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>

using namespace std;

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

for (int val : list) {

cout << val << " ";

}

cout << endl;

}

int main() {

list<int> myList;

myList.push\_back(5);

myList.push\_back(10);

myList.push\_front(3);

myList.push\_back(15);

cout << "Initial list: ";

displayList(myList);

myList.pop\_front();

myList.pop\_back();

cout << "After removing front and back: ";

displayList(myList);

myList.sort();

cout << "Sorted in ascending order: ";

displayList(myList);

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

cout << "Sorted in descending order: ";

displayList(myList);

myList.reverse();

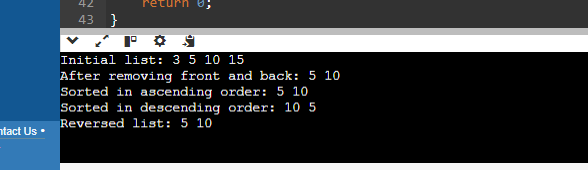
cout << "Reversed list: ";

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.

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

int main() {

vector<float> numbers;

numbers.push\_back(34.8);

numbers.push\_back(29.7);

numbers.push\_back(37.1);

numbers.push\_back(48.8);

numbers.push\_back(60.3);

cout << "Initial vector: ";

for (float num : numbers) {

cout << num << " ";

}

cout << endl;

numbers.erase(numbers.begin() + 3);

std::cout << "Vector after removal: ";

for (float num : numbers) {

cout << num << " ";

}

cout << endl;

float max\_element = \*max(numbers.begin(), numbers.end());

float min\_element = \*min(numbers.begin(), numbers.end());

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

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

float sum = 0.0;

for (float num : numbers) {

sum += num;

}

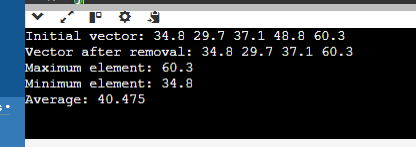
float average = sum / numbers.size();

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() {

queue<string> customerQueue;

int customersServed = 0;

customerQueue.push("Souvik");

customerQueue.push("Baban");

customerQueue.push("Ajit");

customerQueue.push("Arka");

customerQueue.push("DoctorDa");

cout << "Initial queue: ";

queue<string> tempQueue = customerQueue;

while (!tempQueue.empty()) {

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

tempQueue.pop();

}

cout << endl;

while (!customerQueue.empty()) {

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

customerQueue.pop();

customersServed++;

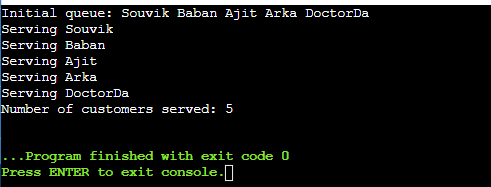
}

cout << "Number of 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.

#include <iostream>

#include <stack>

#include <sstream>

#include <string>

using namespace std;

bool isOperator(const char& token) {

return token == '+'||token == '-'||token == '\*'||token == '/';

}

float applyOperation(const char& operation, float operand1, float operand2) {

switch (operation) {

case '+': return operand1 + operand2;

case '-': return operand1 - operand2;

case '\*': return operand1 \* operand2;

case '/': return operand1 / operand2;

default: throw invalid\_argument("Invalid operation");

}

}

float evaluatePostfixExpression(const std::string& expression) {

stack<float> stack;

istringstream tokens(expression);

string token;

while (tokens >> token) {

if (isOperator(token[0]) && token.size() == 1) {

if (stack.size() < 2) {

throw invalid\_argument("Invalid postfix expression");

}

float operand2 = stack.top(); stack.pop();

float operand1 = stack.top(); stack.pop();

float result = applyOperation(token[0], operand1, operand2);

stack.push(result);

}

else {

stack.push(stof(token));

}

}

if (stack.size() != 1) {

throw invalid\_argument("Invalid postfix expression");

}

return stack.top();

}

void processPostfixExpression() {

string postfixExpression;

cout << "Enter a postfix expression: ";

getline(cin, postfixExpression);

try {

float result = evaluatePostfixExpression(postfixExpression);

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

} catch (const std::exception& e) {

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

}

}

int main() {

processPostfixExpression();

return 0;

}

Output:

