Files I/O in C++

File I/O in C++ involves reading from and writing to files. Here's a basic example demonstrating file input/output operations:

```
#include <iostream>
#include <fstream> // Required for file I/O operations

int main() {
    // Writing to a file
    std::ofstream outFile("output.txt"); // Creates a new file named "output.txt" for writing

    if (outFile.is_open()) { // Check if the file is opened successfully
        outFile << "Hello, World!" << std::endl;
        outFile << "This is a line written to the file." << std::endl;
        outFile.close(); // Close the file when done
        std::cout << "Data written to file successfully." << std::endl;
    }
    else {
        std::cout << "Unable to open file for writing." << std::endl;
        return 1;
    }

    // Reading from a file
    std::ifstream inFile("output.txt"); // Opens the file "output.txt" for reading

    if (inFile.is_open()) { // Check if the file is opened successfully
        std::cout << "Contents of the file:" << std::endl;
        std::string line;
        while (std::getline(inFile, line)) { // Read the file line by line
            std::cout << line << std::endl;
        }
        inFile.close(); // Close the file when done
    }
    else {
        std::cout << "Unable to open file for reading." << std::endl;
        return 1;
    }

    return 0;
}
```

Output:

```
Data written to file successfully.
Contents of the file:
Hello, World!
This is a line written to the file.
```

In this example:

1. We include the <fstream> header for file I/O operations.
2. We create an output file stream outFile to write data to a file named "output.txt". We write some lines of text to the file.
3. After writing, we close the output file stream.
4. We then create an input file stream inFile to read data from the same file "output.txt". We read and output the contents of the file line by line.
5. Finally, we close the input file stream.

Templates

Templates in C++ allow you to write generic code that can work with any data type. They provide a way to create functions, classes, or structures that can operate with any data type without having to write separate implementations for each type. Templates are a powerful feature of C++ that enable code reusability and flexibility.

Example:

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

// Class template for a generic Pair
template<typename T1, typename T2>
class Pair {
private:
    T1 first;
    T2 second;
public:
    Pair(T1 f, T2 s) : first(f), second(s) {}

    void display() {
        cout << "(" << first << ", " << second << ")" << endl;
    }
};

int main() {
    Pair<int, double> pair1(5, 3.14);
    pair1.display();

    Pair<string, char> pair2("Hello", 'W');
    pair2.display();

    return 0;
}
```

Output:

```
(5, 3.14)
(Hello, W)
```