IS 0020
Program Design and Software Tools

Stack/Queue - File Processing

Lecture 10

March 29, 2005

Introduction

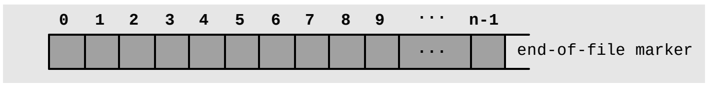
- Storage of data
  - Arrays, variables are temporary
  - Files are permanent
    - Magnetic disk, optical disk, tapes
- In this chapter
  - Create, update, process files
  - Sequential and random access
  - Formatted and raw processing

The Data Hierarchy

- From smallest to largest
  - Bit (binary digit)
    - 1 or 0
    - 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`**)
- 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…

Files and Streams

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



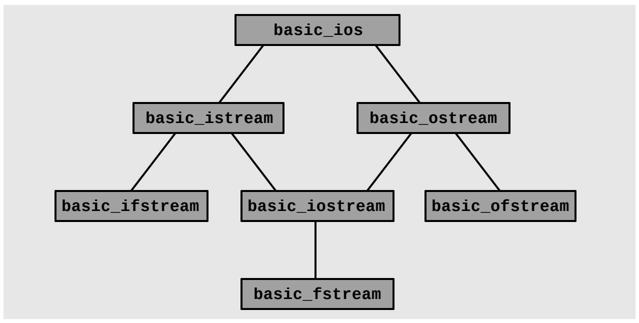
- When file opened
  - Object created, stream associated with it
  - **`cin`**, **`cout`**, etc. created when **`<iostream>`** included
    - Communication between program and file/device

Files and Streams

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

Files and Streams

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



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);
```

Creating a Sequential-Access File

- File-open modes

Mode	Description
ios::app	Write 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 if it exists (this is also the default action for **ios::out**)
ios::binary	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");`**

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 **`failbit`** or **`badbit`** set, otherwise nonzero
      - **`failbit`** set when EOF found
    - **`while ( cin >> myVariable )`**
      - Implicitly converts **`cin`** to pointer
      - Loops until EOF
  - Writing to file (just like **`cout`**)
    - **`outClientFile << myVariable`**
  - Closing file
    - **`outClientFile.close()`**
    - Automatically closed when destructor called

Outline

fig14_04.cpp
(1 of 2)

```
// Fig. 14.4: fig14_04.cpp
// Create a sequential file.
#include <iostream>

using std::cout;
using std::cin;
using std::ios;
using std::cerr;
using std::endl;

#include <fstream>

using std::ofstream;

#include <cstdlib>  // exit prototype

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

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

   } // end if
```

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.

Outline

fig14_04.cpp
(2 of 2)

```

   cout << "Enter the account, name, and balance." << endl
        << "Enter end-of-file t

   int account;
   char name[ 30 ];
   double balance;

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

   } // end while

   return 0;  // ofstream destructor closes file

} // 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()`**.

Outline

fig14_04.cpp
output (1 of 1)

```
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
```

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`**)

Outline

fig14_07.cpp
(2 of 3)

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

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

   } // end if

   int account;
   char name[ 30 ];
   double balance;

   cout << left << setw( 10 ) << "Acco
        << "Name" << "Balance" << endl

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

   return 0; // ifstream destructor closes the file

} // end main
```

Open and test file for input.

Read from file until EOF found.

Outline

fig14_07.cpp
(3 of 3)

fig14_07.cpp
output (1 of 1)

```
// display single record from file
void outputLine( int account, const char * const 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 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

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
- To find pointer location
  - **`tellg`** and **`tellp`**
  - **`location = fileObject.tellg()`**

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

```
300 Worthington 0.00

300 White 0.00 400 Jones 32.87

300 Worthington 0.00ones 32.87
```

Data gets overwritten

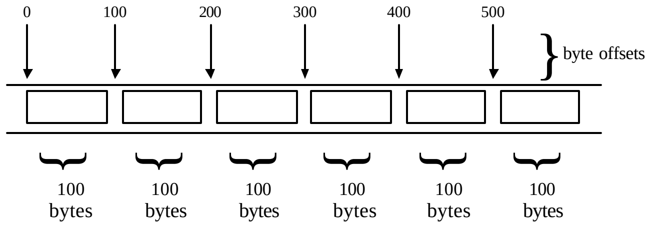
  - Formatted text different from internal representation
  - Problem can be avoided, but awkward

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

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



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 ***

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
- Usually write entire **struct** or object to file

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)
- **`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 *`**

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`**)