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> // for std::sort

using namespace std;

int main() {

// Create a list to store integers

list<int> myList;

// 1. Insert elements at the front and back of the list

myList.push\_back(10); // Insert 10 at the back

myList.push\_front(20); // Insert 20 at the front

myList.push\_back(30); // Insert 30 at the back

myList.push\_front(40); // Insert 40 at the front

// Display the list after insertions

cout << "List after insertions: ";

for (int elem : myList) {

cout << elem << " ";

}

cout << endl;

// 2. Remove elements from the front and back of the list

myList.pop\_front(); // Remove element from the front

myList.pop\_back(); // Remove element from the back

// Display the list after removals

cout << "List after removals: ";

for (int elem : myList) {

cout << elem << " ";

}

cout << endl;

// 3. Sort the list in ascending order

myList.sort(); // Sort the list in ascending order

// Display the list after sorting in ascending order

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

for (int elem : myList) {

cout << elem << " ";

}

cout << endl;

// 4. Sort the list in descending order

myList.sort(greater<int>()); // Sort the list in descending order

// Display the list after sorting in descending order

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

for (int elem : myList) {

cout << elem << " ";

}

cout << endl;

// 5. Reverse the list

myList.reverse(); // Reverse the order of elements in the list

// Display the list after reversing

cout << "List after reversing: ";

for (int elem : myList) {

cout << elem << " ";

}

cout << endl;

return 0;

}

2. 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> // for std::max\_element, std::min\_element

#include <numeric> // for std::accumulate

using namespace std;

int main() {

// Create a vector to store floating-point numbers

vector<float> myVector;

// 1. Add elements to the vector

myVector.push\_back(1.1f); // Add 1.1 to the vector

myVector.push\_back(2.2f); // Add 2.2 to the vector

myVector.push\_back(3.3f); // Add 3.3 to the vector

myVector.push\_back(4.4f); // Add 4.4 to the vector

// Display the vector after adding elements

cout << "Vector after adding elements: ";

for (float elem : myVector) {

cout << elem << " ";

}

cout << endl;

// 2. Remove elements from a specified position

int position = 2; // Position to remove (0-based index)

if (position < myVector.size()) {

myVector.erase(myVector.begin() + position); // Remove element at the specified position

}

// Display the vector after removing an element

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

for (float elem : myVector) {

cout << elem << " ";

}

cout << endl;

// 3. Find the maximum and minimum elements in the vector

if (!myVector.empty()) {

auto maxElement = \*max\_element(myVector.begin(), myVector.end()); // Find the maximum element

auto minElement = \*min\_element(myVector.begin(), myVector.end()); // Find the minimum element

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

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

}

// 4. Calculate the average of the elements

if (!myVector.empty()) {

float sum = accumulate(myVector.begin(), myVector.end(), 0.0f); // Calculate the sum of elements

float average = sum / myVector.size(); // Calculate the average

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

}

// 5. Display the elements of the vector

cout << "Final vector elements: ";

for (float elem : myVector) {

cout << elem << " ";

}

cout << endl;

return 0;

}

3. 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;

int main() {

// Create a queue to store customer IDs

queue<int> ticketQueue;

// Number of customers served

int customersServed = 0;

// 1. Add customers to the queue

ticketQueue.push(1); // Customer 1 added to the queue

ticketQueue.push(2); // Customer 2 added to the queue

ticketQueue.push(3); // Customer 3 added to the queue

ticketQueue.push(4); // Customer 4 added to the queue

// Display the current queue

cout << "Current queue: ";

queue<int> tempQueue = ticketQueue; // Create a temporary queue to display the contents

while (!tempQueue.empty()) {

cout << tempQueue.front() << " "; // Display the front customer

tempQueue.pop(); // Remove the front customer from the temporary queue

}

cout << endl;

// 2. Serve customers (remove from front of the queue)

while (!ticketQueue.empty()) {

cout << "Serving customer: " << ticketQueue.front() << endl; // Serve the front customer

ticketQueue.pop(); // Remove the front customer from the queue

customersServed++; // Increment the number of customers served

// Display the current queue after serving a customer

cout << "Current queue after serving: ";

tempQueue = ticketQueue; // Reset the temporary queue to the current queue

while (!tempQueue.empty()) {

cout << tempQueue.front() << " "; // Display the front customer

tempQueue.pop(); // Remove the front customer from the temporary queue

}

cout << endl;

}

// Display the number of customers served

cout << "Number of customers served: " << customersServed << endl;

return 0;

}

4. 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> // for std::istringstream

using namespace std;

// Function to evaluate a postfix expression

float evaluatePostfix(const string& expression) {

stack<float> evalStack;

istringstream iss(expression);

string token;

// Read tokens from the expression

while (iss >> token) {

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

// If the token is a number (including negative numbers), push it to the stack

evalStack.push(stof(token));

} else {

// The token is an operator, pop two elements from the stack

float operand2 = evalStack.top();

evalStack.pop();

float operand1 = evalStack.top();

evalStack.pop();

// Perform the operation and push the result back to the stack

if (token == "+") {

evalStack.push(operand1 + operand2);

} else if (token == "-") {

evalStack.push(operand1 - operand2);

} else if (token == "\*") {

evalStack.push(operand1 \* operand2);

} else if (token == "/") {

evalStack.push(operand1 / operand2);

}

}

}

// The final result is the only element left in the stack

return evalStack.top();

}

int main() {

// Example postfix expression

string postfixExpression = "3 4 + 2 \* 7 /";

// Evaluate the postfix expression

float result = evaluatePostfix(postfixExpression);

// Display the result

cout << "The result of the postfix expression '" << postfixExpression << "' is: " << result << endl;

return 0;

}