**M. Ali. Arslan**

**19F-0348**

**Lab-04**

**Task # 01**

#include<iostream>

#include<stack>

#include<string>

using namespace std;

class st {

public:

bool op(char c) {

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

return true;

}

else {

return false;

}

}

string PreToPost(string prefix) {

stack < string > s;

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

if (op(prefix[i])) {

string op(1, prefix[i]);

s.push(op);

}

else {

string op1 = s.top();

s.pop();

string op2 = s.top();

s.pop();

s.push(op1 + op2 + prefix[i]);

}

}

return s.top();

}

};

int main() {

st a;

/\*eval p;\*/

string prefix, postfix;

cout << "Enter a prefix expression: ";

cin >> prefix;

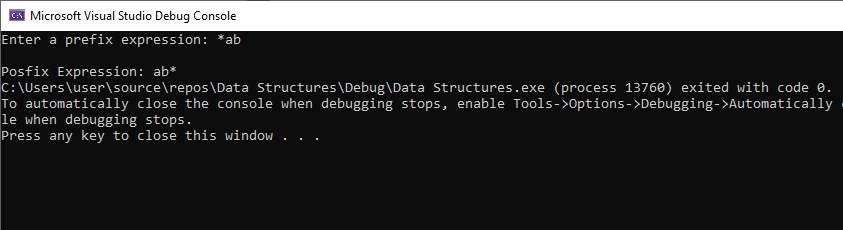
postfix = a.PreToPost(prefix);

cout << "\nPosfix Expression: " << postfix;

/\* p.postfixEval(postfix);\*/

return 0;

}



**Task # 03**

#include <iostream>

using namespace std;

class Node

{

public:

int index;

int data;

Node\* link;

Node()

{

index = 0;

data = 0;

link = NULL;

}

};

class Queue

{

private:

int count;

Node\* front = NULL;

Node\* rear = NULL;

public:

Queue()

: count(0)

{

front = NULL;

rear = NULL;

}

bool isEmpty() {

if (front == NULL && rear == NULL)

{

return true;

}

else

{

return false;

}

};

bool IsFull() const

{

Node\* temp;

try {

temp = new Node;

delete temp;

return false;

}

catch (bad\_alloc exception)

{

return true;

}

}

void enqueue(int d)

{

Node\* ptr = new Node;

if (isEmpty())

{

ptr->data = d;

front = rear = ptr;

}

else

{

ptr->data = d;

rear->link = ptr;

rear = ptr;

}

ptr->index = ++count;

};

int dequeue()

{

int data;

if (isEmpty())

{

cout << "Queue is empty, cannot dequeue!" << endl;

return -1;

}

else if (front == rear)

{

data = front->data;

//cout << "Dequeued: " << data << endl;

front = rear = NULL;

return data;

}

else

{

Node\* ptr = front;

data = front->data;

front = front->link;

ptr = NULL;

//cout << "Dequeued: " << data << endl;

return data;

}

};

void display()

{

if (isEmpty())

{

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

}

else

{

cout << endl;

Node\* temp = front;

while (temp != NULL)

{

cout << temp->data << "-->";

temp = temp->link;

}

cout << endl;

}

};

void rotation()

{

system("cls");

display();

int frontVal = dequeue();

enqueue(frontVal);

cout << "Rotation Performed!" << endl;

display();

}

};

int main()

{

Queue q1;

q1.enqueue(1);

q1.enqueue(2);

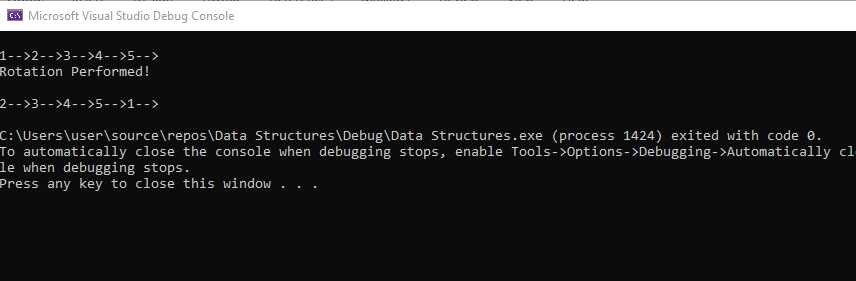
q1.enqueue(3);

q1.enqueue(4);

q1.enqueue(5);

q1.rotation();

}



**Task # 04**

#include <iostream>

using namespace std;

class Node

{

public:

int index;

int data;

Node\* link;

Node()

{

index = 0;

data = 0;

link = NULL;

}

};

class Stack

{

Node\* top;

public:

Stack()

{

top = NULL;

};

bool isempty()

{

if (top == NULL)

return true;

else

return false;

}

bool IsFull() const

{

Node\* temp;

try {

temp = new Node;

delete temp;

return false;

}

catch (bad\_alloc exception)

{

return true;

}

}

void push(int value)

{

Node\* ptr = new Node();

ptr->data = value;

ptr->link = top;

top = ptr;

}

int pop()

{

int x = 0;

if (isempty())

cout << "Stack is Empty";

else

{

Node\* ptr = top;

x = ptr->data;

top = top->link;

delete(ptr);

}

return x;

}

void showTop()

{

if (isempty())

cout << "Stack is Empty";

else

cout << "Element at top is : " << top->data;

}

};

class Queue

{

private:

int count;

Node\* front = NULL;

Node\* rear = NULL;

Stack s1;

public:

Queue()

: count(0)

{

front = NULL;

rear = NULL;

}

bool isEmpty() {

if (front == NULL && rear == NULL)

{

return true;

}

else

{

return false;

}

};

bool IsFull() const

{

Node\* temp;

try {

temp = new Node;

delete temp;

return false;

}

catch (bad\_alloc exception)

{

return true;

}

}

void enqueue(int d)

{

Node\* ptr = new Node;

if (isEmpty())

{

ptr->data = d;

front = rear = ptr;

}

else

{

ptr->data = d;

rear->link = ptr;

rear = ptr;

}

ptr->index = ++count;

};

int dequeue()

{

int data;

if (isEmpty())

{

cout << "Queue is empty, cannot dequeue!" << endl;

return -1;

}

else if (front == rear)

{

data = front->data;

//cout << "Dequeued: " << data << endl;

front = rear = NULL;

return data;

}

else

{

Node\* ptr = front;

data = front->data;

front = front->link;

ptr = NULL;

//cout << "Dequeued: " << data << endl;

return data;

}

};

void display()

{

if (isEmpty())

{

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

}

else

{

cout << endl;

Node\* temp = front;

while (temp != NULL)

{

cout << temp->data << "-->";

temp = temp->link;

}

cout << endl;

}

};

void reverse()

{

display();

Node\* temp = front;

while (!isEmpty())

{

s1.push(dequeue());

};

while (!s1.isempty())

{

enqueue(s1.pop());

};

cout << "Reversed Queue: " << endl;

display();

}

};

int main()

{

Queue q1;

q1.enqueue(1);

q1.enqueue(2);

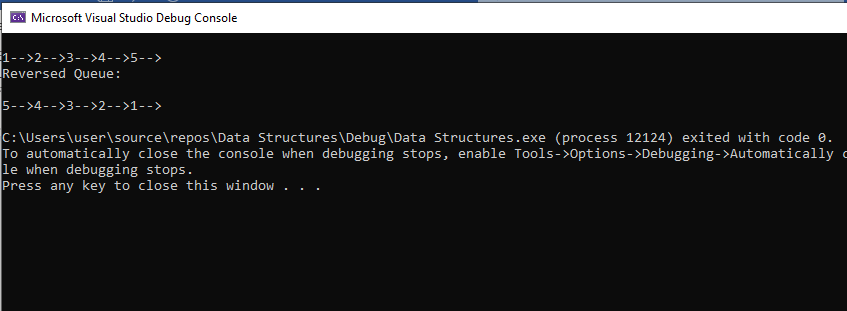
q1.enqueue(3);

q1.enqueue(4);

q1.enqueue(5);

q1.reverse();

}



**Task # 05**

#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 (isempty())

{

front++;

rear++;

data[rear] = x;

}

else if (isfull())

{

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

}

else

{

rear = (rear + 1) % size;

data[rear] = x;

}

}

int dequeue()

{

if (isempty())

{

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 isempty()

{

if (front == -1)

return 1;

else

return 0;

}

int isfull()

{

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 (isempty())

break;

a = dequeue();

if (isempty())

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 << "Enter number of elements: ";

cin >> num;

queue q(num);

cout << "Enter elements of queue: " << endl;

while (num--)

{

cin >> a;

q.enqueue(a);

}

q.sort();

cout << "Sorted Queue is: ";

q.print();

return 0;

}

