CC-211L

Object Oriented Programming

Laboratory 12

File Processing

Version: 1.0.0

Release Date: 13-04-2023

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Files and Streams
- Create a Sequential File
- Read a Sequential File
- Update a Sequential File
- Random Access File
- Create a Random-Access File
- Read a Random-Access File

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Background and Overview:

Files and Streams:

Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.

A stream is an abstraction that represents a device on which operations of input and output are performed. A stream can be represented as a source or destination of characters of indefinite length depending on its usage.

Sequential File:

A sequential file is simply a file where the data is stored one item after another. A more technical definition would be that a sequential file is a collection of data stored on a disk in a sequential, non-indexed, manner. C++ treats all such files simply as a sequence of bytes. Each file ends with a special end of file marker.

Random-Access File:

Random file access enables us to read or write any data in our disk file without having to read or write every piece of data before it while in Sequential files to reach data in the middle of the file you must go through all the data that precedes it. We can quickly search for data, modify data, delete data in a random-access file.

Activities:

Pre-Lab Activities:

Files and Streams:

C++ views each file simply as a sequence of bytes. Each file ends either 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 is created, and a stream is associated with the object.

To perform file processing in C++, headers <iostream> and <fstream> must be included.

Header <fstream> 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.

In addition, the <fstream> library provides typedef aliases for these template specializations:

- ifstream is an alias for basic_ifstream<char>
- ofstream is an alias for basic_ofstream<char>
- fstream is an alias for basic_fstream<char> .

Creating a Sequential File:

Example:

```cpp
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;
int main()
{
    // ofstream constructor opens file
    ofstream outClientFile("clients.dat", ios::out);
    if (!outClientFile) { // overloaded ! operator
        cerr << "File could not be opened" << endl;
        exit(1); // prototype in stdlib.h
    }
    cout << "Enter the account, name, and balance.\n"
        << "Enter end-of-file to end input.\n? ";
    int account;
    char name[30];
    float balance;
    while (cin >> account >> name >> balance) {
        outClientFile << account << ' ' << name
            << ' ' << balance << '\n';
        cout << "? ";
    }
    return 0; // ofstream destructor closes file
}
```

Fig. 01 (Sequential File)

Opening a file:

Following syntax is followed to open a file:

Ofstream outClientFile;

outClientFile.open(Filename, Mode)

Following are the different modes to open a file:

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. Data can be written anywhere in the file.
ios::in	Open file for input.
ios::out	Open file for output.
ios::trunc	Discard's the file content.
ios::binary	Open a file for binary.

Closing a file:

File is closed implicitly when a destructor for the corresponding object is called. It can also be called explicitly by using member function close:

outClientFile.close();

Reading a Sequential file:

Example:

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <stdlib.h>
using namespace std;
void outputLine(int, const char*, double);
int main(){
    // ifstream constructor opens the file
    ifstream inClientFile("clients.dat", ios::in);
    if (!inClientFile) {
        cerr << "File could not be opened\n";
        exit(1);
    }
    int account;
    char name[30];
    double balance;
    cout << setiosflags(ios::left) << setw(10) << "Account"
        << setw(13) << "Name" << "Balance\n";
    while (inClientFile >> account >> name >> balance)
        outputLine(account, name, balance);
    return 0; // ifstream destructor closes the file
}
void outputLine(int acct, const char* name, double bal){
    cout << setiosflags(ios::left) << setw(10) << acct
        << setw(13) << name << setw(7) << setprecision(2)
        << resetiosflags(ios::left)
        << setiosflags(ios::fixed | ios::showpoint)
        << bal << '\n';
}
```

Fig. 02 (Reading Sequential File)

File Content:

Fig. 03 (Reading Sequential File)

Output:

Fig. 04 (Reading Sequential File)

Task 01: Word Occurrence **[Estimated time 15 minutes / 15 marks]**

- Create a text file and write some text in it
- Read the file contents using Sequential Access
- Count the words occurrence and display the most repetitive word/s on the Console

Task 02: Map 2D Array **[Estimated time 15 minutes / 15 marks]**

- Create a text file and write 2D array contents as shown in the figure

```
3 4
1 2 3 4
5 6 7 8
1 2 3 4
```

Fig. 05 (Pre-Lab Task)

- Read the file contents using Sequential Access
- Map the contents to a 2D array and display it on the Console.
- On the first line, there are number of rows and columns being printed.

Task03: Running Calculator **[Estimated time 20 minutes / 20 marks]**

Create a program in cpp in which user will keep on inserting new numbers to get the total sum until user inputs -1. Get each number from user, write it in a file “Numbers.txt” and get the sum of all the numbers written in it. Numbers will be stored with space between them and comma. Like for example enters 5, 10, 40, 50, 28, 27. You will display the sum of all the numbers in file when ever a new number is inserted in file.

In-Lab Activities:

File Position Pointers:

<istream> and <ostream> classes provide member functions for repositioning the file pointer (the byte number of the next byte in the file to be read or to be written). These member functions are:

- seekg (seek get) for istream class
- seekp (seek put) for ostream class

Following are some examples to move a file pointer:
- **inClientFile.seekg(0) -** repositions the file get pointer to the beginning of the file
- **inClientFile.seekg(n, ios:beg) -** repositions the file get pointer to the n-th byte of the file
- **inClientFile.seekg(m, ios:end) -** repositions the file get pointer to the m-th byte from the end of file
- **inClientFile.seekg(0, ios:end) -** repositions the file get pointer to the end of the file

The same operations can be performed with <ostream> function member seekp.

Member functions tellg() and tellp():
Member functions tellg and tellp are provided to return the current locations of the get and put pointers, respectively.

long location = inClientFile.tellg();

To move the pointer relative to the current location use ios:cur
inClientFile.seekg(n, ios:cur) - moves the file get pointer n bytes forward.

Example:
In this example, we open a file named “example.txt” for writing using std::ofstream. We then write some data to the file using the << operator. We use the tellp() function to get the current position of the file pointer and use seekp() to move the file pointer back to the beginning of the file. We then write some more data to the file and use seekp() again to move the file pointer to the end of the file. Finally, we write some more data to the file and close it.

```cpp
#include <iostream>
#include <fstream>

int main() {
    std::ofstream file("example.txt");
    // Write some data to the file
    file << "Hello, world!" << std::endl;
    // Get the current position of the file pointer
    std::ostream::pos_type pos = file.tellp();
    // Move the file pointer back to the beginning of the file
    file.seekp(0);
    // Write some more data to the file
    file << "This is a test" << std::endl;
    // Move the file pointer to the end of the file
    file.seekp(pos);
    // Write some more data to the file
    file << "The end" << std::endl;
    // Clean up
    file.close();
    return 0;
}
```

Fig. 06 (File Position Pointer)

File Content:

Fig. 07 (File Position Pointer)

Example:
In this example, we open a file named "example.txt" and use the tellg() function to get the current position of the file pointer. We then use the seekg() function to move the file pointer to the end of the file and use tellg() again to get the size of the file. We then move the file pointer back to the beginning of the file using seekg() and read the data from the file using the read() function. Finally, we output the data to the console and close the file.

```cpp
#include <iostream>
#include <fstream>
#include <string>
int main() {
    std::ifstream file("example.txt");
    // Get the current position of the file pointer
    std::istream::pos_type pos = file.tellg();
    // Move the file pointer to the end of the file
    file.seekg(0, std::ios::end);
    // Get the size of the file
    std::istream::pos_type size = file.tellg();
    // Move the file pointer back to the beginning of the file
    file.seekg(pos);
    // Read data from the file
    std::string buffer;
    while (file) {
        getline(file, buffer);
        // Output the data to the console
        std::cout << buffer << std::endl;
    }
    // Clean up
    file.close();
    return 0;
}
```

Fig. 08 (File Position Pointer)

Output:

Fig. 09 (File Position Pointer)

Updating a Sequential File:

Data that is formatted and written to a sequential file cannot be modified easily without the risk of destroying other data in the file. If we want to modify a record of data, the new data may be longer than the old one and it could overwrite parts of the record following it.

Example:
Following example demonstrates the overwriting of data while trying to update a sequential file.

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
int main() {
    // open the file for reading and writing
    std::fstream file("contacts.txt", std::ios::in | std::ios::out);
    // search for the record to update
    bool found = false;
    std::string line;
    while (std::getline(file, line)) {
        std::istringstream iss(line);
        std::string first, last, address, city, state, zip;
        std::getline(iss, first, ','); std::getline(iss, last, ','); std::getline(iss, address, ',');
        std::getline(iss, city, ','); std::getline(iss, state, ','); std::getline(iss, zip);
        if (first == "Jane" && last == "Doe") {
            found = true;
            city = "San Francisco";
            state = "CA";
            file.seekg(-(static_cast<int>(line.size()) + 1), std::ios::cur);
            file << first << ',' << last << ',' << address << ','
                << city << ',' << state << ',' << zip << std::endl;
            break;
        }
    }
    file.close(); // close the file
    // print the result
    if (found) std::cout << "Record updated." << std::endl;
    else std::cout << "Record not found." << std::endl;
    return 0;
}
```

Fig. 10 (Updating Sequential File)

Output:

Fig. 11 (Updating Sequential File)

File Content (Before Updating):

Fig. 12 (Updating Sequential File)

File Content (After Updating):

Fig. 13 (Updating Sequential File)

Creating a Random-Access File:
Instant access is possible with random access files. Individual records of a random-access file can be accessed directly without searching many other records.

Example:

```cpp
#include <iostream>
#include <fstream>
using namespace std;
struct clientData {
    int accountNumber;
    char lastName[15];
    char firstName[10];
    float balance;
};
int main()
{
    ofstream outCredit("credit1.dat", ios::out);
    if (!outCredit) {
        cerr << "File could not be opened." << endl;
        exit(1);
    }
        clientData blankClient = { 0, "", "", 0.0 };
    for (int i = 0; i < 100; i++)
        outCredit.write
        (reinterpret_cast<const char*>(&blankClient),
            sizeof(clientData));
    return 0;
}
```

Fig. 14 (Creating a Random-Access File)

The <ostream> member function write outputs a fixed number of bytes beginning at a specific location in memory to the specific stream. When the stream is associated with a file, the data is written beginning at the location in the file specified by the "put" file pointer. The write function expects a first argument of type **"const char *"**, hence we used the reinterpret_cast to convert the address of the blankClient to a const char *. The second argument of write is an integer of type **"size_t"** specifying the number of bytes to written. Thus, the sizeof (clientData).

Writing Data to a Random File:

Example:

```cpp
#include <iostream>
#include <fstream>
using namespace std;
struct clientData {
    int accountNumber;
    char lastName[15];
    char firstName[10];
    float balance;
};
int main(){
    ofstream outCredit("credit.dat", ios::ate);
    if (!outCredit) {
        cerr << "File could not be opened." << endl;
        exit(1);
    }
    cout << "Enter account number (1 to 100, 0 to end input)\n? ";
    clientData client;
    cin >> client.accountNumber;
    while (client.accountNumber > 0 &&
        client.accountNumber <= 100) {
        cout << "Enter lastname, firstname, balance\n? ";
        cin >> client.lastName >> client.firstName >> client.balance;
        outCredit.seekp((client.accountNumber - 1) * sizeof(clientData));
        outCredit.write( reinterpret_cast<const char*>(&client), sizeof(clientData));
        cout << "Enter account number\n? ";
        cin >> client.accountNumber;
    }
    return 0;
}
```

Fig. 15 (Writing Data to a Random-Access File)

Output:

```
Microsoft Visual Studio Debug Console
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Rahman Saad 23.00
Enter account number
? 29
Enter lastname, firstname, balance
? Ali Ahmad 0.00
Enter account number
? 0
```

Fig. 16 (Writing Data to a Random-Access File)

Reading Data from a Random File:

Example:

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
void outputLine(ostream&, const clientData&);
int main(){
    ifstream inCredit("credit.dat", ios::in);
    if (!inCredit) {
        cerr << "File could not be opened." << endl;
        exit(1);
    }
    cout << setiosflags(ios::left) << setw(10) << "Account" << setw(16) << "Last Name"
    << setw(11) << "First Name" << resetiosflags(ios::left) << setw(10) << "Balance" << endl;
    clientData client;
    inCredit.read(reinterpret_cast<char*>(&client),sizeof(clientData));
    while (inCredit && !inCredit.eof()) {
        if (client.accountNumber != 0)
            outputLine(cout, client);
        inCredit.read(reinterpret_cast<char*>(&client), sizeof(clientData));
    }
    return 0;
}
void outputLine(ostream& output, const clientData& c){
    output << setiosflags(ios::left) << setw(10) << c.accountNumber << setw(16) << c.lastName
        << setw(11) << c.firstName << setw(10) << setprecision(2) << resetiosflags(ios::left)
        << setiosflags(ios::fixed | ios::showpoint) << c.balance << '\n';
}
```

Fig. 17 (Reading Data from a Random-Access File)

Output:

```
Microsoft Visual Studio Debug Console
Account   Last Name       First Name    Balance
29        Ali             Ahmad            0.00
37        Rahman          Saad            23.00
```

Fig. 18 (Reading Data from a Random-Access File)

Task 01: Compare Files [Estimated time 20 minutes / 20 marks]

- Take two names of files from the user and create those files.
- Write numbers from 1 to 50 in first file with each number being written on a separate line.
- Similarly Write numbers from 51 to 100 in second file with each number being written in a separate line.
- Now create a Third file named Sum.txt which will contain the Sum of numbers from respective line from both files. For example at line 1 in first file the number 1 will be written and in second file It will be 51, so that is why first line in Sum.txt, the number will be 52 (Sum of the numbers from first line in first and second file.)
- Now Display the Content of Sum.txt in a formatted way.

Task 02: My Editor [Estimated time 20 minutes / 20 marks]

- We will implement our own little editor using File System in cpp. When our editor starts, we want to display the previously written data by user and then accept new data as well and store it back as well in file when user want to exit from editor.
- Create a function load from file which which load all the data from a file named as "Data.txt".
- Display all that data to the user.
- Ask the user to enter new data.
- You will have to implement logic to store data received from file and also the new data written by the user now.
- When user input "exit" then close the editor and write all the data back to file including the old data and the new one as well.
- Remember to clear the file before writing data into it.

Task 03: Student Record [Estimated time 30 minutes / 30 marks]

- Create a class named Student to store information about a student.
- Properties
  - name,
  - ID,
  - GPA
  - list of courses (vector)

- Implement the following methods in the class:
- **void setStudentData():** This method should prompt the user to enter the student's name, ID, GPA, and list of courses, and store the data in the object's member variables.
- **void printStudentData():** This method should print the student's name, ID, GPA, and list of courses to the console.
- **void writeToFile(std::ofstream& file):** This method should write the student's data to the specified file in a custom format. Each line in the file should contain the student's name, ID, GPA, and comma-separated list of courses.
- **void readFromFile(std::ifstream& file):** This method should read the student's data from the specified file in the custom format and store it in the object's member variables.

- In your main() function, create an array of Student objects, prompt the user to enter data for each student, and write the data to a file named "studentRecord.txt". Then, read the data from the file and print it to the console.

Post-Lab Activities:

Task 01: Library Management [Estimated time 60 minutes / 40 marks]

Create a program that simulates a library management system. Your program should have the following classes:

- **Book class:** This class should contain information about a book, including its title, author, publisher, publication year, ISBN, and status (e.g., available, checked out). Implement the following methods in the class:
  - **void setBookData():** This method should prompt the user to enter the book's data, and store the data in the object's member variables.
  - **void printBookData():** This method should print the book's data to the console.
  - **void writeToFile(std::ofstream& file):** This method should write the book's data to the specified file.
  - **void readFromFile(std::ifstream& file):** This method should read the book's data from the specified file and store it in the object's member variables.
- **Library class:** This class should contain information about a library, including its name, address, and list of books. Implement the following methods in the class:
  - **void addBook(Book b):** This method should add the specified book to the library's list of books.
  - **void removeBook(Book b):** This method should remove the specified book from the library's list of books.
  - **void printLibraryData():** This method should print the library's name, address, and list of books to the console.
  - **void writeToFile(std::ofstream& file):** This method should write the library's data to the specified file in a custom format. The first line of the file should contain the library's name and address, followed by a blank line. Each subsequent line should contain the data for one book.
  - **void readFromFile(std::ifstream& file):** This method should read the library's data from the specified file in the custom format and store it in the object's member variables.
- In your main() function, create a Library object, prompt the user to enter data for the library and its books, and write the data to a file named "library.txt". Then, read the data from the file and print it to the console.
- Your program should also allow the user to check out and return books from the library. Implement the following methods for the Library class:
  - **void checkOutBook(Book b):** This method should mark the specified book as checked out.
  - **void returnBook(Book b):** This method should mark the specified book as available again.

Task 02: Students Rubric [Estimated time 30 minutes / 20 marks]

- Make a class rubric with data members:
- regno(RegNo. of student),
- clearity(value should be non negative and less than or equal to 2),
- completeness(value should be non negative and less than or equal to 3),
- accuracy(value should be non-negative and less than or equal to 3),
- time(value should be non negative and less than or equal to 2), and
- total_marks.
- Make 3 objects in Main.
- inputs data for regno, clearity, completeness, accuracy, time, and total_marks=clearity+completeness+accuracy+time for wach object.

- Write all the Data properties of in a file. Properties of each object will be placed in a respective line. For example if we have objects,Data in file will be placed as :

  Regno clearity completeness accuracy time total_marks
  Regno clearity completeness accuracy time total_marks
  Regno clearity completeness accuracy time total_marks
  (Note here that each line has data of one single object).

- Read back the data properties from file and make 3 new objects.
- Display these objects.

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA's will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA's
- **Division of Pre-Lab marks:** **[30 marks]**
  - Task 01: Word Occurrence [15 marks]
  - Task 02: Map 2D Array [15 marks]
- **Division of In-Lab marks:** **[70 marks]**
  - Task 01: Compare Files [20 marks]
  - Task 02: Students Rubric [20 marks]
  - Task 03: Student Record [30 marks]
- **Division of Post-Lab marks:** **[40 marks]**
  - Task 01: Library Management [40 marks]

References and Additional Material:

- **C++ Files and Streams**
  https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm

Lab Time Activity Simulation Log:

- Slot – 01 – 00:00 – 00:15: Class Settlement
- Slot – 02 – 00:15 – 00:40: In-Lab Task
- Slot – 03 – 00:40 – 01:20: In-Lab Task
- Slot – 04 – 01:20 – 02:20: In-Lab Task
- Slot – 05 – 02:20 – 02:45: Evaluation of Lab Tasks
- Slot – 06 – 02:45 – 03:00: Discussion on Post-Lab Task