SEQUENTIAL FILE PROCESSING

Chapter 14

TOPICS

Files and Streams

Creating & Writing to Sequential Files

Reading from a Sequential File

Closing Files

Files and Streams

Files and Streams are Just a Sequence of Bytes

Remember a file is simply a sequence of bytes, ending with an EOF marker

When a file is opened, an object is created and a stream associated with that object

Streams can be input, output, or both

FILE-PROCESSING CLASS TEMPLATES

Must include **<iostream>** and **<fstream>** libraries

<iostream> creates **cin** and **cout**

- Allow program to communicate with external devices

<fstream> includes definitions for using files:

- ifstream – for input
- ofstream – for output

Types of Files

Sequential

- Data stored in some agreed-upon structure
- Data usually accessed from beginning of file

Random-access

- Data stored in a fixed-length structure
- Data can be accessed directly

CREATING & WRITING TO SEQUENTIAL FILES

SEQUENTIAL FILES

No inherent structure in a sequential file

You must structure it appropriately for its use

One use for a file is to store **data:**

- **Field** – single piece of information, for example – an account number
- **Record** – group of fields for one entity, such as all the fields related to one client
- **Record key** – the field used to sort, if sorting is required

Opening a File for Output using the ofstream Class – Format 1

General format:

ofstream variableNameForFileObject{“name of file on disk”, fileMode};

ofstream – class for working with output files

VariableNameForFileObject – variable to be used to refer to the file object in code

Name of file on disk – the actual name of the file as stored on disk, including the file extension; can include a path

File mode – indicates how this file can be used

Opening a File for Output using the ofstream Class – Format 2

Alternate format:

ofstream variableNameForFileObject;

variableNameForFileObject.open(“name of file on disk”, fileMode);

The open function is part of the ofstream class.

File Modes

The file mode indicates how the file will be used

File Mode	Purpose	If File Exists	If File Doesn't Exist
ios::app	Append all output to the end of the file.	New data added to end of existing data	File is created
ios::ate	Open 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.	New data added to end of existing data	File is created
ios::in	Open a file for input.	File is opened for input	Error
ios::out	Open a file for output.	New data added to end of existing data	File is created
ios::trunc	Discard the file's contents (this also is the default action for ios::out).	New data added to end of existing data	File is created
ios::binary	Open a file for binary, i.e., nontext, input or output.	File is opened	File is created

Creating and Writing to a Sequential File – Example

Fig14_02.cpp

Opens file for output using ofstream (line 11)

- File-open mode indicates file will receive output

Tests if file was opened successfully (lines 14-17)

- Uses overloaded negation operator !
- Invokes function **exit** if there was a problem

Creating and Writing to a Sequential File – Example – Note Loop

Loop to read from cin (lines 27-30)

- Continues to execute if user hasn't typed end-of-file indicator

End-of-file indicator:

- Ctrl-z, then Enter key in Windows
- Ctrl-d for Mac/Unix/Linux

WRITING TO A SEQUENTIAL FILE

A file object is a stream, just like cin or cout

So the same stream insertion/extraction operators can be used: << >>

General format for writing to a file:

variableNameForFileObject << *values to write* << endl;

endl only needed if a newline is needed in the file

WRITING TO A SEQUENTIAL FILE – EXAMPLE

Still in same example **Fig14_02.cpp**

Statement to write to file on line 28

READING FROM A SEQUENTIAL FILE

Opening a File for Input using the ifstream Class – Format 1

General format:

ifstream variableNameForFileObject{“name of file on disk”, fileMode};

ifstream – class for working with output files

VariableNameForFileObject – variable to be used to refer to the file object in code

Name of file on disk – the actual name of the file as stored on disk, including the file extension; can include a path

File mode – ios::in

Opening a File for Input using the ifstream Class – Format 2

Alternate format:

ifstream variableNameForFileObject;

variableNameForFileObject.open(“name of file on disk”, fileMode);

The open function is also part of the ifstream class.

READING FROM A SEQUENTIAL FILE – EXAMPLE

Fig14_05.cpp

Opens file for input using ifstream (line 14)

- File-open mode indicates file will provide input to the program

Same test if file was opened successfully (lines 17-20)

- Uses overloaded negation operator !
- Invokes function **exit** if there was a problem

Reading from a Sequential File – Example – Note Loop

Loop to read from file (lines 30-32)

- Continues to execute until the end-of-file

If the EOF indicator is read from the file, the condition on line 30 becomes false, and the loop ends

READING FROM A SEQUENTIAL FILE – CAN USE STREAM OPERATORS

A file object is a stream, just like cin or cout

So the same stream insertion/extraction operators can be used: << >>

General format for reading from a file:

variableNameForFileObject >> ***variable(s) to store data read*****;**

File Position Pointers

Normally, sequential files are read from beginning to end

Every time data is read/written, the current position in file is automatically updated

Current position in file represented by a **file-position pointer**

If we need to read through the file again, we can reposition the file-position pointer using **seek** functions

- **seekg** (seek get) for ifstream
- **seekp** (seek put) for ofstream

SEEK FUNCTIONS

General format:

***variableNameForFileObject*.seekg(*integer*, *direction*);**

First parameter is a **byte number** (as an integer)

Second parameter is the **seek direction**:

- ios::beg – position relative to beginning of file (default)
- ios::cur – position relative to current position in file
- ios::end – position relative to end of file

SEEK EXAMPLES

To put the file-position pointer:	Use these arguments for seek:
At the beginning of the file	fileObject.seekg(0); fileObject.seekp(0);
n bytes forward from current position	fileObject.seekg(n, ios::cur); fileObject.seekp(n, ios::cur);
n bytes back from end of the file	fileObject.seekg(n, ios::end); fileObject.seekp(n, ios::end);
At the end of the file	fileObject.seekg(0, ios::end); fileObject.seekp(0, ios::end);

To Find The Current Position

To locate the current position of the file-position pointer, use the **tell** functions:

General format:

```
longVariable = variableNameForFileObject.tellg();
longVariable = variableNameForFileObject.tellp();
```

Returns the current position of the indicated file pointer as a long data type

UPDATING SEQUENTIAL FILES

Data in sequential files cannot be modified without risk of destroying other data in the file.

If name “White” needed to be changed to “Washington,” the old name cannot simply be overwritten, because the new name requires more space.

123	White	1000

123	Washington

Updating Sequential Files – Not Usually Done in Place

Fields and records in a text file can vary in size.

Records in a sequential file not usually updated in place.

Instead, the entire file is usually rewritten.

Rewriting the entire file is uneconomical to update just one record, but reasonable if a substantial number of records need to be updated.

LONGER EXAMPLE

Fig14_06.cpp

- Program to view accounts from this file in various ways
- Uses seekg and clear functions

clear function

- In this context, resets eof state so we can read through the file again

CLOSING FILES

USE THE CLOSE() FUNCTION TO CLOSE A FILE

When an open file is no longer needed, close it

fileObject.close();

fileObject can be an object of **ifstream** or **ofstream** class

Important for proper cleanup and to prevent memory leaks

Files will be closed automatically when program ends successfully

- But may not be if program crashes while a file is open