Stream Operations

C++ uses streams to perform input and output (I/O) operations. A stream is an abstraction that represents a device where data can be read from or written to, such as the console, a file, or a string.

1 Standard Input and Output

`cin` and `cout` are used for reading from standard input and writing to standard output respectively.

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

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "You are " << age << " years old." << endl;
    return 0;
}
```

2 String Streams

The `sstream` header provides stream classes that operate on strings: `istringstream`, `ostringstream`, and `stringstream`.

- `istringstream` is used to extract data from strings.
- `ostringstream` is used to write data to a string.
- `stringstream` can do both.

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

int main() {
    string data = "42 3.14 hello";
    istringstream iss(data);
    int x;
    double y;
    string word;
    iss >> x >> y >> word;
    cout << x << ", " << y << ", " << word << endl;
    return 0;
}
```

3 File Streams

The `fstream` header provides `ifstream` for reading files and `ofstream` for writing files.

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

int main() {
    ofstream outFile("output.txt");
    outFile << "Hello, file!" << endl;
    outFile.close();

    ifstream inFile("output.txt");
    string line;
    getline(inFile, line);
    cout << "Read from file: " << line << endl;
    inFile.close();
    return 0;
}
```

4 Notes

- Always check if file streams are successfully opened using `is_open()` or by checking the stream object in a condition.
- Use `getline()` for reading lines that may include spaces.
- Use `eof()` or loop conditions like `while(inFile >> var)` to process streams until the end.

5 A more detailed example

This example demonstrates writing structured data to a file and then reading it back line by line.

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

int main() {
    // Write to file
    ofstream outFile("students.txt");
    if (!outFile) {
        cerr << "Failed to open file for writing." << endl;
        return 1;
    }

    vector<string> students = {
        "101 Alice 88.5",
        "102 Bob 92.0",
        "103 Carol 79.5"
    };

    for (const auto& line : students) {
        outFile << line << endl;
    }
    outFile.close();

    // Read from file
    ifstream inFile("students.txt");
    if (!inFile) {
        cerr << "Failed to open file for reading." << endl;
        return 1;
    }

    string line;
    while (getline(inFile, line)) {
        istringstream iss(line);
```

```
        int id;
        string name;
        float score;
        iss >> id >> name >> score;
        cout << "ID: " << id << ", Name: " << name << ", Score: " << score << endl;
    }
    inFile.close();

    return 0;
}
```

Explanation:

- The first part writes a vector of student records to a file using `ofstream`.
- The second part reads each line using `getline()`, then parses each line with `istringstream`.
- The `auto` keyword in the range-based `for` loop lets the compiler deduce the type of each element in the vector. Here, each element is a `std::string`.
- `cerr` is used to print error messages to the standard error stream. It is unbuffered, meaning output is sent directly to the terminal without being stored temporarily in a buffer. This ensures that error messages appear immediately.