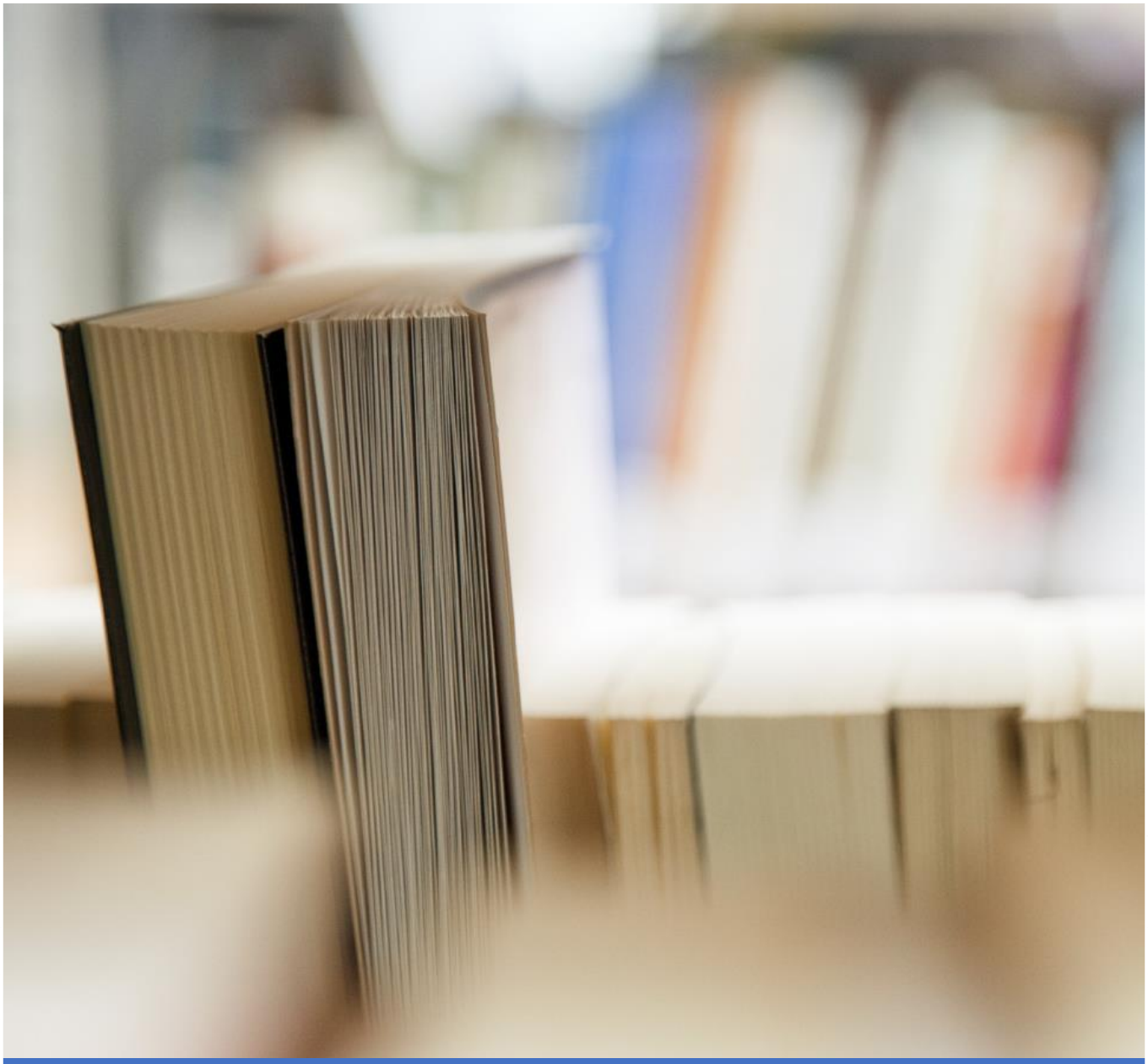

File handling with C++

Object Oriented Programming

CONTENTS

LISTS OF STANDARD FILE HANDLING CLASSES

1. ofstream: This file handling class in C++ signifies the output file stream and is applied to create files for writing information to files.
2. ifstream: This file handling class in C++ signifies the input file stream and is applied for reading information from files.
3. fstream: This file handling class in C++ generally signifies the file stream and can represent both ofstream and ifstream.

All three above classes are derived from fstreambase and the associated iostream class and are explicitly designed to handle disk files.

OPENING AND CLOSING A FILE IN C++

If programmers want to use a disk file for storing data, they need to decide the following things about the file and its intended use. These points that are to be noted are:

- A name for the file.
- Data type and structure of the file.
- Purpose (reading, writing data).
- Opening method.
- Closing the file (after use).

Files can be opened in two ways. they are:

1. Using the constructor function of the class
2. Using member function open of the class

OPENING A FILE IN C++

The first operation generally performed on an object of one of these classes to use a file is the procedure known as opening a file. An open file is represented within a program by a stream, and any input or output task performed on this stream will be applied to the physical file associated with it. The syntax of opening a file in C++ is

open (filename, mode);

There are some mode flags used for file opening. These are:

- ios::app: append mode.
- ios::ate: open a file in this mode for output and read/write control to the end of the file.
- ios::in: open a file in this mode for reading.
- ios::out: open a file in this mode for writing.
- ios::trunk: when any file already exists, its contents will be truncated before the file opening.

CLOSING A FILE IN C++

When any C++ program terminates, it automatically flushes out all the streams, releases all the allocated memory, and closes all the opened files. But it is good to use the close() function to close the file-related streams, which are a member of ifsream, ofstream, and fstream objects.

The structure of using this function is:

void close();

GENERAL FUNCTIONS USED FOR FILE HANDLING

1. open(): To create a file.
2. close(): To close an existing file.
3. get(): to read a single character from the file.
4. put(): to write a single character in the file.
5. read(): to read data from a file.
6. write(): to write data into a file.

READING FROM AND WRITING TO A FILE

While coding in C++, programmers write information to a file from the program using the stream insertion operator (<<) and reads the data using the stream extraction operator (>>). The only difference is that for files, programmers need to use an ofstream or fstream object instead of the cout object and ifstream or fstream object instead of the cin object.

Example:

```
#include <iostream>
#include <fstream.h>


void main () {
  ofstream file;
  file.open ("egone.txt");
  file << "Writing to a file in C++....";
  file.close();
  getch();
}
```

Another program for file handling in C++:

Example:

```
#include <iostream>
#include<fstream.h>


void main()
{
  char c,fn[10];
```

```
 cout<<"Enter the file name....:";
 cin>>fn;
 ifstream in(fn);
 if(!in)
 {
  cout<<"Error! File Does not Exist";
  getch();
  return;
 }
 cout<<endl<<endl;
 while(in.eof()==0)
 {
  in.get(c);
  cout<<c;
 }
 getch();
}
```

Another C++ program to print text on the console:

Example:

```
#include <iostream>
#include<fstream.h>
#include<math.h>

void main()
{
 ofstream fileo("Filethree");
 fileo<<"Hello GS";
 fileo.close();
 ifstream fin("Filethree");
 char ch;
```

```
while(fin)
{
 fin.get(ch);
 cout<<ch;
}
fin.close();
getch();
}
```

CREATE A NEW TEXT FILE AND WRITE SOME TEXT INTO IT.

```
#include <iostream>  // Include the input/output stream library
#include <fstream>   // Include the file stream library
int main() {
  // Create a new file named "test.txt"
  std::ofstream outputFile("test.txt");  // Open/create a file named "test.txt" for writing

  if (outputFile.is_open()) {  // Check if the file was successfully opened
    // Write some text into the file
    outputFile << "C++ is a high-level, general-purpose programming language created by Danish computer scientist Bjarne Stroustrup. \n";  // Write a line of text to the file
    outputFile << "First released in 1985 as an extension of the C programming language, it has since expanded significantly over time. \n";  // Write a line of text to the file
    outputFile << "Modern C++ currently has object-oriented, generic, and functional features, in addition to facilities for low-level memory manipulation.\n";  // Write a line of text to the file
    outputFile << "It is almost always implemented in a compiled language.\n";  // Write a line of text to the file
    outputFile << "Many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Embarcadero, Oracle, and IBM.";  // Write a line of text to the file

    // Close the file
    outputFile.close();  // Close the file after writing

    std::cout << "Text has been written to the file." << std::endl;  // Display a success message
  } else {
    std::cout << "Failed to create the file." << std::endl;  // Display an error message if file creation failed
  }

  return 0;  // Return 0 to indicate successful execution
}
```

OPEN AN EXISTING TEXT FILE AND DISPLAY ITS CONTENTS

```
#include <iostream>   // Including the input/output stream library

#include <fstream>    // Including the file stream library

#include <string>     // Including the string handling library

int main() {
  // Open an existing file named "test.txt"
  std::ifstream inputFile("test.txt");  // Opening the file named "test.txt" for reading

  if (inputFile.is_open()) {  // Checking if the file was successfully opened
    std::string line;  // Declaring a string variable to store each line of text

    while (std::getline(inputFile, line)) {  // Loop through each line in the file
      // Display each line on the console
      std::cout << line << std::endl;  // Output the content of 'line' to the console
    }

    inputFile.close();  // Closing the file after reading
  } else {
    std::cout << "Failed to open the file." << std::endl;  // Display an error message if file opening failed
  }

  return 0;  // Return 0 to indicate successful execution
}
```

COUNT THE NUMBER OF LINES IN A TEXT FILE.

```
#include <iostream>   // Including the input/output stream library

#include <fstream>    // Including the file stream library

#include <string>     // Including the string handling library

int main() {
  // Open the text file
  std::ifstream inputFile("test.txt");  // Opening the file named "test.txt" for reading

  if (inputFile.is_open()) {  // Checking if the file was successfully opened
    std::string line;  // Declaring a string variable to store each line of text
    int lineCount = 0;  // Initializing a variable to count lines

    while (std::getline(inputFile, line)) {  // Loop through each line in the file
      lineCount++;  // Incrementing line count for each line read
    }

    inputFile.close();  // Closing the file after counting lines

    std::cout << "Number of lines in the file: " << lineCount << std::endl;  // Outputting the
total line count
  } else {
    std::cout << "Failed to open the file." << std::endl;  // Display an error message if file
opening failed
  }

  return 0;  // Return 0 to indicate successful execution
}
```

COUNT THE NUMBER OF WORDS IN A TEXT FILE.

```
#include <iostream>     // Including the input/output stream library
#include <fstream>      // Including the file stream library
#include <string>       // Including the string handling library
#include <sstream>      // Including the stringstream library

int main() {
  std::ifstream inputFile("test.txt"); // Open the text file named "test.txt" for reading

  if (inputFile.is_open()) {  // Checking if the file was successfully opened
    std::string line;         // Declaring a string variable to store each line of text
    int wordCount = 0;        // Initializing a variable to count words

    while (std::getline(inputFile, line)) {  // Loop through each line in the file
      std::stringstream ss(line);  // Create a stringstream object with the current line content
      std::string word;  // Declare a string variable to store each word

      while (ss >> word) {  // Extract words from the stringstream
        wordCount++;  // Increment word count for each word extracted
      }
    }

    inputFile.close();  // Closing the file after counting words
    std::cout << "Number of words in the said file: " << wordCount << std::endl;  // Outputting the total word count
  } else {
    std::cout << "Failed to open the file." << std::endl;  // Display an error message if file opening failed
  }
  return 0;  // Return 0 to indicate successful execution
```

```
}
```

COPY THE CONTENTS OF ONE TEXT FILE TO ANOTHER.

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

int main() {
  // Open the input file
  std::ifstream inputFile("test.txt");    // Opening the file named "test.txt" for reading
  // Create or overwrite the output file
  std::ofstream outputFile("test_copy.txt");  // Creating/overwriting the file named "test_copy.txt" for writing

  if (inputFile.is_open() && outputFile.is_open()) {  // Checking if both input and output files were successfully opened
    std::string line;  // Declaring a string variable to store each line of text

    while (std::getline(inputFile, line)) {  // Loop through each line in the input file
      // Write each line to the output file
      outputFile << line << "\n";  // Writing each line to the output file with a newline character
    }

    inputFile.close();   // Closing the input file after copying
    outputFile.close();  // Closing the output file after copying

    std::cout << "File copied successfully." << std::endl;  // Displaying success message
  } else {
    std::cout << "Failed to open the files." << std::endl;  // Display an error message if file opening failed
  }
```

```
return 0;  // Return 0 to indicate successful execution }
```

FIND AND REPLACE A SPECIFIC WORD IN A TEXT FILE

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  std::ifstream inputFile("test.txt"); // Open the input file named "test.txt" for reading
  std::ofstream outputFile("new_test.txt"); // Create or overwrite the output file named "new_test.txt" for writing

  if (inputFile.is_open() && outputFile.is_open()) { // Check if both input and output files were successfully opened
    std::string line; // Declare a string variable to store each line of text
    std::string searchWord = "C++"; // Define the word to search for
```

```
    std::string replaceWord = "CPP"; // Define the word to replace with


    std::cout << "Search word:" << searchWord << std::endl; // Display the word to search for
    std::cout << "Replace word:" << replaceWord << std::endl; // Display the word to replace with


    std::cout << "\nBefore find and replace:" << std::endl; // Display a message before find and replace
    displayFileContent("test.txt"); // Display the content of the input file before find and replace


    while (std::getline(inputFile, line)) { // Loop through each line in the input file
      size_t pos = line.find(searchWord); // Find the position of the search word in the line


      while (pos != std::string::npos) { // Repeat until all occurrences are replaced
        line.replace(pos, searchWord.length(), replaceWord); // Replace the search word with the replace word
        pos = line.find(searchWord, pos + replaceWord.length()); // Find the next occurrence of the search word
      }
      outputFile << line << "\n"; // Write the modified line to the output file
    }


    inputFile.close(); // Close the input file
    outputFile.close(); // Close the output file


    std::cout << "After find and replace:" << std::endl; // Display a message after find and replace
    displayFileContent("new_test.txt"); // Display the content of the output file after find and replace


    std::cout << "\nWord replaced successfully." << std::endl; // Display a success message
  } else {
```

```
    std::cout << "\nFailed to open the files." << std::endl; // Display an error message if file opening failed
  }
  return 0; // Return 0 to indicate successful execution  }
```

APPEND NEW DATA TO AN EXISTING TEXT FILE.

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  displayFileContent("new_test.txt"); // Display content of "new_test.txt" before any modification
  std::cout << std::endl;

  std::ofstream outputFile; // Declare an output file stream object
  // Open the file in append mode
  outputFile.open("new_test.txt", std::ios::app); // Open "new_test.txt" in append mode
```

```
  displayFileContent("new_test.txt"); // Display content of "new_test.txt" after opening in append mode
  std::cout << std::endl;

  if (outputFile.is_open()) { // Check if the file was successfully opened
    std::string newData; // Declare a string to store new data entered by the user

    std::cout << "Enter the data to append: "; // Prompt the user to enter data
    // Read the new data from the user
    std::getline(std::cin, newData); // Get user input for new data

    // Append the new data to the file
    outputFile << newData << std::endl; // Write the new data to the file
    outputFile.close(); // Close the file

    std::cout << "Data appended successfully." << std::endl; // Display a success message
    displayFileContent("new_test.txt"); // Display content of "new_test.txt" after appending data
    std::cout << std::endl;
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }

  return 0; // Return 0 to indicate successful execution
}
```

SORT THE LINES OF A TEXT FILE IN ALPHABETICAL ORDER.

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <vector>     // Including the vector container
#include <algorithm>  // Including algorithms like 'sort'
#include <iterator>   // Including iterator operations

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename);  // Open file with given filename for reading
  std::string line;  // Declare a string to store each line of text

  if (file.is_open()) {  // Check if the file was successfully opened
    std::cout << "File content:" << std::endl;  // Displaying a message indicating file content
    while (std::getline(file, line)) {  // Read each line from the file
      std::cout << line << std::endl;  // Display each line of the file
    }
    file.close();  // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl;  // Display an error message if file opening failed
  }
}

int main() {
  std::ifstream inputFile("test1.txt");  // Open the input file named "test1.txt" for reading
  displayFileContent("test1.txt");  // Display content of "test1.txt"
  std::ofstream outputFile("sorted_test1.txt");  // Create or overwrite the output file named "sorted_test1.txt" for writing
```

```
  if (inputFile.is_open() && outputFile.is_open()) {  // Check if both input and output files were successfully opened
    std::vector<std::string> lines;  // Vector to store the lines of the file
    std::string line;  // Declare a string variable to store each line of text

    while (std::getline(inputFile, line)) {  // Read each line from the input file and store it in the vector
      lines.push_back(line);  // Push each line to the vector
    }

    std::sort(lines.begin(), lines.end());  // Sort the lines in alphabetical order

    std::copy(lines.begin(), lines.end(), std::ostream_iterator<std::string>(outputFile, "\n"));  // Write the sorted lines to the output file

    inputFile.close();  // Close the input file
    outputFile.close();  // Close the output file

    std::cout << "\nLines sorted successfully.\n" << std::endl;  // Display a success message
    displayFileContent("sorted_test1.txt");  // Display content of "sorted_test1.txt"
  } else {
    std::cout << "\nFailed to open the files." << std::endl;  // Display an error message if file opening failed
  }

  return 0;  // Return 0 to indicate successful execution
}
```

MERGE MULTIPLE TEXT FILES INTO A SINGLE FILE

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library
#include <vector>     // Including the vector container

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  std::vector<std::string> inputFiles = { // List of input files
    "test1.txt",
    "test2.txt",
    "test3.txt",
    "test4.txt"
  };
```

```
  std::cout << "Content of test1.txt, test2.txt, test3.txt, text4.txt: " << std::endl;

  displayFileContent("test1.txt"); // Display content of "test1.txt"
  displayFileContent("test2.txt"); // Display content of "test2.txt"
  displayFileContent("test3.txt"); // Display content of "test3.txt"
  displayFileContent("test4.txt"); // Display content of "test4.txt"

  std::string outputFile = "merged_test_file.txt"; // Output file

  std::ofstream mergedFile(outputFile); // Create or overwrite the output file named
"merged_test_file.txt" for writing

  if (mergedFile.is_open()) { // Check if the output file was successfully opened
    for (const auto & inputFile: inputFiles) { // Iterate through each input file
      std::ifstream inputFileStream(inputFile); // Open each input file for reading

      if (inputFileStream.is_open()) { // Check if the input file was successfully opened
        std::string line; // Declare a string to store each line of text

        while (std::getline(inputFileStream, line)) { // Read each line from the input file
          mergedFile << line << "\n"; // Write each line to the merged file
        }

        inputFileStream.close(); // Close the input file
      } else {
        std::cout << "Failed to open input file: " << inputFile << std::endl; // Display an error
message if file opening failed
      }
    }

    mergedFile.close(); // Close the merged file
```

```
    std::cout << "\nFiles merged successfully." << std::endl; // Display a success message
    std::cout << "\nContent of the merged file:" << std::endl;
    displayFileContent("merged_test_file.txt"); // Display content of "merged_test_file.txt"
  } else {
    std::cout << "Failed to open the output file." << std::endl; // Display an error message if output file opening failed
  }

  return 0; // Return 0 to indicate successful execution
}
```

SPLIT A LARGE TEXT FILE INTO SMALLER FILES OF EQUAL SIZE.

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library
#include <vector>     // Including the vector container

// Function to split a file into smaller chunks
void splitFile(const std::string & inputFile, const std::string & outputPrefix, int chunkSize) {
  // Open the input file in binary mode
  std::ifstream input(inputFile, std::ios::binary); // Open the input file in binary mode

  if (input.is_open()) { // Check if the input file was successfully opened
    // Get the file size
    input.seekg(0, std::ios::end); // Move the file pointer to the end of the file
    std::streampos fileSize = input.tellg(); // Get the current position of the file pointer, indicating the file size
    input.seekg(0, std::ios::beg); // Move the file pointer back to the beginning of the file

    // Calculate the number of chunks
    int numChunks = (fileSize + chunkSize - 1) / chunkSize; // Calculate the number of chunks based on file size and chunk size

    // Read and write each chunk
    for (int i = 0; i < numChunks; ++i) { // Iterate through each chunk
      // Create or overwrite the output file with an incremental suffix
      std::ofstream output(outputPrefix + std::to_string(i + 1) + ".txt", std::ios::binary); // Create or overwrite the output file

      if (output.is_open()) { // Check if the output file was successfully opened
        std::vector<char> buffer(chunkSize); // Create a buffer to hold the chunk data
```

```
        // Read a chunk of data from the input file
        input.read(buffer.data(), chunkSize); // Read chunkSize number of bytes into the buffer

        // Write the chunk to the output file
        output.write(buffer.data(), input.gcount()); // Write the read data from the buffer to the output file

        output.close(); // Close the output file
      } else {
        std::cout << "Failed to open output file: " << outputPrefix + std::to_string(i + 1) + ".txt" << std::endl; // Display an error message if output file opening failed
      }
    }

    input.close(); // Close the input file

    std::cout << "File split successfully." << std::endl; // Display a success message after splitting
  } else {
    std::cout << "Failed to open the input file." << std::endl; // Display an error message if input file opening failed
  }
}

int main() {
  std::string inputFile = "merged_test_file.txt"; // Input file
  std::string outputPrefix = "part_"; // Prefix for output files
  int chunkSize = 400; // Chunk size in bytes
  splitFile(inputFile, outputPrefix, chunkSize); // Call the function to split the file
  return 0; // Return 0 to indicate successful execution
}
```

SEARCH FOR A SPECIFIC STRING IN A TEXT FILE AND DISPLAY ITS LINE NUMBER(S)

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library
#include <vector>     // Including the vector container

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

// Function to search for a string in a file and display line numbers where it is found
void searchAndDisplayLineNumbers(const std::string & filename, const std::string & searchStr) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text
  std::vector<int> lineNumbers; // Vector to store line numbers where the search string is found
```

```
  int lineNumber = 1; // Variable to track the current line number


  while (std::getline(file, line)) { // Read each line from the file
    if (line.find(searchStr) != std::string::npos) { // Check if the search string is found in the line
      lineNumbers.push_back(lineNumber); // Store the line number where the search string is found
    }
    lineNumber++; // Increment the line number counter
  }


  file.close(); // Close the file after reading


  if (!lineNumbers.empty()) { // Check if any line numbers were stored
    std::cout << "String \"" << searchStr << "\" found at line number(s): "; // Display a message indicating the search string was found


    for (int i = 0; i < lineNumbers.size(); ++i) { // Loop through the stored line numbers
      std::cout << lineNumbers[i]; // Display each line number


      if (i != lineNumbers.size() - 1) {
        std::cout << ", "; // Display a comma between line numbers, except for the last one
      }
    }


    std::cout << std::endl; // Move to the next line after displaying line numbers
  } else {
    std::cout << "String \"" << searchStr << "\" not found in the file." << std::endl; // Display a message indicating the search string was not found
  }
}
```

```
int main() {
  std::string filename = "test.txt"; // File to search
  displayFileContent("new_test.txt"); // Display content of "new_test.txt"
  std::cout << std::endl; // Output a newline for formatting

  std::string searchStr = "currently"; // String to search
  searchAndDisplayLineNumbers(filename, searchStr); // Search for the string in the file and display line numbers where it's found

  return 0; // Return 0 to indicate successful execution
}
```

ENCRYPT THE CONTENTS OF A TEXT FILE USING A SIMPLE ENCRYPTION ALGORITHM

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

// Function to encrypt a file using a simple algorithm (incrementing ASCII values)
void encryptFile(const std::string & inputFile, const std::string & outputFile) {
  std::ifstream input(inputFile); // Open input file for reading
  std::ofstream output(outputFile); // Open or create output file for writing

  if (input.is_open() && output.is_open()) { // Check if both files were successfully opened
    char ch; // Declare a character variable to read characters from the input file
```

```
    while (input.get(ch)) { // Loop through each character in the input file
      ch++; // Simple encryption algorithm: Increment ASCII value by 1
      output.put(ch); // Write the encrypted character to the output file
    }

    input.close(); // Close the input file
    output.close(); // Close the output file

    std::cout << "File encrypted successfully.\n" << std::endl; // Display a success message
  } else {
    std::cout << "Failed to open the files.\n" << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  std::string inputFile = "test.txt"; // Input file
  displayFileContent("test.txt"); // Display content of "test.txt"
  std::cout << std::endl; // Output a newline for formatting

  std::string outputFile = "encrypted_test.txt"; // Output file for encrypted content
  encryptFile(inputFile, outputFile); // Encrypt "test.txt" and write to "encrypted_test.txt"
  displayFileContent("encrypted_test.txt"); // Display content of "encrypted_test.txt"
  std::cout << std::endl; // Output a newline for formatting

  return 0; // Return 0 to indicate successful execution
}
```

DECRYPT THE CONTENTS OF A TEXT FILE ENCRYPTED USING THE ABOVE ALGORITHM

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

// Function to display the content of a file
void displayFileContent(const std::string & filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // Declare a string to store each line of text

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

// Function to decrypt a file using a simple algorithm (decrementing ASCII values)
void decryptFile(const std::string & inputFile, const std::string & outputFile) {
  std::ifstream input(inputFile); // Open input file for reading
  std::ofstream output(outputFile); // Open or create output file for writing

  if (input.is_open() && output.is_open()) { // Check if both files were successfully opened
```

```
    char ch; // Declare a character variable to read characters from the input file

    while (input.get(ch)) { // Loop through each character in the input file
      ch--; // Simple decryption algorithm: Decrement ASCII value by 1
      output.put(ch); // Write the decrypted character to the output file
    }

    input.close(); // Close the input file
    output.close(); // Close the output file

    std::cout << "File decrypted successfully.\n" << std::endl; // Display a success message
  } else {
    std::cout << "Failed to open the files.\n" << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  std::string inputFile = "encrypted_test.txt"; // Input file (encrypted)
  displayFileContent("encrypted_test.txt"); // Display content of "encrypted_test.txt"
  std::cout << std::endl; // Output a newline for formatting

  std::string outputFile = "decrypted_test.txt"; // Output file (decrypted)
  decryptFile(inputFile, outputFile); // Decrypt "encrypted_test.txt" and write to "decrypted_test.txt"
  displayFileContent("decrypted_test.txt"); // Display content of "decrypted_test.txt"
  std::cout << std::endl; // Output a newline for formatting

  return 0; // Return 0 to indicate successful execution
}
```

READ A CSV FILE AND DISPLAY ITS CONTENTS IN TABULAR FORM

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library
#include <vector>     // Including the vector container library
#include <sstream>    // Including the string stream library

// Function to split a string into tokens based on a delimiter
std::vector<std::string> splitString(const std::string &str, char delimiter) {
  std::vector<std::string> tokens; // Vector to store the split tokens
  std::stringstream ss(str); // Creating a string stream from the input string
  std::string token; // String to store each token

  // Extract tokens using the specified delimiter
  while (std::getline(ss, token, delimiter)) {
    tokens.push_back(token); // Store each token in the vector
  }

  return tokens; // Return the vector of tokens
}

// Function to display the CSV file contents in tabular form
void displayCSVContents(const std::string &filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // String to store each line of the file

  if (file.is_open()) { // Check if the file was successfully opened
    while (std::getline(file, line)) { // Read each line from the file
      std::vector<std::string> tokens = splitString(line, ','); // Split the line into tokens based on comma delimiter
```

```
      for (const std::string &token : tokens) { // Loop through each token in the line
        std::cout << token << "\t"; // Display each token followed by a tab
      }
      std::cout << std::endl; // Output a newline after displaying all tokens in a line
    }

    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
  }
}

int main() {
  std::string filename = "test.csv"; // CSV file to read

  displayCSVContents(filename); // Display contents of the CSV file in tabular form

  return 0; // Return 0 to indicate successful execution
}
```

CALCULATE THE AVERAGE OF NUMBERS STORED IN A FILE

```
#include <iostream>   // Including the input/output stream library
#include <fstream>    // Including the file stream library
#include <string>     // Including the string handling library

// Function to display the content of a file
void displayFileContent(const std::string &filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  std::string line; // String to store each line of the file

  if (file.is_open()) { // Check if the file was successfully opened
    std::cout << "File content:" << std::endl; // Displaying a message indicating file content
    while (std::getline(file, line)) { // Read each line from the file
      std::cout << line << std::endl; // Display each line of the file
    }
    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file
opening failed
  }
}

// Function to calculate the average of numbers from a file
double calculateAverage(const std::string &filename) {
  std::ifstream file(filename); // Open file with given filename for reading
  double sum = 0.0; // Variable to store the sum of numbers
  double count = 0.0; // Variable to count the numbers

  if (file.is_open()) { // Check if the file was successfully opened
    double number; // Variable to store each number read from the file
```

```
    while (file >> number) { // Read each number from the file
      sum += number; // Add the number to the sum
      count++; // Increment the count of numbers
    }

    file.close(); // Close the file
  } else {
    std::cout << "Failed to open the file." << std::endl; // Display an error message if file opening failed
    return 0.0; // Return 0.0 if file opening failed
  }

  if (count > 0) { // Check if numbers were found in the file
    return sum / count; // Return the average of the numbers
  } else {
    std::cout << "No numbers found in the file." << std::endl; // Display a message if no numbers were found
    return 0.0; // Return 0.0 if no numbers were found
  }
}

int main() {
  std::string filename = "sample.txt"; // File containing numbers
  displayFileContent("sample.txt"); // Display content of "sample.txt"
  std::cout << std::endl; // Output a newline for formatting

  double average = calculateAverage(filename); // Calculate the average of numbers in the file
  std::cout << "Average: " << average << std::endl; // Display the calculated average
  return 0; // Return 0 to indicate successful execution
}
```

COURTESY : W3SCHOOLS