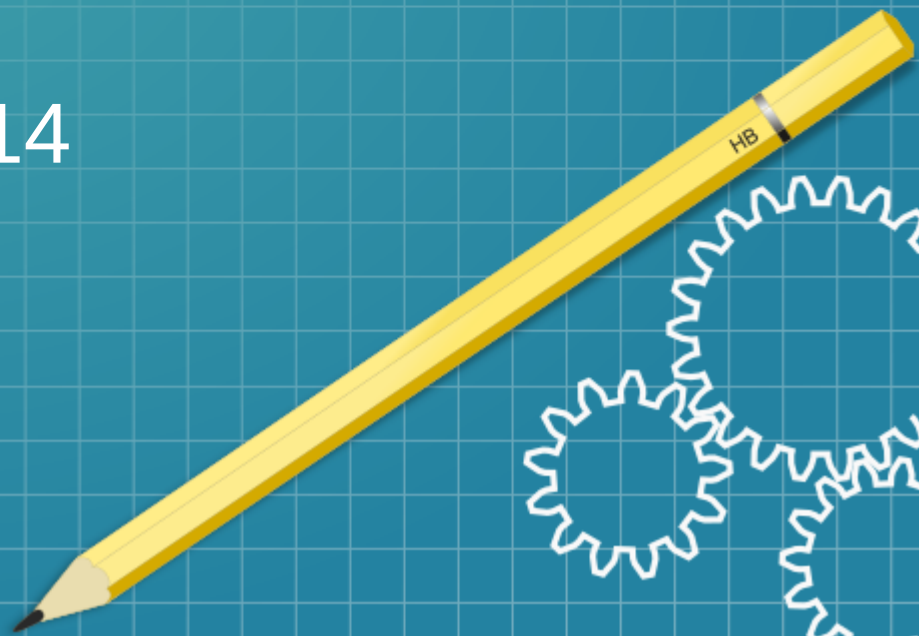
File Processing (Part 1)

Chapter 14



Introduction

C++ views each file as a *sequence of bytes*.

Each file ends with an end-of-file marker or at a specific byte number recorded in an operating-system-maintained administrative data structure.

When a *file is opened*, an *object* associated with the file *is created*, and a *stream is associated* with the object.

When we finished working with a file, we must *close* it.

File processing Class Templates

To perform file processing in C++, include the headers `<iostream>` and `<fstream>`

`<fstream>` header includes the definitions for the stream class templates:
`basic_ifstream` : a subclass of `basic_istream` for file input
`basic_ofstream`: a subclass of `basic_ostream` for file output
`basic_fstream`: a subclass of `basic_iostream` for file input and output

<iostream> header

Each has a predefined specialization for char I/O, and <fstream> provides aliases for these template specializations:
ifstream, alias for basic_ifstream<char>
ofstream, alias for basic_ofstream<char>
fstream is an alias for basic_fstream<char>

All the I/O capabilities described in Chapter 13 also can be applied to file streams.

Opening a File for Writing

Let's create two files: file1.txt and file2.txt and fill them out with some information.

- file1.txt will be used for the account balance
- file2.txt will be used for the record of transactions

file1.txt format:
<account number> <name of account holder> <balance>

file2.txt format:
<account number> <transaction amount>

if balance is negative, it means that the person owes to bank
if transaction amount is negative, it means that amount should be subtracted from the account's balance

see seqFileCreation.cpp

Reading Data From a Sequential File

Let’s read the information from file1.txt:

see readSeqFile.cpp

File Position Pointers

Often programs read *sequentially* from the beginning of a file.

What if we need to find some specific data?

What if we will need to process the same file several times?

File Position Pointers

`istream` and `ostream` provide member functions:

`seekg` : seek get; sets the *position* of the next character to be extracted from the input stream.

`seekp` : seek put; sets the *position* where the next character is to be inserted into the output stream

Both functions reposition the *file-position pointer*.

Each `istream` object has a *get pointer*, the byte number in the file from which the *next input* to occur.
Each `ostream` object has a *put pointer*, the byte number in the file at which the next *output* should be placed.

File Position Pointers

`istream` and `ostream` provide member functions:

`seekg` : seek get; sets the *position* of the next character to be extracted from the input stream.

```
istream& seekg(streampos pos);
```

`seekp` : seek put; sets the *position* where the next character is to be inserted into the output stream

```
ostream& seekp(streampos pos);
```

Both functions reposition the *file-position pointer*.

Each `istream` object has a *get pointer*, the byte number in the file from which the *next input* to occur.
Each `ostream` object has a *put pointer*, the byte number in the file at which the next *output* should be placed.

File Position Pointers

`istream` and `ostream` provide member functions:

`seekg` : seek get;

```
inFile.seekg(0);
```

re-position the file-pointer to the beginning of the file

`seekp` : seek put;

```
// get the pos. of the current char. in the output stream
long  p = outFile.tellp();
outFile.seekp(p-8); // put pointer 8 characters back
```

`tellp()` and `getp()` return the current locations of the get and put pointers.

see seekExamples.cpp

Reading/Writing Quoted Text

Many text files contain *quoted text*.

Assume our account information file was formatted as follows:

```
192 “John Smith” 192.34
```

From C++ 14 we can use *stream manipulator* `quoted`.

It enables a program to read *quoted text* from a *stream,* including any white space characters in the *quoted text*, and discards the double quote delimeters.

```
InFile >> account >> quoted(name) >> balance;
```

Similarly, we can write *quoted text* to a stream.

In-class work

Assume that the file Account.txt contains the account information and is formatted as follows:

<account number> <account holder name, quoted > <balance>
187 “Adam Lee” 12895.60

Assume that we are also given three files with various transactions, formatted as follows:

<account number> <transaction amount>
187 90.50
187 -7658.64

If <transaction amount> is negative, it means that the amount was withdrawn from the account
If <transaction amount> is positive, it means that this amount was deposited into the account.

In-class work

Our goal is to process the Account.txt file and all the transaction files, create a new file and put the updated information there.

The output file name should be “AccountUpdate.txt”

Some comments:
1) You are not allowed to use vectors, arrays, or any other containers
2) The file Account.txt has unique accounts
3) transaction files can have multiple transactions for the same account
4) your code must be well commented (almost every line of the code)

HW assignment

1) Exercises 13.6, 13.7
2) recall the `class Complex`:
re-define the stream extraction operator (input stream) to be able to get the input in the form 4 – 9i from the user. It should determine whether the data entered is valid, and if it is not, it should set `failbit` to indicate improper input.

Self-Study:
read sections 13.7 and 13.8

Optional (for self-development):
Sections 13.6.4