💾 I/O and File Management in C++: Concept of Streams

When you write a C++ program, you often need to **input data** (from the user or a file) and **output data** (to the screen or a file).
This process is called **I/O** — which stands for **Input/Output**.

A key part of this process in C++ is understanding something called **Streams**.

🌊 What is a Stream?

A **stream** in C++ is like a flow of data — imagine water flowing through a pipe.

- Data flows from a **source** (like a file or keyboard) to a **destination** (like the screen or another file).
- Your program can read from or write to this flow, without worrying about where the data physically lives.

🧠 Why Use Streams in C++?

- To handle data smoothly and continuously.
- To simplify reading and writing operations.
- To avoid loading large files into memory all at once.

Streams are part of the **iostream** library in C++, which makes I/O easy to handle!

🔥 Types of Streams in C++

1️⃣ Input Stream

- Used to **read data into** your program.
- Example: Taking input from the **keyboard** or reading from a **file**.
- Object examples: `cin`, `ifstream`.

2️⃣ Output Stream

- Used to **write data out** of your program.
- Example: Printing on the **screen** or writing into a **file**.
- Object examples: `cout`, `ofstream`.

📂 Example: File Reading and Writing in C++

1️⃣ Reading from a File (Input Stream)

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

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

    if (file.is_open()) {
        while (getline(file, content)) {
            cout << content << endl;  // Print the content
        }
        file.close();  // Close the stream
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

In this example:

- The `ifstream` object `file` acts as an **input stream**.
- It reads data from `example.txt` and sends it into the program.

2️⃣ Writing to a File (Output Stream)

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

int main() {
    ofstream file("output.txt");  // Open file for writing

    if (file.is_open()) {
        file << "Hello, world!";  // Write data to the file
        file.close();             // Close the stream
        cout << "Data written to file!" << endl;
    } else {
        cout << "Unable to open file." << endl;
    }

    return 0;
}
```

In this example:

- The `ofstream` object `file` acts as an **output stream**.
- It sends data from the program into `output.txt`.

💡 Real-life Examples of Streams in C++

Source	Stream Type	Destination	Example Use Case
Keyboard	Input	Program	Taking user input with `cin`.
Program	Output	Screen	Displaying messages using `cout`.
File (`.txt`)	Input	Program	Reading data using `ifstream`.
Program	Output	File	Writing logs using `ofstream`.
Microphone	Input	Program	(Advanced — Audio Input)
Program	Output	Speaker	(Advanced — Playing Sound)

📌 Summary

- Streams manage the **flow of data** for input and output in C++.
- They allow your program to **read** from sources or **write** to destinations.
- Streams make C++ programs more flexible and memory-efficient — especially when handling **files** or **continuous input/output**.

💡 C++ I/O: `cin` and `cout` Objects & Stream Classes

When you write C++ programs, you often want to:

- **Get input** from the user (like numbers or text).
- **Show output** to the user (like results or messages).

In C++, this is handled using **I/O streams** (Input/Output streams). Two of the most important stream objects for beginners are:

◇ `cin` — Standard Input Stream

- `cin` stands for **Character Input**.
- Used to **take input from the user** (usually from the keyboard).
- Belongs to the **istream class** (input stream class).

👉 **Example:**

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

int main() {
    int age;
    cout << "Enter your age: ";    // Output message
    cin >> age;                   // Input from user
    cout << "You are " << age << " years old!" << endl;
    return 0;
}
```

☑ Here:

- `cout` displays a message.
- `cin` waits for the user to enter their age and stores it in the `age` variable.

◇ `cout` — Standard Output Stream

- `cout` stands for **Character Output**.
- Used to **display output on the screen**.
- Belongs to the **ostream class** (output stream class).

👉 **Example:**

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

int main() {
    cout << "Hello, World!" << endl;  // Output message
    return 0;
}
```

☑ Here: `cout` sends the text to the **console**. `endl` moves the cursor to a new line.

🧠 C++ Stream Classes Overview

C++ uses **stream classes** to handle all input and output. Here's a simple list:

Class Name	Purpose	Example Object
`istream`	Handles **input** operations	`cin`
`ostream`	Handles **output** operations	`cout`
`ifstream`	Reads data from **files**	`file` (input file)
`ofstream`	Writes data to **files**	`file` (output file)
`fstream`	Used for both **input and output**	`file`

All these classes are part of the `<iostream>` or `<fstream>` libraries in C++.

🔥 How `cin` and `cout` Work Behind the Scenes

- `cin` uses the **>> (extraction operator)** to extract data from the input stream.
- `cout` uses the **<< (insertion operator)** to insert data into the output stream.

🎯 Summary

- `cin` = Used to **take input** from the user (keyboard) — comes from the `istream` class.
- `cout` = Used to **print output** to the screen — comes from the `ostream` class.
- Stream classes handle the **flow of data** in C++, whether it's from the keyboard, screen, or file.
- Using streams makes your program **simple, readable, and efficient**.

💻 Unformatted and Formatted I/O, Manipulators in C++

✸ What is I/O?

In C++, **I/O** (Input/Output) means interacting with the user or external data sources:

- **Input** = Taking data from the user.
- **Output** = Showing data to the user.

Now let's break this into two styles:

◇ Unformatted I/O

Unformatted I/O means reading or displaying data **exactly as it is** — no special format, no alignment, just plain input or output.

In C++, the simplest unformatted I/O uses:

- `cin` for input
- `cout` for output

👉 Example:

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

int main() {
    string name;
    cout << "Enter your name: ";    // Output
    cin >> name;                    // Input
    cout << "Hello " << name << endl;  // Output
    return 0;
}
```

⚡ Note:
If you enter **"John Doe"**, only **"John"** will be read — because `cin` stops at spaces.
This is why it's called **unformatted** — it's simple but not flexible.

◇ Formatted I/O

Formatted I/O lets you control **how data looks** when printed or read. You can set:

- **Alignment** (left, right)
- **Precision** (for decimals)
- **Width of space** for display
- **Fill characters** (like `0` or `*`)

👉 Example:

```
#include <iostream>
#include <iomanip>   // For manipulators like setw, setprecision
using namespace std;

int main() {
    double price = 45.6789;
    cout << "Price without formatting: " << price << endl;
    cout << "Price with formatting: " << fixed << setprecision(2) << price <<
endl;
    return 0;
}
```

🧠 Output:

```
Price without formatting: 45.6789
Price with formatting: 45.68
```

Here, `setprecision(2)` ensures the number has only **2 decimal places** — neat and tidy!

🔧 Manipulators in C++

Manipulators are special functions used with streams (`cin` / `cout`) to control how input and output appear.

Some popular manipulators:

Manipulator	Purpose	Example Usage
`setw()`	Sets the **width** of the output field.	`cout << setw(10) << num;`
`setprecision()`	Sets **decimal precision** for floating numbers.	`cout << setprecision(3) << pi;`
`setfill()`	Fills empty spaces with a **custom character**.	`cout << setfill('*') << setw(5) << 7;`
`fixed`	Prints floating numbers in **fixed-point** notation.	`cout << fixed;`
`scientific`	Prints floating numbers in **scientific** format.	`cout << scientific;`
`left` / `right`	Aligns output **left or right** within the field.	`cout << left << setw(10) << "Hi";`

👉 Example of Multiple Manipulators:

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

int main() {
    int num = 7;
    cout << "Without setw: " << num << endl;
    cout << "With setw(5): " << setw(5) << num << endl;
    cout << "With setw + setfill('*'): " << setfill('*') << setw(5) << num << endl;
    return 0;
}
```

😵 Output:

```
Without setw: 7
With setw(5):     7
With setw + setfill('*'): ****7
```

Summary

Term	Meaning
Unformatted I/O	Simple input/output without styling.
Formatted I/O	Output/input with control over how data looks (width, precision, fill).
Manipulators	Special functions to help format the data neatly when using `cin` and `cout`.

File Stream & File Stream Classes in C++

What is File Handling?

In real-world programs, you often want to **save data** or **load data** even after the program closes — like:

- Saving game progress.
- Storing user information.
- Loading data from a text file.

In C++, this is done using **File Streams**!

What is a File Stream?

A **File Stream** is like a pipe that lets your program **read from** or **write to** files on your computer.

- Data flows from a file **into your program** (input stream).
- Or from your program **into a file** (output stream).

Think of it like water flowing through a hose — files are the buckets, your program is the tap.

🗁 C++ File Stream Classes

C++ provides three main classes for file handling, all defined in:

Class	Purpose	Example Use
`ifstream`	Input File Stream — for **reading** from files.	Load data from a file.
`ofstream`	Output File Stream — for **writing** to files.	Save data to a file.
`fstream`	File Stream — for both **reading & writing**.	Read and write together.

🗎 Example 1: Writing to a File (ofstream)

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

int main() {
    ofstream outFile("example.txt");  // Create and open a file
    outFile << "Hello, File World!";  // Write to the file
    outFile.close();                 // Close the file
    return 0;
}
```

☑ **Explanation:**
This creates a file called `example.txt` and writes `Hello, File World!` into it.

🗎 Example 2: Reading from a File (ifstream)

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

int main() {
    ifstream inFile("example.txt");  // Open file for reading
    string content;
    getline(inFile, content);        // Read one line from the file
    cout << "File says: " << content << endl;
    inFile.close();                  // Close the file
    return 0;
}
```

☑ **Explanation:**
This reads the content from `example.txt` and prints it on the screen.

Example 3: Read & Write (fstream)

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

int main() {
    fstream file("data.txt", ios::out);  // Open for writing
    file << "C++ makes file handling easy!";
    file.close();                        // Close after writing

    file.open("data.txt", ios::in);      // Open for reading
    string text;
    getline(file, text);
    cout << "Stored Text: " << text << endl;
    file.close();                        // Close the file
    return 0;
}
```

Explanation:
This uses `fstream` to first write and then read the same file.

Summary

Term	Meaning
File Stream	A way to handle input/output operations with files.
ifstream	Reads data from a file (Input).
ofstream	Writes data to a file (Output).
fstream	Can both read from and write to files.

Why Use File Streams?

- To store data permanently.
- To load saved data into your program.
- To read and write large data efficiently.

💾 File Management Functions in C++

When you work with files in C++, you don't just read and write — you also manage the file! C++ gives you some simple functions to handle this.

⚙️ Common File Management Functions

Function	Purpose
`open()`	Opens a file for reading or writing.
`close()`	Closes the file after you're done.
`eof()` (end of file)	Checks if the end of the file is reached.
`fail()`	Checks if the file has failed to open.
`good()`	Checks if the file is ready for operations.

💡 Example: Checking if File Opened Successfully

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

int main() {
    ifstream file;
    file.open("data.txt");

    if (!file) {
        cout << "File could not be opened!" << endl;
    } else {
        cout << "File opened successfully!" << endl;
    }

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

🗁 File Modes in C++

When you open a file, you can choose a **mode** to tell the program *how* you want to use the file.

Mode	Meaning
`ios::in`	Open file for reading.
`ios::out`	Open file for writing.
`ios::app`	Append data to the end of the file.
`ios::ate`	Move to the end of the file immediately after opening.
`ios::trunc`	Clear the file if it already exists.
`ios::binary`	Open the file in binary mode (not text).

💡 Example: Opening a File in Append Mode

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

int main() {
    ofstream file("log.txt", ios::app);
    file << "New log entry!\n";
    file.close();
    return 0;
}
```

☑ Explanation:
This adds a new line to `log.txt` without erasing the previous content!

🗲 Binary Files in C++

Most files you read are **text files** (human-readable).
But you can also save data in **binary form** (0s and 1s) — which is faster and more compact for computers.

💡 Example: Writing to a Binary File

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

int main() {
    int num = 1234;
    ofstream file("number.bin", ios::binary);
    file.write((char*)&num, sizeof(num));
    file.close();
    return 0;
}
```

☑ Explanation:
This writes the number `1234` into `number.bin` in binary format.

💡 Example: Reading from a Binary File

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

int main() {
    int num;
    ifstream file("number.bin", ios::binary);
    file.read((char*)&num, sizeof(num));
    cout << "Number is: " << num << endl;
    file.close();
    return 0;
}
```

☑ Explanation:
This reads the saved number from `number.bin` and displays it.

🎲 Random File Access in C++

Sometimes you don't want to read a file from start to end.
You want to jump to a specific **position** in the file — this is called **Random Access**.

⚙️ Functions for Random Access

sFunction	Purpose
`seekg()`	Move the read pointer.
`seekp()`	Move the write pointer.
`tellg()`	Get the current read position.
`tellp()`	Get the current write position.

💡 Example: Jumping in a File

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

int main() {
    fstream file("data.txt", ios::in | ios::out);
    file.seekp(5);  // Move to the 5th byte in the file
    file << "Hi";   // Overwrite starting from position 5
    file.close();
    return 0;
}
```

☑️ Explanation:

This skips the first 5 characters of the file and then writes `Hi` starting from that point.

☑ Summary

Concept	Meaning
File Management Functions	Help you open, close, check, and control files.
File Modes	Decide how the file is used (read, write, append, etc).
Binary Files	Store and read data as 0s and 1s — faster and compact.
Random Files	Let you jump to any position in a file directly.