



# Himalaya College of Engineering

## Advanced C++ Programming Lab Report

### Lab 9: STREAM COMPUTATION

**Prepared By:** Ankit Belbase(HCE081BEI007)

**Subject:** Object-Oriented Programming (OOP)

**Program:** Bachelor of Electronics Engineering

**Institution:** Himalaya College of Engineering

**Date:** July 27,2025

## Lab 9: Stream computation

### Objective:

- To understand the file streams in C++.
- To perform file operations such as reading, writing, and appending data.
- To demonstrate usage of ifstream, ofstream, and fstream classes.

### Theory:

#### Stream computation

Stream computation refers to processing data as it is received using input/output streams. Cin, Cout, ifstream, ofstream, etc. are used in stream computation.

#### Introduction to file handling

File handling in C++ allows programs to store and retrieve data from files using stream classes. Files provide persistent storage for data, enabling input/output operations beyond console interaction.

File stream classes:

- ifstream- used for reading data from files (input file stream).
- ofstream- used for writing data to files (output file stream).
- fstream- used for both reading and writing operations.

#### Opening and closing file

files are opened using the .open() function or constructor, and must be closed using .close() to free resources. File modes (ios::in- open file for reading, ios::out- open files for writing, ios::app- open files for appending at the end, ios::binary- open file in binary mode) control operations.

Basic syntax is:

```
#include <fstream>
```

```
ifstream inFile;    // for reading
```

```
ofstream outFile;   // for writing
```

```
fstream file;       // for both
```

```
inFile.open("input.txt");
```

```
outFile.open("output.txt");
```

```
file.open("data.txt", ios::in | ios::out);
```

```
inFile.close();
```

```
outFile.close();
```

```
file.close();
```

## **Manipulators**

Manipulators are special functions used with cout or ofstream to format the output

endl- move to the next line

setw(n)- set width of the output field

setprecision(n)- set number of digits after decimal

## **Insertion and extraction operator**

Insertion operator is used to write output to screen or file. It works with cout or ofstream.

Eg: cout<<"value is :"<<x;

Extraction operator is used to read input. It works with Cin or ifstream.

Eg: int x;

```
cin>>x;
```

## **Random access and sequential access**

Random access directly jumps to any location in the file like a DVD. You can go to any scene instantly. It uses file pointers (seekg, seekp, tellg, tellp)

Sequential access means reading, or writing data in order, one after another, from the beginning to the end without skipping.

## **Testing errors during file operations with file stream state**

You can use the state flags of file stream objects (std::ifstream, std::ofstream, std::fstream)

to check for errors during file operations. Common state flags include failbit, badbit,

eofbit, and goodbit.

Syntax for Testing Errors:

```
If(!file_stream_object ) {
```

```
// Handle error
```

```
}
```

## **File input/ output with member function**

File Input/Output (I/O) with member functions in Object-Oriented Programming allows file operations, such as reading and writing data, to be handled directly within a class, ensuring encapsulation and abstraction. This approach ties data handling to the object itself, promoting modular and maintainable design, especially for applications requiring persistent storage, like student or employee record systems. It reduces redundant procedural code and supports scalable systems but can be complex for beginners due to stream management and error handling. Common uses include object serialization, file-based logging, and data-intensive applications.

## **Stream state flags in C++**

In C++, file streams use internal status flags to indicate their current state during I/O operations. The goodbit flag shows that the stream is functioning correctly, checked with `good()`. The eofbit flag is set when the end of file is reached, detected using `eof()`. The failbit flag indicates a logical error, such as format mismatch or failure to open a file, checked with `fail()`. The badbit flag signals irrecoverable errors like hardware failure, identified with `bad()`. These flags should be checked after file operations to ensure proper handling of errors and maintain program stability.

## LAB QUESTIONS:

Ques 1 : Write a program in C++ to open an existing text file info.txt, count the number of words and lines, and display the results.

```
#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

    ifstream fin("info.txt");

    if (!fin) {

        cerr << "Error: Could not open file 'info.txt'" << endl;

        return 1;

    }

    string line;

    int lineCount = 0;

    int wordCount = 0;

    while (getline(fin, line)) {

        lineCount++;

        int length = line.length();

        bool inWord = false;

        for (int i = 0; i < length; i++) {

            if (isspace(line[i])) {

                inWord = false;

            } else if (!inWord) {

                wordCount++;

                inWord = true;

            }

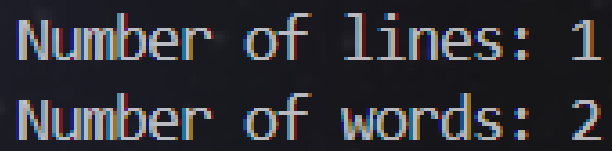
        }

    }

}
```

```
}  
}  
}  
fin.close();  
cout << "Number of lines: " << lineCount << endl;  
cout << "Number of words: " << wordCount << endl;  
return 0;  
}
```

Output:



```
Number of lines: 1  
Number of words: 2
```

Ques 2 : Write a program to make simple library management system of a college. Your program should store and retrieve the information(Book Name, Book ID, Number of books and purchase date). [2076 Chaitra]

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

class Book {
private:
    int bookID;
    char bookName[50];
    int numBooks;
    char purchaseDate[15];

public:
    void input() {
        cout << "Enter Book ID: ";
        cin >> bookID;
        cin.ignore();
        cout << "Enter Book Name: ";
        cin.getline(bookName, 50);
        cout << "Enter Number of Books: ";
        cin >> numBooks;
        cin.ignore();
        cout << "Enter Purchase Date (YYYY-MM-DD): ";
        cin.getline(purchaseDate, 15);
    }

    void display() const {
        cout << "Book ID: " << bookID << "\n"
            << "Book Name: " << bookName << "\n"
            << "Number of Books: " << numBooks << "\n"
            << "Purchase Date: " << purchaseDate << "\n"
            << "-----\n";
    }

    void writeToFile(ofstream& fout) {
        fout.write(reinterpret_cast<char*>(this), sizeof(*this));
    }

    void readFromFile(ifstream& fin) {
        fin.read(reinterpret_cast<char*>(this), sizeof(*this));
    }
}
```

```

};

int main() {
    int n;
    cout << "Enter number of books to add: ";
    cin >> n;
    cin.ignore();

    ofstream fout("library.dat", ios::binary | ios::app);
    if (!fout) {
        cerr << "Error opening file for writing!\n";
        return 1;
    }

    for (int i = 0; i < n; i++) {
        cout << "\nEnter details for book " << (i + 1) << ":\n";
        Book b;
        b.input();
        b.writeToFile(fout);
    }
    fout.close();

    ifstream fin("library.dat", ios::binary);
    if (!fin) {
        cerr << "Error opening file for reading!\n";
        return 1;
    }

    cout << "\n---- Library Book Records ----\n";
    Book b;
    while (fin.read(reinterpret_cast<char*>(&b), sizeof(b))) {
        b.display();
    }
    fin.close();
    return 0;
}

```



## OUTPUT:

```
Enter number of books to add: 2

Enter details for book 1:
Enter Book ID: 123
Enter Book Name: HAri
Enter Number of Books: 1
Enter Purchase Date (YYYY-MM-DD): 2020-43-43

Enter details for book 2:
Enter Book ID: 345
Enter Book Name: Ram
Enter Number of Books: 1
Enter Purchase Date (YYYY-MM-DD): 3211-43-43

---- Library Book Records ----
Book ID: 123
Book Name: HAri
Number of Books: 1
Purchase Date: 2020-43-43
-----
Book ID: 345
Book Name: Ram
Number of Books: 1
Purchase Date: 3211-43-43
-----
```

## DISCUSSION:

Stream computation processes data as a continuous flow rather than all at once. In C++, streams like `cin`, `cout`, `ifstream`, and `ofstream` are used for input and output operations. They make reading and writing data easier, more efficient, and suitable for real-time or large data processing.

## CONCLUSION:

Stream computation simplifies and optimizes data handling by processing information in a flow-oriented manner. In C++, stream objects provide a flexible, safe, and powerful way to perform input/output operations. It enhances program performance and is especially useful for real-time and file-based applications.