



ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

Ενότητα 14: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ



Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



Ενότητα 14

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

Περιεχόμενα ενότητας

- 14.1 Introduction
- 14.2 The Data Hierarchy
- 14.3 Files and Streams
- 14.4 Creating a Sequential-Access File
- 14.5 Reading Data from a Sequential-Access File
- 14.6 Updating Sequential-Access Files
- 14.7 Random-Access Files
- 14.8 Creating a Random-Access File
- 14.9 Writing Data Randomly to a Random-Access File
- 14.10 Reading Data Sequentially from a Random-Access File
- 14.11 Example: A Transaction-Processing Program
- 14.12 Input/Output of Objects

Σκοποί ενότητας

14.1 Introduction

- Storage of data
 - Arrays, variables are temporary
 - Files are permanent
 - Magnetic disk, optical disk, tapes
- In this chapter
 - Create, update, process files
 - Sequential and random access
 - Formatted and raw processing

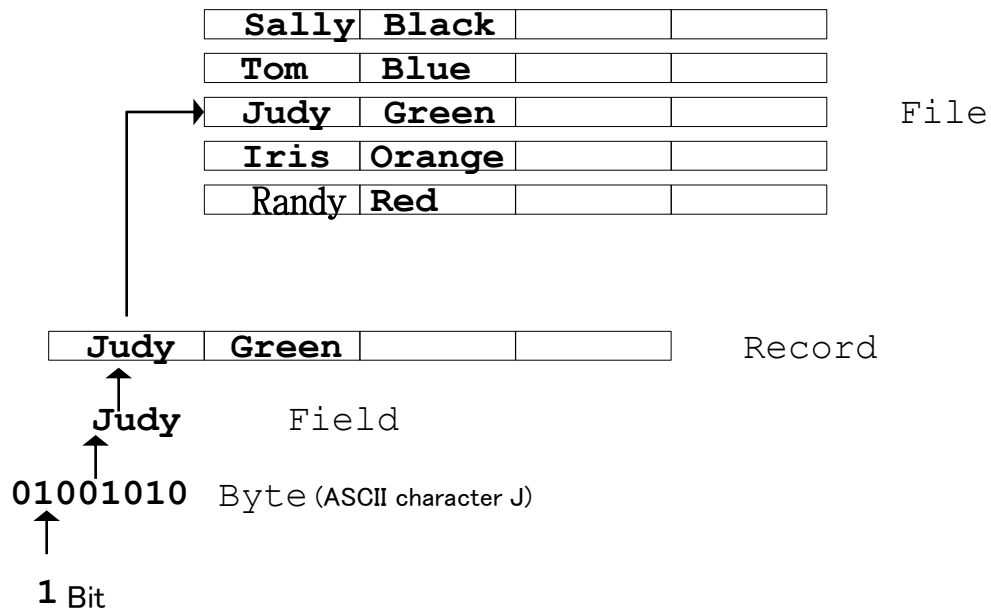
14.2 The Data Hierarchy

- From smallest to largest
 - Bit (binary digit)
 - 1 or 0
 - Everything in computer ultimately represented as bits
 - Cumbersome for humans to use
 - Character set
 - Digits, letters, symbols used to represent data
 - Every character represented by 1's and 0's
 - Byte: 8 bits
 - Can store a character (**char**)
 - Also Unicode for large character sets (**wchar_t**)

14.2 The Data Hierarchy

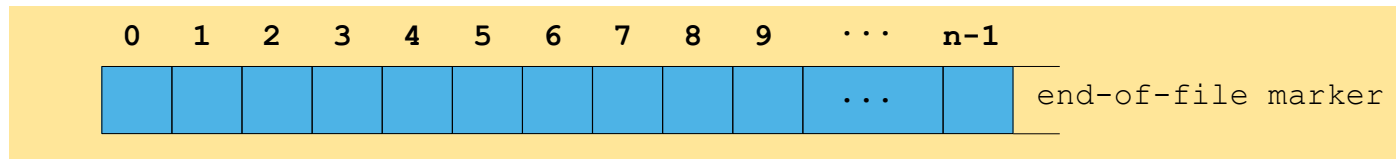
- From smallest to largest (continued)
 - Field: group of characters with some meaning
 - Your name
 - Record: group of related fields
 - **struct** or **class** in C++
 - In payroll system, could be name, SS#, address, wage
 - Each field associated with same employee
 - Record key: field used to uniquely identify record
 - File: group of related records
 - Payroll for entire company
 - Sequential file: records stored by key
 - Database: group of related files
 - Payroll, accounts-receivable, inventory...

14.2 The Data Hierarchy



14.3 Files and Streams

- C++ views file as sequence of bytes
 - Ends with *end-of-file* marker



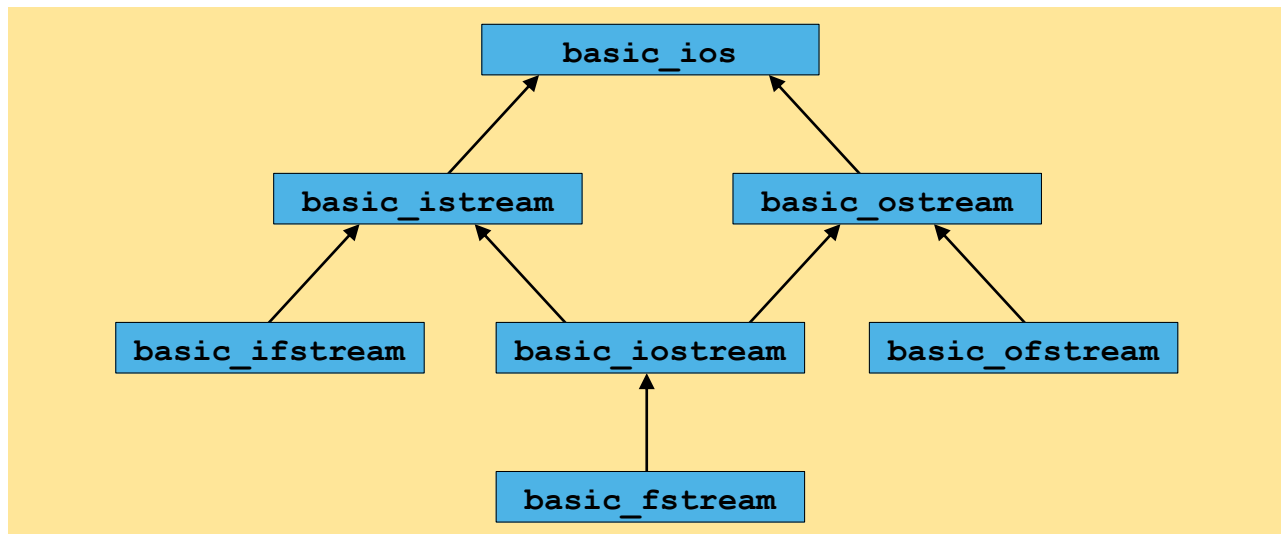
- When file opened
 - Object created, stream associated with it
 - **cin**, **cout**, etc. created when **<iostream>** included
 - Communication between program and file/device

14.3 Files and Streams

- To perform file processing
 - Include `<iostream>` and `<fstream>`
 - Class templates
 - `basic_ifstream` (input)
 - `basic_ofstream` (output)
 - `basic_fstream` (I/O)
 - **typedefs** for specializations that allow **char** I/O
 - `ifstream` (**char** input)
 - `ofstream` (**char** output)
 - `fstream` (**char** I/O)

14.3 Files and Streams

- Opening files
 - Create objects from template
 - Derive from stream classes
 - Can use stream methods from Ch. 12
 - **put**, **get**, **peek**, etc.



14.4 Creating a Sequential-Access File

- C++ imposes no structure on file
 - Concept of "record" must be implemented by programmer
- 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);
```

14.4 Creating a Sequential-Access File

- File-open modes

Mode	Description
<code>ios::app</code>	Write all output to the end of the file.
<code>ios::ate</code>	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.
<code>ios::in</code>	Open a file for input.
<code>ios::out</code>	Open a file for output.
<code>ios::trunc</code>	Discard the file's contents if it exists (this is also the default action for <code>ios::out</code>)
<code>ios::binary</code>	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

- Operations
 - 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

14.4 Creating a Sequential-Access File

- Operations
 - Writing to file (just like **cout**)
 - **outClientFile << myVariable**
 - Closing file
 - **outClientFile.close()**
 - Automatically closed when destructor called

```

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     } // end if
27 }

```

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.

fig14_04.cpp
(1 of 2)

```

28
29 cout << "Enter the account, name, and balance." << endl
30     << "Enter end-of-file to stop." << endl;
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     outFile << 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 ()**.

fig14_04.cpp
(2 of 2)

```
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
```

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)

```

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

```

fig14_07.cpp
(1 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 ) << "Account" <<
45     << "Name" << "Balance" << endl << fix
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.

fig14_07.cpp
(2 of 3)

```

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

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()`
- Upcoming example
 - Credit manager program
 - List accounts with zero balance, credit, and debit

```
1 // Fig. 14.8: fig14_08.cpp
2 // Credit-inquiry program.
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::fixed;
11 using std::showpoint;
12 using std::left;
13 using std::right;
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>
25
```

fig14_08.cpp
(1 of 6)

```

26 enum RequestType { ZERO_BALANCE = 1, CREDIT_BALANCE,
27     DEBIT_BALANCE, END };
28 int getRequest();
29 bool shouldDisplay( int, double );
30 void outputLine( int, const char * const, double );
31
32 int main()
33 {
34     // ifstream constructor opens the file
35     ifstream inClientFile( "clients.dat", ios::in );
36
37     // exit program if ifstream could not open file
38     if ( !inClientFile ) {
39         cerr << "File could not be opened" << endl;
40         exit( 1 );
41
42     } // end if
43
44     int request;
45     int account;
46     char name[ 30 ];
47     double balance;
48
49     // get user's request (e.g., zero, credit or debit balance)
50     request = getRequest();
51

```

fig14_08.cpp
(2 of 6)

```
52 // process user's request
53 while ( request != END ) {
54
55     switch ( request ) {
56
57         case ZERO_BALANCE:
58             cout << "\nAccounts with zero balances:\n";
59             break;
60
61         case CREDIT_BALANCE:
62             cout << "\nAccounts with credit balances:\n";
63             break;
64
65         case DEBIT_BALANCE:
66             cout << "\nAccounts with debit balances:\n";
67             break;
68
69     } // end switch
70
```

fig14_08.cpp
(3 of 6)

```

71 // read account, name and balance from file
72 inClientFile >> account >> name >> balance;
73
74 // display file contents (until eof)
75 while ( !inClientFile.eof() ) {
76
77 // display record
78 if ( shouldDisplay( request, balance ) )
79     outputLine( account, name, balance );
80
81 // read account, name and balance from file
82 inClientFile >> account >> name >> balance;
83
84 } // end inner while
85
86 inClientFile.clear(); // reset eof for next input
87 inClientFile.seekg( 0 ); // move to beginning of file
88 request = getRequest(); // get additional request from user
89
90 } // end outer while
91
92 cout << "End of run." << endl;
93
94 return 0; // ifstream destructor closes the file
95
96 } // end main

```

Use **clear** to reset eof. Use **seekg** to set file position pointer to beginning of file.

fig14_08.cpp
(4 of 6)

```

97
98 // obtain request from user
99 int getRequest()
100 {
101     int request;
102
103     // display request options
104     cout << "\nEnter request" << endl
105         << " 1 - List accounts with zero balances" << endl
106         << " 2 - List accounts with credit balances" << endl
107         << " 3 - List accounts with debit balances" << endl
108         << " 4 - End of run" << fixed << showpoint;
109
110     // input user request
111     do {
112         cout << "\n? ";
113         cin >> request;
114
115     } while ( request < ZERO_BALANCE && request > END );
116
117     return request;
118
119 } // end function getRequest
120

```

fig14_08.cpp
(5 of 6)


```

121 // determine whether to display given record
122 bool shouldDisplay( int type, double balance )
123 {
124     // determine whether to display credit balances
125     if ( type == CREDIT_BALANCE && balance < 0 )
126         return true;
127
128     // determine whether to display debit balances
129     if ( type == DEBIT_BALANCE && balance > 0 )
130         return true;
131
132     // determine whether to display zero balances
133     if ( type == ZERO_BALANCE && balance == 0 )
134         return true;
135
136     return false;
137
138 } // end function shouldDisplay
139
140 // display single record from file
141 void outputLine( int account, const char * const name,
142     double balance )
143 {
144     cout << left << setw( 10 ) << account << setw( 13 ) << name
145         << setw( 7 ) << setprecision( 2 ) << right << balance
146         << endl;
147
148 } // end function outputLine

```

fig14_08.cpp
(6 of 6)

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
-----	-------	--------

fig14_08.cpp
output (1 of 2)

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.

fig14_08.cpp
output (2 of 2)

14.6 Updating Sequential-Access Files

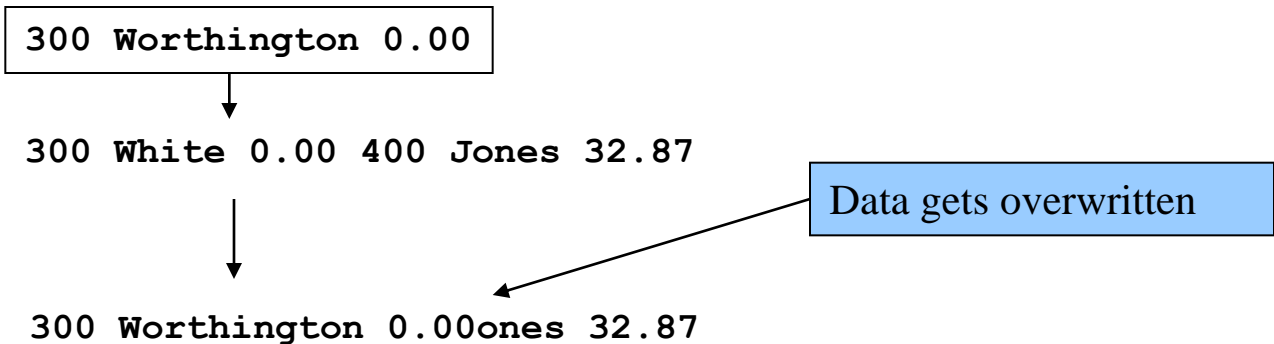
- Updating sequential files

- Risk overwriting other data
- Example: change name "White" to "Worthington"

- Old data

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

- Insert new data



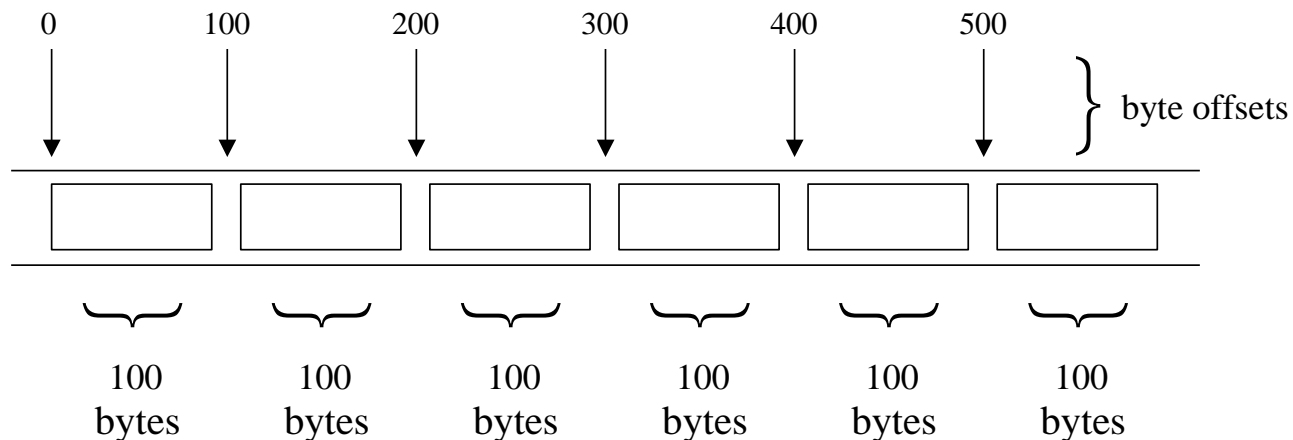
- Formatted text different from internal representation
- Problem can be avoided, but awkward

14.7 Random-Access Files

- Instant access
 - Want to locate record quickly
 - Airline reservations, ATMs
 - Sequential files must search through each one
- Random-access files are solution
 - Instant access
 - Insert record without destroying other data
 - Update/delete items without changing other data

14.7 Random-Access Files

- C++ imposes no structure on files
 - Programmer must create random-access files
 - Simplest way: fixed-length records
 - Calculate position in file from record size and key



14.8 Creating a Random-Access File

- "1234567" (**char ***) vs 1234567 (**int**)
 - **char *** takes 8 bytes (1 for each character + null)
 - **int** takes fixed number of bytes (perhaps 4)
 - 123 same size in bytes as 1234567
- << operator and **write()**
 - **outFile << number**
 - Outputs **number (int)** as a **char ***
 - Variable number of bytes
 - **outFile.write(const char *, size);**
 - Outputs raw bytes
 - Takes pointer to memory location, number of bytes to write
 - Copies data directly from memory into file
 - Does not convert to **char ***

14.8 Creating a Random-Access File

- Example

```
outFile.write( reinterpret_cast<const char *>(&number),  
              sizeof( number ) );
```

- **&number** is an **int ***
 - Convert to **const char *** with **reinterpret_cast**
- **sizeof(number)**
 - Size of **number** (an **int**) in bytes
- **read** function similar (more later)
- Must use **write/read** between compatible machines
 - Only when using raw, unformatted data
- Use **ios::binary** for raw writes/reads

14.8 Creating a Random-Access File

- Usually write entire **struct** or object to file
- Problem statement
 - Credit processing program
 - Store at most 100 fixed-length records
 - Record
 - Account number (key)
 - First and last name
 - Balance
 - Account operations
 - Update, create new, delete, list all accounts in a file
- Next: program to create blank 100-record file

```

1 // Fig. 14.10: clientData.h
2 // Class ClientData definition used in Fig. 14.12–Fig. 14.15.
3 #ifndef CLIENTDATA_H
4 #define CLIENTDATA_H
5
6 #include <iostream>
7
8 using std::string;
9
10 class ClientData {
11
12 public:
13
14     // default ClientData constructor
15     ClientData( int = 0, string = "", string = "", double = 0.0 );
16
17     // accessor functions for accountNumber
18     void setAccountNumber( int );
19     int getAccountNumber() const;
20
21     // accessor functions for lastName
22     void setLastName( string );
23     string getLastName() const;
24

```

Class **ClientData** stores the information for each person. 100 blank **ClientData** objects will be written to a file.

clientData.h
(1 of 2)

```

25 // accessor functions for firstName
26 void setFirstName( string );
27 string getFirstName() const;
28
29 // accessor functions for balance
30 void setBalance( double );
31 double getBalance() const;
32
33 private:
34     int accountNumber;
35     char lastName[ 15 ];
36     char firstName[ 10 ];
37     double balance;
38
39 }; // end class ClientData
40
41 #endif

```

Put limits on the size of the first and last name. **accountNumber** (an **int**) and **balance** (**double**) are already of a fixed size.

clientData.h
(2 of 2)

```

1 // Fig. 14.11: ClientData.cpp
2 // Class ClientData stores customer's credit information.
3 #include <iostream>
4
5 using std::string;
6
7 #include <cstring>
8 #include "clientData.h"
9
10 // default ClientData constructor
11 ClientData::ClientData( int accountNumberValue,
12     string lastNameValue, string firstNameValue,
13     double balanceValue )
14 {
15     setAccountNumber( accountNumberValue );
16     setLastName( lastNameValue );
17     setFirstName( firstNameValue );
18     setBalance( balanceValue );
19
20 } // end ClientData constructor
21
22 // get account-number value
23 int ClientData::getAccountNumber() const
24 {
25     return accountNumber;
26
27 } // end function getAccountNumber

```

ClientData.cpp
(1 of 4)

```

28
29 // set account-number value
30 void ClientData::setAccountNumber( int accountNumberValue )
31 {
32     accountNumber = accountNumberValue;
33
34 } // end function setAccountNumber
35
36 // get last-name value
37 string ClientData::getLastName() const
38 {
39     return lastName;
40
41 } // end function getLastName
42
43 // set last-name value
44 void ClientData::setLastName( string lastNameString )
45 {
46     // copy at most 15 characters from string to lastName
47     const char *lastNameValue = lastNameString.data();
48     int length = strlen( lastNameValue );
49     length = ( length < 15 ? length : 14 );
50     strncpy( lastName, lastNameValue, length );
51
52     // append null character to lastName
53     lastName[ length ] = '\0';

```

ClientData.cpp
(2 of 4)

```

54
55 } // end function setLastName
56
57 // get first-name value
58 string ClientData::getFirstName() const
59 {
60     return firstName;
61 }
62 // end function getFirstName
63
64 // set first-name value
65 void ClientData::setFirstName( string firstNameString )
66 {
67     // copy at most 10 characters from string to firstName
68     const char *firstNameValue = firstNameString.data();
69     int length = strlen( firstNameValue );
70     length = ( length < 10 ? length : 9 );
71     strncpy( firstName, firstNameValue, length );
72
73     // append new-line character to firstName
74     firstName[ length ] = '\\0';
75
76 } // end function setFirstName
77

```

ClientData.cpp
(3 of 4)

```
78 // get balance value
79 double ClientData::getBalance() const
80 {
81     return balance;
82 }
83 // end function getBalance
84
85 // set balance value
86 void ClientData::setBalance( double balanceValue )
87 {
88     balance = balanceValue;
89 }
90 // end function setBalance
```

ClientData.cpp
(4 of 4)

```

1 // Fig. 14.12: fig14_12.cpp
2 // Creating a randomly accessed file.
3 #include <iostream>
4
5 using std::cerr;
6 using std::endl;
7 using std::ios;
8
9 #include <fstream>
10
11 using std::ofstream;
12
13 #include <cstdlib>
14 #include "clientData.h" // ClientData class definition
15
16 int main()
17 {
18     ofstream outCredit( "credit.dat", ios::binary );
19
20     // exit program if ofstream could not open file
21     if ( !outCredit ) {
22         cerr << "File could not be opened." << endl;
23         exit( 1 );
24     } // end if
25 }

```

Open a file for raw writing using an **ofstream** object and **ios::binary**.

fig14_12.cpp
(1 of 2)


```

26
27 // create ClientData with no information
28 ClientData blankClient;
29
30 // output 100 blank records to file
31 for ( int i = 0; i < 100; i++ )
32     outCredit.write(
33         reinterpret_cast< const char * >( &blankClient ),
34         sizeof( ClientData ) );
35
36 return 0;
37
38 } // end main

```

Create a blank object. Use **write** to output the raw data to a file (passing a pointer to the object and its size).

fig14_12.cpp
(2 of 2)

14.9 Writing Data Randomly to a Random-Access File

- Use **seekp** to write to exact location in file
 - Where does the first record begin?
 - Byte 0
 - The second record?
 - Byte 0 + sizeof(object)
 - Any record?
 - (Recordnum - 1) * sizeof(object)

```
1 // Fig. 14.13: fig14_13.cpp
2 // Writing to a random access file.
3 #include <iostream>
4
5 using std::cerr;
6 using std::endl;
7 using std::cout;
8 using std::cin;
9 using std::ios;
10
11 #include <iomanip>
12
13 using std::setw;
14
15 #include <fstream>
16
17 using std::ofstream;
18
19 #include <cstdlib>
20 #include "clientData.h" // ClientData class definition
21
```

fig14_13.cpp
(1 of 4)

```

22 int main()
23 {
24     int accountNumber;
25     char lastName[ 15 ];
26     char firstName[ 10 ];
27     double balance;
28
29     ofstream outCredit( "credit.dat", ios::binary );
30
31     // exit program if ofstream cannot open file
32     if ( !outCredit ) {
33         cerr << "File could not be opened." << endl;
34         exit( 1 );
35
36     } // end if
37
38     cout << "Enter account number "
39           << "(1 to 100, 0 to end input)\n? ";
40
41     // require user to specify account number
42     ClientData client;
43     cin >> accountNumber;
44     client.setAccountNumber( accountNumber );
45

```

Open file for raw (binary) writing.

Get account number, put into object. It has not yet been written to file.

fig14_13.cpp
(2 of 4)

```

46 // user enters information, which is copied into file
47 while ( client.getAccountNumber() > 0 &&
48         client.getAccountNumber() <= 100 ) {
49
50 // user enters last name, first name and balance
51 cout << "Enter lastname, firstname, balance\n? ";
52 cin >> setw( 15 ) >> lastName;
53 cin >> setw( 10 ) >> firstName;
54 cin >> balance;
55
56 // set record lastName, firstName and
57 client.setLastName( lastName );
58 client.setFirstName( firstName );
59 client.setBalance( balance );
60
61 // seek position in file of user-specified record
62 outCredit.seekp( ( client.getAccountNumber() - 1 ) *
63                 sizeof( ClientData ) );
64
65 // write user-specified information in file
66 outCredit.write(
67                 reinterpret_cast< const char * >( &client ),
68                 sizeof( ClientData ) );
69

```

Position **outCredit** to the proper location in the file (based on the account number).

Write **ClientData** object to file at specified position.

fig14_13.cpp
(3 of 4)

```
70     // enable user to specify another account number
71     cout << "Enter account number\n? ";
72     cin >> accountNumber;
73     client.setAccountNumber( accountNumber );
74
75 } // end while
76
77 return 0;
78
79 } // end main
```

fig14_13.cpp
(4 of 4)

```
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
```

Notice that accounts can be created in any order.

**fig14_13.cpp
output (1 of 1)**

14.10 Reading Data Sequentially from a Random-Access File

- **read** - similar to **write**
 - Reads raw bytes from file into memory
 - `inFile.read(reinterpret_cast<char *>(&number), sizeof(int));`
 - **&number**: location to store data
 - **sizeof(int)**: how many bytes to read
 - Do not use `inFile >> number` with raw bytes
 - `>>` expects `char *`
- Upcoming program
 - Output data from a random-access file
 - Go through each record sequentially
 - If no data (`accountNumber == 0`) then skip


```

1 // Fig. 14.14: fig14_14.cpp
2 // Reading a random access file.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::ios;
8 using std::cerr;
9 using std::left;
10 using std::right;
11 using std::fixed;
12 using std::showpoint;
13
14 #include <iomanip>
15
16 using std::setprecision;
17 using std::setw;
18
19 #include <fstream>
20
21 using std::ifstream;
22 using std::ofstream;
23
24 #include <cstdlib> // exit protooype
25 #include "clientData.h" // ClientData class definition
26

```

fig14_14.cpp
(1 of 3)

```

27 void outputLine( ostream&, const ClientData & );
28
29 int main()
30 {
31     ifstream inCredit( "credit.dat", ios::in );
32
33     // exit program if ifstream cannot open file
34     if ( !inCredit ) {
35         cerr << "File could not be opened." << endl;
36         exit( 1 );
37
38     } // end if
39
40     cout << left << setw( 10 ) << "Last Name" <<
41         << setw( 10 ) << right << "Balance" << endl;
42
43     ClientData client; // create record
44
45     // read first record from file
46     inCredit.read( reinterpret_cast< char * >( &client ),
47         sizeof( ClientData ) );
48
49

```

Read **sizeof(ClientData)** bytes and put into object **client**. This may be an empty record.

fig14_14.cpp
(2 of 3)

```

50 // read all records from file
51 while ( inCredit && !inCredit.eof() ) {
52
53     // display record
54     if ( client.getAccountNumber() != 0 )
55         outputLine( cout, client );
56
57     // read next from file
58     inCredit.read( reinterpret_cast<char * >( &client ),
59                 sizeof( ClientData ) );
60
61 } // end while
62
63 return 0;
64
65 } // end main
66
67 // display single record
68 void outputLine( ostream &output, const ClientData &record )
69 {
70     output << left << setw( 10 ) << record.getAccountNumber()
71         << setw( 16 ) << record.getLastName().data()
72         << setw( 11 ) << record.getFirstName().data()
73         << setw( 10 ) << setprecision( 2 ) << right << fixed
74         << showpoint << record.getBalance() << endl;
75
76 } // end outputLine

```

Loop exits if there is an error reading (`inCredit == 0`) or EOF is found (`inCredit.eof() == 1`)

Output non-empty accounts. Note that `outputLine` takes an `ostream` argument. We could easily output to another file (opened with an `ofstream` object, which derives from `ostream`).

fig14_14.cpp
(3 of 3)

Account	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

fig14_14.cpp
output (1 of 1)

14.11 Example: A Transaction-Processing Program

- Instant access for bank accounts
 - Use random access file (data in **client.dat**)
- Give user menu
 - Option 1: store accounts to **print.txt**

Account	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

- Option 2: update record

Enter account to update (1 - 100): 37			
37	Barker	Doug	0.00
Enter charge (+) or payment (-): +87.99			
37	Barker	Doug	87.99

14.11 Example: A Transaction-Processing Program

- Menu options (continued)

- Option 3: add new record

```
Enter new account number (1 - 100): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```

- Option 4: delete record

```
Enter account to delete (1 - 100): 29
Account #29 deleted.
```

- To open file for reading and writing

- Use **fstream** object
- "Or" file-open modes together

```
fstream inOutCredit( "credit.dat", ios::in | ios::out );
```

```

1 // Fig. 14.15: fig14_15.cpp
2 // This program reads a random access file sequentially, updates
3 // data previously written to the file, creates data to be placed
4 // in the file, and deletes data previously in the file.
5 #include <iostream>
6
7 using std::cout;
8 using std::cerr;
9 using std::cin;
10 using std::endl;
11 using std::ios;
12 using std::left;
13 using std::right;
14 using std::fixed;
15 using std::showpoint;
16
17 #include <fstream>
18
19 using std::ofstream;
20 using std::ostream;
21 using std::fstream;
22
23 #include <iomanip>
24
25 using std::setw;
26 using std::setprecision;
27
28 #include <cstdlib> // exit prototype
29 #include "clientData.h" // ClientData class definition

```

fig14_15.cpp
(1 of 14)

```

30
31 int enterChoice();
32 void printRecord( fstream& );
33 void updateRecord( fstream& );
34 void newRecord( fstream& );
35 void deleteRecord( fstream& );
36 void outputLine( ostream&, const ClientData & );
37 int getAccount( const char * const );
38
39 enum Choices { PRINT = 1, UPDATE, NEW, DELETE };
40
41 int main()
42 {
43     // open file for reading and writing
44     fstream inOutCredit( "credit.dat", ios::in | ios::out );
45
46     // exit program if fstream cannot open file
47     if ( !inOutCredit ) {
48         cerr << "File could not be opened." << endl;
49         exit ( 1 );
50
51     } // end if
52

```

Open file for reading and writing
(**fstream** object needed).

fig14_15.cpp
(2 of 14)


```

53     int choice;
54
55     // enable user to specify action
56     while ( ( choice = enterChoice() ) != END ) {
57
58         switch ( choice ) {
59
60             // create text file from record file
61             case PRINT:
62                 printRecord( inOutCredit );
63                 break;
64
65             // update record
66             case UPDATE:
67                 updateRecord( inOutCredit );
68                 break;
69
70             // create record
71             case NEW:
72                 newRecord( inOutCredit );
73                 break;
74
75             // delete existing record
76             case DELETE:
77                 deleteRecord( inOutCredit );
78                 break;
79

```

fig14_15.cpp
(3 of 14)

```

53  int choice;
54
55  // enable user to specify action
56  while ( ( choice = enterChoice() ) != END ) {
57
58      switch ( choice ) {
59
60          // create text file from record file
61          case PRINT:
62              printRecord( inOutCredit );
63              break;
64
65          // update record
66          case UPDATE:
67              updateRecord( inOutCredit );
68              break;
69
70          // create record
71          case NEW:
72              newRecord( inOutCredit );
73              break;
74
75          // delete existing record
76          case DELETE:
77              deleteRecord( inOutCredit );
78              break;
79

```

Displays menu and returns user's choice.

fig14_15.cpp
(4 of 14)

```

80         // display error if user does not select valid choice
81         default:
82             cerr << "Incorrect choice" << endl;
83             break;
84
85     } // end switch
86
87     inOutCredit.clear(); // reset end-of-file indicator
88
89 } // end while
90
91 return 0;
92
93 } // end main
94
95 // enable user to input menu choice
96 int enterChoice()
97 {
98     // display available options
99     cout << "\nEnter your choice" << endl
100         << "1 - store a formatted text file of accounts" << endl
101         << "    called \"print.txt\" for printing" << endl
102         << "2 - update an account" << endl
103         << "3 - add a new account" << endl
104         << "4 - delete an account" << endl
105         << "5 - end program\n? ";

```

fig14_15.cpp
(5 of 14)

```

106
107     int menuChoice;
108     cin >> menuChoice; // receive choice from user
109
110     return menuChoice;
111
112 } // end function enterChoice
113
114 // create formatted text file for printing
115 void printRecord( fstream &readFromFile )
116 {
117     // create text file
118     ofstream outPrintFile( "print.txt", ios::out );
119
120     // exit program if ofstream cannot create file
121     if ( !outPrintFile ) {
122         cerr << "File could not be created." << endl;
123         exit( 1 );
124     } // end if
125
126
127     outPrintFile << left << setw( 10 ) << "Account" << setw( 16 )
128         << "Last Name" << setw( 11 ) << "First Name" << right
129         << setw( 10 ) << "Balance" << endl;
130

```

Output to `print.txt`. First, print the header for the table.

fig14_15.cpp
(6 of 14)

```

131 // set file-position pointer to beginning of record file
132 readFromFile.seekg( 0 );
133
134 // read first record from record file
135 ClientData client;
136 readFromFile.read( reinterpret_cast< char * >( &client ),
137     sizeof( ClientData ) );
138
139 // copy all records from record file into text file
140 while ( !readFromFile.eof() ) {
141
142     // write single record to text file
143     if ( client.getAccountNumber() != 0 )
144         outputLine( outPrintFile, client );
145
146     // read next record from record file
147     readFromFile.read( reinterpret_cast< char * >( &client ),
148         sizeof( ClientData ) );
149
150 } // end while
151
152 } // end function printRecord
153

```

Go to front of file, read account data, and print record if not empty.

Note that **outputLine** takes an **ostream** object (base of **ofstream**). It can easily print to a file (as in this case) or **cout**.

fig14_15.cpp
(7 of 14)

```

154 // update balance in record
155 void updateRecord( fstream &updateFile )
156 {
157     // obtain number of account to update
158     int accountNumber = getAccount( "Enter account to update" );
159
160     // move file-position pointer to correct record in file
161     updateFile.seekg(
162         ( accountNumber - 1 ) * sizeof( ClientData ) );
163
164     // read first record from file
165     ClientData client;
166     updateFile.read( reinterpret_cast< char * >( &client ),
167         sizeof( ClientData ) );
168
169     // update record
170     if ( client.getAccountNumber() != 0 ) {
171         outputLine( cout, client );
172
173         // request user to specify transaction
174         cout << "\nEnter charge (+) or payment (-): ";
175         double transaction; // charge or payment
176         cin >> transaction;
177
178         // update record balance
179         double oldBalance = client.getBalance();
180         client.setBalance( oldBalance + transaction );
181         outputLine( cout, client );
182

```

This is **fstream** (I/O) because we must read the old balance, update it, and write the new balance.

fig14_15.cpp
(8 of 14)

```

183     // move file-position pointer to correct record in file
184     updateFile.seekp(
185         ( accountNumber - 1 ) * sizeof( ClientData ) );
186
187     // write updated record over old record in file
188     updateFile.write(
189         reinterpret_cast< const char * >( &client ),
190         sizeof( ClientData ) );
191
192 } // end if
193
194 // display error if account does not exist
195 else
196     cerr << "Account #" << accountNumber
197         << " has no information." << endl;
198
199 } // end function updateRecord
200
201 // create and insert record
202 void newRecord( fstream &insertInFile )
203 {
204     // obtain number of account to create
205     int accountNumber = getAccount( "Enter new account number" );
206
207     // move file-position pointer to correct record in file
208     insertInFile.seekg(
209         ( accountNumber - 1 ) * sizeof( ClientData ) );

```

This is **fstream** because we read to see if a non-empty record already exists. If not, we write a new record.

fig14_15.cpp
(9 of 14)

```

210
211 // read record from file
212 ClientData client;
213 insertInFile.read( reinterpret_cast< char * >( &client ),
214     sizeof( ClientData ) );
215
216 // create record, if record does not previously exist
217 if ( client.getAccountNumber() == 0 ) {
218
219     char lastName[ 15 ];
220     char firstName[ 10 ];
221     double balance;
222
223     // user enters last name, first name and balance
224     cout << "Enter lastname, firstname, balance\n? ";
225     cin >> setw( 15 ) >> lastName;
226     cin >> setw( 10 ) >> firstName;
227     cin >> balance;
228
229     // use values to populate account values
230     client.setLastName( lastName );
231     client.setFirstName( firstName );
232     client.setBalance( balance );
233     client.setAccountNumber( accountNumber );
234

```

fig14_15.cpp
(10 of 14)


```

235     // move file-position pointer to correct record in file
236     insertInFile.seekp( ( accountNumber - 1 ) *
237         sizeof( ClientData ) );
238
239     // insert record in file
240     insertInFile.write(
241         reinterpret_cast< const char * >( &client ),
242         sizeof( ClientData ) );
243
244 } // end if
245
246 // display error if account previously exists
247 else
248     cerr << "Account #" << accountNumber
249         << " already contains information." << endl;
250
251 } // end function newRecord
252

```

fig14_15.cpp
(11 of 14)

```

253 // delete an existing record
254 void deleteRecord( fstream &deleteFromFile )
255 {
256     // obtain number of account to delete
257     int accountNumber = getAccount( "Enter account to delete" );
258
259     // move file-position pointer to correct record in file
260     deleteFromFile.seekg(
261         ( accountNumber - 1 ) * sizeof( ClientData ) );
262
263     // read record from file
264     ClientData client;
265     deleteFromFile.read( reinterpret_cast< char * >( &client ),
266         sizeof( ClientData ) );
267
268     // delete record, if record exists in file
269     if ( client.getAccountNumber() != 0 ) {
270         ClientData blankClient;
271
272         // move file-position pointer to correct record in file
273         deleteFromFile.seekp( ( accountNumber - 1 ) *
274             sizeof( ClientData ) );
275

```

fstream because we read to check if the account exists. If it does, we write blank data (erase it). If it does not exist, there is no need to delete it.

fig14_15.cpp
(12 of 14)

```

276     // replace existing record with blank record
277     deleteFromFile.write(
278         reinterpret_cast< const char * >( &blankClient ),
279         sizeof( ClientData ) );
280
281     cout << "Account #" << accountNumber << " deleted.\n";
282
283 } // end if
284
285 // display error if record does not exist
286 else
287     cerr << "Account #" << accountNumber << " is empty.\n";
288
289 } // end deleteRecord
290
291 // display single record
292 void outputLine( ostream &output, const ClientData &record )
293 {
294     output << left << setw( 10 ) << record.getAccountNumber()
295         << setw( 16 ) << record.getLastName().data()
296         << setw( 11 ) << record.getFirstName().data()
297         << setw( 10 ) << setprecision( 2 ) << right << fixed
298         << showpoint << record.getBalance() << endl;
299
300 } // end function outputLine
301

```

outputLine is very flexible, and can output to any **ostream** object (such as a file or **cout**).

fig14_15.cpp
(13 of 14)

```
302 // obtain account-number value from user
303 int getAccount( const char * const prompt )
304 {
305     int accountNumber;
306
307     // obtain account-number value
308     do {
309         cout << prompt << " (1 - 100): ";
310         cin >> accountNumber;
311
312     } while ( accountNumber < 1 || accountNumber > 100 );
313
314     return accountNumber;
315
316 } // end function getAccount
```

fig14_15.cpp
(14 of 14)

14.12 Input/Output of Objects

- I/O of objects
 - Chapter 8 (overloaded >>)
 - Only object's data transmitted
 - Member functions available internally
 - When objects stored in file, lose type info (class, etc.)
 - Program must know type of object when reading
 - One solution
 - When writing, output object type code before real object
 - When reading, read type code
 - Call proper overloaded function (**switch**)

Τέλος Ενότητας

