TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

CHYASAL, LALITPUR

Lab Report No: -09

Title: - Steam Computation

Submitted by: -

Name: Aastha Gaire

Roll NO: - HCE081BEI003

Submitted To: -

Department of Electronics and Computer Engineering

Checked by: -

Date of submission: - 2082/04/11

OBJECTIVE

- To comprehend the C++ idea of streams.
- To implement input and output operations using stream classes.
- To become familiar with reading from and writing to files using file streams.
- To distinguish between fstream, ifstream/ofstream, and cin/cout.

THEORY

In C++, stream computation refers to the process of performing input and output (I/O) operations using streams. A stream is an abstraction representing a continuous flow of data between a program and external sources or destinations, such as the keyboard, console, or files. Streams provide a uniform, device-independent, and object-oriented interface to handle data transfer. This abstraction hides the complexities of device-specific operations, allowing programmers to read from or write to different devices seamlessly.

Conceptually, a stream is a sequence of bytes flowing into or out of a program. Input streams transfer data into the program (for example, cin reads input from the keyboard).Output streams transfer data out of the program (for example, cout writes output to the console).Besides console I/O, streams can also be used to read from or write to files or memory buffers.

Why to Use Streams?

1. Simplifies I/O operations: Streams allow programmers to use simple syntax (>> and <<) for input and output.

2. Device independence: The same stream operations work with different data sources/destinations (console, files, memory).

3. Supports formatted and unformatted data: Enables flexible data handling (numbers, text, binary).

4. Error handling: Streams provide built-in mechanisms to detect and manage errors during I/O.

5. Extensibility: Stream classes can be extended for custom devices or data formats**.**

Types of Streams in C++

- **Standard I/O Streams**:

    a. cin → standard input (keyboard)

b. cout → standard output (console)
c. cerr → standard error (unbuffered)
d. clog → standard error (buffered)

- **File I/O Streams**:

1. ifstream → input file stream (for reading from files)
2. ofstream → output file stream (for writing to files)
3. fstream → input/output file stream (for both reading and writing)

2. Input/Output Using cin and cout.

Example:

```
int age;
cout << "Enter your age: ";
cin >> age;
cout << "You are " << age << " years old.";
```

3. File Handling in C++

File streams are part of the <fstream> header in C++. They allow programs to store and retrieve data from disk files, which is essential for data persistence.

- **Opening a File**:

  Files can be opened using constructors or the open() function.

```
ofstream fileOut("data.txt");
ifstream fileIn("data.txt");
```

- **Writing to a File**:

```
ofstream file("example.txt");
file << "Hello, file!";
file.close();
```

- **Reading from a File**:

```
ifstream file("example.txt");
string line;
```

```
while(getline(file, line)) {
    cout << line << endl;
}
file.close();
```

- **Checking File Status**:
  Always verify that the file has been opened successfully using .is_open() method or by checking the stream object.

4. **File Modes**

Different modes are used to open a file:

- ios::in – open for reading
- ios::out – open for writing
- ios::app – append mode
- ios::trunc – truncate file if exists
- ios::binary – binary mode

Example:

```
fstream file("file.txt", ios::in | ios::out | ios::app);
```

Advantages of Using File Streams

- Enables persistent data storage.
- Allows large amounts of data to be processed without overloading memory.
- Provides a mechanism for reading and writing data in a structured way.

LAB PROGRAMS

1) Write a program to store and retrieve 'n' records of items (item_ID, name, price, mfd_date, company) in Inventory system.

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
using namespace std;
struct Item {
    int item_ID;
    string name;
    double price;
    string mfd_date;
    string company;
};
int main() {
    int n;
    cout << "Enter number of items: ";
    cin >> n;
    cin.ignore();
    vector<Item> inventory;
    for (int i = 0; i < n; ++i) {
        Item temp;
        cout << "\nEnter details for item " << i + 1 << ":\n";
        cout << "Item ID: ";
        cin >> temp.item_ID;
        cin.ignore();
    cout << "Name: ";
        getline(cin, temp.name);
    cout << "Price: ";
        cin >> temp.price;
        cin.ignore();
    cout << "Manufacturing Date (YYYY-MM-DD): ";
        getline(cin, temp.mfd_date);
    cout << "Company: ";
        getline(cin, temp.company);
    inventory.push_back(temp);
    }
    ofstream fout("inventory.txt");
    if (!fout) {
      cout << "Error opening file for writing.\n";
        return 1;
    }
    for (const auto& item : inventory) {
        fout << item.item_ID << "," << item.name << "," << item.price << "," << item.mfd_date << "," << item.company << endl;
    }
    fout.close();

    cout << "\n--- Inventory Records ---\n";
    for (const auto& item : inventory) {
        cout << "Item ID: " << item.item_ID << endl;
        cout << "Name: " << item.name << endl;
        cout << "Price: $" << item.price << endl;
        cout << "Manufacturing Date: " << item.mfd_date << endl;
        cout << "Company: " << item.company << endl;
        cout << "--------------------------\n";
    }
    cout << "Inventory has been saved to inventory.txt\n";
        return 0;
}
```

```
Enter number of items: 1

Enter details for item 1:
Item ID: 14
Name: Earbuds
Price: 24999
Manufacturing Date (YYYY-MM-DD): 2024-08-15
Company: Samsung

--- Inventory Records ---
Item ID: 14
Name: Earbuds
Price: $24999
Manufacturing Date: 2024-08-15
Company: Samsung
--------------------------
Inventory has been saved to inventory.txt
```

2) Write a program to write the information of students in a file. And also display their details in console.

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

struct Student {
 int roll;
 string name;
  int age;
 string course;
};

int main() {
   int n;
cout << "Enter number of students:";
cin >> n;
 cin.ignore();
 vector<Student> students;
  for (int i = 0; i < n; ++i) {
  Student s;
cout << "\nEnter details for student " << i + 1 << ":\n";
     cout << "Roll number: ";
        cin >> s.roll;
        cin.ignore();
        cout << "Name: ";
        getline(cin, s.name);
        cout << "Age: ";
        cin >> s.age;
        cin.ignore();
        cout << "Course: ";
        getline(cin, s.course);
```

```
    students.push_back(s);
}

ofstream fout("students.txt");
if (!fout) {
    cout << "Error opening file for writing.\n";
    return 1;
}

for (const auto& s : students) {
    fout << s.roll << "," << s.name << "," << s.age << "," << s.course << endl;
}
fout.close();

cout << "\n—— Student Details ——\n";
for (const auto& s : students) {
    cout << "Roll: " << s.roll << endl;
    cout << "Name: " << s.name << endl;
    cout << "Age: " << s.age << endl;
    cout << "Course: " << s.course << endl;
    cout << "----------------------------\n";
}

return 0;
```

```
Enter number of students: 1

Enter details for student 1:
Roll number: 003
Name: Aastha Gaire
Age: 20
Course: BEI

ûûû Student Details ûûû
Roll: 3
Name: Aastha Gaire
Age: 20
Course: BEI
----------------------------
```

}

Discussion:

Stream computation in C++ is a core concept for handling input and output operations, whether through the console or files. In this lab, we explored both standard input/output using cin and cout, as well as file operations using ifstream, ofstream, and fstream. These allowed us to understand how data is read into and written out of a C++ program.

An important lesson was the need for proper file handling—such as verifying if a file is available before accessing it, and ensuring it is properly closed after use. We also learned about different file modes that influence how data is processed, whether it's being read, written, or appended. Overall, the lab helped us grasp how streams function and how essential they are in building effective C++ programs.

Conclusion:

The stream computation lab in C++ helped us build a strong foundation in handling input and output processes within a program. We explored how cin and cout are used for user interaction, and how file streams facilitate reading and writing data to files. Utilizing stream classes from the <iostream> and <fstream> libraries made these operations more efficient and adaptable.

We also learned the significance of checking whether files open correctly, managing errors effectively, and choosing the correct file modes based on the task. Understanding these stream operations is essential for creating practical applications that require reliable data input, output, and storage.