**ALI HASSAN 03-135211-005**

**ASSIGNMENT 5**

**TASK 1**

#include <iostream>

using namespace std;

#define size 5

class Stack {

int\* stack\_arr;

int top;

int stack\_current;

public:

Stack();

void push(int);

int pop();

bool stack\_isEmpty();

bool stack\_isFull();

void display();

void enQueue(int, Stack\*);

int deQueue(Stack\*);

int totalVal();

};

Stack::Stack() {

stack\_arr = new int[size];

top = -1;

stack\_current = 0;

}

bool Stack::stack\_isEmpty() {

if (top == -1)

return true;

return false;

}

bool Stack::stack\_isFull() {

if (top == size - 1)

return true;

return false;

}

void Stack :: push(int val) {

if (stack\_isFull()) {

cout << "Cannot add more values, stack full!" << endl;

exit(0);

}

else {

stack\_arr[++top] = val;

stack\_current++;

}

}

int Stack::pop() {

int val = 0;

if (stack\_isEmpty()) {

cout << "cannot remove more values, stack empty!" << endl;

}

else {

val = stack\_arr[top];

top--;

}

return val;

}

void Stack::display() {

cout << "Qeueue: ";

for (int i = top; i >= 0; i--) {

cout << stack\_arr[i] << "\t";

}

}

void Stack::enQueue(int val, Stack\* s1) {

while (!s1->stack\_isEmpty()) {

this->push(s1->pop());

}

s1->push(val);

while (!this->stack\_isEmpty()) {

s1->push(this->pop());

}

}

int Stack:: deQueue(Stack \*s1) {

int val = 0;

string err = "Queue is empty!\n";

try {

if (s1->stack\_isEmpty())

throw err;

else {

val = s1->pop();

}

} catch(string err){

cout << err;

}

return val;

}

int Stack::totalVal() {

return stack\_current;

}

int main() {

Stack s1, s2;

s2.enQueue(4, &s1);

s2.enQueue(2, &s1);

s2.enQueue(0, &s1);

s1.display();

cout << endl;

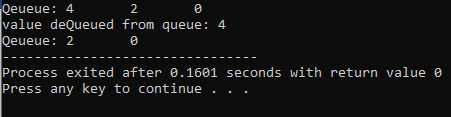
cout << "value deQueued from queue: " << s1.deQueue(&s1) << endl;

s1.display();

return 0;

}

**Output:**

****

**Task 2:**

#include <iostream>

using namespace std;

#define que 5

class Queue {

public:

int\* arr;

int front;

int back;

int current;

//public:

Queue();

bool isEmpty();

bool isFull();

void enQueue(int);

int deQueue();

void display();

int queueLenght() {

return current;

}

};

Queue::Queue() {

arr = new int[que];

front = -1;

back = -1;

current = 0;

}

bool Queue::isEmpty() {

if (current == 0)

return true;

return false;

}

bool Queue::isFull() {

if (current == que)

return true;

return false;

}

void Queue::enQueue(int val) {

if (isFull()) {

cout << "Cannot add" << val << ", queue overflow" << endl;

}

else {

arr[++back] = val;

if (front == -1)

front++;

current++;

}

}

void Queue::display() {

for (int i = front; i <= back; i++) {

cout << arr[i] << "\t";

}

}

int Queue::deQueue() {

int val = 0;

if (isEmpty()) {

cout << "Dequeue stopped, queue underflow." << endl;

}

else {

val = arr[front];

front++;

current--;

}

return val;

}

template <class T>

T reveseQueue(Queue\* two) {

int\* elements;

int arr\_size = two->current;

elements = new int[arr\_size];

int i = 0;

while (!two->isEmpty()) {

elements[i] = two->deQueue();

i++;

}

for (int i = arr\_size - 1; i >= 0; i--) {

two->enQueue(elements[i]);

}

delete[] elements;

return \*two;

}

int main() {

Queue one, two;

one.enQueue(6);

one.enQueue(4);

one.enQueue(5);

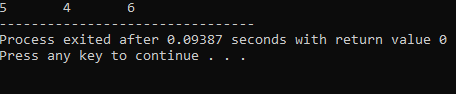
one = reveseQueue<Queue>(&one);

one.display();

return 0;

}

**Output:**

****

**Task 3:**

#include <iostream>

using namespace std;

#define que 5

class Stack {

int\* stack\_arr;

int top;

int stack\_current;

public:

Stack();

void push(int);

int pop();

bool stack\_isEmpty();

bool stack\_isFull();

void display();

};

Stack::Stack() {

stack\_arr = new int[que];

top = -1;

stack\_current = 0;

}

bool Stack::stack\_isEmpty() {

if (top == -1)

return true;

return false;

}

bool Stack::stack\_isFull() {

if (top == que - 1)

return true;

return false;

}

void Stack::push(int val) {

if (stack\_isFull()) {

cout << "Cannot add more values, stack full!" << endl;

exit(0);

}

else {

stack\_arr[++top] = val;

stack\_current++;

}

}

int Stack::pop() {

int val = 0;

if (stack\_isEmpty()) {

cout << "cannot remove more values, stack empty!" << endl;

}

else {

val = stack\_arr[top];

top--;

}

return val;

}

void Stack::display() {

cout << "Qeueue: ";

for (int i = top; i >= 0; i--) {

cout << stack\_arr[i] << "\t";

}

}

class Queue {

public:

int\* arr;

int front;

int back;

int current;

//public:

Queue();

bool isEmpty();

bool isFull();

void enQueue(int);

int deQueue();

void display();

int queueLenght() {

return current;

}

};

Queue::Queue() {

arr = new int[que];

front = -1;

back = -1;

current = 0;

}

bool Queue::isEmpty() {

if (current == 0)

return true;

return false;

}

bool Queue::isFull() {

if (current == que)

return true;

return false;

}

void Queue::enQueue(int val) {

if (isFull()) {

cout << "Cannot add" << val << ", queue overflow" << endl;

}

else {

arr[++back] = val;

if (front == -1)

front++;

current++;

}

}

void Queue::display() {

for (int i = front; i <= back; i++) {

cout << arr[i] << "\t";

}

}

int Queue::deQueue() {

int val = 0;

if (isEmpty()) {

cout << "Dequeue stopped, queue underflow." << endl;

}

else {

val = arr[front];

front++;

current--;

}

return val;

}

template <class T>

T reveseQueue2(Queue\* two) {

Stack s;

while (!two->isEmpty()) {

s.push(two->deQueue());

}

while (!s.stack\_isEmpty()) {

two->enQueue(s.pop());

}

return \*two;

}

int main() {

Queue one, two;

one.enQueue(6);

one.enQueue(4);

one.enQueue(5);

one.enQueue(4);

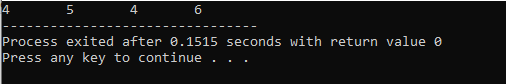
one = reveseQueue2<Queue>(&one);

one.display();

return 0;

}

**Output:**

****

**Task 4:**

#include <iostream>

using namespace std;

#define que 5

template <class T>

T queueCount(T cunt) {

return cunt;

}

class Queue {

int\* arr;

int front;

int back;

int current;

public:

Queue();

bool isEmpty();

bool isFull();

void enQueue(int);

int deQueue();

void display();

int queueLenght() {

return current;

}

Queue reverseQueue(Queue\*);

};

Queue::Queue() {

arr = new int[que];

front = -1;

back = -1;

current = 0;

}

bool Queue::isEmpty() {

if (current == 0)

return true;

return false;

}

bool Queue::isFull() {

if (current == que)

return true;

return false;

}

void Queue::enQueue(int val) {

if (isFull()) {

cout << "Cannot add" << val << ", queue overflow" << endl;

}

else {

arr[++back] = val;

if (front == -1)

front++;

current++;

}

}

void Queue::display() {

for (int i = front; i <= back; i++) {

cout << arr[i] << "\t";

}

}

int Queue::deQueue() {

int val = 0;

if (isEmpty()) {

cout << "Dequeue stopped, queue underflow." << endl;

}

else {

val = arr[front];

front++;

current--;

}

return val;

}

Queue Queue::reverseQueue(Queue\* two) {

for (int i = this->back; i >= this->front; i--) {

two->enQueue(this->arr[i]);

}

return \*two;

}

int main() {

Queue one, two;

one.enQueue(6);

one.enQueue(5);

one.enQueue(4);

cout << queueCount<int>(one.queueLenght());

return 0;

}

**Output:**

