Object-Oriented Programming

CL-1004

Lab 10

File Handling

National University of Computer and Emerging Sciences–NUCES – Karachi

Contents

1. Introduction to Filing

Filing is an essential concept in programming that deals with storing data permanently on a storage device, such as a hard drive. Unlike variables or arrays that lose their data once a program ends, files allow data to persist even after the program has terminated. In C++, file handling provides mechanisms to perform input and output operations on files, making it possible to save program outputs for future use or read existing data for processing. This is particularly useful in real-world applications like data logging, configuration storage, or user data management.

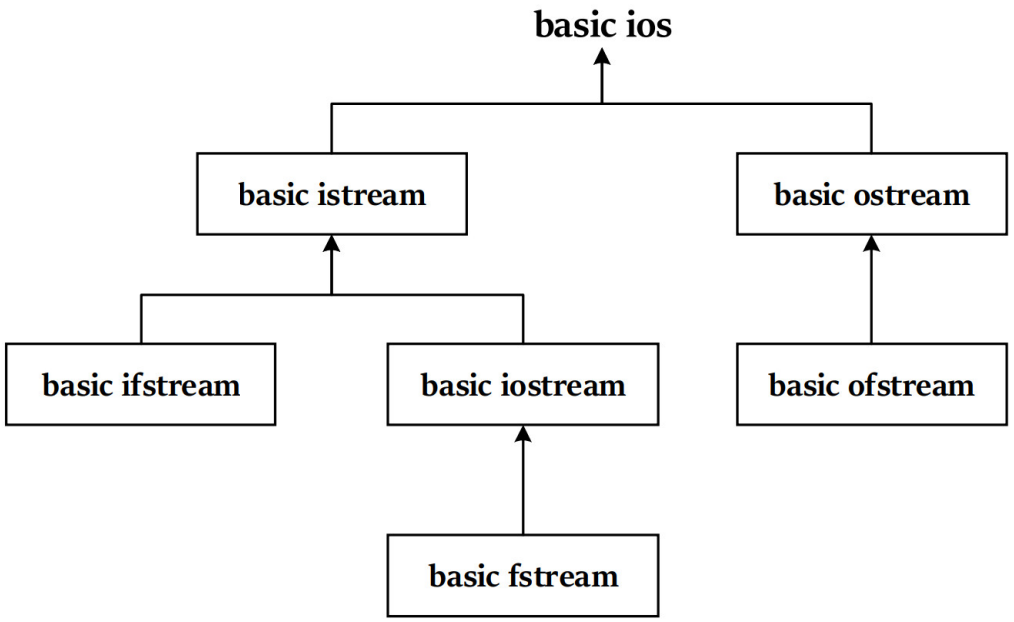
2. Streams

In C++, all input and output operations are handled using streams, which are sequences of bytes that flow between a program and external devices. When performing input operations, data flows from an external source (like a keyboard, file, or network) into the program's memory.

Conversely, in output operations, data flows from the program's memory to an external destination (like a screen, printer, or file).

The standard I/O operations we commonly use, such as **cin** for input and **cout** for output, are based on streams provided by the iostream library. These standard streams are just one part of a larger system of file and device communication in C++.

2.1. Stream I/O Template Hierarchy



C++ provides a flexible and powerful file handling mechanism through three main classes derived from the stream hierarchy:

- **ifstream** (input file stream): Used to **read** data from files.
- **ofstream** (output file stream): Used to **write** data to files.
- **fstream** (file stream): A versatile class used for **both** reading and writing operations on files.

These classes are part of the **<fstream>** header file and provide functions and operators to manage file operations easily. By using these classes, you can read characters, lines, or even entire objects from files, and similarly, write various forms of data into files.

3. Opening and Writing into a File

Before performing any read or write operation on a file in C++, the file must first be opened. This step establishes a link between the program and the physical file.

ofstream is used to write to files (stands for output file stream).

A file can be opened in two ways:

1. **Using the constructor:**
   You can pass the file name directly while creating an object of **ifstream**, **ofstream**, or **fstream**.

Syntax:

```
ofstream file("filename.txt");   // opens file for writing

file << "Write this to the file.";  // writes to file
file.close();
```

Example 01:

```
#include <iostream>
#include <fstream>  // required for file handling

using namespace std;

int main() {
    // create an ofstream object and open file using constructor
    ofstream outputFile("example.txt");

    // check if the file opened successfully
    if (!outputFile) {
        cout << "Error: File could not be created!" << endl;
        return 1;
    }
```

```
    // write data to the file
    outputFile << "Hello, this file was opened using the constructor
method.\n";
    outputFile << "File handling in C++ is fun and useful!\n";

    // close the file
    outputFile.close();

    cout << "Data successfully written to example.txt" << endl;

    return 0;
}
```

Output	Data successfully written to example.txt

example.txt
Hello, this file was opened using the constructor method. File handling in C++ is fun and useful!

Example 02:

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

using namespace std;

class FileHandler {
private:
    ofstream outputFile;
    string filename;

public:
    // constructor that opens the file
    FileHandler(string fname) : filename(fname) {
        outputFile.open(filename);
        if (!outputFile) {
            cerr << "Error: File could not be created!" << endl;
            exit(1);
        }
    }

    // method to write data to the file
    void writeData(string data) {
```

```
        if (outputFile.is_open()) {
            outputFile << data;
        }
    }

    // method to close the file
    void closeFile() {
        if (outputFile.is_open()) {
            outputFile.close();
            cout << "Data successfully written to " << filename << endl;
        }
    }

};

int main() {

    FileHandler fileHandler("example.txt");

    // write data to the file
    fileHandler.writeData("Hello, this file was opened using the
constructor method.\n");
    fileHandler.writeData("File handling in C++ is fun and useful!\n");

    // close the file explicitly
    fileHandler.closeFile();

    return 0;
}
```

Output	Data successfully written to example.txt

example.txt

Hello, this file was opened using the constructor method.

File handling in C++ is fun and useful!

2. **Using the open() method:**

   This method allows more flexibility, especially when dealing with multiple files or dynamically assigned file names.

Syntax:

```
ofstream file;
file.open("filename.txt");        // open file
file << "Write this to the file."; // write data
file.close();
```

Example 03:

data.txt | (before)

```
// C++
```

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

int main() {
    ofstream file;

    // open file in append mode
    file.open("data.txt", ios::app);

    if (!file) {
        cout << "Failed to open file!" << endl;
        return 1;
    }

    file << "This line is appended using the open() method.\n";
    file.close();

    cout << "Data written successfully." << endl;
    return 0;
}
```

Output	Data written successfully.

data.txt | (after)

```
// C++
```

```
This line is appended using the open() method.
```

Example 04:

data.txt | (before)

```
// C++
```

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

class FileAppender {
private:
    ofstream file;

public:
    // constructor opens the file immediately
    FileAppender(string filename) {
        file.open(filename, ios::app);
        if (!file) {
            cout << "Failed to open file!" << endl;
            exit(1);
        }
    }

    // Simple method to append text
    void append(string text) {
        file << text;
    }

    // destructor closes the file automatically
    ~FileAppender() {
        file.close();
        cout << "Data written successfully." << endl;
    }
};

int main() {
    FileAppender myFile("data.txt");
    myFile.append("This line is appended using the open() method.\n");

    // file closes automatically when myFile goes out of scope
    return 0;
}
```

Output	Data written successfully.

data.txt | (after)

```
// C++


This line is appended using the open() method.
```

3.1. Mode Flags and Description

The mode specifies how the file should be opened. Some common modes include:

- ios::in – open for reading
- ios::out – open for writing
- ios::app – append to the end of the file
- ios::ate – move to the end of the file, but allow writing anywhere
- ios::binary – open in binary mode
- ios::trunc – truncate the file if it already exists
- ios::nocreate – open only if the file exists
- ios::noreplace – open only if the file does not exist

When opening an input file, it must already exist, otherwise the open operation fails and the stream enters a failure state. However, when opening an output file, it is automatically created if it doesn't exist. If the output file already exists, its previous contents will be erased unless a mode like **ios::app** is used.

By correctly opening and closing files, you ensure smooth and error-free file processing in your C++ programs.

Example 05:

```cpp
#include <iostream>
#include <fstream>  // for file operations
using namespace std;

int main() {
    ofstream file;

    // open file using open() method
    file.open("output.txt");

    // check if file is open
    if (!file) {
        cout << "Error opening file!" << endl;
        return 1;
    }
```

```
    // write to the file
    file << "This is a basic example of writing to a file in C++.\n";
    file << "File handling makes data storage easier.\n";

    // close the file
    file.close();

    cout << "Data written to file successfully." << endl;
    return 0;
}
```

Output	File written successfully.

output.txt
This is a basic example of writing to a file in C++. File handling makes data storage easier.

Example 06:

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

class FileWriter {
    ofstream file;

public:
    FileWriter(string filename) {
        file.open(filename);
        if (!file) {
            cerr << "Error opening file!" << endl;
            exit(1);
        }
    }

    void write(string content) {
        file << content;
    }

    ~FileWriter() {
        file.close();
        cout << "Data written to file successfully." << endl;
    }
};
```

```cpp
int main() {
    FileWriter writer("output.txt");
    writer.write("This is a basic example of writing to a file in
C++.\n");
    writer.write("File handling makes data storage easier.\n");

    return 0;
}
```

Output	File written successfully.

output.txt

```
This is a basic example of writing to a file in C++.
File handling makes data storage easier.
```

4. getline() function and Read from Files

getline() is used to read an entire line (until newline \n) from a file or input.

Syntax:

```cpp
getline(stream, string_variable);
```

- **stream**: can be **cin**, **ifstream** object, etc.
- **string_variable**: where the line will be stored.

Example 01:

example.txt

```
This is line 1 of the example file.
Here comes line 2 with some basic text.
Line 3 shows how files store data persistently.
The fourth line contains numbers: 42, 7.5, 100.
Finally, line 5 demonstrates end-of-file handling.
```

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

int main()
{

    ifstream inFile("example.txt");

    // variable to store each line read from file
    string line;

    // check if file opened successfully
    if (inFile.is_open()){
        // read file line by line until end of file
        while (getline(inFile, line))
        {
            // print current line to console
            cout << line << endl;
        }
        // close the file when done reading
        inFile.close();
    }
    else{

        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

Output	This is line 1 of the example file. Here comes line 2 with some basic text. Line 3 shows how files store data persistently. The fourth line contains numbers: 42, 7.5, 100. Finally, line 5 demonstrates end-of-file handling.

Example 02:

example.txt
This is line 1 of the example file. Here comes line 2 with some basic text. Line 3 shows how files store data persistently.

```
The fourth line contains numbers: 42, 7.5, 100.
Finally, line 5 demonstrates end-of-file handling.
```

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

class FileReader {
private:
    ifstream file;
    string filename;

public:
    // constructor takes filename and opens the file
    FileReader(const string& fname) : filename(fname) {
        file.open(filename);
    }

    // check if file is open
    bool isOpen() const {
        return file.is_open();
    }

    // read and display file contents
    void displayContents() {
        string line;
        while (getline(file, line)) {
            cout << line << endl;
        }
    }

    // destructor closes the file automatically
    ~FileReader() {
        if (file.is_open()) {
            file.close();
        }
    }
};

int main() {

    FileReader reader("example.txt");

    if (reader.isOpen()) {
        reader.displayContents();
    }
```

```
    else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

Output	This is line 1 of the example file. Here comes line 2 with some basic text. Line 3 shows how files store data persistently. The fourth line contains numbers: 42, 7.5, 100. Finally, line 5 demonstrates end-of-file handling.

4.1. Reading Word by Word

Example 03:

readfileexample.txt
HelloWorld! Programming
3.14 42@test 100%
C++ File-Read Example
Space Tab Separated
Goodbye! Visit_Again :)

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

using namespace std;

int main() {
    ifstream inFile("readfileexample.txt");
    string word;

    if (inFile.is_open()) {
        while (inFile >> word) {
            cout << word << endl;
        }
        inFile.close();
    } else {
        cout << "File not found." << endl;
    }

    return 0;
}
```

Output	HelloWorld! Programming 3.14 42@test 100% C++ File-Read Example Space Tab Separated Goodbye! Visit_Again :)

Example 04:

readfileexample.txt
HelloWorld! Programming
3.14 42@test 100%
C++ File-Read Example
Space Tab Separated
Goodbye! Visit_Again :)

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

class FileWordReader {
private:
    ifstream file;

public:
    // constructor that opens the file
    FileWordReader(const string& filename) {
        file.open(filename);
        if (!file.is_open()) {
            cerr << "File not found." << endl;
            exit(1);
        }
    }

    // reads and displays words one by one
    void displayWords() {
        string word;
```

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

    // destructor closes the file automatically
    ~FileWordReader() {
        if (file.is_open()) {
            file.close();
        }
    }
};

int main() {
    FileWordReader reader("readfileexample.txt");
    reader.displayWords();
    return 0;
}
```

Output	HelloWorld! Programming 3.14 42@test 100% C++ File-Read Example Space Tab Separated Goodbye! Visit_Again :)

4.2. Reading Character by Character

Example 05:

readfileexample.txt
"In the middle of difficulty lies opportunity." -Albert_Einstein

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

int main() {
    ifstream inFile("readfileexample.txt");
    char ch;
```

```
    if (inFile.is_open()) {
        while (inFile.get(ch)) {  // reads one character at a time
            cout << ch;
        }
        inFile.close();
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

Output	"In the middle of difficulty lies opportunity." -Albert_Einstein

Example 06:

readfileexample.txt
"In the middle of difficulty lies opportunity." -Albert_Einstein

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

class CharFileReader {
private:
    ifstream file;

public:
    // constructor opens the file
    CharFileReader(const string& filename) {
        file.open(filename);
        if (!file.is_open()) {
            cerr << "Unable to open file." << endl;
            exit(1);
        }
    }

    // read and display file contents character by character
    void displayChars() {
        char ch;
        while (file.get(ch)) {
            cout << ch;
        }
    }
```

```
    // destructor closes the file
    ~CharFileReader() {
        if (file.is_open()) {
            file.close();
        }
    }
};

int main() {
    CharFileReader reader("readfileexample.txt");
    reader.displayChars();
    return 0;
}
```

Output	"In the middle of difficulty lies opportunity." - Albert_Einstein

5. read() and write() Functions

These functions are used for binary file input/output. They work with character arrays (buffers) and are typically used when you want to read or write raw binary data instead of plain text.

1. **write()** Function (for output)
   Syntax:

```
ofstream.write((char*)&variable, sizeof(variable));
```

   - **write()** is a member of **ofstream** or **fstream**.
   - It takes a pointer to a character buffer and the number of bytes to write.
   - Used for writing binary data.

2. **read()** Function (for input)
   Syntax:

```
ifstream.read((char*)&variable, sizeof(variable));
```

   - **read()** is a member of **ifstream** or **fstream**.
   - It reads binary data into a buffer (e.g., a structure or array).

Example 01:

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


class Student {
public:
    int id;
    char name[50];
};

int main() {
    // ---------- writing to file ----------
    Student s1 = {101, "Abdullah Shaikh"};
    ofstream outFile("student.dat", ios::binary);
    outFile.write((char*)&s1, sizeof(s1));
    outFile.close();

    // ---------- reading from file ----------
    Student s2;
    ifstream inFile("student.dat", ios::binary);
    inFile.read((char*)&s2, sizeof(s2));
    inFile.close();

    // ---------- displaying read data ----------
    cout << "Student ID: " << s2.id << endl;
    cout << "Student Name: " << s2.name << endl;

    return 0;
}
```

Output	Student ID: 101 Student Name: Abdullah Shaikh

- **student.dat** is a **binary file**.
- We use **write()** to store structure data in binary format.
- We use **read()** to load that binary data back into another structure.
- **ios::binary** is essential when working with binary files to avoid data corruption.

6. File Pointers

File pointers track positions in a file for read/write operations. C++ uses two pointers:

1. **Get Pointer**: Tracks the **read position** (input operations via **ifstream/fstream**).
2. **Put Pointer**: Tracks the **write position** (output operations via **ofstream/fstream**).

When working with files in C++, file pointers (also called stream pointers) help track the current read/write positions. C++ provides four key functions to manage these positions:

- **seekg()** – Sets the get pointer (input position).
- **tellg()** – Gets the current position of the get pointer.
- **seekp()** – Sets the put pointer (output position).
- **tellp()** – Gets the current position of the put pointer.

	Function	Purpose	Stream Type	Operation
associated with Get Pointer	**seekg()**	Set read position	ifstream, fstream	Read
	tellg()	Get read position		
associated with Put Pointer	**seekp()**	Set write position	ofstream, fstream	Write
	tellp()	Get write position		

These functions are essential for random file access, where we need to read or write data at specific locations rather than sequentially.

6.1. Why Do We Need These Functions?

Scenario 1: Sequential Access vs. Random Access

- **Sequential Access:** Reading/writing data from start to end (like a tape recorder).
- **Random Access**: Jumping to any position in the file (like a CD player).

Scenario 2: Modifying Specific Parts of a File

- **Example:** Updating a record in a database file without rewriting the entire file.
- **Example:** Reading a specific chunk of a large binary file (e.g., extracting a portion of an image).

Scenario 3: Multi-Operation File Handling

- Switching between reading and writing operations requires resetting the file pointer positions.

6.2. seekg() Function

Moves the **input** (**read**) pointer to a specified position.

Syntax:

```
file.seekg(offset, direction);  // relative positioning
file.seekg(position);           // absolute positioning
```

Parameters:

- **offset**: Number of bytes to move (can be negative).
- **direction**:
  - **ios::beg** – From the **beginning** of the file.
  - **ios::cur** – From the **current** position.
  - **ios::end** – From the **end** of the file.

6.3. tellg() Function

Returns the **current** position of the **get** pointer.

Syntax:

```
streampos pos = file.tellg();
```

Example 01:

input.txt
HelloWorldProgrammingInC++

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

int main() {
    ifstream in("input.txt", ios::binary);
    if (!in) {
        cerr << "File not found!";
        return 1;
    }

    // read first 5 bytes (properly null-terminated)
    char buffer[6] = {0};  // initialize with zeros
    in.read(buffer, 5);
```

```
    cout << "First 5 bytes: " << buffer << endl;

    // move get pointer 10 bytes from start
    in.seekg(10, ios::beg);
    streampos pos = in.tellg();
    cout << "Current position: " << pos << endl;

    // read next 5 bytes (using write for safety)
    in.read(buffer, 5);
    cout << "Bytes 10-14: ";
    cout.write(buffer, 5);
    cout << endl;

    in.close();
    return 0;
}
```

Output	First 5 bytes: Hello Current position: 10 Bytes 10-14: Progr

- **ifstream**: Input file stream class
- **ios::binary**: Opens file in binary mode (no **character translation**)
- **Error check**: Returns if file can't be opened
- **buffer[6]** instead of **[5]** - extra space for null terminator
- **= {0}** initializes all elements to zero (includes null terminator)
- **read()** gets **5** bytes, with proper termination
- **seekg(10, ios::beg):** Moves read pointer to 10th byte from start
- **tellg()**: Gets current position (should output 10)

6.4. seekp() Function

Moves the **output (write)** pointer to a specified position.

Syntax:

```
file.seekp(offset, direction);  // relative positioning
file.seekp(position);           // absolute positioning
```

6.5. tellp() Function

Returns the **current position** of the **put** pointer.

Syntax:

```
streampos pos = file.tellp();
```

Example 02:

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

int main() {
    ofstream out("output.txt", ios::binary);
    if (!out) {
        cerr << "File creation failed!";
        return 1;
    }

    // write at start
    out.write("HELLO", 5);
    streampos pos = out.tellp();
    cout << "Position after write: " << pos << endl; // 5

    // move put pointer to 10th byte and write
    out.seekp(10, ios::beg);
    out.write("WORLD", 5); // bytes 10-14: WORLD

    out.close();
    return 0;
}
```

Output	Position after write: 5

output.txt
HELLO WORLD

Example 03:

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

int main() {
    // Open in read/write mode (create if missing)
    fstream file("data.txt", ios::in | ios::out | ios::trunc |
ios::binary);
    if (!file) {
        cerr << "File error!";
        return 1;
    }

    // write data
    string quote = "In the middle of difficulty lies opportunity. -
Albert_Einstein ";
    file.write(quote.c_str(), quote.size());

    // reset to beginning and read first 5 bytes
    file.seekg(0);
    char buffer[6] = {0};
    file.read(buffer, 5);
    cout << "First 5 bytes: " << buffer << endl;

    // overwrite 8th-13th bytes
    file.seekp(7);
    file.write("MIDDLE", 6);

    // read entire file directly
    file.seekg(0);  // rewind to start
    string content;
    getline(file, content, '\0');  // read until null character (entire
file)
    cout << "Final content: " << content << endl;

    file.close();
    return 0;
}
```

Output	First 5 bytes: In th Final content: In the MIDDLE of difficulty lies opportunity. -Albert_Einstein

data.txt
In the MIDDLE of difficulty lies opportunity. -Albert_Einstein

- Opens **data.txt** with these modes:
  - **ios::in**: **Read** mode
  - **ios::out**: **Write** mode
  - **ios::trunc**: **Clear** file if it exists (or **create new**)
  - **ios::binary**: Treat as **binary**
- **file.write(quote.c_str(), quote.size());** writes the quote string to file:
  - **.c_str():** Converts string to C-style char array
  - **.size():** Writes exact length (no null terminator needed)
- **seekg(0):** Moves read pointer to start
- **Reads** "**In th**" (first 5 chars)
- Buffer is null-terminated for safe printing
- **getline(..., '\0')** reads until EOF (treats entire file as one line)
- Output shows the modified text with "**MIDDLE**" replacement

Problem: Student Marks Record System

Question:

Create a simple program to manage student marks. The program should allow:

1. Writing data (roll number, name, marks) of students to a file.
2. Reading and displaying all student records from the file.

The system should work in a way that a user can first input student data, which is saved in a file called **students.txt**, and then retrieve and display that data when needed.

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


class Student {
public:
```

```
    int rollNo;
    string name;
    float marks;
};

void writeToFile() {
    ofstream outFile("students.txt", ios::app); // app to append data

    Student s;
    cout << "Enter Roll No: ";
    cin >> s.rollNo;
    cin.ignore(); // clear newline from buffer
    cout << "Enter Name: ";
    getline(cin, s.name);
    cout << "Enter Marks: ";
    cin >> s.marks;

    if (outFile.is_open()) {
        outFile << s.rollNo << "," << s.name << "," << s.marks << endl;
        outFile.close();
        cout << "Student data saved successfully!" << endl;
    } else {
        cout << "Unable to open file for writing." << endl;
    }
}

void readFromFile() {
    ifstream inFile("students.txt");
    string line;

    cout << "\n--- Student Records ---\n";
    if (inFile.is_open()) {
        while (getline(inFile, line)) {
            cout << line << endl;
        }
        inFile.close();
    } else {
        cout << "Unable to open file for reading.\n";
    }
}

int main() {
    int choice;

    do {
        cout << "\n1. Add Student Record\n";
        cout << "2. View All Records\n";
        cout << "3. Exit\n";
```

```
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1: writeToFile(); break;
            case 2: readFromFile(); break;
            case 3: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n";
        }
    } while (choice != 3);

    return 0;
}
```

Output

```
1. Add Student Record
2. View All Records
3. Exit
Enter your choice: 1
Enter Roll No: 0000
Enter Name: Abdullah
Enter Marks: 100
Student data saved successfully!

1. Add Student Record
2. View All Records
3. Exit
Enter your choice: 2

--- Student Records ---
0,Abdullah,100

1. Add Student Record
2. View All Records
3. Exit
Enter your choice: 3
Exiting...
```

students.txt

```
0,Abdullah,100
```