Day-14:

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

for (auto it = lst.begin(); it != lst.end(); ++it) {

std::cout << \*it << " ";

}

std::cout << std::endl;

}

int main() {

std::list<int> myList;

myList.push\_front(10);

myList.push\_back(12);

myList.push\_front(4);

myList.push\_back(20);

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

displayList(myList);

myList.pop\_front();

myList.pop\_back();

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

displayList(myList);

myList.sort();

std::cout << "List sorted in ascending order: ";

displayList(myList);

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

std::cout << "List sorted in descending order: ";

displayList(myList);

myList.reverse();

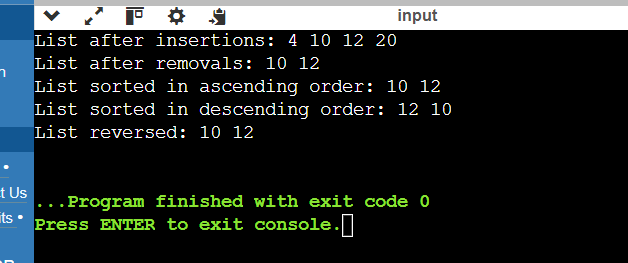
std::cout << "List reversed: ";

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:

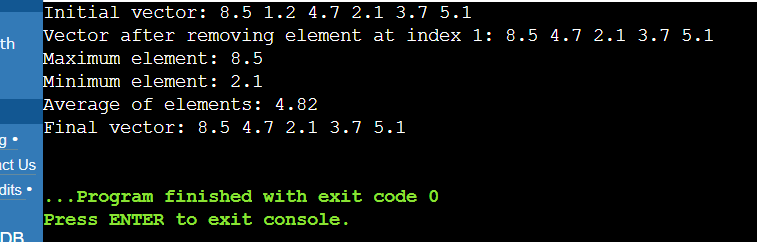
Add elements to the vector.

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.



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>

void displayQueue(const std::queue<int>& q) {

std::queue<int> tmpQueue = q;

std::cout << "Current queue: ";

while (!tmpQueue.empty()) {

std::cout << tmpQueue.front() << " ";

tmpQueue.pop();

}

std::cout << std::endl;

}

int main() {

std::queue<int> customerQueue;

int customersServed = 0;

customerQueue.push(1);

customerQueue.push(2);

customerQueue.push(3);

customerQueue.push(4);

customerQueue.push(5);

displayQueue(customerQueue);

while (!customerQueue.empty()) {

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

customerQueue.pop();

customersServed++;

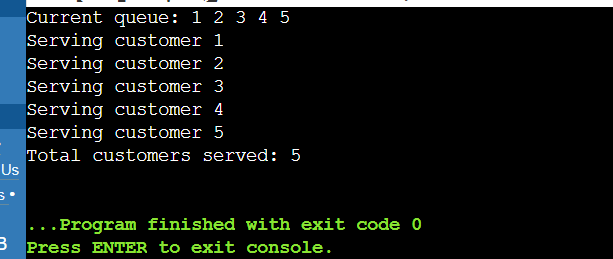
}

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

}

}

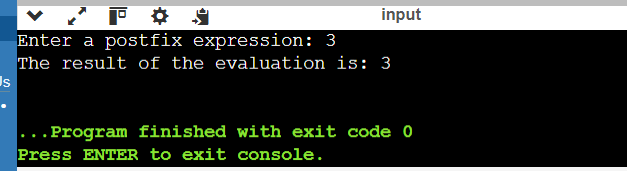
int main() {

processPostfixExpression();

return 0;

}

Output:



#include<iostream>

#include<fstream>

#include<string>

using namespace std;

int main()

{

string fileName;

char choice;

cout<<"enter the file name ";

cin>>fileName;

cout<<"enter 'r' to read from the file or 'w' to write to the file:";

cin>>choice;

if(choice=='r')

{

ifstream inputFile(fileName);

if(inputFile.is\_open())

{

string line;

while(getline(inputFile,line))

{

cout<<line<<endl;

}

inputFile.close();

}

else{

cout<<"error opening file for reading"<<endl;

}

}

else if(choice=='w')

{

ofstream outputFile(fileName);

if(outputFile.is\_open())

{

string content;

cout<<"enter the content to write to the file:";

cin.ignore();

getline(cin,content);

outputFile<<content<<endl;

outputFile.close();

cout<<"content written to the file successfully"<<endl;

}

else{

cout<<"error opening file for writing"<<endl;

}

}

else{

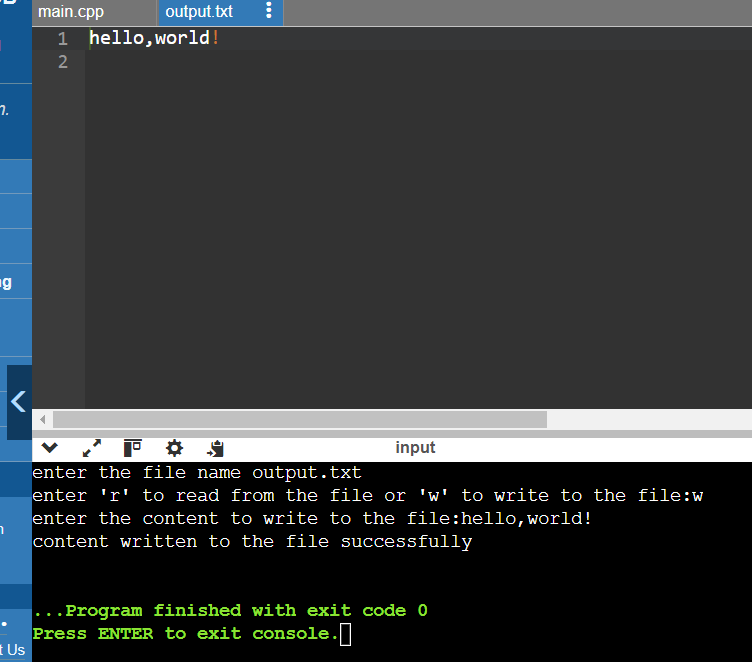
cout<<"invalid choice. please enter 'r' or 'w' "<<endl;

}

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.

Questions:

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

What is the purpose of the ifstream class in C++?

How can you check if a file was successfully opened?

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

How do you properly close a file after reading?

#include <iostream>

#include <fstream>

int main() {

std::ifstream inFile;

inFile.open("input.txt");

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

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

return 1;

}

inFile.close();

return 0;

}

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

The ifstream class in C++ is specifically designed for reading from files. It inherits from the istream class and provides methods and operators to facilitate reading operations (like >> for formatted input and getline() for reading lines) from files.