**Name: Talha Abdullah Bangyal**

**Roll Number: 22F-3194**

**Section: BAI-3A**

**Lab: 5**

**--------------------------------------------------------------------**

**Starter Task 1**

#include<iostream>

using namespace std;

class IntStack

{

private:

int\* stackArray;

int stackSize;

int top;

public:

IntStack(int);

~IntStack();

bool push(int);

bool pop(int&);

bool isFull();

bool isEmpty();

};

IntStack::IntStack(int size) //constructor

{

stackArray = new int[size];

stackSize = size;

top = -1;

}

IntStack::~IntStack(void) //destructor

{

delete[] stackArray;

}

bool IntStack::isFull(void)

{

if (top == stackSize - 1)

return true;

else

return false;

// return (top == stackSize-1);

}

bool IntStack::isEmpty(void)

{

return (top == -1);

}

bool IntStack::push(int num)

{

if (isFull())

{

cout << "The stack is full.\n";

return false;

}

top++;

stackArray[top] = num;

return true;

}

bool IntStack::pop(int& num)

{

if (isEmpty())

{

cout << "The stack is empty.\n";

return false;

}

num = stackArray[top];

top--;

return true;

}

int main() {

}

**Starter Task 2**

#include<iostream>

using namespace std;

class node {

public:

int data; node\* next;

};

class Stack {

node\* top;

public:

Stack();

~Stack();

void Push(int newelement);

bool Pop(int&);

bool IsEmpty();

void makeNull();

};

Stack::Stack() {

top = NULL;

}

bool Stack::IsEmpty() {

return (top == NULL);

}

void Stack::Push(int newelement) {

node\* newptr = new node;

newptr->data = newelement;

newptr->next = top;

top = newptr;

}

bool Stack::Pop(int& returnvalue) {

if (IsEmpty()){

cout << "underflow error";

return false;

}

node\* tempptr = top;

returnvalue = top->data;

top = top->next;

delete tempptr;

return true;

}

void Stack::makeNull() {

int x;

while (Pop(x));

}

Stack::~Stack() {

makeNull();

}

int main() {

}

**Task 1**

#include<iostream>

#include<string>

using namespace std;

struct node {

char data;

node\* next;

};

class Stack {

node\* top;

public:

Stack() { top = NULL; }

~Stack() { makeNull(); }

void Push(char newelement);

bool Pop(char&);

bool IsEmpty();

void makeNull();

char Top();

};

void Stack::Push(char newelement) {

node\* newptr = new node;

newptr->data = newelement;

newptr->next = top;

top = newptr;

}

bool Stack::Pop(char& returnvalue) {

if (IsEmpty()) {

cout << "underflow error\n";

return false;

}

node\* tempptr = top;

returnvalue = top->data;

top = top->next;

delete tempptr;

return true;

}

bool Stack::IsEmpty() {

return (top == NULL);

}

void Stack::makeNull() {

char x;

while (Pop(x));

}

char Stack::Top() {

if (IsEmpty()) {

cout << "Stack is empty\n";

return '\0';

}

return top->data;

}

int precedence(char c) {

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

return 1;

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

return 2;

else

return 0;

}

void infixToPostfix(string infix, string& postfix) {

Stack s;

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

if (infix[i] == ' ') continue;

else if (infix[i] == '(') s.Push(infix[i]);

else if (infix[i] == ')') {

char c;

while (!s.IsEmpty() && s.Top() != '(') {

s.Pop(c);

postfix += c;

}

if (s.IsEmpty() || s.Top() != '(') {

cout << "Invalid infix expression\n";

return;

}

s.Pop(c);

}

else if (isalpha(infix[i])) postfix += infix[i];

else {

while (!s.IsEmpty() && s.Top() != '(' && precedence(s.Top()) >= precedence(infix[i])) {

char c;

s.Pop(c);

postfix += c;

}

s.Push(infix[i]);

}

}

while (!s.IsEmpty()) {

char c;

s.Pop(c);

postfix += c;

}

}

int main() {

string infix, postfix;

cout << "Enter infix expression: ";

getline(cin, infix);

infixToPostfix(infix, postfix);

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

return 0;

}

**Task 2**

#include<iostream>

#include<string>

using namespace std;

struct node {

char data;

node\* next;

};

class Stack {

node\* top;

public:

Stack() { top = NULL; }

~Stack() { makeNull(); }

void Push(char newelement);

bool Pop(char&);

bool IsEmpty();

void makeNull();

char Top();

};

void Stack::Push(char newelement) {

node\* newptr = new node;

newptr->data = newelement;

newptr->next = top;

top = newptr;

}

bool Stack::Pop(char& returnvalue) {

if (IsEmpty()) {

cout << "underflow error";

return false;

}

node\* tempptr = top;

returnvalue = top->data;

top = top->next;

delete tempptr;

return true;

}

bool Stack::IsEmpty() {

return (top == NULL);

}

void Stack::makeNull() {

char x;

while (Pop(x));

}

char Stack::Top() {

if (IsEmpty()) {

cout << "Stack is empty";

return '\0';

}

return top->data;

}

int precedence(char c) {

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

return 1;

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

return 2;

else

return 0;

}

void infixToPrefix(string infix, string& prefix) {

Stack s;

string temp = infix;

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

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

if (temp[i] == ' ') continue;

else if (temp[i] == '(') {

char c;

while (!s.IsEmpty() && s.Top() != ')') {

s.Pop(c);

prefix += c;

}

if (s.IsEmpty() || s.Top() != ')') {

cout << "Invalid infix expression";

return;

}

s.Pop(c);

}

else if (temp[i] == ')') s.Push(temp[i]);

else if (isalpha(temp[i])) prefix += temp[i];

else {

while (!s.IsEmpty() && s.Top() != ')' && precedence(s.Top()) >= precedence(temp[i])) {

char c;

s.Pop(c);

prefix += c;

}

s.Push(temp[i]);

}

}

while (!s.IsEmpty()) {

char c;

s.Pop(c);

prefix += c;

}

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

}

int main() {

string infix, prefix;

cout << "Enter infix expression: ";

getline(cin, infix);

infixToPrefix(infix, prefix);

cout << "Prefix expression: " << prefix << endl;

return 0;

}

**Task 3**

#include<iostream>

#include<string>

using namespace std;

struct node {

char data;

node\* next;

};

class Stack {

node\* top;

public:

Stack() { top = NULL; }

~Stack() { makeNull(); }

void Push(char newelement);

bool Pop(char&);

bool IsEmpty();

void makeNull();

char Top();

};

void Stack::Push(char newelement) {

node\* newptr = new node;

newptr->data = newelement;

newptr->next = top;

top = newptr;

}

bool Stack::Pop(char& returnvalue) {

if (IsEmpty()) {

cout << "underflow error";

return false;

}

node\* tempptr = top;

returnvalue = top->data;

top = top->next;

delete tempptr;

return true;

}

bool Stack::IsEmpty() {

return (top == NULL);

}

void Stack::makeNull() {

char x;

while (Pop(x));

}

char Stack::Top() {

if (IsEmpty()) {

cout << "Stack is empty";

return '\0';

}

return top->data;

}

int precedence(char c) {

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

return 1;

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

return 2;

else if (c == '!' || c == '~' || c == '+' + 256 || c == '-' + 256) // unary operators

return 3;

else

return 0;

}

void infixToPostfix(string infix, string& postfix) {

Stack s;

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

if (infix[i] == ' ') continue;

else if (infix[i] == '(') s.Push(infix[i]);

else if (infix[i] == ')') {

char c;

while (!s.IsEmpty() && s.Top() != '(') {

s.Pop(c);

postfix += c;

}

if (s.IsEmpty() || s.Top() != '(') {

cout << "Invalid infix expression";

return;

}

s.Pop(c);

}

else if (isalpha(infix[i])) postfix += infix[i];

else if (infix[i] == '!' || infix[i] == '~' || infix[i] == '+' + 256 || infix[i] == '-' + 256) { // unary operators

s.Push(infix[i]);

}

else {

while (!s.IsEmpty() && s.Top() != '(' && precedence(s.Top()) >= precedence(infix[i])) {

char c;

s.Pop(c);

postfix += c;

}

s.Push(infix[i]);

}

}

while (!s.IsEmpty()) {

char c;

s.Pop(c);

postfix += c;

}

}

int main() {

string infix, postfix;

cout << "Enter infix expression: ";

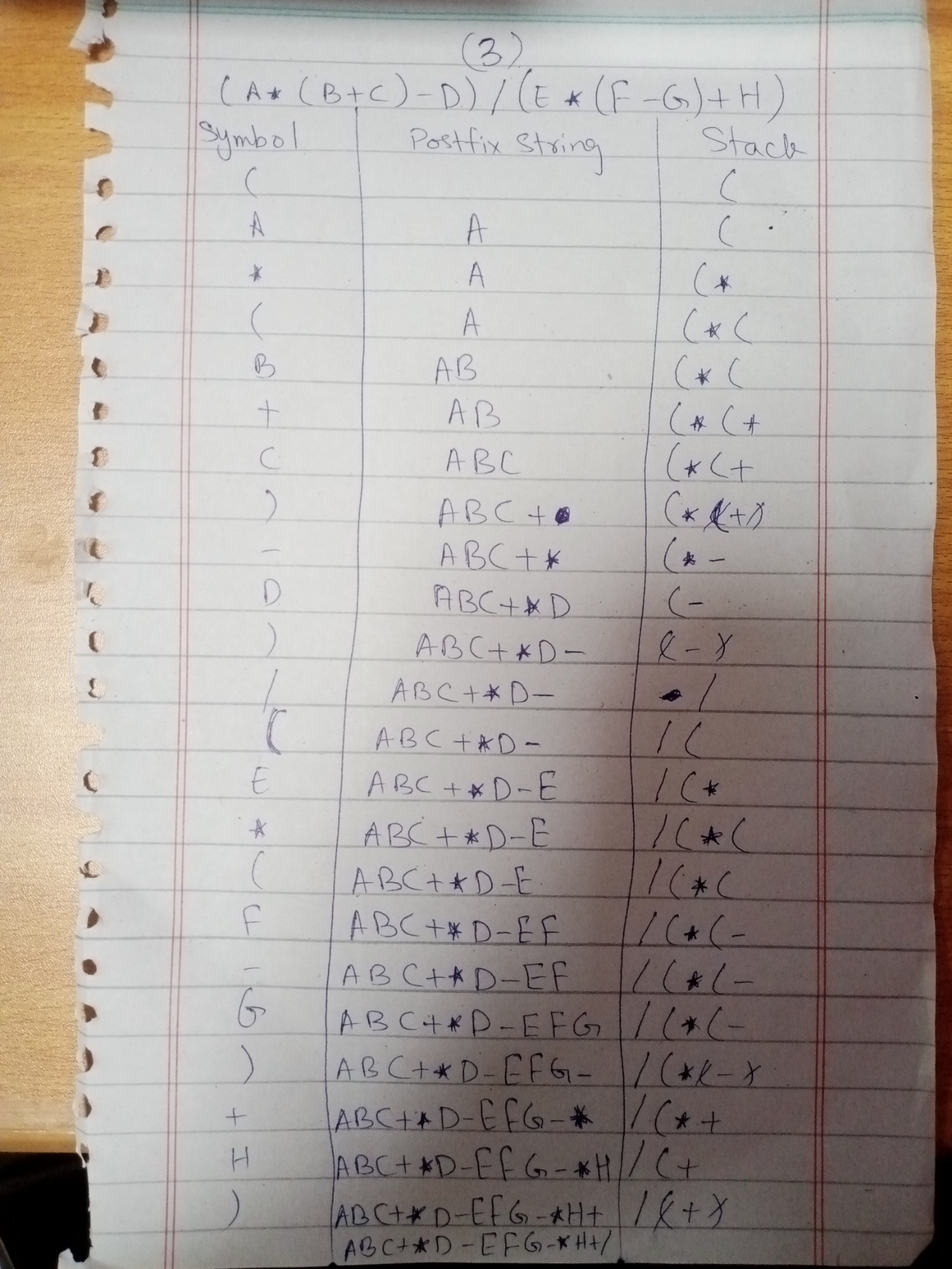
getline(cin, infix);

infixToPostfix(infix, postfix);

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

return 0;

}

**Task 4 (1)**

**Task 4 (3)**

**A piece of paper with writing on it

Description automatically generated**