Advanced Files

CS 1: Problem Solving & Program Design Using C++

Objectives

- How do we operate on them
- Format the output to look pretty
- Passing them into functions
- Testing for errors
- Functions for reading and writing them
- Working with multiple files
- Binary files
- Creating records with structures
- Random-access files
- Reading and writing to the same file

File Operations

- FILE: a set of data stored on a computer, often on a disk drive
- Programs can read from and write to files
- Used in many applications:
    - Word processing
    - Databases
    - Spreadsheets
    - Compilers

Using Files

- Requires fstream header file (i.e. #include <fstream>)
    - Use ifstream data type for input files
    - Use ofstream data type for output files
    - Use fstream data type for both input, output files
- Use >> to read from a file
- Use << to write to a file
- Can use eof member function to test for end of input file

fstream Object

- fstream object can be used for either input or output
- Must specify mode on the open statement
- Sample modes:

  ios::in – input

  ios::out – output
- Can be combined on open call:

```
dFile.open("class.txt", ios::in | ios::out);
```

File Access Flags

Flag	Description
ios::app	• Append mode • If the file already exists, its contents are preserved and all output is written to the end of the file • By default, this flag causes the file to be created if it does not exist
ios::ate	• If the file already exists, the program goes to the end of it • Output may be written anywhere in the file
ios::binary	• Binary mode • When a file is opened in binary mode, data is written to or read from it in pure binary mode (the default mode is text)

File Access Flags (2)

Flag	Description
ios::in	• Input mode • Data will be read from the file • If the file does not exist, it will not be created and the open function will fail
ios::out	• Output mode • Data will be written to the file • By default, the file’s contents will be deleted if it already exists
ios::trunc	• If the file already exists, its contents will be deleted (truncated) • Default mode used by ios::out

Using Files Example

```
#include <fstream>
using namespace std;

int main()
{
        // copy 10 numbers between files
        // open the files
        fstream infile("input.txt", ios::in);
        fstream outfile("output.txt", ios::out);
        int num;
```

Using Files Example (2)

```
        for (int i = 1; i <= 10; i++)
        {
                infile >> num;    // use the files
                outfile << num;
        }
        infile.close();     // close the files
        outfile.close();

        return 0;
}
```

Default File Open Modes

- ifstream
  - Open for input only
  - File cannot be written to
  - Open fails if file does not exist
- ofstream
  - Open for output only
  - File cannot be read from
  - File created if no file exists
  - File contents erased if file exists

More File Open Details

- Can use filename, flags in definition:

```
ifstream gradeList("grades.txt");
```

- File stream object set to 0 (false) if open failed:

```
if (!gradeList) ...
```

- Can also check fail member function to detect file open error:

```
if (gradeList.fail()) ...
```

File Output Formatting

- Use the same techniques with file stream objects as with cout: showpoint, setw(x), showprecision(x), etc.
- Requires iomanip to use manipulators

File Output Formatting Example

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

int main()
{
        fstream dataFile;
        double num = 17.816392;

        dataFile.open("numfile.txt", ios::out); // Open in output mode
```

<u>numfile.txt</u>

```
17.816392
17.8164
17.816
17.82
17.8
```

File Output Formatting Example (2)

```
dataFile << fixed; // Format for fixed-point notation
dataFile << num << endl; // Write the number

dataFile << setprecision(4); // Format for 4 decimal places
dataFile << num << endl; // Write the number

dataFile << setprecision(3); // Format for 3 decimal places
dataFile << num << endl; // Write the number
```

File Output Formatting Example (3)

```
        dataFile << setprecision(2); // Format for 4 decimal places
        dataFile << num << endl; // Write the number

        dataFile << setprecision(1); // Format for 3 decimal places
        dataFile << num << endl; // Write the number

        cout << "Done." << endl;
        dataFile.close();
        return 0;
}
```

Passing File Streams Objects to Functions

- It is very useful to pass file stream objects to functions
- Be sure to always pass file stream objects by reference

Passing File Stream Objects to Functions Example

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
using namespace std;

bool openFileIn(fstream &, string);
void showContents(fstream &);

int main ()
{
		fstream dataFile;
```

<u>demofile.txt</u>

Jones

Smith

Willis

Davis

Passing File Stream Objects to Functions Example (2)

```
        if (openFileIn(dataFile, "demofile.txt"))
        {
                cout << "File opened successfully." << endl;
                cout << "Now reading data from the file." << endl;
                showContents(dataFile);
                dataFile.close();
                cout << endl << "Done." << endl;
        }
        else
        {
                cout << "File open error!" << endl;
        }

        return 0;
}
```

Passing File Stream Objects to Functions Example (3)

```
bool openFileIn(fstream &file, string name)
{
        file.open(name.c_str(), ios::in);
        if (file.fail())
        {
                return false;
        }
        else
        {
                return true;
        }
}
```

Passing File Stream Objects to Functions Example (4)

```
void showContents(fstream &file)
{
        string line;

        while (file >> line)
        {
                cout << line << endl;
        }
}
```

More Detailed Error Testing

- Can examine error state bits to determine stream status
- Bits tested/cleared by stream member functions

Error State Bit	Description
`ios::eofbit`	Set when end of file detected
`ios::failbit`	Set when operation failed
`ios::hardfail`	Set when error occurred and no recovery
`ios::badbit`	Set when invalid operation attempted
`ios::goodbit`	Set when no other bits are set

Member Functions/Flags

Functions	Description
`eof()`	true if `eofbit` set, false otherwise
`fail()`	true if `failbit` or `hardfail` set, false otherwise
`bad()`	true if `badbit` set, false otherwise
`good()`	true if `goodbit` set, false otherwise
`clear()`	clear all flags (no arguments), or clear a specific flag

Member Functions for Reading and Writing Files

- Functions that may be used for input with whitespace, to perform single character I/O, or to return to the beginning of an input file
- Member functions:
  - getline: reads input including whitespace
  - get: reads a single character
  - put: writes a single charact

The getline Function

- Three arguments:
  - Name of a file stream object
  - Name of a string object
  - Delimiter character of your choice
    - If left out, '\n' is default for third argument
- Examples, using the file stream object myFile, and the string objects name and address:

```
getline(myFile, name);

getline(myFile, address, '\t');
```

getline Function Example

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
using namespace std;

int main()
{
        string input;
        fstream nameFile;

        nameFile.open("murphy.txt", ios::in);
```

murphy.txt
Charlie Murphy
47 Jones Circle
Almond, NC 28702

getline Function Example (2)

```
        if (nameFile)
        {
                getline(nameFile, input);
                while (nameFile)
                {
                        cout << input << endl;
                        getline(nameFile, input);
                }

                nameFile.close();
        }
        else
        {
                cout << "ERROR:  Cannot open file." << endl;
        }
        return 0;
}
```

Single Character I/O

- get: read a single character from a file

```
char letterGrade;
gradeFile.get(letterGrade);
```

  - Will read any character, including whitespace

- put: write a single character to a file

```
reportFile.put(letterGrade);
```

Working with Multiple Files

- Can have more than file open at a time in a program
- Files may be open for input or output
- Need to define file stream object for each file

Working with Multiple Files Example

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
using namespace std;

int main()
{
        string fileName;
        char ch;
        ifstream inFile;
        ofstream outFile("out.txt");
```

hownow.txt
how now brown cow.
How Now?

Working with Multiple Files Example (2)

```
cout << "Enter a file name : ";
cin >> fileName;

inFile.open(fileName.c_str());

if (inFile)
{
        inFile.get(ch);

        while (inFile)
        {
                outFile.put(toupper(ch));
                inFile.get(ch);
        }
```

Working with Multiple Files Example (3)

```
          inFile.close();
          outFile.close();
          cout << "File conversion done." << endl;
     }
     else
     {
          cout << "Cannot open " << fileName << endl;
     }
     return 0;
}
```

Binary Files

- Binary file contains unformatted, non-ASCII data
- Indicate by using binary flag on open:

```
inFile.open("nums.dat", ios::in | ios::binary);
```

Binary Files (2)

- Use read and write instead of <<, >>

```
char ch;

// read in a letter from file

inFile.read(&ch, sizeof(ch));
```

  - ch&: address of where to put the data being read in; the read function expects to read chars
  - sizeof(ch): how many bytes to read from the file

```
// send a character to a file

outFile.write(&ch, sizeof(ch));
```

Binary Files (3)

- To read, write non-character data, must use a typecast operator to treat the address of the data as a character address

```
int num;
// read in a binary number from a file
inFile.read(reinterpret_cast<char *>&num, sizeof(num));
```

  - reinterpret_cast<char *> treats the address of num as the address of a char

```
// send a binary value to a file
outf.write(reinterpret_cast<char *>&num, sizeof(num));
```

Creating Records with Structures

- Can write structures to, read structures from files
- To work with structures and files
    - Use ios::binary file flag upon open
    - Use read, write member functions

Creating Records with Structures Example

```
struct TestScore
{
        int studentId;
        double score;
        char grade;
};
TestScore oneTest;
...
// write out oneTest to a file
gradeFile.write(reinterpret_cast<char *> (&oneTest), sizeof(oneTest));
```

Random Access Files

- SEQUENTIAL ACCESS: start at beginning of file and go through data in file, in order, to end
  - To access 100th entry in file, go through 99 preceding entries first
- RANDOM ACCESS: access data in a file in any order
  - Can access 100th entry directly

Random Access Member Functions

- seekg (seek get): used with files open for input
- seekp (seek put): used with files open for output
- Used to go to a specific position in a file

Random Access Member Functions (2)

- seekg,seekp arguments:
  - OFFSET: number of bytes, as a long
  - MODE FLAG: starting point to compute offset
- Examples:

```
inData.seekg(25L, ios::beg);
// set read position at 26th byte
// from beginning of file
outData.seekp(-10L, ios::cur);
// set write position 10 bytes
// before current position
```

Important Note on Random Access

- If eof is true, it must be cleared before seekg or seekp

```
gradeFile.clear();
gradeFile.seekg(0L, ios::beg);
 // go to the beginning of the file
```

Random Access Information

- tellg member function: return current byte position in input file

```
long int whereAmI;
whereAmI = inData.tellg();
```

- tellp member function: return current byte position in output file

```
whereAmI = outData.tellp();
```

Opening a File for Both Input and Output

- File can be open for input and output simultaneously
- Supports updating a file:
  - Read data from file into memory
  - Update data
  - Write data back to file
- Use fstream for file object definition:

```
fstream gradeList("grades.dat", ios::in | ios::out);
```

- Can also use ios::binary flag for binary data

Summary

- How do we operate on them
  - ifstream, ofstream, fstream
  - File access flags (e.g. ios::in, ios::out)
- Format the output to look pretty
  - Use the same techniques as cout (i.e. showpoint, showprecision(x), setw(x))
- Passing them into functions
  - Pass by reference
- Testing for errors
  - Bits: ios::eofbit, ios::failbit, etc.
  - Flags: bad(), good(), clear(), etc.
- Functions for reading and writing them
  - getline(), get(), put()

Summary (2)

- Working with multiple files
  - Files may be open for input or output
  - File streams need to be opened for each file
- Binary files
  - Contain unformatted, non-ASCII data
  - Uses ios::binary flag
- Creating records with structures
  - Use ios::binary flag, as well as read(), write() functions
- Random-access files
  - Allows us to look for a specific position for a specific file using seekg() (input) and seekp() (output)
- Reading and writing to the same file
  - Ability to open a file for input and output simultaneously, as long as ios::in and ios::out flags are used for it