Problem 1: Read from a File

Task:

Write a C++ program that reads a text file named input.txt and prints its content to the console.

Code:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream inputFile("input.txt");

if (!inputFile) {

cerr << "Error opening file." << endl;

return 1;

}

string line;

while (getline(inputFile, line)) {

cout << line << endl;

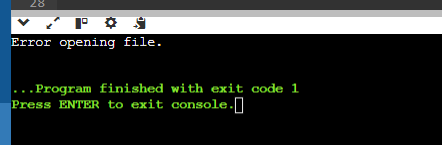
}

inputFile.close();

return 0;

}

Output:



How do you open a file for reading in C++?

Ans: To open a file for reading in C++, we use the std::ifstream class.

What is the purpose of the ifstream class in C++?

Ans: The std::ifstream (input file stream) class is used for reading from files. It provides various functions to facilitate reading data from a file, such as reading character by character, line by line, or even formatted data.

How can you check if a file was successfully opened?

Ans: To check if a file was successfully opened, you can use the is\_open() method or simply use the ifstream object in a boolean context.

What function do you use to read a line from a file?

Ans: To read a line from a file, we use the std::getline function.

How do you properly close a file after reading?

Ans: After you are done reading from a file, you should close it using the close() method to free the associated resources.

Problem 2: Write to a File

Task:

Write a C++ program that writes the following lines to a file named output.txt:

bash

Copy code

Hello, world!

This is a test file.

Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outputFile("output.txt");

if (!outputFile) {

cerr << "Error opening file for writing." << endl;

return 1;

}

outputFile << "Hello, world!" << endl;

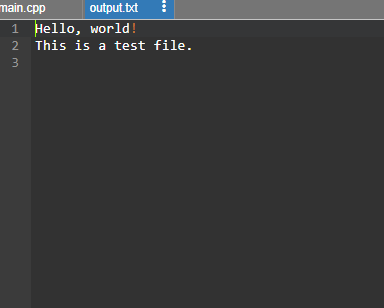
outputFile << "This is a test file." << endl;

outputFile.close();

return 0;

}

Output:



How do you open a file for writing in C++?

Ans: To open a file for writing in C++, we use the std::ofstream class.

What is the purpose of the ofstream class in C++?

Ans: The std::ofstream (output file stream) class is used for writing to files. It provides various functions to facilitate writing data to a file, such as writing strings, characters, or formatted data.

How can you handle errors if the file fails to open for writing?

Ans: To handle errors when a file fails to open for writing, you can check if the file stream object is in a good state. This can be done using the is\_open() method or simply using the ofstream object in a boolean context.

How do you write a string to a file in C++?

Ans: To write a string to a file, you use the << operator with the ofstream object.

What is the importance of closing a file after writing to it?

Ans: Closing a file after writing to it is important because it:

1. Ensures that all data written to the file is properly saved and not stuck in the buffer.
2. Frees the file handle and other system resources associated with the file.

Problem 3: Append to a File

Task:

Write a C++ program that appends the following line to a file named log.txt:

bash

Copy code

New log entry.

Code:

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outputFile("log.txt", ios\_base::app);

if (!outputFile) {

cerr << "Error opening file for appending." << endl;

return 1;

}

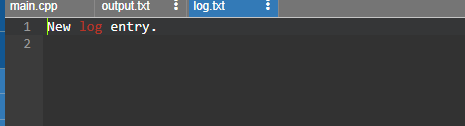
outputFile << "New log entry." << endl;

outputFile.close();

return 0;

}

Output:



How do you open a file for appending in C++?  
In C++, you can open a file for appending by using the open method of the std::ofstream class with the std::ios\_base::app mode flag.

What is the difference between opening a file in write mode and append mode?

Write mode: When a file is opened in write mode, if the file already exists, its content is truncated (erased), and new content is written from the beginning of the file. If the file does not exist, it is created.

Append mode: When a file is opened in append mode, if the file already exists, new content is written at the end of the file without erasing the existing content. If the file does not exist, it is created. The position for writing is always at the end of the file.

How do you use the ofstream class to append data to a file?

You can use the ofstream class to append data to a file by opening the file with the std::ios\_base::app flag.

What happens if the file does not exist when you try to open it in append mode?

If the file does not exist when you try to open it in append mode (std::ios\_base::app), the file will be created. The new content will then be written to this newly created file.

How can you ensure data integrity when appending to a file?

**Check if the file opened successfully**: Ensure that the file was opened without errors before attempting to write to it.

**lush the stream**: After writing to the file, you can explicitly flush the output stream to ensure that all data is written to the file immediately.

Problem 4: Copy a File

Task:

Write a C++ program that copies the content of a file named source.txt to another file named destination.txt.

include <fstream>

#include <iostream>

using namespace std;

int main() {

ifstream sourceFile("source.txt", ios::binary);

ofstream destinationFile("destination.txt", ios::binary);

if (!sourceFile) {

cerr << "Error: Could not open source file." << endl;

return 1;

}

if (!destinationFile) {

cerr << "Error: Could not open destination file." << endl;

return 1;

}

destinationFile << sourceFile.rdbuf();

sourceFile.close();

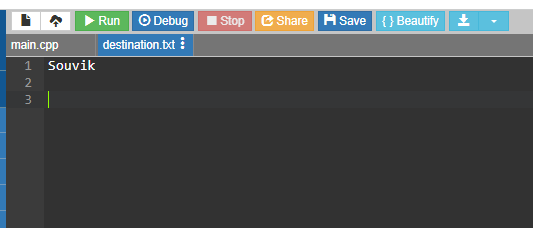
destinationFile.close();

cout << "File copied successfully." << endl;

return 0;

}

Output:



How do you read from one file and write to another file in C++?

In C++, you can read from one file and write to another using std::ifstream for reading and std::ofstream for writing.

How can you efficiently copy the contents of a file in C++?

A more efficient way to copy the contents of a file in C++ is to use binary mode and copy the buffer directly:

#include <fstream>

#include <iostream>

int main() {

std::ifstream sourceFile("source.txt", std::ios::binary);

std::ofstream destinationFile("destination.txt", std::ios::binary);

if (!sourceFile.is\_open() || !destinationFile.is\_open()) {

std::cerr << "Error opening files!" << std::endl;

return 1;

}

destinationFile << sourceFile.rdbuf();

sourceFile.close();

destinationFile.close();

return 0;

}

What are the potential errors you should handle when copying a file?

1. The source file may not exist.
2. Issues like running out of disk space or file system corruption.
3. Problems during the read/write operations.

How do you check the end-of-file (EOF) condition when reading a file?

You can check the end-of-file (EOF) condition using the eof() method of the stream.

How do you ensure both files are properly closed after the copy operation?

To ensure both files are properly closed, always use the close() method for both input and output files. Additionally, consider using RAII (Resource Acquisition Is Initialization) by leveraging the destructors of ifstream and ofstream which automatically close the files when they go out of scope.

Problem 5: Count Words in a File

Task:

Write a C++ program that reads a file named data.txt and counts the number of words in the file.

#include <fstream>

#include <iostream>

#include <string>

int main() {

std::ifstream file("data.txt");

if (!file.is\_open()) {

std::cerr << "Error: Could not open file." << std::endl;

return 1;

}

std::string word;

int wordCount = 0;

while (file >> word) {

++wordCount;

}

if (file.eof()) {

std::cout << "End of file reached." << std::endl;

} else if (file.fail()) {

std::cerr << "Error: Reading from file failed." << std::endl;

}

file.close();

std::cout << "Number of words in the file: " << wordCount << std::endl;

return 0;

}

How do you define a word in the context of reading from a file?

n the context of reading from a file, a "word" is typically defined as a sequence of characters separated by whitespace (spaces, newlines, tabs). This definition can be expanded to include handling punctuation and special characters, but the basic idea is to use whitespace as the delimiter.

What functions can you use to read words from a file in C++?

In C++, you can use the >> operator with an ifstream object to read words from a file. This operator automatically treats any whitespace (space, newline, tab) as a delimiter.

How do you handle different word delimiters (spaces, newlines, etc.)?

The >> operator in C++ treats any whitespace character as a delimiter, so it naturally handles spaces, newlines, and tabs. If you need to handle custom delimiters, you can use the std::getline function with a custom delimiter and process the resulting strings.

How can you keep track of the word count while reading the file?

To keep track of the word count while reading the file, you can simply increment a counter each time a word is successfully read.

How do you handle large files to avoid memory issues while counting words?

When dealing with large files, you want to ensure that you read the file in a memory-efficient manner. The ifstream object handles this efficiently as it reads the file in chunks rather than loading the entire file into memory at once. Here are a few tips:

1. Read and process the file line by line or word by word to avoid loading the entire file into memory.
2. Ensure that the file stream is buffered correctly to handle large files efficiently.