**Problem 1. Read from a file.**

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

Ans : #include <iostream>

#include <fstream>

#include <string>

int main() {

std::ifstream inputFile("input.txt");

if (!inputFile) {

std::cerr << "Unable to open file input.txt";

return 1;

}

std::string line;

while (std::getline(inputFile, line)) {

std::cout << line << std::endl;

}

inputFile.close();

return 0;

}

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

Ans : To open a file for reading in C++, you typically use the ifstream class from the <fstream> header. Here’s how you can open a file named filename.txt for reading.

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

Ans : The ifstream class in C++ stands for input file stream. It is used for handling input operations on files. Specifically, it is used to read data from files. Instances of ifstream provide member functions to open a file, read from it, and close it when done.

4. How can you check if a file was successfully opened?

Ans : After attempting to open a file with an ifstream object, you can check if the file was successfully opened using the is\_open() member function of the ifstream class.

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

Ans : To read a line from a file opened with ifstream, you can use the getline() function. This function reads characters from the input stream until a newline character ('\n') is encountered or the specified delimiter is reached.

6. How do you properly close a file after reading?

Ans : To properly close a file after reading from it, you use the close() member function of the ifstream class. This function flushes any buffered data to the file and releases any associated resources. Always ensure to close the file after you have finished reading from it to free up system resources and allow other processes to access the file if needed.

**Problem 2. Write to a file.**

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

Ans : #include <iostream>

#include <fstream>

int main() {

// Open the file for writing

std::ofstream outputFile("output.txt");

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

std::cerr << "Failed to open output.txt for writing\n";

return 1;

}

outputFile << "Hello, world!\n";

outputFile << "This is a test file.\n";

outputFile.close();

return 0;

}

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

Ans : To open a file for writing in C++, you use the ofstream (output file stream) class from the <fstream> header. You provide the file name to the constructor of ofstream or use the open() member function after declaring an ofstream object.

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

Ans : The ofstream class in C++ is used for handling output operations on files. It allows you to write data to files, creating new files or overwriting existing ones. Instances of ofstream provide member functions to open a file, write to it, and close it when done.

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

Ans : You can check if the file opened successfully using the is\_open() member function of ofstream. If is\_open() returns false, it indicates that the file failed to open. In such cases, you can handle the error by printing an error message to std::cerr or taking appropriate action based on your program's requirements.

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

Ans : To write a string to a file using ofstream, you can use the insertion operator (<<) or the write() member function. The insertion operator (<<) is straightforward for writing formatted data, including strings, to the file.

outputFile << "Hello, world!\n";

outputFile << "This is a test file.\n";

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

Ans : Closing a file after writing to it is important for several reasons:

* It ensures that all data is flushed from memory to the file, making sure that all written content is saved.
* It releases system resources associated with the file, allowing other programs or processes to access it.
* It prevents potential data corruption or loss in case of unexpected program termination or errors.

**Problem 3: Append to a File**

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

Ans : #include <iostream>

#include <fstream>

int main() {

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

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

std::cerr << "Failed to open log.txt for appending\n";

return 1;

}

outputFile << "New log entry.\n";

outputFile.close();

return 0;

}

2. How do you open a file for appending in C++?

Ans : To open a file for appending in C++, you can use the std::ofstream class with the std::ios::app flag. This flag stands for append mode and ensures that any output operations are performed at the end of the file, preserving the existing contents. This line opens the file log.txt for appending.

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

Ans :  **Write mode (std::ios::out)**: When you open a file in write mode, it truncates the file (deletes all existing content) and starts writing from the beginning. If the file does not exist, it creates a new one.

 **Append mode (std::ios::app)**: Opening a file in append mode preserves the existing content and appends new data to the end of the file. If the file does not exist, it creates a new file.

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

Ans: You use the std::ofstream class with the std::ios::app flag to open the file for appending. After opening the file, you can use the insertion operator (<<) or the write() member function to append data to the file. This line appends the string "New log entry." followed by a newline to the end of the log.txt file.

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

Ans : If the file specified does not exist when you try to open it in append mode (std::ios::app), C++ will create a new file with the specified name and then open it for appending. This behavior is different from write mode (std::ios::out), which would create a new file and truncate any existing content.

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

Ans : To ensure data integrity when appending to a file:

* Always check if the file opened successfully after attempting to open it in append mode.
* Properly handle any errors that occur during file operations.

**Problem 4: Copy a File**

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

Ans : #include <iostream>

#include <fstream>

int main() {

std::ifstream sourceFile("source.txt");

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

std::cerr << "Failed to open source.txt\n";

return 1;

}

std::ofstream destinationFile("destination.txt");

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

std::cerr << "Failed to open destination.txt\n";

sourceFile.close();

return 1;

}

char ch;

while (sourceFile.get(ch)) {

destinationFile.put(ch);

}

sourceFile.close();

destinationFile.close();

std::cout << "File copied successfully.\n";

return 0;

}

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

Ans : To read from one file and write to another file in C++, you typically use two different stream objects: one for reading (std::ifstream) and another for writing (std::ofstream). You can read data from the source file using input operations (get(), getline(), etc.) and write data to the destination file using output operations (put(), write(), << operator).

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

Ans : Here, sourceFile.get(ch) reads a character from sourceFile into ch, and destinationFile.put(ch) writes the character ch to destinationFile. This loop continues until sourceFile.get(ch) returns false, indicating the end of the file.

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

Ans : Potential errors to handle when copying a file include:

* Failure to open the source file (sourceFile.is\_open() returns false).
* Failure to open the destination file (destinationFile.is\_open() returns false).
* Errors that occur during reading from the source file (sourceFile.fail()).
* Errors that occur during writing to the destination file (destinationFile.fail()).

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

Ans : You can check for the end-of-file (EOF) condition in C++ using the stream's eof() function or by checking the result of input operations (get(), getline(), etc.). In the provided program, we use sourceFile.get(ch), which returns false when it reaches the end of the file.

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

Ans : To ensure both files are properly closed after the copy operation, you should explicitly call the close() function on both ifstream and ofstream objects.

sourceFile.close();

destinationFile.close();

**Problem 5: Count Words in a File**

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

Ans : #include <iostream>

#include <fstream>

#include <string>

#include <sstream>

int countWords(const std::string &str) {

std::stringstream ss(str);

std::string word;

int count = 0;

while (ss >> word) {

count++;

}

return count;

}

int main() {

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

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

std::cerr << "Failed to open data.txt\n";

return 1;

}

std::string line;

int wordCount = 0;

while (std::getline(inputFile, line)) {

wordCount += countWords(line);

}

inputFile.close();

std::cout << "Number of words in data.txt: " << wordCount << std::endl;

return 0;

}

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

Ans : In this program, a word is defined as a sequence of characters separated by whitespace characters (spaces, tabs, newlines, etc.). Punctuation marks and other non-alphanumeric characters are considered part of words unless explicitly filtered out.

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

Ans : You can use std::ifstream along with getline() to read lines from a file and then std::stringstream with operator>> to extract words from each line.

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

Ans : The std::stringstream object splits the input string (line in this case) into words based on whitespace by default (operator>> reads until whitespace is encountered). This effectively handles spaces, tabs, newlines, etc., as delimiters.

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

Ans : We maintain a counter wordCount to accumulate the total number of words encountered across all lines in the file. Each time countWords() function is called with a line, it returns the number of words in that line, which is then added to wordCount.

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

Ans :

* The program reads the file line by line using std::getline(), which is memory efficient because it only loads one line into memory at a time.
* std::stringstream is used to parse each line into words, ensuring that memory consumption remains manageable even for large files.
* By closing the file (inputFile.close()) after reading, we release system resources promptly.