Module 5 Critical Thinking

Dennis Weddig

Colorado State University Global

CSC450: Programming III

Dr. Jack Li

9/14/2025 11:59pm

Module 5 Critical Thinking

Repository location for Module 5 Critical Thinking assignment:

https://github.com/denniswed/csc450/tree/main/Module5/crit_think

Code:

```
/*

* Program: Create a C++ program that will obtain input from a user and store it
into the provided CSC450_CT5_mod5.txt.

* Your program should append it to the provided text file, without deleting the
existing data:
a. Store the provided data in the CSC450_CT5_mod5.txt file.
b. Create a reversal method that will reverse all of the characters in the CSC450_CT5_mod5.txt
file and store the result in a CSC450-mod5-reverse.txt file.

* Things I added:
- File size checks to prevent excessive memory usage (FIO19-C)
- RAII principles for file handling to ensure proper resource management
- Input validation to avoid excessively long lines (STR50-CPP)
- Exception handling for robust error management (ERR50-CPP)
- Use of std::filesystem for safer file operations (C++17 feature)
- Encapsulation of functionality within a class for better organization (OOP58-CPP)
- Clear separation of concerns with helper methods for file operations
- Use of constants for file names and limits to avoid magic numbers
- copying the orginal file from backup so we start with a clean file reach run
*/

#include <algorithm>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>

class FileProcessor {
private:
static constexpr const char* INPUT_FILE = "CSC450_CT5_mod5.txt";
static constexpr const char* OUTPUT_FILE = "CSC450-mod5-reverse.txt";
static constexpr const char* BACKUP_FILE = "CSC450_CT5_mod5 copy.txt";
static constexpr size_t MAX_FILE_SIZE = 10'000'000; // 10MB limit
```

```
// Helper method for safe file reading (FIO51-CPP)
std::string readFileContent(const std::string& filename) const {
// Check file size first (FIO19-C)
std::error_code ec;
auto fileSize = std::filesystem::file_size(filename, ec);
if (ec || fileSize > MAX_FILE_SIZE) {
throw std::runtime_error("File too large or inaccessible: " + filename);
}

// Use RAII for file management
std::ifstream file(filename, std::ios::binary);
if (!file) {
throw std::runtime_error("Cannot open file: " + filename);
}

// Reserve space to avoid multiple allocations
std::string content;
content.reserve(static_cast<size_t>(fileSize));

// Read entire file
content.assign(std::istreambuf_iterator<char>(file), std::istreambuf_iterator<char>());

return content;
}

// Helper method for safe file writing (FIO51-CPP)
void writeFileContent(const std::string& filename, const std::string& content) const {
// Use RAII for file management
std::ofstream file(filename, std::ios::binary);
if (!file) {
throw std::runtime_error("Cannot create file: " + filename);
}

file << content;

// Explicitly check for write errors (FIO04-CPP)
if (!file.good()) {
throw std::runtime_error("Write error occurred for file: " + filename);
}
}

// Helper method for safe file appending (FIO51-CPP)
void appendToFile(const std::string& filename, const std::string& content) const {
std::ofstream file(filename, std::ios::app | std::ios::binary);
```

```
if (!file) {
throw std::runtime_error("Cannot open file for appending: " + filename);
}

file << content;

if (!file.good()) {
throw std::runtime_error("Append error occurred for file: " + filename);
}
}

public:
FileProcessor() noexcept = default;

// Non-copyable, non-movable for simplicity (OOP58-CPP)
FileProcessor(const FileProcessor&) = delete;
FileProcessor& operator=(const FileProcessor&) = delete;
FileProcessor(FileProcessor&&) = delete;
FileProcessor& operator=(FileProcessor&&) = delete;

/**
* Restores the input file from backup copy
* @return true if successful, false otherwise
*/
bool restoreFromBackup() const noexcept {
try {
std::cout << "=== Restoring Original File ===\n";

// Check if backup file exists
if (!std::filesystem::exists(BACKUP_FILE)) {
std::cerr << "Backup file not found: " << BACKUP_FILE << '\n';
return false;
}

// Read backup content
std::string backupContent = readFileContent(BACKUP_FILE);

// Delete existing input file if it exists
std::error_code ec;
if (std::filesystem::exists(INPUT_FILE)) {
std::filesystem::remove(INPUT_FILE, ec);
if (ec) {
std::cerr << "Failed to delete existing file: " << INPUT_FILE << '\n';
return false;
}
```

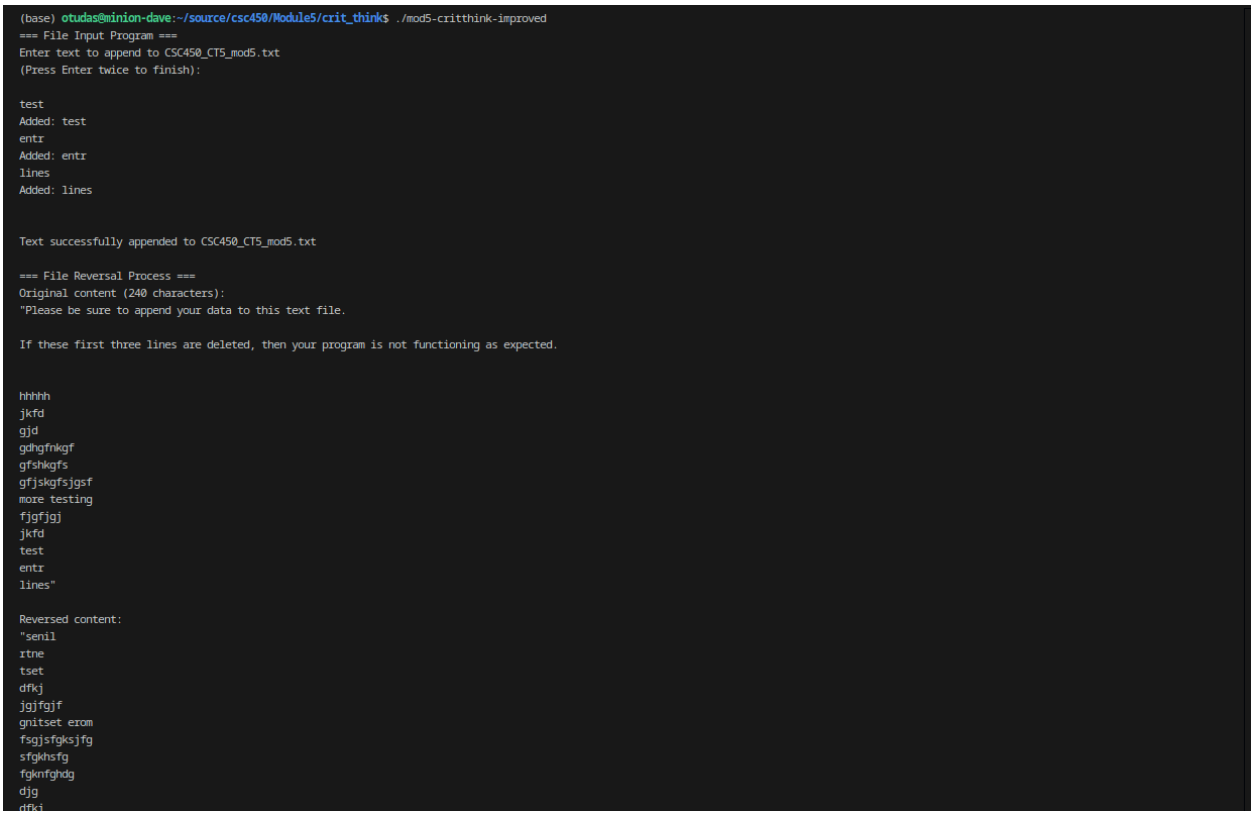
```
}

// Write backup content to input file
writeFileContent(INPUT_FILE, backupContent);

std::cout << "Successfully restored " << INPUT_FILE << " from " << BACKUP_FILE << '\n';
return true;

} catch (const std::exception& e) {
std::cerr << "Error restoring from backup: " << e.what() << '\n';
return false;
}
}

/**
* Safely appends user input to the specified file
* @return true if successful, false otherwise
*/
bool appendUserInput() {
try {
std::cout << "=== File Input Program ===\n"
<< "Enter text to append to " << INPUT_FILE << "\n"
<< "(Press Enter twice to finish):\n\n";

std::vector<std::string> lines;
std::string line;
bool hasContent = false;

// Collect all input first (safer approach)
while (std::getline(std::cin, line)) {
if (line.empty() && hasContent) {
break; // Stop on empty line after content
}

if (!line.empty()) {
// Input validation (STR50-CPP)
if (line.length() > 1000) { // Reasonable line length limit
std::cerr << "Warning: Line too long, truncating\n";
line.resize(1000);
}

hasContent = true;
lines.push_back(line);
std::cout << "Added: " << line << '\n';
}
```

```
}

if (!hasContent) {
std::cout << "No content entered.\n";
return false;
}

// Build content string
std::string content;
for (const auto& inputLine : lines) {
content += inputLine + '\n';
}

// Append to file using RAII
appendToFile(INPUT_FILE, content);

std::cout << "\nText successfully appended to " << INPUT_FILE << '\n';
return true;

} catch (const std::exception& e) {
std::cerr << "Error appending to file: " << e.what() << '\n';
return false;
}
}

/**
* Safely reads input file, reverses all characters, and writes to output file
* @return true if successful, false otherwise
*/
bool reverseFileContent() const noexcept {
try {
std::cout << "\n=== File Reversal Process ===\n";

// Read file content safely
std::string content = readFileContent(INPUT_FILE);

if (content.empty()) {
std::cout << "Input file is empty. Nothing to reverse.\n";
return false;
}

// Remove trailing newline for clean reversal
if (!content.empty() && content.back() == '\n') {
content.pop_back();
}
```

```
std::cout << "Original content (" << content.length() << " characters):\n"
<< "\"" << content << "\"\n\n";

// Reverse all characters (safe with std::string)
std::reverse(content.begin(), content.end());

std::cout << "Reversed content:\n"
<< "\"" << content << "\"\n\n";

// Write to output file
writeFileContent(OUTPUT_FILE, content + '\n');

std::cout << "Reversed content written to " << OUTPUT_FILE << '\n';
return true;

} catch (const std::exception& e) {
std::cerr << "Error during reversal process: " << e.what() << '\n';
return false;
}
}

/**
* Displays the contents of both files for verification
*/
void displayFileContents() const noexcept {
std::cout << "\n=== File Contents Verification ===\n";

displaySingleFile(INPUT_FILE);
displaySingleFile(OUTPUT_FILE);
}

private:
void displaySingleFile(const std::string& filename) const noexcept {
try {
std::cout << "\nContents of " << filename << ":\n" << std::string(50, '-') << '\n';

std::string content = readFileContent(filename);
std::istringstream stream(content);
std::string line;
int lineNum = 1;

while (std::getline(stream, line)) {
std::cout << lineNum++ << ": " << line << '\n';
}
```

```
} catch (const std::exception& e) {
std::cout << "Could not read file " << filename << ": " << e.what() << '\n';
}
}
};

int main() {
try {
FileProcessor processor;

// Step 1: Get user input and append to file
if (processor.appendUserInput()) {
// Step 2: Reverse the file content
if (processor.reverseFileContent()) {
// Step 3: Display results
processor.displayFileContents();

std::cout << "\n=== Program Completed Successfully ===\n"
<< "✓ User input appended to CSC450_CT5_mod5.txt\n"
<< "✓ Reversed content saved to CSC450-mod5-reverse.txt\n";
}
}

return EXIT_SUCCESS;

} catch (const std::exception& e) {
std::cerr << "Fatal error: " << e.what() << '\n';
return EXIT_FAILURE;
}
}
```

Screenshots of above execution (its in multiple parts):



```
dfkj
hhhhh


.detcepxe sa gninoitcnuf ton si margorp ruoy neht ,deteled era senil eerht tsrif eseht fI

.elif txet siht ot atad ruoy dneppa ot erus eb esaelP"

Reversed content written to CSC450-mod5-reverse.txt

=== File Contents Verification ===

Contents of CSC450_CT5_mod5.txt:
--------------------------------------------------
1: Please be sure to append your data to this text file.
2:
3: If these first three lines are deleted, then your program is not functioning as expected.
4:
5:
6: hhhhh
7: jkfd
8: gjd
9: gdhgfnkgf
10: gfshkgfs
11: gfjskgfsjgsf
12: more testing
13: fjgfjgj
14: jkfd
15: test
16: entr
17: lines

Contents of CSC450-mod5-reverse.txt:
--------------------------------------------------
1: senil
2: rtne
3: tset
4: dfkj
5: jgjfgjf
6: gnitset erom
7: fsgjsfgksjfg
8: sfgkhsfg
9: fgknfghdg
10: djg
11: dfkj
12: hhhhh
13:
```

```
Contents of CSC450-mod5-reverse.txt:
--------------------------------------------------
1: senil
2: rtne
3: tset
4: dfkj
5: jgjfgjf
6: gnitset erom
7: fsgjsfgksjfg
8: sfgkhsfg
9: fgknfghdg
10: djg
11: dfkj
12: hhhhh
13:
14:
.detcepxe sa gninoitcnuf ton si margorp ruoy neht ,deteled era senil eerht tsrif eseht fI
16:
.elif txet siht ot atad ruoy dneppa ot erus eb esaelP

=== Program Completed Successfully ===
✓ User input appended to CSC450_CT5_mod5.txt
✓ Reversed content saved to CSC450-mod5-reverse.txt
(base) otudas@minion-dave:~/source/csc450/Module5/crit_thinks$
```

References

CPlusPlus.com. (n.d.). *Input/output with files*. *CPlusPlus.com*.

https://cplusplus.com/doc/tutorial/files/

CPlusPlus.com. (n.d.). *<fstream>*. *CPlusPlus.com*. https://cplusplus.com/reference/fstream/

CPlusPlus.com. (n.d.). *std::ios_base*. *CPlusPlus.com*.

https://cplusplus.com/reference/ios/ios_base/

Cppreference.com contributors. (n.d.). *Input/output library*. *cppreference*.

https://en.cppreference.com/w/cpp/io

Cppreference.com contributors. (n.d.). *std::basic_fstream*. *cppreference*.

https://en.cppreference.com/w/cpp/io/basic_fstream

Cppreference.com contributors. (n.d.). *std::filesystem*. *cppreference*.

https://en.cppreference.com/w/cpp/filesystem

GeeksforGeeks. (n.d.). *File handling through C++ classes*. *GeeksforGeeks*.

https://www.geeksforgeeks.org/file-handling-c-classes/

GeeksforGeeks. (n.d.). *C++ stream classes structure*. *GeeksforGeeks*.

https://www.geeksforgeeks.org/c-stream-classes-structure/

Software Engineering Institute. (n.d.). Secure coding in C++11 and C++14. *SEI Blog, Carnegie Mellon University*. https://www.sei.cmu.edu/blog/secure-coding-in-c11-and-c14/