

Sequential-Access Files

Lab 12: Processing Data in Files

Fall 2021

Rung-Bin Lin

International Bachelor Program in Informatics

Yuan Ze University

Dec. 28, 2021

Outline

- ▶ Introduction
- ▶ Data Hierarchy
- ▶ Files and streams
- ▶ Creating a Sequential File
- ▶ Reading Data from a Sequential File
- ▶ Updating Sequential Files
- ▶ See Chapter 8 for Details

8.1 Introduction

- ▶ Storage of data in memory is temporary.
- ▶ **Files** are used for **data persistence**—permanent retention of data.
- ▶ Computers store files on **secondary storage devices**, such as hard disks, CDs, DVDs, flash drives and tapes.
- ▶ Here, we explain how to build C++ programs that create, update and process sequential files.

8.2 Data Hierarchy

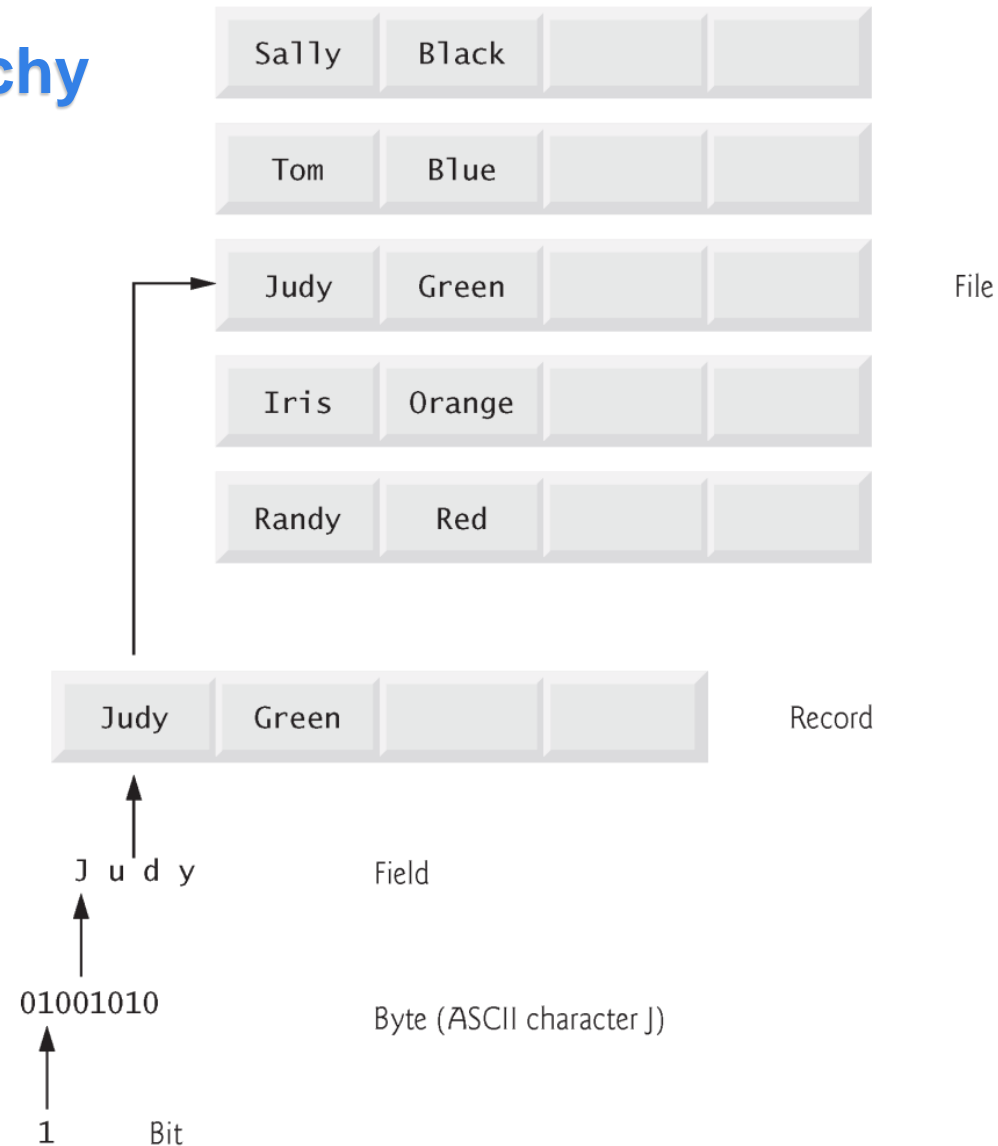


Fig. 8.1 | Data hierarchy.

8.3 Files and Streams

- ▶ C++ views each file as a sequence of bytes (Fig. 8.2). It imposes no structure on a file.



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

- ▶ Each file ends generally with an **end-of-file marker**.
- ▶ 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.

8.3 Files and Streams (cont.)

- ▶ To perform file processing in C++, header files `<iostream>` and `<fstream>` must be included.
- ▶ Three kinds of file streams
 - **ifstream**: Input file stream for reading (input)
 - **ofstream**: Output file stream for writing (output)
 - **fstream**: File stream for reading and writing (input and output)

Example of Creating a Sequential File

```
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
```

In CodeBlock, here this file is created and located at the same directory of the program that creates this file.

```

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

```

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. (Part 2 of 2.)

Create and Open an ofstream File at the Same Time

- ▶ In Fig. 8.3, `ofstream outClientFile("clients.txt", ios::out);` creates an ofstream object `outClientFile` and open it at the same time for output by associating it with the file named `clients.txt`.
 - 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).
 - **By default**, ofstream objects are opened for **output**, so the open mode is not required in the **constructor** call.
- ▶ **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.

File Open Modes

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

Fig. 8.4 | File open modes.

◆ Default open modes:

- `ifstream` is `ios::in`
- `ofstream` is `ios::out`
- `fstream` is `ios::in | ios::out`

◆ **`ios::trunc`** is used when we would like to overwrite (replacing old data with new data)

Create and Open an ofstream File at Different Times

- ▶ An **ofstream** object can be created without opening a specific file—a file can be attached to the object later.
 - For example, the statement below creates an **ofstream** object named **outClientFile**.

ofstream outClientFile;

- ▶ The **ofstream** member function **open()** opens a file and attaches it to an **existing** **ofstream** object as follows:

```
const string fileName = "clients.txt";  
outClientFile.open(filename, ios::out);
```

Terminating Input from Keyboard and Closing a File

- ▶ Once the user enters the **end-of-file indicator (^Z)**, Input from the **keyboard** are terminated.
- ▶ You can close the **ofstream** object explicitly, using member function **close ()** in the statement, for example **outClientFile.close()**.

8.5 Reading Data from a Sequential File

- ▶ Creating an `ifstream` object opens a file for input.
- ▶ The `ifstream` constructor can receive the filename and the file open mode as arguments.

Note that if a file is opened for input and it does not exist, this will cause an error. Especially, this will occur when you want to create an `fstream` file for input&output, but the file does not exist.

To solve this problem, you can first create the file with output mode, then close it, and then re-opened it with input&output mode.

```
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
```

Fig. 8.6 | Reading and printing a sequential file. (Part I of 3.)

```
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
```

Fig. 8.6 | 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. 8.6 | Reading and printing a sequential file. (Part 3 of 3.)

Open a File for Reading

- ▶ Line 15, `ifstream inClientFile("clients.txt", ios::in);` creates an `ifstream` object and open the object by associating it with a file named `clients.txt`.
 - An `ifstream` object is opened for input by **default**, so to open `clients.txt` for input we could have used the statement
 - `ifstream inClientFile("clients.txt");`
- ▶ Just as with an `ofstream` object, an `ifstream` object can be created **first** and opened **later** by associating it with an existing 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 close the file.

Where to read or write next?

- ▶ By a **file-position pointer** which is associated with reading (or writing).
- ▶ It is an integer value that specifies the location in the file as **a number of bytes from the file's starting location**.
- ▶ If a file is opened for input and output, it has two file-position pointers, one for input and the other for output.
- ▶ When a record is read or written, its file-position pointer is advanced to the location where the next record starts.
- ▶ A file-position pointer can be set to a certain location using **seekg** member function for an input stream and **seekp** member function for an output stream.

Reading All the File Data Several Times

- ▶ 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.
- ▶ To process the file sequentially several times, each time we must reposition a file-position pointer to the beginning of the file. Both `ifstream` and `ofstream` provide member functions for repositioning a file-position pointer.
 - `Seekg(n, dir)` for `ifstream` to set the “**get**” file-position pointer for “**read**”
 - `Seekp(n, dir)` for `ofstream` to set the “**put**” file-position pointer for “**write**”
 - `seekg(n, dir)` and `seekp(n, dir)` for `fstream`
- ▶ **If a file is open for both read and write, the “get” and “put” file position pointers are in fact the same one. In other words, there is only one file position pointer for any kind of files.**
- ▶ A file position pointer is modified based on the two arguments **n** and **dir**. The argument **n** normally is a **long integer**.

Modifying a File-Position Pointer (cont. from previous page)

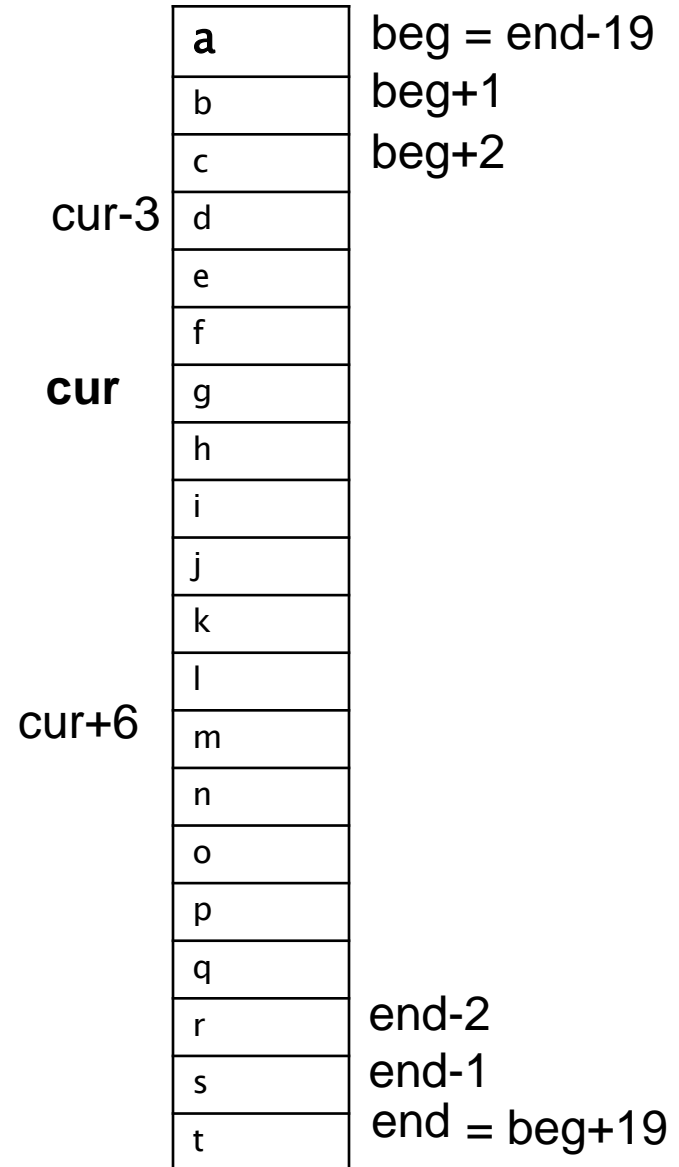
- ▶ The first argument **n** specified the number of bytes used to calculate the value of the file-position pointer.
- ▶ The second argument **dir** specifies the **seek direction**, which can be
 - `ios::beg` (the **default**) for positioning relative to the **beginning** of a stream,
 - `ios::cur` for positioning relative to the **current** position in a stream or
 - `ios::end` for positioning relative to the **end** of a stream
- ▶ The value of the file-position pointer is calculated as follows:
 - If **dir** is `ios::beg`, the value is **beg+n**.
 - If **dir** is `ios::cur`, the value is **cur+n**.
 - If **dir** is `ios::end`, the value is **end-n**.
- ▶ The statement `inClientFile.seekg(0);` repositions the file-position pointer to the beginning of the file (location **beg**, i.e., 0) attached to `inClientFile`.

Some examples of Setting File-Position Pointer's Value

- `// 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 `ofstream` member function `seekp`.
- ▶ Member functions `tellg` and `tellp` are provided to return the current locations of the “get” and “put” pointers, respectively.

Example of File-Position Pointers

- File-position pointers are maintained in the file object. You typically need not explicitly specify the pointer when you read/ write a file.
- If you would like to read a file again from the beginning of the file, you have two ways:
 - Close the file and then re-opened it. Not suggested.
 - Reposition the file-position pointer to the beginning of the file using `fileObject.Seekg(0,beg)` for input.
- If you would like to overwrite the file from the beginning of the file, you should use `fileObject.Seekp(0,beg)`.
- To update a file, you should be more cautious. You normally can not just write the data at the location (pointed by `cur`) where the original data is stored.



Example of Processing a File

- ▶ Figure 8.7 enables a credit manager to **display** the account information 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 received in the past)

```

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

```

Fig. 8.7 | Credit inquiry program. (Part 1 of 7.)


```

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

```

Fig. 8.7 | Credit inquiry program. (Part 2 of 7.)

The statement in line 52 should read a record first and then check **eof** (end of file) in the while loop below. If this statement is omitted, the program will not work correctly.

```
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
```

Fig. 8.7 | Credit inquiry program. (Part 3 of 7.)

```
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. 8.7 | 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
```

Fig. 8.7 | Credit inquiry program. (Part 5 of 7.)

```

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

Fig. 8.7 | Credit inquiry program. (Part 6 of 7.)

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. (Part 7 of 7.)

8.6 Updating Sequential Files

- ▶ Data that is formatted and written to a sequential file as shown in Section 8.4 cannot be modified without the risk of destroying other data in the file.
- ▶ For example, if the name “white” needs to be changed to “worthington,” the old name cannot be overwritten without corrupting the file.
 - The record for white was written to the file as
 - 300 white 0.00
 - If this record were rewritten beginning at the same location in the file using the longer name, the record would be
 - 300 worthington 0.00
 - The new record contains six more characters than the original record. Therefore, the characters beyond the second “o” in “worthington” would overwrite the beginning of the next sequential record in the file.

Updating Sequential Files (cont.)

- ▶ The problem is that, in the **formatted input/output model** using the stream insertion operator << and the stream extraction operator >>, fields—and **hence records—can vary in size.**
 - For example, values 7, 14, -117, 2074, and 27383 are all **ints**, which store the same number of “raw data” bytes internally (**typically four bytes** on today’s popular 32-bit machines). Hence, if they are stored in file with a **binary open mode**, all integers use four bytes.
 - However, these integers become different-sized fields when output as **formatted text** (character sequences, ASCII code for example). It uses one byte to store 7, two bytes to store 14, four bytes for -117, four bytes for 2047 and five bytes for 27383.
 - **Therefore, the formatted input/output model usually is not used to update records in place.**

Suggestion for Updating File

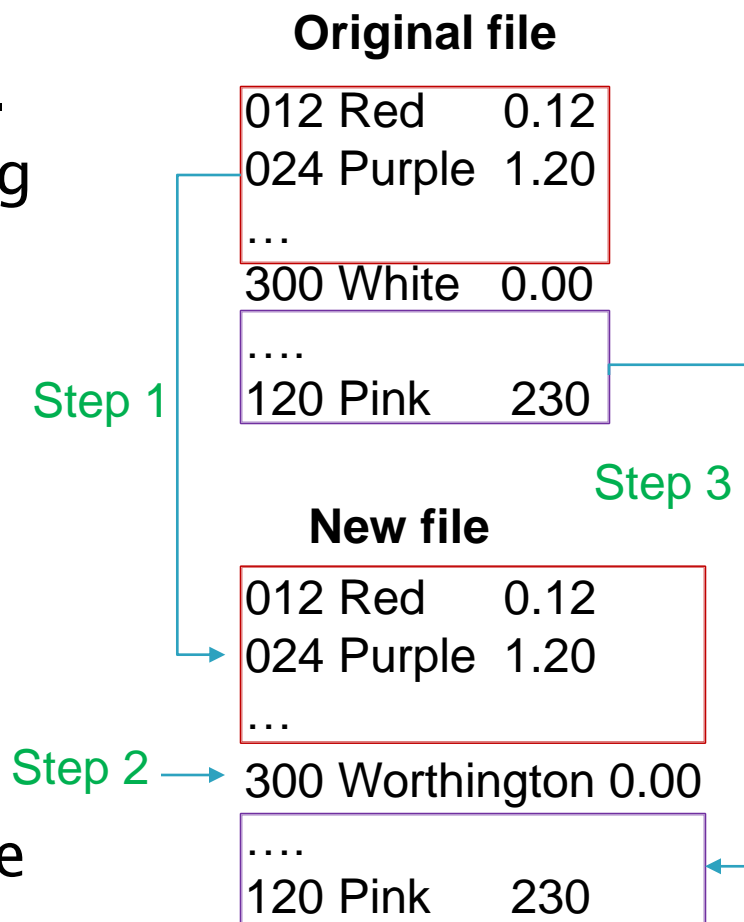
- ▶ Such updating can be done with formatted I/O, but a bit awkwardly.
- ▶ For example, to make the preceding name change

Step1: the records before 300 White 0.00 in a sequential file could be copied to a new file

Step2: the updated record then written to the new file

Step 3: the records after 300 White 0.00 copied to the new file.

- ▶ This requires processing *every* record in the file to update *only* one record.
- ▶ If many records are being updated in one pass of the file, though, this technique can be acceptable.



Lab 12: File Processing

- ▶ You are given a file name `clientFile.txt` that consists of some bank account records. Each record has four fields. The first field is `firstName`, the second field is `lastName`, the third field is `actBalance` which is a real number. The fourth field is `the date and time` when the record is created. You are asked to do the following tasks:

Step 1: Check whether `clientFile.txt` exists or not. If it does exist, print the prompt message “Existing file content: “ and then display the file content. Otherwise, print the prompt message “Creating a new file: “.

Step 2: Print the prompt message “Enter first name, last name, and balance: “.

Step 3: Read `firstName`, `lastName`, `actBalance` from keyboard and form a record. Add this record to the end of this file if the last name and first name are not the same as any existing one in the file. Repeat this step until a `ctrl ^z` is entered from keyboard to stop reading data.

Step 4: Display the file content.

Requirements

- ▶ The number of accounts is at most 1000.
- ▶ You must create the following function for Step 3:
`void createRecord(fstream &fptr, string firstN, string lastN, double actBal);`
 - ✓ This function stores a record into the file. `fptr` is a file object, `firstN` is first name, `lastN` is last name, and `actBal` is account balance. You should store date and time in the number of seconds passed since 00:00:00 GMT, Jan 1, 1970, to the end of each record.
 - ✓ You should not add a record whose last name and first name are the same as any existing one in the file.
- ▶ You should not change `int main()` and `void displayFile(ifstream &)`.
- ▶ Check TA grading notes to understand how your program will be tested.

Time and Date in C++

```
#include <iostream>
#include <ctime>      /* time_t, time, ctime */
using namespace std;
int main ()
{
    time_t currentTime;
    time (&currentTime);
    // get the time and date which is the total number of seconds passed since
    00:00:00 GMT, Jan 1, 1970
    Cout << "current time is: " << currentTime << endl;
    cout << "Current time is: " << ctime (&currentTime) << endl;
    // ctime converts the number of seconds into ASCII representation of date and
    time.
    return 0;
}
```

Below is the output of running the program.

```
Current time is: 1543654269
Current time is: Sat Dec 01 16:51:09 2018
```

int main()

```
int main()
{
    fstream clientFile;
    fstream tempFile;
    string firstName;
    string lastName;
    double actBalance;
    // const char *fileName = "clientFile.txt";
    const string fileN = "clientFile.txt";
    //exit program if unable to create file
    clientFile.open(fileN, ios::in);
    if ( clientFile ){ // overloaded ! operator
        cout << "Existing file content: " << endl;
        displayFile(clientFile);
    }
    else {
        cout << "Creating a new file: " << endl;
        clientFile.open(fileN, ios::out);
    }
    clientFile.close();
    clientFile.open(fileN, ios::in | ios::out);
    cout << "Enter first name, last name, and balance:" << endl;
    while(cin >> firstName >> lastName >> actBalance)
        createRecord(clientFile, firstName, lastName, actBalance);
    displayFile(clientFile);
    clientFile.close();
}
```

void displayFile(fstream&)

//This function prints all the records stored in a file.

```
void displayFile(fstream &fptr)
{
    string str1;
    string str2;
    double balance;
    int numAct = 0;
    time_t transTime;
    fptr.clear();
    fptr.seekg(0);
    while(fptr >> str1 >> str2 >> balance >> transTime) {
        cout << setw(15) << str1 << ' ' << setw(15) << str2 << ' '
        << setw(7) << balance << "    " << ctime(&transTime);
        numAct++;
    }
    cout << "Number of records in the file: " << numAct << endl;
}
```

Hints

- ▶ If a file is open for both read and write, the “get” and “put” file position pointers are in fact the same one. In other words, there is only one file position pointer for any kind of files.
- ▶ You may have to close a file object and then re-open the file object with different file-open mode when removing a record in the file.
- ▶ Before checking whether an account for the given first name and last name already exists in the file, you may have to reposition the file-position pointer to the beginning of the file before you do search again. **Before doing so, you may have to first use `clear()` member function to reset eof flag.**
- ▶ Note that a **file object of `fstream`** has **two** file-position pointers, one for reading and the other for writing.
- ▶ A file is opened once. It should have a corresponding close. That is, if `fileObject.open(...)` is called, there should be a `fileObject.close(...)`.

Input & Output (First Run)

- Each line consists of a record as shown below.

Sample input	Sample output
Lisa Lin -23	Lisa Lin -23 Tue Dec 28 13:51:45 2021
James Liu -3	James Liu -3 Tue Dec 28 13:51:45 2021
Shih-Yi Chung 19.8	Shih-Yi Chung 19.8 Tue Dec 28 13:51:45 2021
Peter Lin 120	Peter Lin 120 Tue Dec 28 13:51:45 2021
Tomas Lee 5	Tomas Lee 5 Tue Dec 28 13:51:45 2021
James Liu -3	Tom Huang 5 Tue Dec 28 13:51:45 2021
Tom Huang 5	Rung-Bin Lin -34 Tue Dec 28 13:51:46 2021
Peter Lin 123	Number of records in the file: 7
Rung-Bin Lin -34	

```

Creating a new file:
Enter first name, last name, and balance:
Lisa Lin -23
James Liu -3
Shih-Yi Chung 19.8
Peter Lin 120
Tomas Lee 5
James Liu -3
Tom Huang 5
Peter Lin 123
Rung-Bin Lin -34
^Z
          Lisa      Lin      -23      Tue Dec 28 13:51:45 2021
          James     Liu       -3      Tue Dec 28 13:51:45 2021
          Shih-Yi   Chung    19.8    Tue Dec 28 13:51:45 2021
          Peter     Lin      120     Tue Dec 28 13:51:45 2021
          Tomas     Lee       5      Tue Dec 28 13:51:45 2021
          Tom       Huang     5      Tue Dec 28 13:51:45 2021
          Rung-Bin  Lin      -34     Tue Dec 28 13:51:46 2021
Number of records in the file: 7
    
```

Sample
output

Input & Output (Second Run)

This run is conducted with an existing file clientFile.txt.

```
Existing file content:
  Lisa      Lin      -23    Tue Dec 28 16:14:05 2021
  James     Liu       -3     Tue Dec 28 16:14:05 2021
  Shih-Yi   Chung    19.8   Tue Dec 28 16:14:05 2021
  Peter     Lin       120    Tue Dec 28 16:14:05 2021
  Tomas     Lee        5     Tue Dec 28 16:14:05 2021
  Tom       Huang     5     Tue Dec 28 16:14:05 2021
  Rung-Bin  Lin      -34    Tue Dec 28 16:14:06 2021
Number of records in the file: 7
Enter first name, last name, and balance:
Rung-Bin Lin -34
Mary Chuang -2
Gerge Huang -5
James Lin -3
Cheng-Yi Lim -12
Tom Huang 5
Tomson Lee 57
Gane Eye -20
Mary Lin -20
Min-Han Liu 0
Henry Lin 0
Henry Wang 20
Jane Wilson 0
Donal Trump 546
^Z
  Lisa      Lin      -23    Tue Dec 28 16:14:05 2021
  James     Liu       -3     Tue Dec 28 16:14:05 2021
  Shih-Yi   Chung    19.8   Tue Dec 28 16:14:05 2021
  Peter     Lin       120    Tue Dec 28 16:14:05 2021
  Tomas     Lee        5     Tue Dec 28 16:14:05 2021
  Tom       Huang     5     Tue Dec 28 16:14:05 2021
  Rung-Bin  Lin      -34    Tue Dec 28 16:14:06 2021
  Mary     Chuang     -2     Tue Dec 28 16:14:35 2021
  Gerge    Huang     -5     Tue Dec 28 16:14:35 2021
  James    Lin       -3     Tue Dec 28 16:14:35 2021
  Cheng-Yi Lim     -12     Tue Dec 28 16:14:35 2021
  Tomson   Lee      57     Tue Dec 28 16:14:35 2021
  Gane     Eye     -20     Tue Dec 28 16:14:35 2021
  Mary     Lin     -20     Tue Dec 28 16:14:35 2021
  Min-Han  Liu      0      Tue Dec 28 16:14:35 2021
  Henry    Lin      0      Tue Dec 28 16:14:35 2021
  Henry    Wang     20     Tue Dec 28 16:14:35 2021
  Jane     Wilson   0      Tue Dec 28 16:14:35 2021
  Donal    Trump    546    Tue Dec 28 16:14:36 2021
Number of records in the file: 19
```

Original
accounts

Added
accounts

Sample
output

TA Grading Notes (1)

- ▶ To test a program, ask students to run the program twice. **Before running the program, remove clientFile.txt from work library.**
 - The first time runs the program with the first part of public test cases. This should create clientFile.txt.
 - The second time runs the program with the file created in the first time and the second part of public test cases.
- ▶ Should have the following statement in the main() function:
void createRecord(fstream &fptr, string firstN, string lastN, double actBal)
This function should be contained in a loop.

TA Grading Notes (2)

- ▶ Ask students to explain how to remove duplicate records.
- ▶ **main() and displayFile(fstream &) functions should not be modified.**
- ▶ Should make sure that the duplicate records are correctly removed.