Lecture 17: Sequential File Access

Class page: https://github.com/tsung-wei-huang/cs1410-40

Dr. Tsung-Wei Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Learning Objective

- 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.

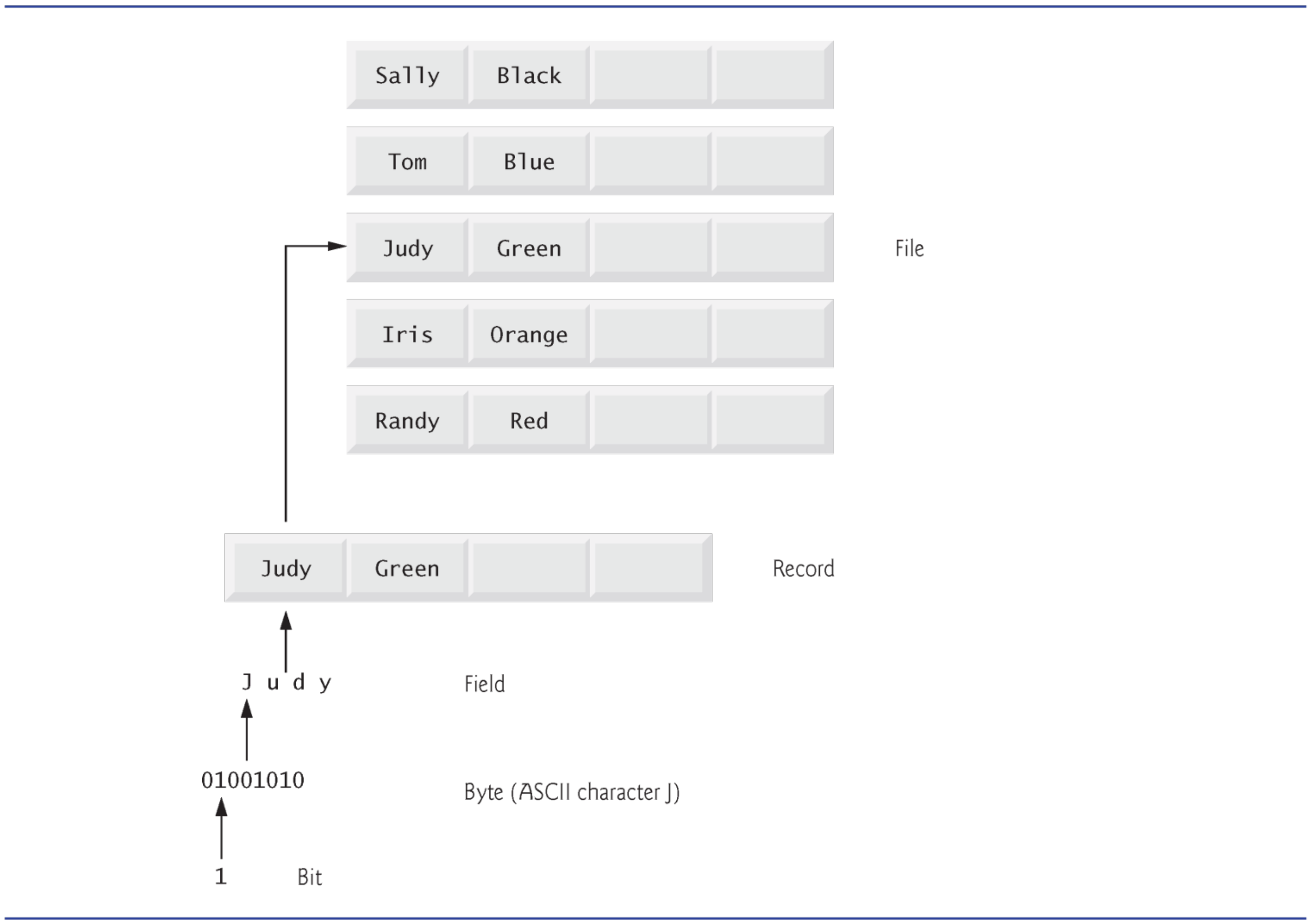
Introduction

- 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.
- In this chapter, we explain how to build C++ programs that create, update and process sequential files.

Data Hierarchy

- **Ultimately, all data items that digital computers process are reduced to combinations of zeros and ones.**
- **The smallest data item that computers support is called a bit**
  - Each data item, or bit, can assume either the value 0 or the value 1.
- **The decimal digits (0–9), letters (A–Z and a–z) and special symbols (e.g., $, @, %, &, *, …) are referred to as characters.**
  - People create programs and data items with characters.
- **Every character in a computer's character set is represented as a pattern of 1s and 0s.**
  - Computers manipulate and process these characters as patterns of bits.
- **Bytes are composed of eight bits.**

Data Hierarchy



File and Stream

- ❑ C++ views each file as a sequence of bytes.
- ❑ Each file ends either with an end-of-file marker or at a specific byte number recorded in operating.
- ❑ When a file is opened, an object is created, and a stream is associated with the object.
- ❑ The streams associated with these objects provide communication channels between a program and a particular file or device.

Creating a Sequential File

```
// Create a sequential file.
#include <iostream>
#include <string>
#include <fstream> // file stream
#include <cstdlib>
using namespace std;

int main()
{
   // ofstream constructor opens file
   ofstream outClientFile( "clients.txt", ios::out );

   // exit program if unable to create file
   if ( !outClientFile ) // overloaded ! operator
   {
      cerr << "File could not be opened" << endl;
      exit( 1 );
   } // end if

   cout << "Enter the account, name, and balance." << endl
      << "Enter end-of-file to end input.\n? ";

```

Creating a Sequential File

```cpp
    int account; // customer's account number
    string name; //customer's name
    double balance; // amount of money customer owes company

    // read account, name and balance from cin, then place in file
    while ( cin >> account >> name >> balance )
    {
       outClientFile << account << ' ' << name << ' ' << balance << endl;
       cout << "? ";
    } // end while
 } // end main
```

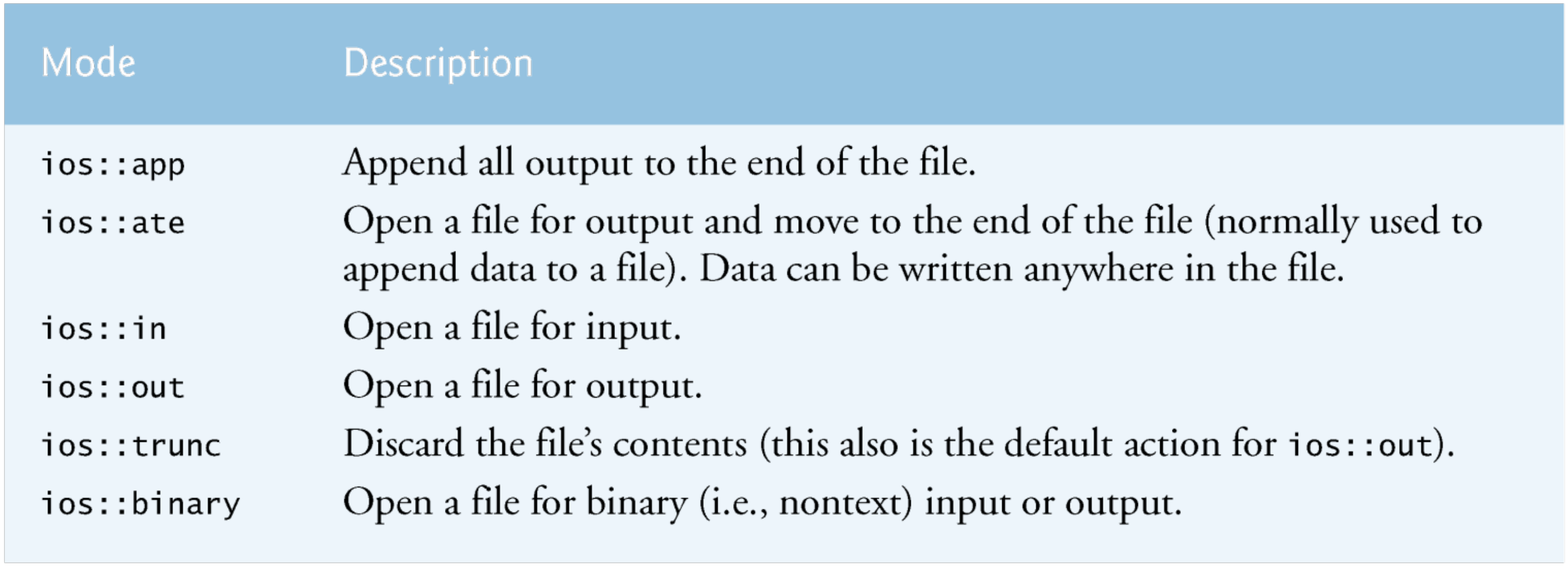
```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Fig. 8.3 | Creating a sequential file. (Part 2 of 2.)

Creating a Sequential File

- **In the previous example, the file is to be opened for output, so an `ofstream` object is created.**
  - This establishes a "line of communication" with the file.
  - By default, `ofstream` objects are opened for output.
- **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
- **If the specified file does not yet exist, then the `ofstream` object creates the file, using that filename.**
- **Existing files opened with mode `ios::out` are truncated**
  - All data in the file is discarded without warning.

Modes



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.

Creating a Sequential File

- 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.txt", ios::out);`

Creating a Sequential File

- **After creating an `ofstream` object, the program tests whether the open operation was successful before using it.**
- **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,
  - 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. (0: normally, others: error)
  - The calling environment (most likely the operating system) uses the value returned by `exit` to respond appropriately to the error.

Reading a Sequential File

- **Files store data so it may be retrieved for processing when needed.**
- **The next example reads records from the `clients.txt` file and displays the contents of these records.**
- **Creating an `ifstream` object opens a file for input.**
  - It 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.txt` 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.**

Reading a Sequential File

```
// Reading and printing a sequential file.
#include <iostream>
#include <fstream> // file stream
#include <iomanip>
#include <string>
#include <cstdlib>
using namespace std;

void outputLine( int, const string, double ); // prototype

int main()
{
   // ifstream constructor opens the file
   ifstream inClientFile( "clients.txt", ios::in );

   // exit program if ifstream could not open file
   if ( !inClientFile )
   {
      cerr << "File could not be opened" << endl;
      exit( 1 );
   } // end if

```

Reading a Sequential File

```cpp
    int account; // customer’s account number
    string name; // customer’s name
    double balance; //amount of money customer owes company

    cout << left << setw( 10 ) << "Account" << setw( 13 )
       << "Name" << "Balance" << endl << fixed << showpoint;

    // display each record in file
    while ( inClientFile >> account >> name >> balance )
       outputLine( account, name, balance );
} // end main

// display single record from file
void outputLine( int account, const string name, double balance )
{
    cout << left << setw( 10 ) << account << setw( 13 ) << name
       << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
} // 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
```

Reading a Sequential File

- ❑ **Objects of class `ifstream` are opened for input by default, so we could use the statement**
  - `ifstream inClientFile( "clients.txt" );`
- ❑ **An `ifstream` object can also be created without opening a specific file**
  - ❑ A file can be attached to it later.
- ❑ **Each time line 32 executes, it reads another record from the file into the variables `account`, `name` and `balance`.**
  - ❑ Similar to using `cin`, but replace `cin` by `inClientFile`.
- ❑ **When the end of file has been reached, the `while` condition returns `false` and terminates the `while` statement and the program.**

Reading a Sequential 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`

Example

- 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`.

Summary

- ❑ Creating a file using ofstream
- ❑ Reading a file using ifstream