TASK 1:

#include<iostream>

#include<string>

using namespace std;

struct Node

{

char data;

Node\* next;

};

class List

{

private:

Node\* head;

int length;

public:

List();

bool IsEmpty() const;

char ReadHead() const;

void InsertAtTop(int value);

string DeleteFromHead();

};

List::List()

{

length = 0;

head = NULL;

}

bool List::IsEmpty() const

{

return (length == 0 || head == NULL);

}

char List::ReadHead() const

{

char data = ' ';

if (head != NULL)

{

data = head->data;

}

return data;

}

void List::InsertAtTop(int value)

{

Node\* newNode = new Node;

newNode->data = value;

newNode->next = head;

head = newNode;

length++;

}

string List::DeleteFromHead()

{

string data = "";

if (head != NULL)

{

Node\* temp = head;

head = temp->next;

data = temp->data;

delete temp;

length--;

}

return data;

}

class Stack

{

private:

List list;

public:

bool empty() const;

char top() const;

void push(int value);

string pop();

};

bool Stack::empty() const

{

return list.IsEmpty();

}

char Stack::top() const

{

return list.ReadHead();

}

void Stack::push(int value)

{

list.InsertAtTop(value);

}

string Stack::pop()

{

return list.DeleteFromHead();

}

class Expression

{

public:

int precedence(char op) {

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

return 1;

if (op == '\*' || op == '/')

return 2;

return 0;

}

int applyOp(int a, int b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

}

}

int evaluate(string tokens) {

int i;

Stack values;

Stack ops;

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

if (tokens[i] == ' ')

continue;

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

ops.push(tokens[i]);

}

else if (isdigit(tokens[i])) {

int val = 0;

while (i < tokens.length() && isdigit(tokens[i]))

{

val = (val \* 10) + (tokens[i] - '0');

i++;

}

values.push(val);

i--;

}

else if (tokens[i] == ')')

{

while (!ops.empty() && ops.top() != '(')

{

int val2 = values.top();

values.pop();

int val1 = values.top();

values.pop();

char op = ops.top();

ops.pop();

values.push(applyOp(val1, val2, op));

}

if (!ops.empty())

ops.pop();

}

else

{

while (!ops.empty() && precedence(ops.top())

>= precedence(tokens[i])) {

int val2 = values.top();

values.pop();

int val1 = values.top();

values.pop();

char op = ops.top();

ops.pop();

values.push(applyOp(val1, val2, op));

}

ops.push(tokens[i]);

}

}

while (!ops.empty()) {

int val2 = values.top();

values.pop();

int val1 = values.top();

values.pop();

char op = ops.top();

ops.pop();

values.push(applyOp(val1, val2, op));

}

return values.top();

}

};

void main()

{

string str;

Expression E;

cout << "Please enter an infix Expression:" << endl;

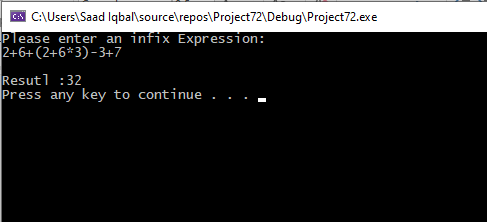
getline(cin, str);

cout << endl;

cout << "Resutl :" << E.evaluate(str) << endl;

system("pause");

}



TASK 2:

#include <iostream>

#include <stack>

using namespace std;

bool isOperator(char x)

{

switch (x) {

case '+':

case '-':

case '/':

case '\*':

return true;

}

return false;

}

struct node

{

string data;

node\* next;

};

class linkedliststack

{

private:

node\* top;

public:

linkedliststack()

{

top = NULL;

}

string push(string data)

{

node\* temp = new node;

if (!temp)

{

cout << "Stack Overflow";

}

else

{

temp->data = data;

temp->next = top;

top = temp;

}

return data;

}

int isEmpty()

{

return top == NULL;

}

string peek()

{

if (!isEmpty())

return top->data;

else

exit(1);

}

void pop()

{

struct node\* temp;

if (top == NULL)

{

cout << "Stack Underflow" << endl;

}

else

{

temp = top;

top = top->next;

temp->next = NULL;

delete temp;

}

}

};

string preToPost(string pre\_exp)

{

linkedliststack s;

int length = pre\_exp.size();

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

{

if (isOperator(pre\_exp[i]))

{

string op1 = s.peek();

s.pop();

string op2 = s.peek();

s.pop();

string temp = op1 + op2 + pre\_exp[i];

s.push(temp);

}

else

{

s.push(string(1, pre\_exp[i]));

}

}

return s.peek();

}

int main()

{

string pre\_exp;

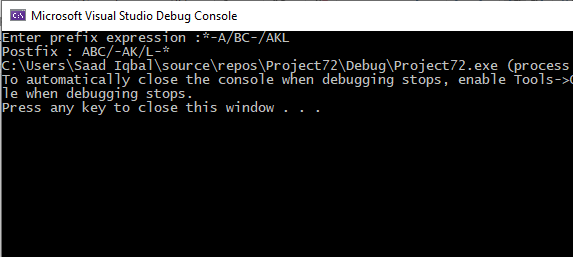
cout << "Enter prefix expression :";

cin >> pre\_exp;

cout << "Postfix : " << preToPost(pre\_exp);

return 0;

}



TASK 3:

#include<iostream>

using namespace std;

class queue

{

int data[200], front, rear, size;

public:

queue(int x)

{

front = -1;

rear = -1;

size = x;

}

void enqueue(int x)

{

if (is\_empty())

{

front++;

rear++;

data[rear] = x;

}

else if (is\_full())

{

cout << "Queue is full " << endl;

}

else

{

rear = (rear + 1) % size;

data[rear] = x;

}

}

int dequeue()

{

if (is\_empty())

{

return 0;

}

else if (front == rear)

{

int a = data[front];

front = rear = -1;

return a;

}

else

{

int a = data[front];

front = (front + 1) % size;

return a;

}

}

int is\_empty()

{

if (front == -1)

return 1;

else

return 0;

}

int is\_full()

{

if ((rear + 1) % size == front)

return 1;

else

return 0;

}

void print()

{

int a = front;

do

{

cout << data[a] << " ";

a = (a + 1) % size;

} while (a != (rear + 1) % size);

cout << endl;

}

void rotate()

{

int temparr[100] = { 0 };

int i = 0;

while (!(front < 0))

{

temparr[i] = dequeue();

i++;

}

int c = 0;

int size = 0;

size = sizeof(temparr) / sizeof(int);

int k = 0;

while (!(temparr[k] == 0))

{

k++;

c++;

}

int temp = temparr[0];

temparr[0] = temparr[c - 1];

temparr[c - 1] = temp;

int kk = 0;

while (!(temparr[kk] == 0))

{

enqueue(temparr[kk]);

kk++;

}

}

};

int main()

{

int a, num;

cout << "Please Enter the number of elements : ";

cin >> num;

queue q(num);

cout << "Please enter element in queue " << endl;

while (num--)

{

cin >> a;

q.enqueue(a);

}

cout << "Original Queue " << endl;

q.print();

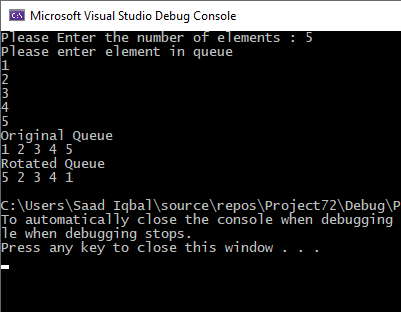
q.rotate();

cout << "Rotated Queue" << endl;

q.print();

return 0;

}



TASK 4:

#include<iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

}\*heading;;

class queue

{

int size;

Node\* temp;

Node\* tail;

public:

queue()

{

heading = NULL;

tail = NULL;

size = 0;

}

~queue()

{}

void enqueue(int data)

{

temp = new Node;

temp->data = data;

temp->next = NULL;

if (heading == NULL)

{

tail = temp;

heading = temp;

}

else

{

tail->next = temp;

tail = temp;

}

size++;

}

void display()

{

if (heading == NULL)

{

cout << "Queue is empty \n";

}

else

{

Node\* ptr = heading;

cout << "Queue elements are: front->";

while (ptr != NULL)

{

cout << ptr->data << "->";

ptr = ptr->next;

}

cout << "rear";

cout << endl;

}

}

Node\* reverse(Node\* node)

{

if (node == NULL)

return NULL;

if (node->next == NULL) {

heading = node;

return node;

}

Node\* node1 = reverse(node->next);

node1->next = node;

node->next = NULL;

return node;

}

};

int main()

{

queue g;

int op;

g.enqueue(4);

g.enqueue(6);

g.enqueue(2);

g.enqueue(5);

g.enqueue(3);

a: cout << "Press 1 to enqueue values :"<<endl;

cout << "Press 2 to display values :" << endl;

cout << "Press 3 to reverse values :" << endl;

cout << "Press 4 to exit the program:" << endl;

cin >> op;

while (1)

{

if (op == 1)

{

g.enqueue(4);

g.enqueue(6);

g.enqueue(2);

g.enqueue(5);

g.enqueue(3);

goto a;

}

if (op == 2)

{

g.display();

goto a;

}

if (op == 3)

{

g.reverse(heading);

cout << "After reversing :" << endl;

g.display();

goto a;

}

if(op==4)

{

break;

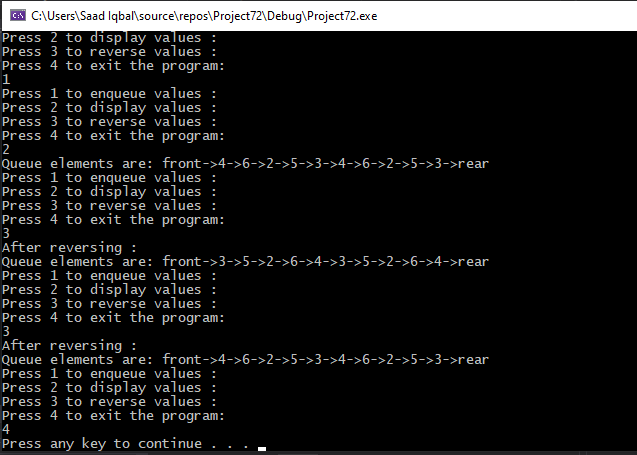
}

}

system("pause");

return 0;

}



TASK 5

#include<iostream>

using namespace std;

class queue

{

int data[200], front, rear, size;

public:

queue(int x)

{

front = -1;

rear = -1;

size = x;

}

void enqueue(int x)

{

if (is\_empty())

{

front++;

rear++;

data[rear] = x;

}

else if (is\_full())

{

cout << "Queue is full " << endl;

}

else

{

rear = (rear + 1) % size;

data[rear] = x;

}

}

int dequeue()

{

if (is\_empty())

{

return 0;

}

else if (front == rear)

{

int a = data[front];

front = rear = -1;

return a;

}

else

{

int a = data[front];

front = (front + 1) % size;

return a;

}

}

int is\_empty()

{

if (front == -1)

return 1;

else

return 0;

}

int is\_full()

{

if ((rear + 1) % size == front)

return 1;

else

return 0;

}

void print()

{

int a = front;

do

{

cout << data[a] << " ";

a = (a + 1) % size;

} while (a != (rear + 1) % size);

cout << endl;

}

void sort()

{

int a, b, i = 0, j, check;

int l = size;

while (l--)

{

if (is\_empty())

break;

a = dequeue();

if (is\_empty())

break;

b = dequeue();

j = size - i - 1;

while (j--)

{

if (a < b)

{

if (front != 0)

{

enqueue(a);

a = dequeue();

}

else

break;

}

else

{

if (front != 0)

{

enqueue(b);

b = dequeue();

}

else

break;

}

}

if (a > b)

{

enqueue(b);

enqueue(a);

}

else

{

enqueue(a);

enqueue(b);

}

front = 0;

rear = size - 1;

i++;

}

}

};

int main()

{

int a, num;

cout << "Please Enter the number of elements : ";

cin >> num;

queue q(num);

cout << "Please enter element in queue " << endl;

while (num--)

{

cin >> a;

q.enqueue(a);

}

cout << "Original Queue " << endl;

q.print();

q.sort();

cout << "Sorted Queue" << endl;

q.print();

return 0;

}

