**Date:11/07/2024**

**Day\_14\_Assignments**

Problem 1: List Operations

Description:

Write a program that uses the std::list container to manage a collection of integers. Your program should perform the following operations:

Insert elements at the front and back of the list.

Remove elements from the front and back of the list.

Sort the list in ascending and descending order.

Reverse the list.

Display the elements of the list.

#include <iostream>

#include <list>

#include <algorithm>

void display(const std::list<int>& lst) {

for (const int& elem : lst) {

std::cout << elem << " ";

}

std::cout << std::endl;

}

int main() {

std::list<int> myList;

// Insert elements at the front and back

myList.push\_back(5);

myList.push\_front(10);

myList.push\_back(15);

myList.push\_front(20);

std::cout << "List after inserting elements at front and back: ";

display(myList);

// Remove elements from the front and back

myList.pop\_front();

myList.pop\_back();

std::cout << "List after removing elements from front and back: ";

display(myList);

// Sort the list in ascending order

myList.sort();

std::cout << "List after sorting in ascending order: ";

display(myList);

// Sort the list in descending order

myList.sort(std::greater<int>());

std::cout << "List after sorting in descending order: ";

display(myList);

// Reverse the list

myList.reverse();

std::cout << "List after reversing: ";

display(myList);

return 0;

}

Problem 2: Vector Manipulation

Description:

Create a program that uses the std::vector container to store a collection of floating-point numbers. The program should:

Add elements to the vector.

Remove elements from a specified position.

Find the maximum and minimum elements in the vector.

Calculate the average of the elements.

Display the elements of the vector.

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

void display(const std::vector<float>& vec) {

for (const float& elem : vec) {

cout << elem << " ";

}

cout << endl;

}

int main() {

std::vector<float>Vector;

Vector.push\_back(5.5);

Vector.push\_back(1.1);

Vector.push\_back(18.2);

Vector.push\_back(27.8);

cout << "Vector after adding elements: ";

display(Vector);

Vector.erase(Vector.begin() + 2);

cout << "Vector after removing element at position 2: ";

display(Vector);

auto maxElem = \*max\_element(Vector.begin(), Vector.end());

auto minElem = \*min\_element(Vector.begin(), Vector.end());

cout << "Maximum element: " << maxElem <<endl;

cout << "Minimum element: " << minElem <<endl;

float sum = std::accumulate(Vector.begin(), Vector.end(), 0.0f);

float average = sum /Vector.size();

cout << "Average of elements: " << average <<endl;

cout << "Final vector elements: ";

display(Vector);

return 0;

}

Problem 3: Queue Simulation

Description:

Implement a program using the std::queue container to simulate a ticketing system. The program should:

Add customers to the queue.

Serve customers (remove from front of the queue).

Display the current queue.

Display the number of customers served.

#include <iostream>

#include <queue>

using namespace std;

void display(queue<string> q) {

while (!q.empty()) {

cout << q.front() << " ";

q.pop();

}

cout<<endl;

}

int main() {

queue<string> ticket;

int custServed = 0;

ticket.push("Cust 1");

ticket.push("Cust 2");

ticket.push("Cust 3");

cout << "Current queue: "<<endl;

display(ticket);

while (!ticket.empty()) {

cout << ticket.front() << "served"<< endl;

ticket.pop();

custServed++;

cout << "Queue after serving a customer:";

display(ticket);

}

cout << "Total number of customers served:" << custServed <<endl;

return 0;

}

problem 4 :

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

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

if (!inputFile) {

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

return 1;

}

std::string line;

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

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

}

inputFile.close();

return 0;

}

To use the ifstream class in C++ to open a file for reading. std::ifstream inputFile("input.txt"); is one example.

In C++, reading from files is done via the ifstream class. It offers a file stream input interface for operations.

It can use the operator! or fail() method of the ifstream object to determine whether a file was successfully opened.

if (!inputFile) or if (inputFile.fail()), for instance. The std::getline function is used to read a line from a file.

std::getline(inputFile, line); is one example.

After reading a file, it calls the close function on the ifstream object to close it correctly. Take inputFile.close(); as an example.

In C++, you can open a file for reading using the ifstream class from the <fstream> library. Here are the steps to open a file for reading:

1. **Include the <fstream> header**: This header contains the definitions for file stream classes like ifstream.
2. **Create an ifstream object**: This object will be used to handle the input file stream.
3. **Open the file**: You can open the file either by passing the filename to the ifstream constructor or by calling the open member function on the ifstream object.
4. **Check if the file is open**: It's good practice to check if the file was opened successfully.

Problem 5

problem 2 :

\*CODE:

#include <iostream>

#include <fstream>

#include <string>

int main() {

// Open a file for writing

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

// Check if the file was successfully opened

if (!outputFile) {

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

return 1;

}

// Write lines to the file

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

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

// Properly close the file after writing

outputFile.close();

return 0;

}

Use the ofstream class in C++ to open a file for writing. std::ofstream outputFile("output.txt"); is one example.

Writing to files in C++ is done via the ofstream class. It offers a file stream output interface for operations.

can check the operator! or fail() method of the ofstream object to handle failures in the event that the file cannot be opened for writing.

if (!outputFile) or if (outputFile.fail()), for instance.

In C++, you use the \\ operator with the ofstream object to write a string to a file.

Take outputFile \\ "Hello, world!" \\ std::endl; as an example.

It's crucial to close a file after writing to it in order to ensure that resources are released,

all data is correctly flushed from the buffer to the file, and no data corruption happens.

It also opens the file for access by other processes or applications.

problem 6 :

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

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

if (!logFile) {

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

return 1;

}

logFile << "New log entry." << std::endl;

logFile.close();

return 0;

}

1. Use the ofstream class in C++ to open a file for appending

while in the std::ios\_base::app mode. std::ofstream logFile("log.txt", std::ios\_base::app); is one example.

2.When a file is opened in write mode (std::ios\_base::out), any existing file will be truncated, meaning its contents will be removed.

If a file is opened in append mode (std::ios\_base::app),

it will create it if it doesn't already exist, preserve its contents, and append new data at the end.

3. Users must open a file in the std::ios\_base::app mode in order to append data to it using the ofstream class.

std::ofstream logFile("log.txt", std::ios\_base::app); is one example.

4. The file will be created if it doesn't already exist when you try to open it in add mode.

5. Use file locks or other synchronization techniques as an option if more than one process or thread is accessing the file at once.

Address any mistakes that can arise when operating on the file.

problem 7 :

#include <iostream>

#include <fstream>

#include <string>

int main() {

// Open the input file

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

// Check if the file was successfully opened

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

std::cerr << "Failed to open the file." << std::endl;

return 1;

}

// Initialize variables

std::string word;

int wordCount = 0;

// Read and count words from the file

while (inputFile >> word) {

wordCount++;

}

// Close the file

inputFile.close();

// Output the word count

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

return 0;

}

Including the Necessary Headers:

#include <iostream>: For console input and output.

#include <fstream>: For file stream operations.

#include <string>: For using the std::string class.

Opening the File:

std::ifstream inputFile("data.txt");: Creates an ifstream object and attempts to open data.txt for reading.

Checking if the File was Successfully Opened:

if (!inputFile.is\_open()) { ... }: Checks if the file was successfully opened. If not, an error message is printed and the program exits.

Counting Words in the File:

while (inputFile >> word) { ... }: This loop reads words from the file using the >> operator and increments the wordCount for each word read.

Closing the File:

inputFile.close();: Closes the file after reading is done to free up resources.

Outputting the Word Count:

std::cout << "Number of words in the file: " << wordCount << std::endl;: Displays the total number of words counted from the file.

This program efficiently counts the words in the file data.txt without loading the entire file into memory at once, making it suitable for handling large files as well. Adjustments can be made for different delimiters or specific word definitions as needed.