicates that the program terminated normally; any other value indicates that the program terminated due to an error.
  - The calling environment (most likely the operating system) uses the value returned by exit to respond appropriately to the error.

8.4 Creating a Sequential File

- The user enters the end-of-file key combination to inform the program to process no additional data—this sets the "end-of-file indicator" in the cin object.

Computer system	Keyboard combination
UNIX/Linux/Mac OS X	<*Ctrl-d*> (on a line by itself)
Microsoft Windows	<*Ctrl-z*> (sometimes followed by pressing *Enter*)
VAX (VMS)	<*Ctrl-z*>

- Later in the chapter, we'll use the eof member function to test for end-of-file in an input file.
- Line 31 writes a set of data to the file clients.txt, using the stream insertion operator << and the outClientFile object associated with the file at the beginning of the program.
- The data may be retrieved by a program designed to read the file
- The file created is simply a text file (can be viewed by any text editor)

8.4 Creating a Sequential File

- Once the user enters the end-of-file indicator, main terminates.
- This implicitly invokes outClientFile's destructor, which closes the clients.txt file.
- You also can close the ofstream object explicitly, using member function close in the statement

Good Programming Practice 8.1

Open a file for input only (using `ios::in`*) if the file's contents should not be modified. This prevents unintentional modification of the file's contents and is an example of the principle of least privilege.*

8.5 Reading Data from a Sequential File

```
1  // Fig. 8.6: Fig08_06.cpp
2  // Reading and printing a sequential file.
3  #include <iostream>
4  #include <fstream> // file stream
5  #include <iomanip>
6  #include <string>
7  #include <cstdlib>
8  using namespace std;
9
10 void outputLine( int, const string, double ); // prototype
11
12 int main()
13 {
14    // ifstream constructor opens the file
15    ifstream inClientFile( "clients.txt", ios::in );
16
17    // exit program if ifstream could not open file
18    if ( !inClientFile )
19    {
20       cerr << "File could not be opened" << endl;
21       exit( 1 );
22    } // end if
23
24    int account; // customer's account number
25    string name; // customer's name
26    double balance; //amount of money customer owes company
27
28    cout << left << setw( 10 ) << "Account" << setw( 13 )
29       << "Name" << "Balance" << endl << fixed << showpoint;
```

```
30
31      // display each record in file
32      while ( inClientFile >> account >> name >> balance )
33         outputLine( account, name, balance );
34   } // end main
35
36   // display single record from file
37   void outputLine( int account, const string name, double balance )
38   {
39      cout << left << setw( 10 ) << account << setw( 13 ) << name
40         << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
41   } // 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
```

Fig. 8.6 | Reading and printing a sequential file.

- Objects of class ifstream are opened for input by default, so to open clients.txt for input we could have used the statement

  ifstream inClientFile("clients.txt");

- An ifstream object can be created without opening a specific file, because a file can be attached to it later.

8.5 Reading Data from a Sequential File

- Each time line 32 executes, it reads another record from the file into the variables account, name and balance.
- When the end of file has been reached, the while condition returns false), terminating the while statement and the program; this causes the ifstream destructor function to run, closing the file.
- To retrieve data sequentially from a file, programs normally start reading from the beginning of the file and read all the data consecutively until the desired data is found.
- It might be necessary to process the file sequentially several times (from the beginning of the file) during program execution.
- Both istream and ostream provide member functions for repositioning the file-position pointer (the byte number of the next byte in the file to be read or written).
  - seekg (" seek get") for istream
  - seekp (" seek put") for ostream

8.5 Reading Data from a Sequential File

- Each istream object has a "get pointer," which indicates the byte number in the file from which the next input is to occur, and each ostream object has a "put pointer," which indicates the byte number in the file at which the next output should be placed.
- The statement

  inClientFile.seekg(0);

  repositions the file-position pointer to the beginning of the file (location 0) attached to inClientFile.
- The argument to seekg normally is a long integer.
- A second argument can be specified to indicate the seek direction
  - ios::beg (the default) for positioning relative to the beginning of a stream,
  - ios::cur for positioning relative to the current position in a stream or
  - ios::end for positioning relative to the end of a stream

8.5 Reading Data from a Sequential File

- The file-position pointer is an integer value that specifies the location in the file as a number of bytes from the file's starting location (also referred to as the offset from the beginning of the file).
- Some examples of positioning the "get" file-position pointer are
  - // position to the nth byte of fileObject (assumes ios::beg)
    fileObject.seekg(n);
  - // position n bytes forward in fileObject
    fileObject.seekg(n, ios::cur);
  - // position n bytes back from end of fileObject
    fileObject.seekg(n, ios::end);
  - // position at end of fileObject
    fileObject.seekg(0, ios::end);
- The same operations can be performed using ostream member function seekp.
- Member functions tellg and tellp are provided to return the current locations of the "get" and "put" pointers, respectively.

```
1  // Fig. 8.7: Fig08_08.cpp
2  // Credit inquiry program.
3  #include <iostream>
4  #include <fstream>
5  #include <iomanip>
6  #include <string>
7  #include <cstdlib>
8  using namespace std;
9
10 enum RequestType { ZERO_BALANCE = 1, CREDIT_BALANCE, DEBIT_BALANCE, END };
11 int getRequest();
12 bool shouldDisplay( int, double );
13 void outputLine( int, const string, double );
14
15 int main()
16 {
17    // ifstream constructor opens the file
18    ifstream inClientFile( "clients.txt", ios::in );
19
20    // exit program if ifstream could not open file
21    if ( !inClientFile )
22    {
23       cerr << "File could not be opened" << endl;
24       exit( 1 );
25    } // end if
26
27    int request; // request type: zero, credit or debit balance
28    int account; // customer's account number
29    string name; // customer's name
30    double balance; // amount of money customer owes company
31
```



```
32    // get user's request (e.g., zero, credit or debit balance)
33    request = getRequest();
34
35    // process user's request
36    while ( request != END )
37    {
38       switch ( request )
39       {
40          case ZERO_BALANCE:
41             cout << "\nAccounts with zero balances:\n";
42             break;
43          case CREDIT_BALANCE:
44             cout << "\nAccounts with credit balances:\n";
45             break;
46          case DEBIT_BALANCE:
47             cout << "\nAccounts with debit balances:\n";
48             break;
49       } // end switch
50
51       // read account, name and balance from file
52       inClientFile >> account >> name >> balance;
53
54       // display file contents (until eof)
55       while ( !inClientFile.eof() )
56       {
57          // display record
58          if ( shouldDisplay( request, balance ) )
59             outputLine( account, name, balance );
60
61          // read account, name and balance from file
62          inClientFile >> account >> name >> balance;
63       } // end inner while
```

```
64
65          inClientFile.clear(); // reset eof for next input
66          inClientFile.seekg( 0 ); // reposition to beginning of file
67          request = getRequest(); // get additional request from user
68       } // end outer while
69
70       cout << "End of run." << endl;
71    } // end main
72
73    // obtain request from user
74    int getRequest()
75    {
76       int request; // request from user
77
78       // display request options
79       cout << "\nEnter request" << endl
80          << " 1 - List accounts with zero balances" << endl
81          << " 2 - List accounts with credit balances" << endl
82          << " 3 - List accounts with debit balances" << endl
83          << " 4 - End of run" << fixed << showpoint;
84
85       do // input user request
86       {
87          cout << "\n? ";
88          cin >> request;
89       } while ( request < ZERO_BALANCE && request > END );
90
91       return request;
92    } // end function getRequest
93
```



```
94    // determine whether to display given record
95    bool shouldDisplay( int type, double balance )
96    {
97       // determine whether to display zero balances
98       if ( type == ZERO_BALANCE && balance == 0 )
99          return true;
100
101      // determine whether to display credit balances
102      if ( type == CREDIT_BALANCE && balance < 0 )
103         return true;
104
105      // determine whether to display debit balances
106      if ( type == DEBIT_BALANCE && balance > 0 )
107         return true;
108
109      return false;
110   } // end function shouldDisplay
111
112   // display single record from file
113   void outputLine( int account, const string name, double balance )
114   {
115      cout << left << setw( 10 ) << account << setw( 13 ) << name
116         << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
117   } // end function outputLine
```

```
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

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 2

Accounts with credit balances:
400       Stone         -42.16

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 3

Accounts with debit balances:
100       Jones          24.98
200       Doe           345.67
500       Rich          224.62

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 4
End of run.
```

Fig. 8.7 | Credit inquiry program.