**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.

A: #include <iostream>

#include <list>

using namespace std;

void displayList(const list<int>& lst) {

for (int val : lst) {

cout << val << " ";

}

cout << endl; }

int main() {

list<int> myList; // Insert elements at the front and back

myList.push\_front(10);

myList.push\_front(20);

myList.push\_back(30);

myList.push\_back(40);

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

displayList(myList);

myList.pop\_front();

myList.pop\_back(); // Remove elements from the front and back

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

displayList(myList);

myList.sort();

cout << "List sorted in ascending order: "; // Sort the list in ascending order

displayList(myList);

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

cout << "List sorted in descending order: "; // Sort the list in descending order

displayList(myList);

myList.reverse();

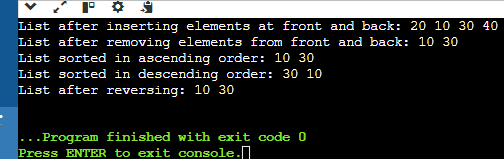
cout << "List after reversing: ";

displayList(myList);

return 0;

}

OUTPUT:



**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.

A: #include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

int main() {

std::vector<float> vec;

vec.push\_back(10.5);

vec.push\_back(20.2);

vec.push\_back(5.8);

vec.push\_back(30.1);

vec.push\_back(15.9);

cout << "Initial vector: ";

for (std::vector<float>::iterator it = vec.begin(); it != vec.end(); ++it) {

cout << \*it << " ";

}

cout << endl;

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

cout << "After removing element at position 2: ";

for (std::vector<float>::iterator it = vec.begin(); it != vec.end(); ++it) {

cout << \*it << " ";

}

cout << endl;

float max = \*std::max\_element(vec.begin(), vec.end()); // Find maximum and minimum elements

float min = \*std::min\_element(vec.begin(), vec.end());

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

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

float sum = std::accumulate(vec.begin(), vec.end(), 0.0); // Calculate average

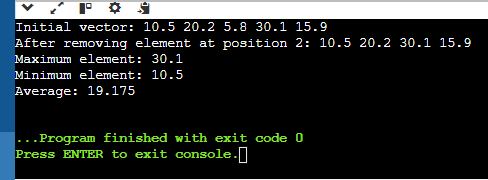
float average = sum / vec.size();

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

return 0;

}

OUTPUT:



**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>

#include <string>

using namespace std;

int main() {

std::queue<std::string> customerQueue;

int customersServed = 0;

// Add customers to the queue

customerQueue.push("A");

customerQueue.push("B");

customerQueue.push("C");

customerQueue.push("D");

customerQueue.push("E");

// Display the current queue

cout << "Initial Queue: ";

std::queue<std::string> tempQueue = customerQueue;

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

cout << endl;

// Serve customers (remove from front of the queue)

while (!customerQueue.empty()) {

cout << "Serving: " << customerQueue.front() << endl;

customerQueue.pop();

customersServed++;

}

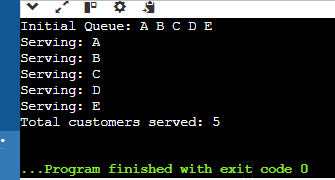
// Display the number of customers served

cout << "Total customers served: " << customersServed << endl;

return 0;

}

OUTPUT:



**Problem 4: Stack Operations**

Description:

Write a program using the std::stack container to evaluate a postfix expression. The program should:

Read a postfix expression.

Use a stack to evaluate the expression.

Display the result of the evaluation.

A: #include <iostream>

#include <stack>

#include <sstream>

#include <string>

using namespace std;

// Function to evaluate a postfix expression

int evaluatePostfix(const string& expression) {

stack<int> s;

istringstream iss(expression);

string token;

while (iss >> token) {

if (isdigit(token[0]) || (token[0] == '-' && token.size() > 1)) {

s.push(stoi(token));

} else {

int op2 = s.top(); s.pop();

int op1 = s.top(); s.pop();

if (token == "+") s.push(op1 + op2);

else if (token == "-") s.push(op1 - op2);

else if (token == "\*") s.push(op1 \* op2);

else if (token == "/") s.push(op1 / op2); }

}

return s.top(); }

int main() {

string postfixExpression;

cout << "Enter a postfix expression: ";

getline(cin, postfixExpression);

try {

int result = evaluatePostfix(postfixExpression);

cout << "The result of the evaluation is: " << result << endl;

} catch (const exception& e) {

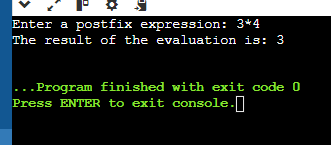
cerr << "Error: " << e.what() << endl;

}

return 0;

}

OUTPUT:



------------------------------------------------------------------------------------------------------------------------------------------

### 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.

### A: #include <iostream>

### #include <fstream>

### #include <string>

### using namespace std;

### int main() {

### ifstream inputFile("input.txt");

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

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

### return 1; // Return with an error code

### } string line;

### while (getline(inputFile, line)) { // Read the file line by line and print to the console

### cout << line << endl;

### }

### inputFile.close();

### return 0;

### }

### OUTPUT:

### 

### Questions:

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

### A: You open a file for reading in C++ using the ifstream class.

### ifstream inputFile("input.txt");

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

### A: The ifstream class is used to read data from files. It stands for input file stream.

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

### A: You can check if a file was successfully opened by calling the is\_open() member function on the ifstream object.

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

### A: To read a line from a file, you use the getline() function.

### 5. How do you properly close a file after reading?

### A: You properly close a file after reading by calling the close() member function on the ifstream object.

### 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.

### #include <iostream>

### #include <fstream>

### using namespace std;

### int main() {

### ofstream outputFile("output.txt");

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

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

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

### outputFile.close();

### cout << "Content written to the file successfully." << endl;

### } else {

### cout << "Error opening file for writing." << endl;

### }

### return 0;

### }

### OUTPUT:

### 

### Questions:

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

### A: Use the ofstream class to create an output file stream and open the file.

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

### A: The ofstream class is used for writing to files. It provides facilities to write data in various formats.

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

### A: Check if the file stream is successfully opened using the is\_open() member function.

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

### A: Use the insertion operator (<<) to write strings to the file.

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

### A: Closing a file ensures that all data is properly flushed to the file and resources are released.

### 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.

### #include <iostream>

### #include <fstream>

### using namespace std;

### int main() {

### ofstream logFile("log.txt", ios::app);

### if (logFile.is\_open()) {

### logFile << "New log entry." << endl;

### logFile.close();

### cout << "Content appended to the file successfully." << endl;

### } else {

### cout << "Error opening file for appending." << endl;

### }

### return 0;

### }

### OUTPUT:

### 

### Questions:

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

### A: Use the ofstream class with the ios::app flag to open the file in append mode.

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

### A: Write mode (ios::out) truncates the existing content, while append mode (ios::app) preserves the existing content and adds new content at the end.

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

### A: Open the file using ofstream with the ios::app flag.

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

### A:The file will be created if it does not exist.

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

### A: Ensure that the file is successfully opened before appending and properly close the file after the operation.

### 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 <iostream>

### #include <fstream>

### #include <string>

### using namespace std;

### int main() {

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

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

### if (sourceFile.is\_open() && destFile.is\_open()) {

### destFile << sourceFile.rdbuf();

### sourceFile.close();

### destFile.close();

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

### } else {

### cout << "Error opening source or destination file." << endl; }

### return 0;

### }

### OUTPUT:

### 

### Questions:

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

### A: Use ifstream to read from the source file and ofstream to write to the destination file.

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

### Use the rdbuf() method to copy the file contents in binary mode.

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

### A: Errors in opening the source or destination file and errors during the read/write operations.

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

### A: The ifstream automatically sets the EOF flag when the end of the file is reached, which can be checked using the eof() member function.

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

### A: Use the close() member function for both ifstream and ofstream objects.

### 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 <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream inputFile("data.txt");

if (inputFile.is\_open()) {

string word;

int wordCount = 0;

while (inputFile >> word) {

wordCount++; }

inputFile.close();

cout << "Total number of words: " << wordCount << endl;

} else {

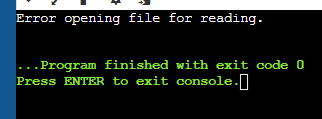
cout << "Error opening file for reading." << endl;

}

return 0;

}

OUTPUT:



Questions:

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

A: A word is typically defined as a sequence of characters separated by whitespace (spaces, newlines, etc.).

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

A: Use the extraction operator (>>) to read words from the file.

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

A: The extraction operator (>>) automatically handles spaces, newlines, and other whitespace as delimiters.

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

A: Increment a counter variable each time a word is successfully read.

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

A: Since only one word is read into memory at a time, large files are handled efficiently without significant memory usage.