CHAPTER-8

File Handling and Streams

Stream Classes

- In C++, file streams are used to read from and write to files. To work with files, we use the classes defined in the <fstream> library. These classes allow you to open, read, write, and close files.
- There are three main classes in <fstream>:

1. Ifstream(Input File Stream)

- The ifstream class is used for reading data from files. It is part of the <fstream> library in C++, which provides classes for file input/output operations.
- When you use ifstream, you can open a file, read its contents, and then close it after finishing the operations.

2. ofstream (Output File Stream)

- The ofstream class is used for writing data to files. It is part of the <fstream> library, which provides functionality for file input/output operations.
- When you use ofstream, you can create a new file (or open an existing one) and write data to it. If the file already exists, it will be overwritten by default. If the file doesn't exist, it will be created.

Prepared by Ayush Lamsal ||
layush788@gmail.com

Stream Classes

fstream (File Stream) in C++

- The fstream class in C++ is a combination of both ifstream (for reading from files) and ofstream (for writing to files). It allows you to read from and write to the same file, making it a versatile option for file handling in C++.
- When you use fstream, you can open a file for both input and output operations. It's particularly useful when you want to perform both reading and writing on the same file without needing to use two separate objects.

Note: To perform file processing in C++, the header files <iostream> and <fstream> must be included in our C++ source file.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Example (Here the output is in the same line)

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

int main() {
    ofstream out("output.txt"); // Create and open the file abd store in the object  out

    if (out.is_open()) {
        out << "Hello, this is a out write example";
        out << "Hello, this is a out write example 2";
        out.close(); // Always close the file after use
        cout << "Data written successfully!" ;
    } else {
        cout << "Failed to open the out." << endl;
    }

    return 0;
}
```

```
Hello, this is a out write exampleHello, this is a out write example 2
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

Example (For the multiple line output)

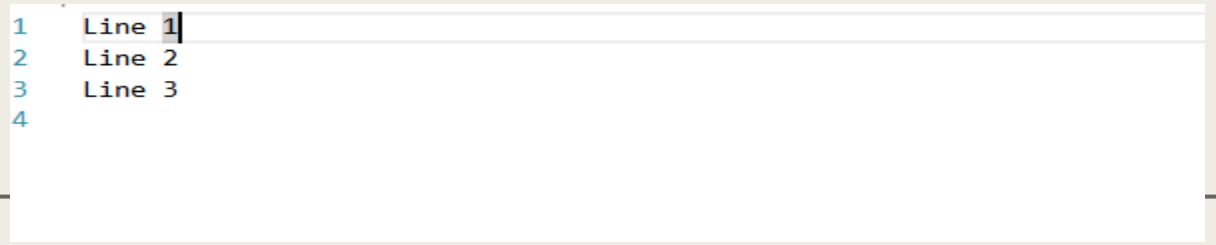
Yes, to write multiple lines to a file in C++, you should use either \n or endl after each line to ensure they appear on separate lines in the file.

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

int main() {
    ofstream file("output.txt");

    if (file.is_open()) {
        file << "Line 1\n";
        file << "Line 2\n";
        file << "Line 3\n";
        file.close();
        cout << "Lines written using \\n!" << endl;
    }

    return 0;
}
```



Prepared by Ayush Lamsal ||
layush788@gmail.com

Contd...

One thing you have noticed in the previous example is that it has overwritten the connect

What if we want to append the content?

Prepared by Ayush Lamsal | |
layush788@gmail.com

Append In File

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

int main() {
    ofstream file("output.txt", ios::app); // Open in append mode

    //ios: input/output strea
    //app: append mode

    if (file.is_open()) {
        file << "New line appended\n";
        file << "Another line appended\n";
        file.close();
        cout << "Data appended successfully!" << endl;
    } else {
        cout << "Failed to open the file." << endl;
    }

    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

Reading from file using ifstream

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

int main() {
    ifstream file("output.txt");  // Open file for reading
    string line;

    if (file.is_open()) {
        while (getline(file, line)) {

            // while (getline(file, line)):
            // ➜ Reads each line from the file
            // ➜ Stores it into the string line
            // ➜ Repeats until end of file (EOF) is reached


            cout << line << endl;  // Print each line
        }
        file.close();  // Close the file
    } else {
        cout << "Unable to open the file." << endl;
    }

    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

For reading a single line

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

int main() {
    ifstream file("output.txt");
    string line;

    if (file.is_open()) {
        if (getline(file, line)) {  // Read only one line
            cout << "First line: " << line << endl;
        } else {
            cout << "File is empty or could not read." << endl;
        }
        file.close();
    } else {
        cout << "Unable to open the file." << endl;
    }

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

For reading a single line

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

int main() {
    ifstream file("output.txt");
    string line;

    if (file.is_open()) {
        if (getline(file, line)) {  // Read only one line
            cout << "First line: " << line << endl;
        } else {
            cout << "File is empty or could not read." << endl;
        }
        file.close();
    } else {
        cout << "Unable to open the file." << endl;
    }

    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Fstream in c++

- fstream is a file stream class that supports both input and output operations on files.
- You can read from and write to the same file using a single fstream object.
- When using fstream in C++, you need to specify the mode you want to open the file with because fstream can handle both reading and writing.

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

int main() {
    fstream file("example.txt", ios::out);  // Open for writing (overwrite)

   // !file means the file stream failed to open or is not ready for input/output operations.
    if (!file) {
        cout << "Failed to open file for writing." << endl;
        return 1;
    }

    // Write some text to the file
    file << "Hello, file!\n";
    file << "This is a simple fstream example.\n";

    file.close();  // Close after writing

    // Now open the same file for reading
    file.open("example.txt", ios::in);
    if (!file) {
        cout << "Failed to open file for reading." << endl;
        return 1;
    }

    string line;
    // Read and print the file line by line
    while (getline(file, line)) {
        cout << line << endl;
    }

    file.close();

    return 0;
}
```

Binary file vs character file

Binary File:

- A binary file stores data in the form of raw bytes (binary format), exactly as it is in memory. It can contain any kind of data — images, audio, video, executable code, or any non-text information. Binary files are not human-readable because they do not represent data as characters.
- Example file types: .exe, .jpg, .mp3, .dat
- Requires opening the file in binary mode (e.g., using ios::binary in C++) to read/write correctly.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Binary file vs character file

Character File

- A character file (or text file) stores data as a sequence of readable characters, usually encoded in ASCII, UTF-8, or similar encoding standards. It contains only text and can be opened and read easily with text editors.
- Example file types: .txt, .csv, .html, .cpp
- Readable and editable by humans.
- Usually opened in text mode by default.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Binary file vs character file

Feature	Binary File	Character (Text) File
Data Format	Stores raw binary data (bytes)	Stores readable characters (text)
Readability	Not human-readable	Human-readable
Content	Images, audio, executables, any data	Only text data
File Size	Usually smaller, more compact	Usually larger due to encoding
File Mode in C++	Open with `ios::binary`	Open in default text mode
Portability	May vary across systems (binary format differences)	More portable across systems
Editing	Needs special software	Can edit with any text editor
Error Detection	Harder to detect and fix errors without special tools	Easier to detect and fix errors manually or with text tools

Prepared by Ayush Lamsal ||
layush788@gmail.com

Console I/O operation

Console I/O operation refers to the operations that **involve taking input from the console** (i.e., from the keyboard) and **displaying outp**ut to the screen.

There are **two types of console I/O operations** in C++:

1. Unformatted I/O operations

It refer to input/output that deals with **raw character data** without any automatic formatting or data type conversion.

It allows reading and writing **exact characters**, including whitespace (spaces, tabs, newlines).

The following are the operation of unformatted console input/output operation

Prepared by Ayush Lamsal | |
layush788@gmail.com

Unformatted I/O operations

1. get():

- get() is a method of the cin object used to input a single character from the keyboard.
- But its main property is that it allows wide space and new line character also

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

int main() {
    char ch;
    cout << "Enter any character: ";
    ch = cin.get();  // Reads one character including space or newline
    cout << "You entered: ";
    cout.put(ch);
    cout << endl;
    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

```
Enter any character: abcdefghijkl
You entered: a
PS C:\Users\ayush\OneDrive\Desktop\oop> cd "c:\Users\ayush\OneDrive\Desktop\oop\" ; if ($?) { g++ if_Strea
Enter any character:  abcdefghijkl
You entered:
```

Unformatted I/O operations

2. Put():

put() is a method of the cout object used to print a specified single character on the screen (monitor).

Syntax

```
cout.put(character);
```

Example

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

int main() {
    char ch = 'X';
    cout.put(ch);    // Prints 'X' on the screen
    cout.put('\n');  // Prints a newline
    return 0;
}
```

Unformatted I/O operations

3. getline():

getline is a method of the cin object used to input a string that can contain multiple characters, including spaces.

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

int main() {
    char name[100];
    cout << "Enter your full name: ";
    cin.getline(name, 100);  // Reads a line including spaces into char array
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

Unformatted I/O operations

4. Write():

The write() method is a member function of output stream objects such as cout (which is of type ostream).

It is used to output a specified number of characters from a given character array (C-style string) directly to the output stream without any formatting or interpretation.

Note: The write() method outputs a specified number of characters from a character array (C-style string) directly to the console or output stream without any formatting.

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

int main() {
    char ch[] = "Hello, World!";
    cout.write(ch, 8);  // Prints first 8 characrers
    cout.put('\n');     // Prints a newline
    return 0;
}
```

Prepared by Ayush Lamsal | |
layush788@gmail.com

Unformatted I/O operations

5. Cin:

cin is used to take input from the console and store it in a variable.

```
cin >> variable;
```

- `cin` : The standard input stream object.
- `>>` : Extraction operator used to take input.
- `variable` : The variable where input is stored.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Unformatted I/O operations

6. Cout:

cout is a method used to print variables or strings to the console.

```
cout << variable_or_string;
```

- `cout` is the output stream object.
- `<<` is the insertion operator.
- `variable_or_string` is the data you want to print.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Formatted I/O

- in C++, formatted I/O means showing the output in a clean, organized, and easy-to-read way. It is important in professional programs where the output should be properly aligned and well-structured.
- C++ provides support for this through the <iomanip> header file, which includes Input/Output manipulators to help format the output neatly.
- Here's a list of different functions provided in the <iomanip> header file in C++

Prepared by Ayush Lamsal ||
layush788@gmail.com

Formatted I/O

Function	Description
`setw(n)`	Sets the **width** of the output field to `n` characters.
`setfill(char)`	Fills extra space with the specified **character**.
`setprecision(n)`	Sets the number of digits after the **decimal point**.
`setf(arg1, arg2)`	Sets a formatting **flag** (e.g., fixed, scientific).
`unsetf(arg2)`	**Removes** a specific formatting flag.
`setbase(arg)`	Sets the **number base**: 10 for decimal, 8 for octal, 16 for hex.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Formatted I/O

- Setw():

It specifies how many spaces wide the printed value should be. If the actual output is smaller, it will add extra spaces (or a fill character) to make it fit that width. This helps align output neatly in columns or tables.

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

int main() {
    int num = 42;

    cout << "Without setw: " << num << endl;

    cout << "With setw(5): " << setw(5) << num << endl;

    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

```
Without setw: 4222
With setw(5):  4222
```

Formatted I/O

setfill(char):

- setfill(char) is used to specify the character that fills the extra space when you use setw().
- By default, setw() fills empty spaces with spaces ' '. Using setfill(), you can change this to any character you want, like *, -, or 0.

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

int main() {
    int num = 42;

    cout<<num<<endl;

    cout << setw(5) << setfill('*') << num << endl;

    return 0;
}
```

```
42
***42
```

Formatted I/O

Setprecision()

- It is a manipulator used to control the number of digits displayed when printing floating-point numbers (float, double).
- By default, setprecision(n) sets the total number of significant digits shown, including digits before and after the decimal point.
- When used together with the manipulator fixed, setprecision(n) sets the number of digits to show after the decimal point only.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Formatted I/O

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

int main() {
    double number = 123.456789;

    // Without fixed: setprecision(n) sets total significant digits (before + after decimal)
    cout << "Default precision (5): " << setprecision(5) << number << endl;

    // With fixed: setprecision(n) sets digits only after the decimal point
    cout << "Fixed precision (5): " << fixed << setprecision(5) << number << endl;

    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

Formatted I/O

Setprecision()

- It is a manipulator used to control the number of digits displayed when printing floating-point numbers (float, double).
- By default, setprecision(n) sets the total number of significant digits shown, including digits before and after the decimal point.
- When used together with the manipulator fixed, setprecision(n) sets the number of digits to show after the decimal point only.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Formatted I/O

setf

- It is used to turn on special formatting options for output in C++.
- It tells the computer how to show numbers or text — like aligning text left or right, or showing numbers in fixed or scientific form.
- It helps make the output look neat and clear.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Formatted I/O

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

int main() {
    int num = 123;

    // Set alignment to left
    //adjust field means set allgnement
    cout.setf(ios::left, ios::adjustfield);
    cout << setw(10) << num << "END" << endl;

    // Set alignment to right
    cout.setf(ios::right, ios::adjustfield);
    cout << setw(10) << num << "END" << endl;

    return 0;
}
```

Prepared by Ayush Lamsal ||
layush788@gmail.com

Formatted I/O

- Example of setf and unsetf

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

int main() {
    int num = 123;

    // Set left alignment
    cout.setf(ios::left, ios::adjustfield);
    cout << setw(10) << num << "END" << endl;  // Output: 123       END

    // Remove left alignment (reset alignment)
    cout.unsetf(ios::adjustfield);            // Unset any existing alignment

    // Set right alignment
    cout.setf(ios::right, ios::adjustfield);
    cout << setw(10) << num << "END" << endl;  // Output:       123END

    return 0;
}
```

Opening and closing a file

We have already covered this topic

Prepared by Ayush Lamsal | |
layush788@gmail.com

List of file mode in c++

We have already covered this topic

File Mode	Meaning
`ios::in`	Open file for **reading**
`ios::out`	Open file for **writing**
`ios::app`	Open file in **append** mode (add at the end)
`ios::ate`	Open file and move pointer to the **end**
`ios::trunc`	**Truncate** the file (clear contents if exists)

Prepared by Ayush Lamsal | |
layush788@gmail.com

Assignment

Submit examples of each file mode by today to earn 2 additional practical marks.

The submission deadline is before 8 PM today

Prepared by Ayush Lamsal | |
layush788@gmail.com

Example

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

int main() {
    // Open existing file in read-write mode and start at the end
    fstream file("output.txt", ios::in | ios::out | ios::ate);

    if (!file) {
        cout << "Cannot open file." << endl;
        return 1;
    }

    // Write new text at the end of the file
    file << "Adding this line using ios::ate.\n";

    // Move pointer to the beginning to read entire file content
    file.seekg(0);

    cout << "File content after appending:\n";
    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }

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

layush788@gmail.com

Example

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

int main() {
    // Open existing file in read-write mode and start at the end
    fstream file("output.txt",ios::ate);

    if (!file) {
        cout << "Cannot open file." << endl;
        return 1;
    }

    // Write new text at the end of the file
    file << "Adding this line using ios::ate.\n";

    // Move pointer to specific poistion
    file.seekg(20);

    cout << "File content after appending:\n";
    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }

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

C++ error handling during file operation

It is quite common that errors may occur during file operations. There may be different reasons for errors arising while working with files.

The following are common problems that lead to errors during file operations:

Common Problems Leading to Errors During File Operations

1. **Trying to open a file for reading that does not exist.**
   The program will fail if it attempts to read a file that is not present.
2. **Trying to read beyond the total number of characters in a file.**
   Reading past the end of the file causes errors or unexpected behavior.
3. **Performing a read operation on a file opened only for writing, or a write operation on a file opened only for reading.**
   File mode restrictions can cause errors when using unsupported operations.
4. **Trying to perform operations on a file that has not been opened.**
   Any attempt to read from or write to a file that isn't open will result in an error.

Prepared by Ayush Lamsal | |
layush788@gmail.com

Error Handling Functions in C++ File Streams

When working with file streams in C++, it is important to check the state of the stream to handle errors properly. The following functions help detect the status of a stream:

1. `bad()`
   - Returns **true** if a **serious error** occurs, such as a hardware failure or corrupted file.
   - When `bad()` is true, the stream is typically unusable.
2. `fail()`
   - Returns **true** if an input/output operation **fails**.
   - This includes format errors, trying to read/write in the wrong mode, or `bad()` errors.
   - It indicates the stream is in a failed state.
3. `eof()` (End Of File)
   - Returns **true** if the **end of the file** has been reached during reading.
   - This means no more data is available to read.
4. `good()`
   - Returns **true** if the stream is in a **good state** and ready for I/O operations.
   - No errors or end-of-file conditions are present.

Prepared by Ayush Lamsal ||
layush788@gmail.com

true means 1
false means 0

If bad() is true , the error is irrecoverable and we stop
If bad() is false, it means no serious hardware error happened.
But there could still be other errors, like:

I. Wrong data type (fail() is true)
II. End of file (eof() is true)

eof Example(Extra)

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

int main() {
    ifstream file("example.txt");  // Open file

    if (!file) {
        cout << "File not found!" << endl;
        return 1;
    }

    char ch;
    while (!file.eof()) {
        file.get(ch);
        cout << ch;
    }

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

Note: get() reads one character, getline() reads a full line.

Prepared by Ayush Lamsal ||
layush788@gmail.com

Example

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

int main() {
    ifstream file("output.txt");  // Open file

    if (!file) {
        cout << "Error: Cannot open file.\n";
        return 1;
    }

    int num;
    file >> num;  // Try to read an integer

    // Check error states
    cout << "bad(): " << file.bad() << " (serious I/O error)\n";
    cout << "fail(): " << file.fail() << " (logical error or extraction failed)\n";
    cout << "eof(): " << file.eof() << " (end of file reached)\n";
    cout << "good(): " << file.good() << " (no errors, stream is good)\n";

    if (!file.fail()) {
        cout << "Number read from file: " << num << endl;
    } else {
        cout << "Failed to read a valid integer.\n";
    }

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

full line.

The End

Prepared by Ayush Lamsal ||
layush788@gmail.com