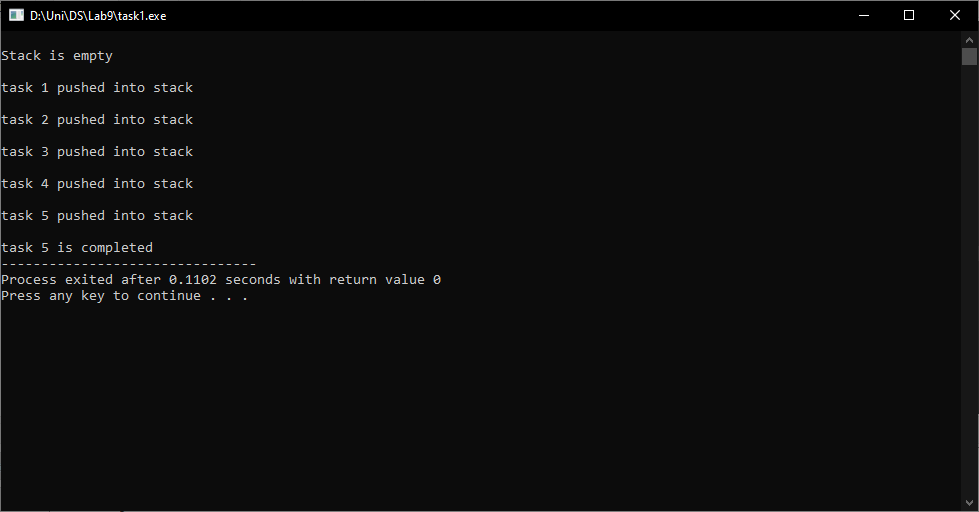
**DS LAB 9**

**SYED YOUSHA**

**K226007**

**BSR-3C**

Task 1:



#include<iostream>

using namespace std;

#define max 5

class stack

{

int top;

public:

string a[max];

stack()

{

top = -1;

}

bool push(string x)

{

if(top >= (max - 1))

{

cout<< "\nStack overflow";

}

else

{

a[++top] = x;

cout<<endl;

cout << x <<" pushed into stack\n";

return true;

}

}

string pop()

{

if(top < 0)

{

cout<<"\nStack underflow";

return 0;

}

else

{

string x = a[top];

top--;

cout<<endl;

cout<<x<<" is completed";

return x;

}

}

void peek()

{

if(top < 0)

{

cout<<"\nStack is empty";

return;

}

else

{

string x = a[top];

cout<<x;

}

}

bool isEmpty()

{

return (top < 0);

}

bool isFull()

{

return (top == max);

}

};

int main()

{

stack obj;

if(obj.isEmpty())

cout<<"\nStack is empty\n";

obj.push("task 1");

obj.push("task 2");

obj.push("task 3");

obj.push("task 4");

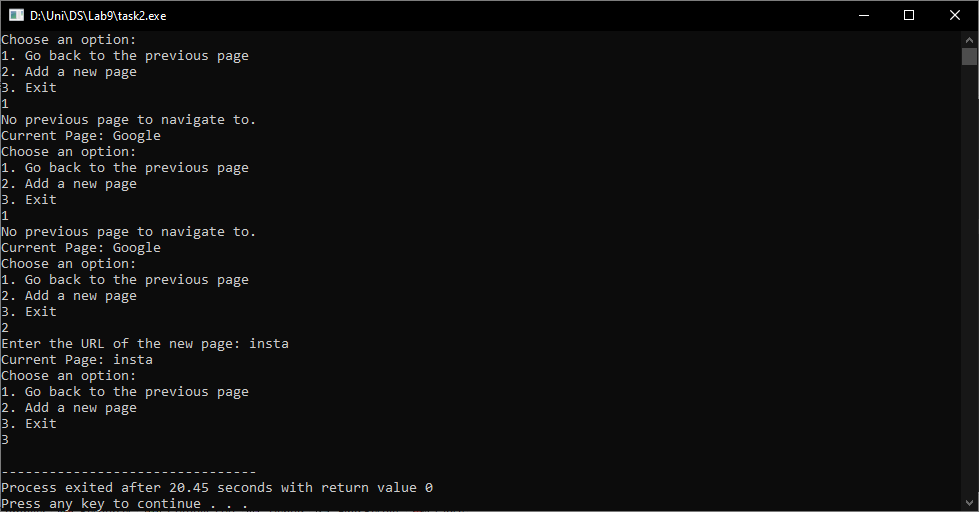
obj.push("task 5");

obj.pop();

return 0;

}

Task2:



#include <iostream>

#include <string>

#include <stack>

using namespace std;

struct WebPage {

string url;

WebPage\* next;

};

void addWebPage(WebPage\*& head, string url) {

WebPage\* newPage = new WebPage;

newPage->url = url;

newPage->next = head;

head = newPage;

}

bool navigateBack(WebPage\*& head, stack<string>& pageStack) {

if (pageStack.empty()) {

cout << "No previous page to navigate to." << endl;

return false;

}

string currentUrl = head->url;

string previousUrl = pageStack.top();

pageStack.push(currentUrl);

WebPage\* temp = head;

head = head->next;

delete temp;

cout << "Navigating back to: " << previousUrl << endl;

return true;

}

int main() {

WebPage\* browsingHistory = NULL;

stack<string> pageStack;

addWebPage(browsingHistory, "Instagram");

addWebPage(browsingHistory, "LinkedIn");

addWebPage(browsingHistory, "Twitter");

addWebPage(browsingHistory, "Facebook");

addWebPage(browsingHistory, "Google");

while (true) {

cout << "Current Page: " << browsingHistory->url << endl;

cout << "Choose an option:\n";

cout << "1. Go back to the previous page\n";

cout << "2. Add a new page\n";

cout << "3. Exit\n";

int choice;

cin >> choice;

if (choice == 1) {

navigateBack(browsingHistory, pageStack);

} else if (choice == 2) {

string newUrl;

cout << "Enter the URL of the new page: ";

cin >> newUrl;

addWebPage(browsingHistory, newUrl);

pageStack.push(browsingHistory->url);

} else if (choice == 3) {

break;

} else {

cout << "Invalid choice. Please enter 1, 2, or 3." << endl;

}

}

while (browsingHistory != NULL) {

WebPage\* temp = browsingHistory;

browsingHistory = browsingHistory->next;

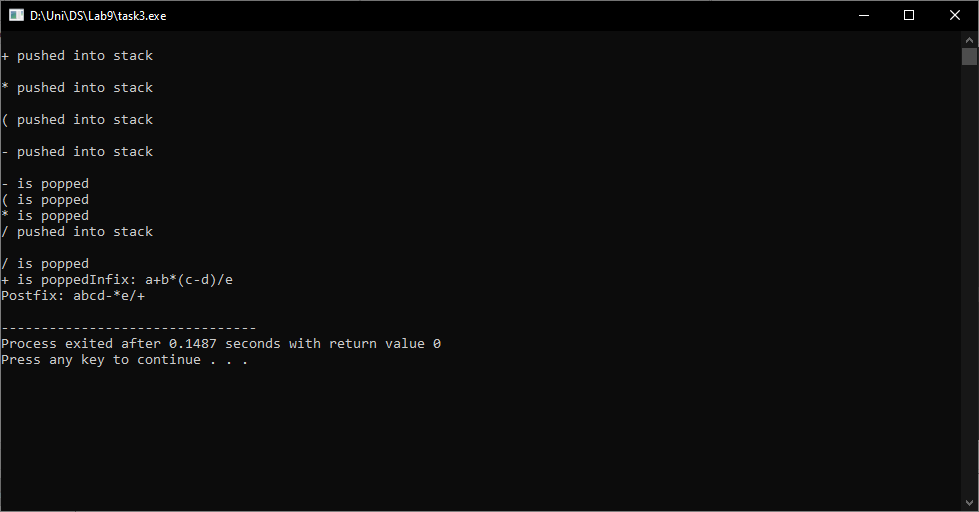
delete temp;

}

return 0;

}

Task 3:



#include<iostream>

#include<string>

using namespace std;

class Stack

{

public:

int top;

int max;

char \*a;

Stack(int m)

{

top = -1;

max = m;

a = new char[m];

}

bool push(char x)

{

if (top >= (max - 1))

{

cout << "\nStack overflow";

return false;

}

else

{

a[++top] = x;

cout << endl;

cout << x << " pushed into stack\n";

return true;

}

}

char pop()

{

if (top < 0)

{

cout << "\nStack underflow";

return 0;

}

else

{

char x = a[top];

top--;

cout << endl;

cout << x << " is popped";

return x;

}

}

char peek()

{

if (top < 0)

{

cout << "\nStack is empty";

return 0;

}

else

{

char x = a[top];

return x;

}

}

bool isEmpty()

{

return (top < 0);

}

bool isFull()

{

return (top == max - 1);

}

~Stack()

{

delete[] a;

}

};

int getPrecedence(char c)

{

if (c == '^')

return 3;

else if (c == '\*' || c == '/')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1;

}

string infixToPostfix(string infix)

{

string postfix = "";

Stack s(infix.length());

for (int i = 0; i < infix.length(); i++)

{

char c = infix[i];

if (isalpha(c))

{

postfix += c;

}

else if (c == '(')

{

s.push(c);

}

else if (c == ')')

{

while (!s.isEmpty() && s.peek() != '(')

{

char op = s.pop();

postfix += op;

}

if (s.peek() == '(')

{

s.pop();

}

}

else

{

while (!s.isEmpty() && getPrecedence(c) <= getPrecedence(s.peek()))

{

char op = s.pop();

postfix += op;

}

s.push(c);

}

}

while (!s.isEmpty())

{

char op = s.pop();

postfix += op;

}

return postfix;

}

//k226007

int main()

{

string infix = "a+b\*(c-d)/e";

string postfix = infixToPostfix(infix);

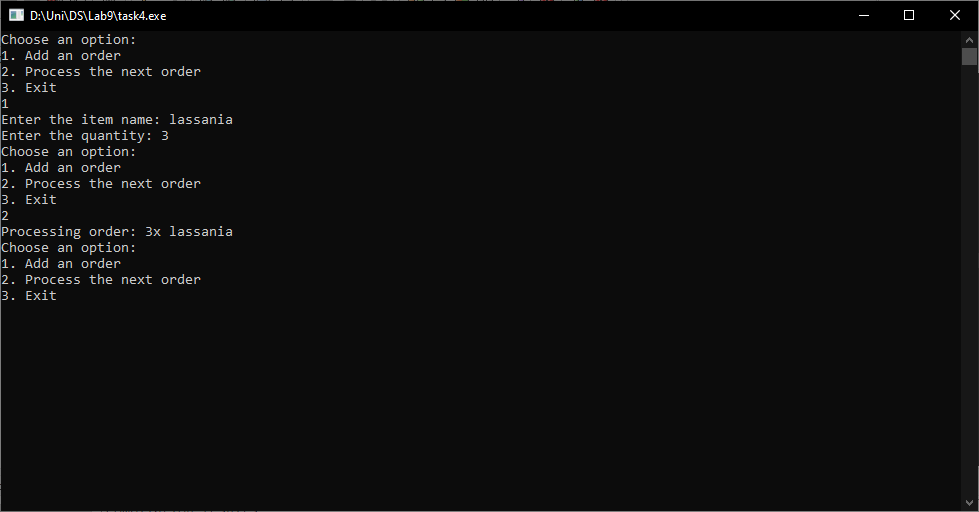
cout << "Infix: " << infix << endl;

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

return 0;

}

Task 4:



#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Order {

string itemName;

int quantity;

};

void addOrder(queue<Order>& orderQueue, const Order& order) {

orderQueue.push(order);

}

bool processOrder(queue<Order>& orderQueue) {

if (!orderQueue.empty()) {

Order order = orderQueue.front();

cout << "Processing order: " << order.quantity << "x " << order.itemName << endl;

orderQueue.pop();

return true;

} else {

cout << "No orders to process." << endl;

return false;

}

}

int main() {

queue<Order> orderQueue;

while (true) {

cout << "Choose an option:\n";

cout << "1. Add an order\n";

cout << "2. Process the next order\n";

cout << "3. Exit\n";

int choice;

cin >> choice;

if (choice == 1) {

Order order;

cout << "Enter the item name: ";

cin >> order.itemName;

cout << "Enter the quantity: ";

cin >> order.quantity;

addOrder(orderQueue, order);

} else if (choice == 2) {

processOrder(orderQueue);

} else if (choice == 3) {

break;

} else {

cout << "Invalid choice. Please enter 1, 2, or 3." << endl;

}

}

while (!orderQueue.empty()) {

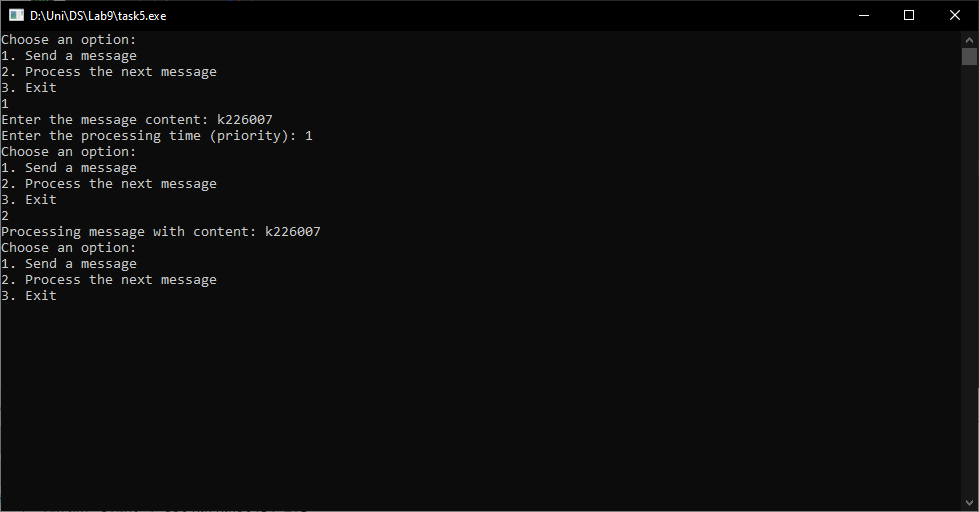
processOrder(orderQueue);

}

return 0;

}

Task 5:



#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Message {

string content;

int processingTime;

};

struct CompareMessages {

bool operator()(const Message& msg1, const Message& msg2) {

return msg1.processingTime > msg2.processingTime;

}

};

int main() {

priority\_queue<Message, vector<Message>, CompareMessages> messageQueue;

while (true) {

cout << "Choose an option:\n";

cout << "1. Send a message\n";

cout << "2. Process the next message\n";

cout << "3. Exit\n";

int choice;

cin >> choice;

if (choice == 1) {

Message message;

cout << "Enter the message content: ";

cin.ignore();

getline(cin, message.content);

cout << "Enter the processing time (priority): ";

cin >> message.processingTime;

messageQueue.push(message);

} else if (choice == 2) {

if (!messageQueue.empty()) {

Message nextMessage = messageQueue.top();

messageQueue.pop();

cout << "Processing message with content: " << nextMessage.content << endl;

} else {

cout << "No messages to process." << endl;

}

} else if (choice == 3) {

break;

} else {

cout << "Invalid choice. Please enter 1, 2, or 3." << endl;

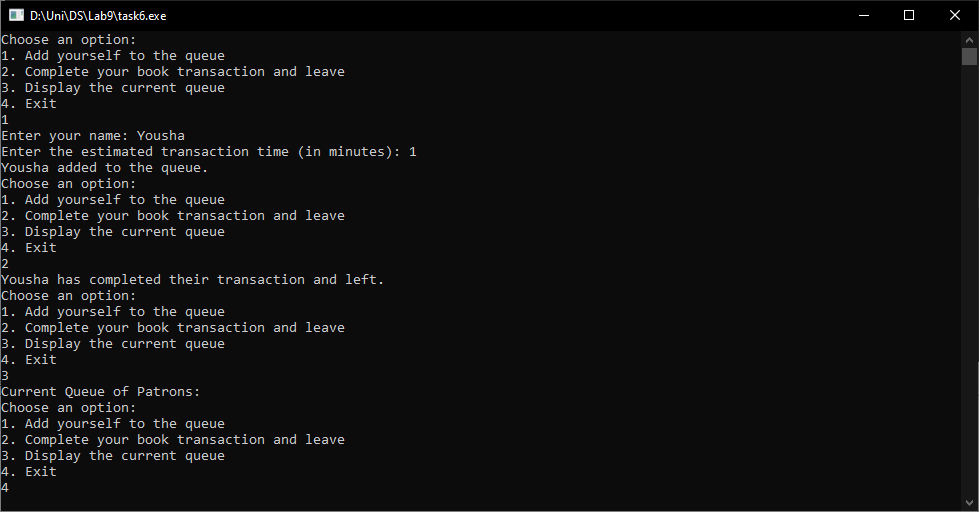
}

}

return 0;

}

Task 6:



#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Patron {

string name;

int transactionTime;

};

int main() {

queue<Patron> patronQueue;

while (true) {

cout << "Choose an option:\n";

cout << "1. Add yourself to the queue\n";

cout << "2. Complete your book transaction and leave\n";

cout << "3. Display the current queue\n";

cout << "4. Exit\n";

int choice;

cin >> choice;

if (choice == 1) {

Patron patron;

cout << "Enter your name: ";

cin >> patron.name;

cout << "Enter the estimated transaction time (in minutes): ";

cin >> patron.transactionTime;

patronQueue.push(patron);

cout << patron.name << " added to the queue.\n";

} else if (choice == 2) {

if (!patronQueue.empty()) {

Patron nextPatron = patronQueue.front();

patronQueue.pop();

cout << nextPatron.name << " has completed their transaction and left.\n";

} else {

cout << "No patrons in the queue.\n";

}

} else if (choice == 3) {

cout << "Current Queue of Patrons:\n";

queue<Patron> tempQueue = patronQueue;

while (!tempQueue.empty()) {

Patron currentPatron = tempQueue.front();

tempQueue.pop();

cout << currentPatron.name << " (Estimated Time: " << currentPatron.transactionTime << " minutes)\n";

}

} else if (choice == 4) {

break;

} else {

cout << "Invalid choice. Please enter 1, 2, 3, or 4." << endl;

}

}

return 0;

}