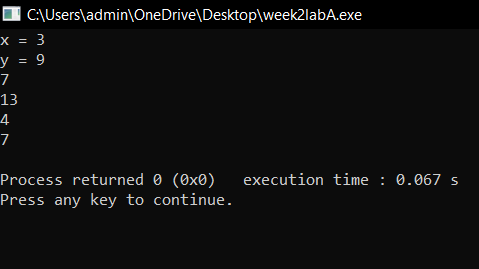
**DS LAB- ODD 2024**

**MOHD TARIQ SAMI – 9923103011**

**WEEK 2 LAB A**

Q1:



2.

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class CustomStack {

private:

Node\* top;

public:

CustomStack() {

top = NULL;

}

bool isEmpty() {

return top == NULL;

}

void push(int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = top;

top = newNode;

}

int pop() {

if (isEmpty()) {

cout<< "Stack is empty." << endl;

return -1; // Return a sentinel value

}

Node\* temp = top;

int poppedValue = temp->data;

top = top->next;

delete temp;

return poppedValue;

}

int peek() {

if (isEmpty()) {

cout << "Stack is empty." << endl;

return -1; // Return a sentinel value

}

return top->data;

}

};

bool is\_prime(int n) {

if (n <= 1) {

return false;

}

for (int i = 2; i \* i <= n; ++i) {

if (n % i == 0) {

return false;

}

}

return true;

}

void prime\_factors(int n) {

CustomStack stack;

for (int i = 2; i <= n; ++i) {

while (is\_prime(i) && n % i == 0) {

stack.push(i);

n /= i;

}

}

cout << "Prime factors in descending order: ";

while (!stack.isEmpty()) {

cout << stack.pop() << " ";

}

cout << endl;

}

int main() {

int n;

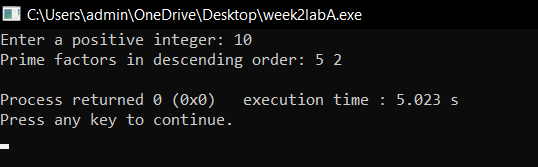
cout << "Enter a positive integer: ";

cin >> n;

prime\_factors(n);

return 0;

}



3. #include <iostream>

using namespace std;

#define MAX\_SIZE 100

class Stack {

private:

int arr[MAX\_SIZE];

int top;

public:

Stack() : top(-1) {}

bool isEmpty() const {

return top == -1;

}

bool isFull() const {

return top == MAX\_SIZE - 1;

}

void push(int value) {

if (isFull()) {

cout << "Stack is full!" << endl;

return;

}

arr[++top] = value;

}

int pop() {

if (isEmpty()) {

cout << "Stack is empty!" << endl;

return -1; // Indicates error

}

return arr[top--];

}

int peek() const {

if (isEmpty()) {

cout << "Stack is empty!" << endl;

return -1; // Indicates error

}

return arr[top];

}

void print() const {

for (int i = top; i >= 0; --i) {

cout << arr[i] << " ";

}

cout << endl;

}

};

void splitStack(Stack &original, Stack &bottomHalf, Stack &topHalf) {

int temp[MAX\_SIZE];

int size = 0;

while (!original.isEmpty()) {

temp[size++] = original.pop();

}

int mid = size / 2;

for (int i = mid - 1; i >= 0; --i) {

bottomHalf.push(temp[i]);

}

for (int i = size - 1; i >= mid; --i) {

topHalf.push(temp[i]);

}

}

void combineStacks(Stack &first, Stack &second) {

Stack tempStack;

while (!first.isEmpty()) {

tempStack.push(first.pop());

}

while (!second.isEmpty()) {

first.push(second.pop());

}

while (!tempStack.isEmpty()) {

first.push(tempStack.pop());

}

}

int main() {

Stack originalStack;

Stack bottomHalfStack;

Stack topHalfStack;

for (int i = 1; i <= 10; ++i) {

originalStack.push(i);

}

cout << "Original Stack: ";

originalStack.print();

splitStack(originalStack, bottomHalfStack, topHalfStack);

cout << "Bottom Half Stack: ";

bottomHalfStack.print();

cout << "Top Half Stack: ";

topHalfStack.print();

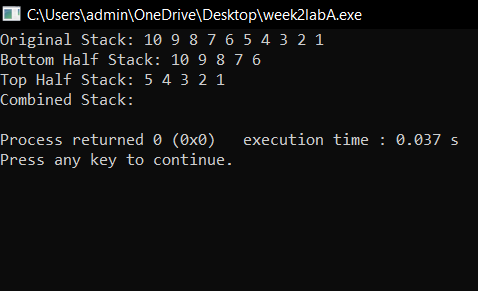
combineStacks(bottomHalfStack, topHalfStack);

cout << "Combined Stack:

originalStack.print();

return 0;

}



4. #include <iostream>

#include <stack>

using namespace std;

void convertToBase(int decimalNumber, int base) {

if (base < 2 || base > 9) {

cout << "Base must be between 2 and 9." << endl;

return;

}

stack<int> remainderStack;

if (decimalNumber == 0) {

cout << "0" << endl;

return;

}

while (decimalNumber > 0) {

int remainder = decimalNumber % base;

remainderStack.push(remainder);

decimalNumber /= base;

}

while (!remainderStack.empty()) {

cout << remainderStack.top();

remainderStack.pop();

}

cout << endl;

}

int main() {

int decimalNumber, base;

cout << "Enter a decimal number: ";

cin >> decimalNumber;

cout << "Enter the base (between 2 and 9): ";

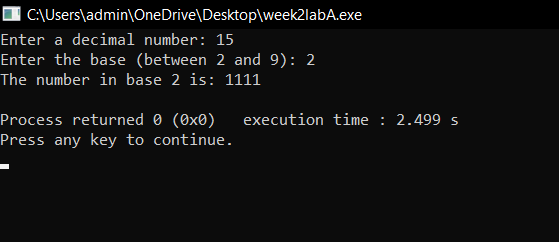
cin >> base;

cout << "The number in base " << base << " is: ";

convertToBase(decimalNumber, base);

return 0;

}



Q5

#include <iostream>

#include <stack>

#include <string>

#include <cctype>

#include <algorithm>

#include <cmath>

using namespace std;

// Function to check if the character is an operator

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

// Function to convert postfix to prefix

string postfixToPrefix(string postfix) {

stack<string> s;

for (char c : postfix) {

if (isOperator(c)) {

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

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

string temp = c + op2 + op1;

s.push(temp);

} else {

s.push(string(1, c));

}

}

return s.top();

}

// Function to convert prefix to postfix

string prefixToPostfix(string prefix) {

stack<string> s;

reverse(prefix.begin(), prefix.end());

for (char c : prefix) {

if (isOperator(c)) {

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

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

string temp = op1 + op2 + c;

s.push(temp);

} else {

s.push(string(1, c));

}

}

return s.top();

}

// Function to determine precedence of operators

int precedence(char c) {

if (c == '+' || c == '-') return 1;

if (c == '\*' || c == '/') return 2;

return 0;

}

// Function to convert infix to postfix

string infixToPostfix(string infix) {

stack<char> s;

string postfix = "";

for (char c : infix) {

if (isalnum(c)) {

postfix += c;

} else if (c == '(') {

s.push(c);

} else if (c == ')') {

while (!s.empty() && s.top() != '(') {

postfix += s.top();

s.pop();

}

s.pop();

} else {

while (!s.empty() && precedence(s.top()) >= precedence(c)) {

postfix += s.top();

s.pop();

}

s.push(c);

}

}

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

// Function to evaluate a postfix expression

int evaluatePostfix(string postfix) {

stack<int> s;

for (char c : postfix) {

if (isdigit(c)) {

s.push(c - '0');

} else {

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

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

switch (c) {

case '+': s.push(op1 + op2); break;

case '-': s.push(op1 - op2); break;

case '\*': s.push(op1 \* op2); break;

case '/': s.push(op1 / op2); break;

}

}

}

return s.top();

}

int main() {

// Example input

string infix = "((4+9\*6)-((8-6)/2\*4)\*9/3)";

// Convert infix to postfix

string postfix = infixToPostfix(infix);

cout << "Postfix expression: " << postfix << endl;

// Evaluate postfix expression

int result = evaluatePostfix(postfix);

cout << "Evaluation result: " << result << endl;

// Convert postfix to prefix

string prefixFromPostfix = postfixToPrefix(postfix);

cout << "Prefix from Postfix: " << prefixFromPostfix << endl;

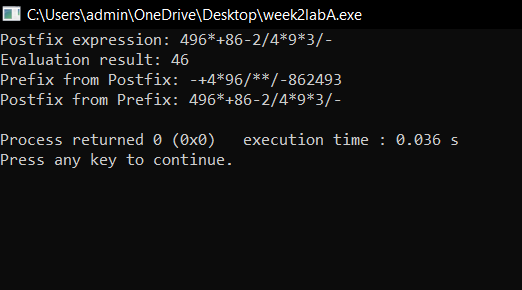
// Convert prefix to postfix

string postfixFromPrefix = prefixToPostfix(prefixFromPostfix);

cout << "Postfix from Prefix: " << postfixFromPrefix << endl;

return 0;

}



Q6

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool isMatchingPair(char char1, char char2) {

if (char1 == '(' && char2 == ')')

return true;

else if (char1 == '[' && char2 == ']')

return true;

else if (char1 == '{' && char2 == '}')

return true;

return false;

}

bool areSymbolsBalanced(string expr) {

stack<char> s;

for (char c : expr) {

// If the character is an opening bracket, push it to the stack

if (c == '(' || c == '[' || c == '{') {

s.push(c);

}

else if (c == ')' || c == ']' || c == '}') {

if (s.empty() || !isMatchingPair(s.top(), c)) {

return false;

} else {

s.pop(); // Pop the matching opening bracket from the stack

}

}

}

return s.empty();

}

int main() {

string expr;

cout << "Enter an expression with symbols: ";

cin >> expr;

if (areSymbolsBalanced(expr)) {

cout << "Symbols are balanced!" << endl;

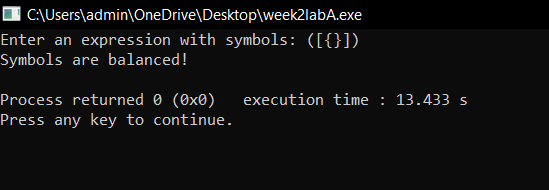
} else {

cout << "Symbols are not balanced!" << endl;

}

return 0;

}



**Q7**.

#include <iostream>

#include <queue>

#include <string>

using namespace std;

string compressText(const string& text) {

queue<char> q;

string result = "";

for (char c : text) {

if (c != ' ') {

q.push(c);

}

}

while (!q.empty()) {

char current = q.front();

q.pop();

int count = 1;

while (!q.empty() && q.front() == current) {

count++;

q.pop();

}

result += current;

if (count > 1) {

result += to\_string(count);

}

}

return result;

}

int main() {

string input;

cout << "Enter the text: ";

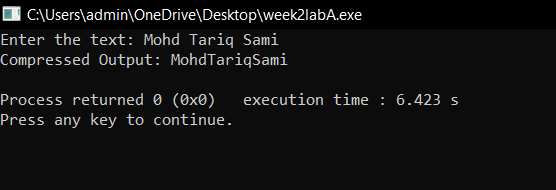
getline(cin, input);

string compressedText = compressText(input);

cout << "Compressed Output: " << compressedText << endl;

return 0;

}



Q8.

#include <iostream>

#include <queue>

using namespace std;

void moveNthFront(queue<int>& q, int n) {

if (n <= 0 || n > q.size()) {

cout << "Invalid value of n." << endl;

return;

}

queue<int> tempQueue;

int targetElement;

for (int i = 1; i < n; ++i) {

tempQueue.push(q.front());

q.pop();

}

targetElement = q.front();

q.pop();

while (!q.empty()) {

tempQueue.push(q.front());

q.pop();

}

q.push(targetElement);

while (!tempQueue.empty()) {

q.push(tempQueue.front());

tempQueue.pop();

}

}

int main() {

queue<int> q;

q.push(1);

q.push(2);

q.push(3);

q.push(4);

q.push(5);

q.push(6);

int n = 2;

cout << "Original Queue: ";

queue<int> temp = q; // Copy to display the queue

while (!temp.empty()) {

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

temp.pop();

}

cout << endl;

moveNthFront(q, n);

cout << "Queue after moving nth element to front: ";

while (!q.empty()) {

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

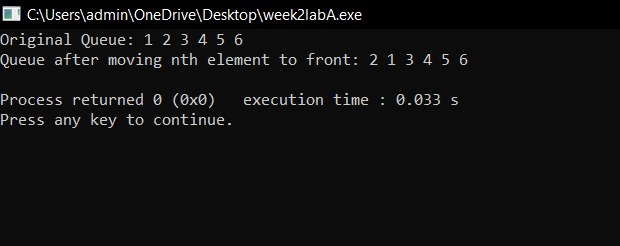
q.pop();

}

cout << endl;

return 0;

}



Q9.

#include <iostream>

#include <queue>

#include <stack>

#include <string>

using namespace std;

bool isAlphabetic(char c) {

return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');

}

char toLowerCase(char c) {

if (c >= 'A' && c <= 'Z') {

return c + ('a' - 'A');

}

return c;

}

bool isPalindrome(const string& text) {

queue<char> q;

stack<char> s;

for (char c : text) {

if (isAlphabetic(c)) {

}

q.pop();

s.pop();

}

return true;

}

int main() {

string input;

cout << "Enter a line of text: ";

getline(cin, input);

if (isPalindrome(input)) {

cout << "The text is a palindrome!" << endl;

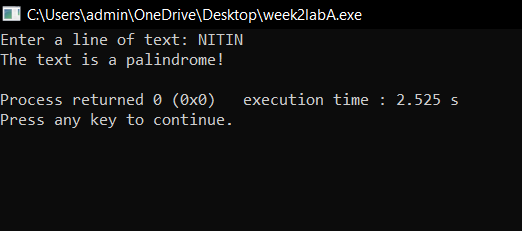
} else {

cout << "The text is not a palindrome." << endl;

}

return 0;

}



Q10.

#include <iostream>

#include <string>

using namespace std;

string extractAndReverse(const string& input) {

size\_t start = input.find("X");

size\_t mid = input.find("Y", start + 1);

size\_t mid\_end = input.rfind("Y");

size\_t end = input.find("X", mid\_end + 1);

if (start != string::npos && mid != string::npos && mid\_end != string::npos && end !=

string::npos) {

string subStr = input.substr(mid + 1, mid\_end - mid - 1);

int len = subStr.length();

for (int i = 0; i < len / 2; ++i) {

char temp = subStr[i];

subStr[i] = subStr[len - i - 1];

subStr[len - i - 1] = temp;

}

return subStr;

} else {

return ""; // Return an empty string if S1 or S2 isn't found correctly

}

}

int main() {

string input = "ABXNNYPEROYABCDCXT";

cout << "Input String: " << input << endl;

string result = extractAndReverse(input);

cout << "Output: " << result << endl;

return 0;

}

