C++ File Handling

Table of Contents

1. Introduction to File Handling

File handling in C++ allows programs to store and retrieve data from external files. This enables data persistence beyond program execution and provides a way to process large amounts of data.

C++ provides robust file handling capabilities through the `<fstream>` library, which includes three main classes:

- `ifstream` for reading from files
- `ofstream` for writing to files
- `fstream` for both reading and writing

2. Basic File Operations

Opening Files

To work with files, you first need to include the necessary header:

```
#include <fstream>
```

Then create a file stream object and open a file:

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

int main() {
    // Creating an output file stream
    ofstream outFile("example.txt");

    if (outFile.is_open()) {
        cout << "File opened successfully!" << endl;
        // File operations go here
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

File Modes

When opening a file, you can specify different modes:

```
ofstream outFile("example.txt", ios::out);    // Output (default for ofstream)
ifstream inFile("example.txt", ios::in);      // Input (default for ifstream)
fstream ioFile("example.txt", ios::in | ios::out); // Both input and output
```

Common file modes:

Mode	Description
`ios::in`	Open for reading
`ios::out`	Open for writing (creates file if doesn't exist)
`ios::app`	Append to end of file
`ios::ate`	Set initial position to end of file
`ios::trunc`	Erase content if file exists
`ios::binary`	Open in binary mode

You can combine modes using the bitwise OR operator (`|`):

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

3. Writing to Files

Text Mode(Default): Writing to a file in text mode:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open file for writing
    ofstream outFile("example.txt");

    if (outFile.is_open()) {
        // Write to the file
        outFile << "Hello, World!" << endl;
        outFile << "This is a line of text." << endl;
        outFile << "Numbers can be written too: " << 42 << endl;
        // Close the file
        outFile.close();
        cout << "Data written to file successfully!" << endl;
    } else {
        cout << "Failed to open file!" << endl;
    }
    return 0;}
```

Binary Mode

Writing to a file in binary mode:

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

struct Person {
    char name[50];
    int age;
    double salary;
};

int main() {
    Person person = {"John Doe", 30, 50000.50};

    // Open file in binary mode
    ofstream outFile("person.bin", ios::binary);

    if (outFile.is_open()) {
        // Write binary data
        outFile.write(reinterpret_cast<char*>(&person),  sizeof(Person));

        // Close the file
        outFile.close();
        cout << "Binary data written successfully!" << endl;
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

4. Reading from Files

Reading Character by Character

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

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

    if (inFile.is_open()) {
        char ch;

        // Read file character by character
        while (inFile.get(ch)) {
            cout << ch;
        }

        // Close the file
        inFile.close();
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

Reading Line by Line

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

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

    if (inFile.is_open()) {
        string line;

        // Read file line by line
        while (getline(inFile, line)) {
            cout << line << endl;
        }

        // Close the file
        inFile.close();
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

Reading Word by Word

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

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

    if (inFile.is_open()) {
        string word;

        // Read file word by word
        while (inFile >> word) {
            cout << word << " ";
        }

        // Close the file
        inFile.close();
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

Reading the Entire File

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

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

    if (inFile.is_open()) {
        // Read entire file into a string stream
        stringstream buffer;
        buffer << inFile.rdbuf();

        // Get the content as a string
        string content = buffer.str();

        cout << "File content:" << endl;
        cout << content << endl;

        // Close the file
        inFile.close();
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

5. File Pointers and Positioning

Getting Current Position

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

int main() {
    fstream file("example.txt", ios::in | ios::out);

    if (file.is_open()) {
        // Get current position
        streampos position = file.tellg();
        cout << "Current read position: " << position << endl;

        // Move to 10th byte
        file.seekg(10);

        // Get new position
        position = file.tellg();
        cout << "New read position: " << position << endl;
        file.close();
    }
    return 0;}
```

Moving to Specific Positions

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

int main() {
    fstream file("example.txt", ios::in | ios::out);

    if (file.is_open()) {
        // Move to the beginning
        file.seekg(0, ios::beg);
        // Move to the end
        file.seekg(0, ios::end);
        // Get file size
        streampos fileSize = file.tellg();
        cout << "File size: " << fileSize << " bytes" << endl;
        file.close();
    }
    return 0;
}
```

6. Error Handling

Checking File Status

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

int main() {
    ifstream inFile("nonexistent.txt");

    if (!inFile) {
        cerr << "Error: File could not be opened!" << endl;
        return 1;
    }
    // Check for errors during operations
    int value;
    inFile >> value;

    if (inFile.fail()) {
        cerr << "Error: Failed to read from file!" << endl;
        inFile.close();
        return 1;
    }

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

Exception Handling

```
#include <iostream>
#include <fstream>
#include <stdexcept>
using namespace std;
int main() {
    try {
        ifstream inFile("data.txt");
        if (!inFile) {
            throw runtime_error("Could not open file");
        }
        // File operations
        inFile.close();
    }
    catch (const exception& e) {
        cerr << "Exception: " << e.what() << endl;
        return 1;

        }
    return 0;       }
```

7. Working with Binary Files

Reading from a binary file:

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

struct Person {
    char name[50];
    int age;
    double salary;
};

int main() {
    Person person;

    // Open binary file for reading
    ifstream inFile("person.bin", ios::binary);

    if (inFile.is_open()) {
        // Read binary data
        inFile.read(reinterpret_cast<char*>(&person), sizeof(Person));

        // Display the data
        cout << "Name: " << person.name << endl;
        cout << "Age: " << person.age << endl;
        cout << "Salary: " << person.salary << endl;

        inFile.close();
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

8. Advanced File Operations

Appending to Files

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

int main() {
    // Open file in append mode
    ofstream outFile("log.txt", ios::app);

    if (outFile.is_open()) {
        outFile << "This line will be appended to the end of the file." << endl;
        outFile.close();
        cout << "Data appended successfully!" << endl;
    } else {
        cout << "Failed to open file!" << endl;
    }

    return 0;
}
```

Random Access

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

struct Record {
    int id;
    char name[50];
    double score;
};

int main() {
    fstream file("records.bin", ios::in | ios::out | ios::binary);

    if (!file) {
        // File doesn't exist, create it
        ofstream createFile("records.bin", ios::binary);

        // Add some records
        Record records[3] = {
            {1, "Alice", 95.5},
            {2, "Bob", 87.3},
            {3, "Charlie", 92.7}
        };

        for (int i = 0; i < 3; i++) {
            createFile.write(reinterpret_cast<char*>(&records[i]), sizeof(Record));
        }

        createFile.close();

        // Reopen in read/write mode
        file.open("records.bin", ios::in | ios::out | ios::binary);
    }

    // Access the second record (index 1)
    file.seekg(sizeof(Record) * 1);

    Record record;
    file.read(reinterpret_cast<char*>(&record),  sizeof(Record));

    cout << "ID: " << record.id << endl;
    cout << "Name: " << record.name << endl;
    cout << "Score: " << record.score << endl;

    // Modify the record
    record.score = 91.0;
    // Move back to write the modified record
    file.seekp(sizeof(Record) * 1);
    file.write(reinterpret_cast<char*>(&record), sizeof(Record));

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

9. Best Practices

1. **Always close files**: Use proper file closing to prevent resource leaks.
2. **Check if files are open**: Always verify that files opened successfully before performing operations.
3. **Use appropriate modes**: Choose the correct file mode based on your needs.
4. **Handle errors gracefully**: Implement proper error handling to make your program robust.
5. **Use binary mode for binary data**: When working with non-text data, always use binary mode.
6. **Use exceptions for error handling**: Consider using exception handling for more complex file operations.
7. **Use streams for formatting**: Take advantage of stream formatting capabilities for better output.
8. **Buffer management**: For performance-critical applications, consider adjusting buffer sizes.

```
// Setting buffer size
file.rdbuf()->pubsetbuf(buffer, bufferSize);
```

9. **Close files before program exit**: Make sure all files are closed before your program terminates.
10. **Use RAII pattern**: Consider using Resource Acquisition Is Initialization pattern with file streams.

```
{
    ofstream file("example.txt");
    // File operations
    // File automatically closed when going out of scope
}
```

10. Practical Examples

Example 1: Simple Text Editor

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

void viewFile(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "File not found or cannot be opened." << endl;
        return;
    }

    string line;
    int lineNum = 1;

    cout << "\n--- File Content ---\n";
    while (getline(file, line)) {
        cout << lineNum++ << ": " << line << endl;
    }
    cout << "--------------------------------\n";

    file.close();
}

void editFile(const string& filename) {
    ofstream file(filename);
    if (!file) {
        cout << "Cannot create or open file for writing." << endl;
        return;
    }

    cout << "Enter file content (type 'END' on a new line to finish):\n";
    string line;

    while (true) {
        getline(cin, line);
        if (line == "END") break;
        file << line << endl;
    }
    file.close();
    cout << "File saved successfully!" << endl;
}

int main() {
    string filename;
    int choice;

    cout << "Simple Text Editor" << endl;
    cout << "Enter filename: ";
    getline(cin, filename);

    do {
        cout << "\nMenu:\n";
        cout << "1. View file\n";
        cout << "2. Edit file\n";
        cout << "3. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;
        cin.ignore(); // Clear the newline

        switch (choice) {
            case 1:
                viewFile(filename);
                break;
            case 2:
                editFile(filename);
                break;
            case 3:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice!" << endl;
        }
    } while (choice != 3);

    return 0;
}
```

***********************Only for those who want to explore more*****************************

Example 2: CSV File Processor

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

struct Student {
    string name;
    int id;
    double gpa;
};

vector<Student> readCSV(const string& filename) {
    vector<Student> students;
    ifstream file(filename);

    if (!file) {
        cout << "File not found or cannot be opened." << endl;
        return students;
    }

    string line, cell;
    // Skip header
    getline(file, line);

    while (getline(file, line)) {
        stringstream lineStream(line);
        Student student;

        // Parse CSV row
        getline(lineStream, cell, ',');
        student.name = cell;

        getline(lineStream, cell, ',');
        student.id = stoi(cell);

        getline(lineStream, cell, ',');
        student.gpa = stod(cell);

        students.push_back(student);
    }
```

```cpp
    file.close();
    return students;
}
void writeCSV(const string& filename, const vector<Student>& students) {
    ofstream file(filename);

    if (!file) {
        cout << "Cannot create or open file for writing." << endl;
        return;
    }

    // Write header
    file << "Name,ID,GPA" << endl;

    // Write data
    for (const auto& student : students) {
        file << student.name << "," << student.id << "," << student.gpa << endl;
    }

    file.close();
    cout << "CSV file written successfully!" << endl;
}

int main() {
    // Sample data
    vector<Student> students = {
        {"Alice Smith", 10001, 3.8},
        {"Bob Johnson", 10002, 3.2},
        {"Charlie Brown", 10003, 3.9},
        {"Diana Prince", 10004, 4.0}
    };
    // Write to CSV
    writeCSV("students.csv", students);

    // Read from CSV
    vector<Student> readStudents = readCSV("students.csv");

    // Display read data
    cout << "\nStudent records from CSV:\n";
    cout << "----------------------------------------\n";
    for (const auto& student : readStudents) {
        cout << "Name: " << student.name
             << ", ID: " << student.id
             << ", GPA: " << student.gpa << endl;
    }
    return 0;
}
```