Lecture 23:
File Processing

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
May 5th, 2010

17.4 Creating a Sequential File (cont.)

- In Fig. 17.4, the file is to be opened for output, so an `ofstream` object is created.
- Two arguments are passed to the object's constructor—the filename and the file-open mode (line 12).
- For an `ofstream` object, the file-open mode can be either `ios::out` to output data to a file or `ios::app` to append data to the end of a file (without modifying any data already in the 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, then the `ofstream` object creates the file, using that filename.
- The `ofstream` constructor opens the file—this estab-lishes 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.
- Figure 17.5 lists the file-open modes.

Common Programming Error 17.1

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.

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.

Fig. 17.5 | File open modes.

17.4 Creating a Sequential File (cont.)

- An `ofstream` object can be created without opening a specific file—a file can be attached to the object later.
- For example, the statement
  - `ofstream outClientFile;`
- creates an `ofstream` object named `outClientFile`.
- The `ofstream` member function open opens a file and attaches it to an existing `ofstream` object as follows:
  - `outClientFile.open("clients.dat", ios::out);`

17.4 Creating a Sequential File (cont.)

- 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.

17.4 Creating a Sequential File (cont.)

- The `operator void *` function can be used to test an input object for end-of-file instead of calling the `eof` member function explicitly on the input object.
- Figure 17.6 lists the keyboard combinations for entering end-of-file for various computer systems.

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*>

Fig. 17.6 | End-of-file key combinations for various popular computer systems.

Performance Tip 17.1

Closing files explicitly when the program no longer needs to reference them can reduce resource usage (especially if the program continues execution after closing the files).

17.5 Reading Data from a Sequential File (cont.)

- Creating an `ifstream` object opens a file for input.
- The `ifstream` constructor can receive the filename and the file open mode as arguments.
- Line 15 creates an `ifstream` object called `inClientFile` and associates it with the `clients.dat` file.
- The arguments in parentheses are passed to the `ifstream` constructor function, which opens the file and establishes a "line of communication" with the file.

Good Programming Practice 17.1

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.*

```
1  // Fig. 17.7: Fig17_07.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.dat", 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
```

Fig. 17.7 | Reading and printing a sequential file. (Part 1 of 3.)

```
24       int account;
25       string name;
26       double balance;
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
```

Fig. 17.7 | Reading and printing a sequential file. (Part 2 of 3.)

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

Fig. 17.7 | Reading and printing a sequential file. (Part 3 of 3.)

17.5 Reading Data from a Sequential File (cont.)

- Objects of class `ifstream` are opened for input by de-fault.
- We could have used the statement
  - `ifstream inClientFile( "clients.dat" );`
- to open `clients.dat` for input.
- Just as with an `ofstream` ob-ject, an `ifstream` object can be created without opening a specific file, because a file can be attached to it later.
- Line 32 reads a set of data (i.e., a record) from the 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 implicit call to `operator void *` in the `while` condition returns the null pointer (which converts to the `bool` value `false`), the `ifstream` destructor function closes the file and the program terminates.

17.5 Reading Data from a Sequential File (cont.)

- 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 the execution of a program.
- Both `istream` and `ostream` provide member functions for repositioning the file-position pointer (the byte num-ber of the next byte in the file to be read or written).
  - seekg (" seek get") for `istream`
  - seekp (" seek put") for `ostream`

17.5 Reading Data from a Sequential File (cont.)

- 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 indi-cates 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.

17.5 Reading Data from a Sequential File (cont.)

- A second argu-ment can be specified to indicate the seek direction, which can be
  - `ios::beg` (the de-fault) 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
- 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 (this is also re-ferred to as the offset from the beginning of the file).

17.5 Reading Data from a Sequential File (cont.)

- Some examples of po-sitioning 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`.

17.5 Reading Data from a Sequential File (cont.)

- Member functions `tellg` and `tellp` are provided to return the current locations of the "get" and "put" pointers, respectively.
- Figure 17.8 enables a credit manager to display the account informa-tion for those customers with
  - zero balances (i.e., customers who do not owe the company any money),
  - credit (negative) balances (i.e., customers to whom the company owes money), and
  - debit (positive) balances (i.e., customers who owe the company money for goods and services re-ceived in the past)

```
1  // Fig. 17.8: Fig17_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.dat", ios::in );
19
20    // exit program if ifstream could not open file
21    if ( !inClientFile )
22    {
23       cerr << "File could not be opened" << endl;
```

Fig. 17.8 | Credit inquiry program. (Part 1 of 7.)

```
24          exit( 1 );
25       } // end if
26
27       int request;
28       int account;
29       string name;
30       double balance;
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;
```

Fig. 17.8 | Credit inquiry program. (Part 2 of 7.)

```
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
```

Fig. 17.8 | Credit inquiry program. (Part 3 of 7.)

```
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
```

Fig. 17.8 | Credit inquiry program. (Part 4 of 7.)

```
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
```

Fig. 17.8 | Credit inquiry program. (Part 5 of 7.)

```
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
```

Fig. 17.8 | Credit inquiry program. (Part 6 of 7.)

```
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. 17.8 | Credit inquiry program. (Part 7 of 7.)

Questions