**Data Structure**

**Name:** Faizan Tariq

**Roll No:** 22F-3858

**Section:** BS (CS) 3D

**Submitted To:** Tayyeb Javed

**Assignment 2**

**TASK 1 :**

#include<iostream>

#include<string>

using namespace std;

// Global variables here

string resultString = " ";

void menu() {

cout << "\n1. Calculate expression as Postfix.\n2. Calculate expression as Prefix.\n3. Quit.\n";

}

struct Node {

string data;

Node\* next;

Node() {

data = " ";

next = NULL;

}

Node(string Data) {

data = Data;

next = NULL;

}

};

class StackClass {

private:

Node\* top;

public:

StackClass() {

top = NULL;

}

bool isEmpty() {

return (top == NULL);

}

bool push(string data) {

Node\* newNode = new Node(data);

if (isEmpty()) {

top = newNode;

return true;

}

newNode->next = top;

top = newNode;

return true;

}

string pop() {

if (isEmpty()) {

cout << "Stack is empty.\n";

return " ";

}

string value = top->data;

Node\* deleteNode = top;

top = top->next;

delete deleteNode;

return value;

}

string giveTopValue() {

if (isEmpty()) {

cout << "Stack is empty.\n";

return " ";

}

return top->data;

}

void makeNull() {

while (!isEmpty()) {

pop();

}

}

} stakData, stakCalculate;

bool isOperator(string op) {

if (op == "+" || op == "-" || op == "\*" || op == "/" || op == "%")

return true;

return false;

}

int getPrecedence(string op) {

if (op == "+" || op == "-")

return 1;

if (op == "\*" || op == "/" || op == "%")

return 2;

return 0;

}

int performOperation(string op ,int operand1, int operand2) {

if (op == "+")

return operand1 + operand2;

else if (op == "-")

return operand1 - operand2;

else if (op == "\*")

return operand1 \* operand2;

else if (op == "/")

{

if (operand2 == 0) {

cout << "\n Denominator cannot be zero. \n";

return 0;

}

return operand1 / operand2;

}

else if (op == "%")

{

if (operand2 == 0) {

cout << "\n Denominator cannot be zero. \n";

return 0;

}

return operand1 % operand2;

}

else

cout << "\nInvalid Operator!!\n";

return 0;

}

void evaluatePreexpression(string ch) { //symb = next inputS character;

if (!isOperator(ch) && ch != "(" && ch != ")" && ch!=" ") { //if (symb is an operand)

string revCh = "";

for (int i = ch.length()-1; i >= 0; i--) // reverse ch

{

revCh += ch[i];

}

cout << endl << revCh << " pushed in stack"<<endl;

stakCalculate.push(revCh); //push(opndstk, symb)

}

else { // if ch is operator

int operand1, operand2;

operand1 =stoi(stakCalculate.pop());

operand2 = stoi(stakCalculate.pop()); // convert string into int

// now perfrom calculation and store them in result using operand1 and operand2 and ch will be operator

int result = performOperation(ch, operand1, operand2); // perform operaiton according to ch

cout << "\nResult of " << operand1 << " " << ch << " " << operand2 << " is :" << result;

stakCalculate.push(to\_string(result)); // convert result to string and paush in stack

}

}

void evaluatePostexpression(string ch) {

if (!isOperator(ch) && ch != "(" && ch != ")") {

cout << endl << ch << " pushed in stack." << endl;

stakCalculate.push(ch); // Push operands onto the stack

}

else { // If ch is an operator

int operand1 = stoi(stakCalculate.pop());

int operand2 = stoi(stakCalculate.pop());

int result = performOperation(ch, operand2, operand1);

cout << "\nResult of " << operand2 << " "<< ch <<" " << operand1 << " is :" << result;

stakCalculate.push(to\_string(result));

}

}

void convertToPostfix(string ch) {

if (!isOperator(ch) && ch != "(" && ch != ")") {

resultString += ch + " ";

}

else if (ch == ")") {

while (!stakData.isEmpty() && stakData.giveTopValue() != "(") {

resultString += stakData.pop() + " ";

}

if (!stakData.isEmpty() && stakData.giveTopValue() == "(") {

stakData.pop();

}

}

else if (ch == "(") {

stakData.push(ch);

}

else {

while (!stakData.isEmpty() && getPrecedence(ch) <= getPrecedence(stakData.giveTopValue())) {

resultString += stakData.pop() + " ";

}

stakData.push(ch);

}

}

void convertPostExpression() {

resultString = "";

cout << "\nEnter Your Expression (with spaces): ";

string expression, token;

getline(cin, expression); // get input with spaces

bool continueReadingNumber = false; // Flag to check if we are reading a number

string currentNumber = ""; // Store the current number being read

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

if (expression[i] == ' ') {

if (continueReadingNumber) {

convertToPostfix(currentNumber);

continueReadingNumber = false;

currentNumber = "";

}

}

else {

continueReadingNumber = true;

currentNumber += expression[i];

}

}

if (continueReadingNumber) { // If the last token was a number, handle it

convertToPostfix(currentNumber);

}

while (!stakData.isEmpty()) {

resultString += stakData.pop() + " ";

}

cout << "\nYour Postfix Expression: " << resultString << endl;

cout << "Now Calculating POSTFix expression.\n";

stakCalculate.makeNull();

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

if (resultString[i] == ' ') {

continue;

}

else {

while(resultString[i] !=' '){

token += resultString[i];

i++;

}

evaluatePostexpression(token);

token = "";

}

}

string result = " ";

result = stakCalculate.pop();

cout << "\nResult of overall expression is: " << result << endl;

}

void convertPreExpression() {

resultString = "";

cout << "\nEnter Your Expression (with spaces): ";

string expression, token;

getline(cin, expression);

string currentNumber = "";

bool continueReadingNumber = false;

for (int i = 0; i < expression.length(); i++) { //1st invert parantheses

if (expression[i] == '(') {

expression[i] = ')';

}

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

expression[i] = '(';

}

}

string reverseExpression = "";

for (int i = expression.length() - 1; i >= 0; i--) { // Reverse the original expression

reverseExpression += expression[i];

}

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

if (reverseExpression[i] == ' ') {

continue;

}

else {

while (reverseExpression[i] != ' ' && i < reverseExpression.length()) {

token += reverseExpression[i];

i++;

}

convertToPostfix(token);

token = "";

}

}

while (!stakData.isEmpty()) {

resultString += stakData.pop() ;

}

// Reverse the postfix result to get the prefix expression

string prefixExpression = "";

for (int i = resultString.length() - 1; i >= 0; i--) {

prefixExpression += resultString[i];

}

cout << "\nYour Prefix Expression: " << prefixExpression << endl;

// Now calculate the preExpression

cout << "Now calculating your Prefix expression.\n";

stakCalculate.makeNull(); // make sure the stack is already empty if not then make it empty

token = " "; // reset token value

string token2 = "";

for (int i = prefixExpression.length()-1; i >= 0; i--) {

token2 = "";

if (prefixExpression[i] == ' ') {

continue;

}

else {

while (i >=0 && prefixExpression[i] != ' ') {

token2 += prefixExpression[i];

i--;

}

evaluatePreexpression(token2);

}

}

string result = stakCalculate.pop();

cout << "\nResult of the prefix expression is: " << result << endl;

}

int main() {

int choice = 0;

while (1) {

menu();

cout << "Select Option :";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

convertPostExpression();

break;

case 2:

convertPreExpression();

break;

default:

return 0;

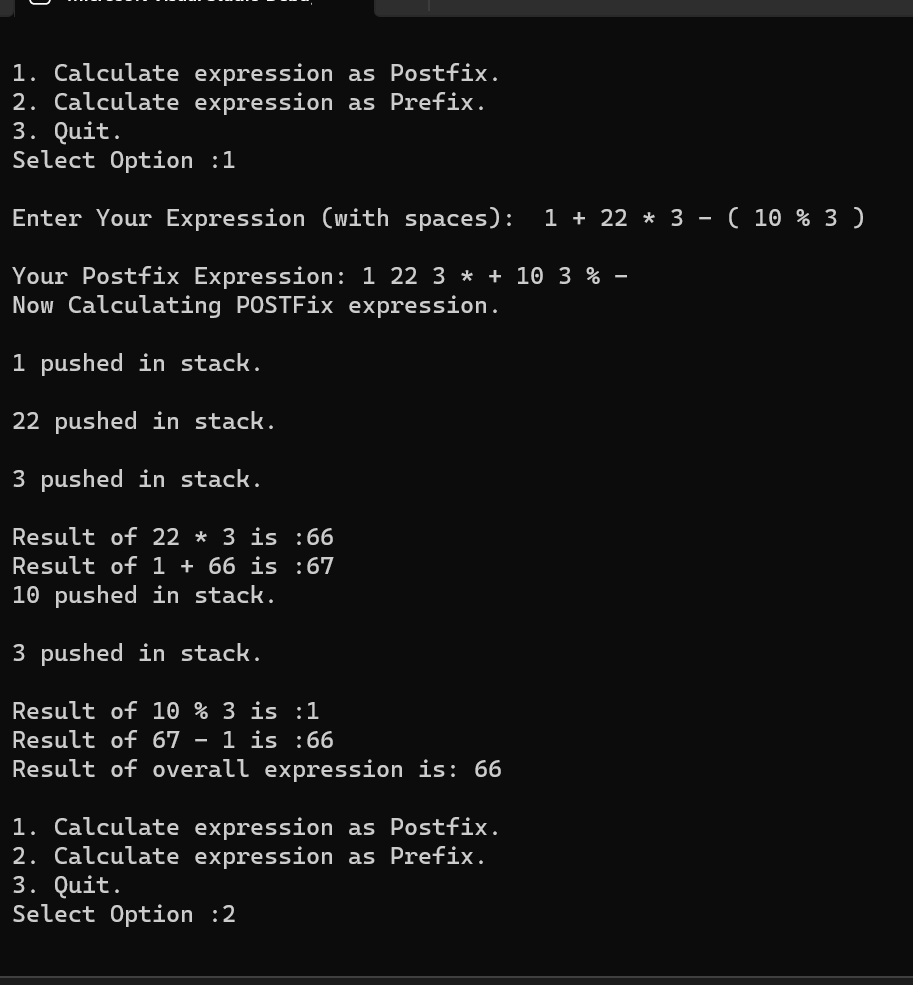
}

}

return 0;

}

**OUTPUT:**

****

**A screenshot of a computer program

Description automatically generated**

**TASK 2:**

#include<iostream>

#include<string>

using namespace std;

struct Node {

string command;

Node\* next;

Node() {

next = NULL;

command = "";

}

Node(string command) {

next = NULL;

this->command = command;

}

};

class undoRedo{

Node\* top;

bool enabled;

public:

undoRedo() {

top = NULL;

enabled = false;

}

string getEnabled() {

return enabled ?"Enabled":"Disabled";

}

void setEnabled(bool value) {

enabled = value;

}

bool isEmpty() {

return top == NULL;

}

bool push(string command) {

Node\* newNode = new Node(command);

if (isEmpty()) {

top = newNode;

enabled = true;

return true;

}

newNode->next = top;

top = newNode;

enabled = true;

return true;

}

bool pop(string & returnData) {

if (isEmpty()) {

return false;

}

returnData = top->command;

Node\* DeleteNode = top;

top = top->next;

delete DeleteNode;

if (isEmpty()) {

enabled = false;

}

return true;

}

void makeNull() {

string temp;

while (!isEmpty()) {

pop(temp);

}

}

};

int main() {

cout << "LIST OF COMMANDS :\n->undo.\n->redo.\n->add(Any text to add).\n->delete.\n->quit.";

undoRedo undo, redo;

string commands = " ";

string deletedCommand = " ";

while (1) {

cout << "\n[undo: " << undo.getEnabled() << " , " << "redo: " << redo.getEnabled() << "]"<<endl;

cout << "Type your command : ";

cin >> commands;

if (commands == "undo") {

if (undo.pop(deletedCommand)) {

redo.push(deletedCommand);

}

}

else if (commands == "redo") {

if (redo.pop(deletedCommand)) {

undo.push(deletedCommand);

}

}

else if (commands == "delete") { // if command is delete

if (undo.pop(deletedCommand)) {

redo.makeNull();

redo.push(deletedCommand);

cout <<endl<< deletedCommand << " undone.\n";

}

}

else if (commands == "quit" || commands == "Quit" || commands == "QUIT") {

break;

}

else { // for add text

undo.push(commands);

redo.makeNull();

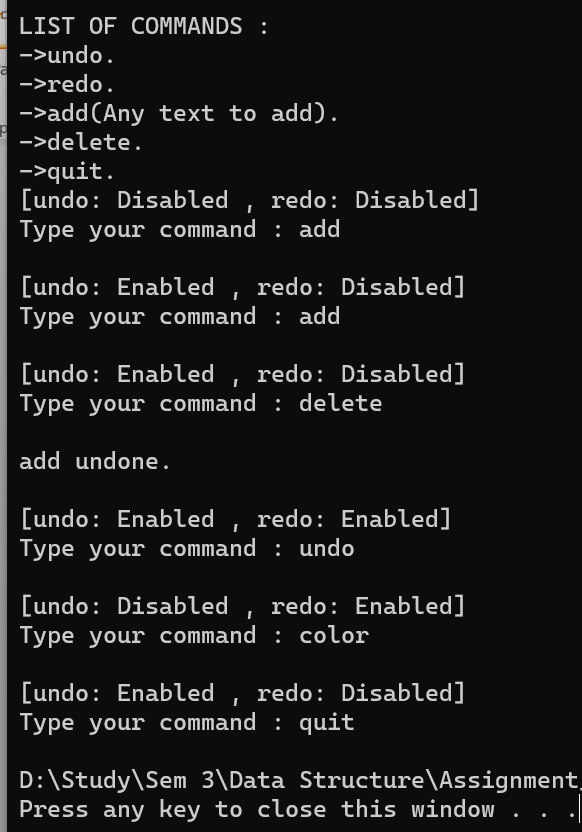
}

}

return 0;

}

**OUTPUT :**

****

**TASK 3:**

#include<iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node() {

data = 0;

next = NULL;

}

Node(int data) {

this->data = data;

next = NULL;

}

};

class Stack {

Node\* top;

public:

Stack() {

top = NULL;

}

bool isEmpty() {

return top == NULL;

}

bool push(int data) {

Node\* newNode = new Node(data);

newNode->next = top;

top = newNode;

return true;

}

bool pop(int& tempData) {

if (isEmpty()) {

return false;

}

Node\* tempNode = top;

tempData = tempNode->data;

top = tempNode->next;

delete tempNode;

}

void displayStack() {

Node\* cur = top;

cout << endl;

while (cur != NULL) {

cout << cur->data << " ";

cur = cur->next;

}

cout << endl;

}

~Stack() {

int n;

while (top != NULL) {

pop(n);

}

delete top;

}

};

class queueStack {

Stack S1, S2;

int numItems;

public:

queueStack() {

numItems = 0;

}

bool isEmpty() {

return numItems == 0;

}

void enqueue(int value) {

S1.push(value);

numItems++;

}

int dequeue(){

int value =0 ;

while (!S1.isEmpty()) {

S1.pop(value);

S2.push(value); // store data of S1 to S2 the stack will become inverted

}

S2.pop(value);

numItems--;

return value;

}

~queueStack() {

while (!isEmpty()) {

dequeue();

}

}

};

int main() {

queueStack Q1;

Q1.enqueue(178);

Q1.enqueue(-43);

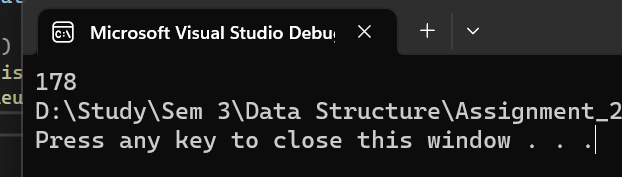
Q1.enqueue(953);

cout << Q1.dequeue();

return 0;

}

**OUTPUT :**

****

**TASK 4 :**#include<iostream>

using namespace std;

class deque {

int\* dequeArr;

int maxSize, rear, front;

int numItems;

public:

deque(int size) {

maxSize = size;

dequeArr = new int[maxSize];

rear = front = -1;

numItems = 0;

for (int i = 0; i < maxSize; i++) {

dequeArr[i] = 0;

}

}

bool isEmpty() {

return numItems == 0;

}

bool isFull() {

return numItems == maxSize - 1;

}

void insertFront(int value) {

if (!isFull()) {

if (isEmpty()) { // for insert at first time

rear = (rear + 1) % maxSize;

front = (front + 1) % maxSize;

dequeArr[rear] = value;

}

else if (rear <= front) {

if (rear <= 0) rear = maxSize - 1;

else rear--;

dequeArr[rear] = value;

}

else {

if (front <= 0) front = maxSize - 1;

else front--;

dequeArr[front] = value;

}

numItems++;

}

else {

cout << "\nQueue is full.\n";

}

}

void insertBack(int value) {

if (!isFull()) {

if (isEmpty()) { // for insert at first time

rear = (rear + 1) % maxSize;

front= (front + 1) % maxSize;

dequeArr[rear] = value;

}

else if (rear >= front) {

rear = (rear + 1) % maxSize;

dequeArr[rear] = value;

}

else {

front = (front + 1) % maxSize;

dequeArr[front] = value;

}

numItems++;

}

else {

cout << "\nQueue is full.\n";

}

}

void removeFront() {

if (!isEmpty()) {

dequeArr[front] = 0;

if (numItems == 1) {

front = -1;

rear = -1;

}

else

front = (front + 1) % maxSize;

}

else {

cout << "\nQueue is empty.\n";

}

numItems--;

}

void removeBack() {

if (!isEmpty()) {

dequeArr[rear] = 0;

if (numItems == 1) {

rear = -1;

front = -1;

}

else

rear = (rear - 1) % maxSize;

}

else {

cout << "\nQueue is empty.\n";

}

numItems--;

}

void displayQueue() {

cout << "\nFront :" << front << "| Rear :" << rear << endl;

for (int i = 0; i < maxSize; i++)

{

cout << dequeArr[i]<<" ";

}

cout << endl;

}

~deque() {

delete[] dequeArr;

maxSize = -1;

rear = -1;

front = -1;

}

};

int main() {

deque Q(10);

Q.insertBack(3);

Q.displayQueue();

Q.insertBack(14);

Q.displayQueue();

Q.insertFront(90);

Q.displayQueue();

Q.removeFront();

Q.displayQueue();

Q.removeBack();

Q.displayQueue();

Q.removeFront();

Q.displayQueue();

return 0;

}

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**