RIPHAH
FACULTY OF REHABILITATION
& ALLIED HEALTH SCIENCES

NAME	**SAID RASOOL**
SAP	**55691**
SEC	**SE 3-2**

LAB_14:

TASK:01.

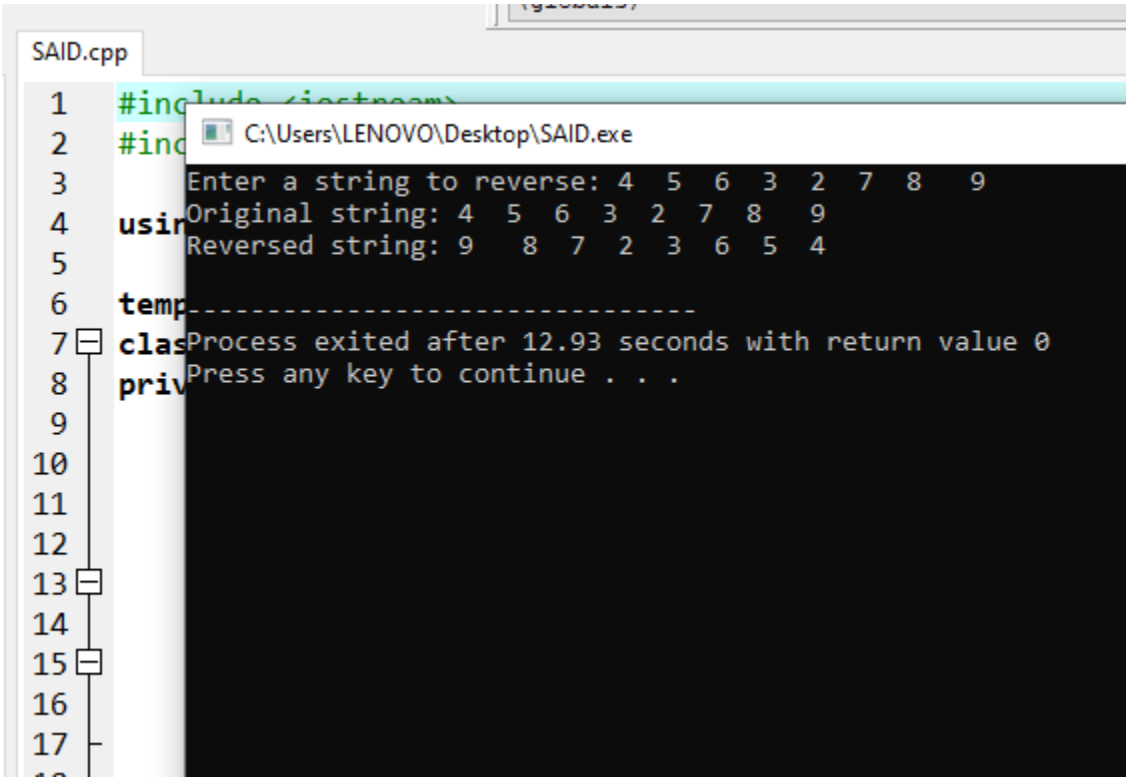
INPUT:

```
#include <iostream>

#include <string>


using namespace std;


template <typename T>

class Stack {

private:

    T* elements;

    int topIndex;

    int capacity;


    void resize(int newCapacity) {
```

```
        T* newElements = new T[newCapacity];
        for (int i = 0; i <= topIndex; ++i) {
            newElements[i] = elements[i];
        }
        delete[] elements;
        elements = newElements;
        capacity = newCapacity;
    }

public:
    // Constructor
    Stack(int initialCapacity = 10)
        : elements(new T[initialCapacity]), topIndex(-1), capacity(initialCapacity) {}

    // Destructor
    ~Stack() {
        delete[] elements;
    }


    void add(const T& item) {
        if (topIndex + 1 == capacity) {
            resize(capacity * 2);
        }
        elements[++topIndex] = item;
```

```
}



void remove() {
    if (isEmpty()) {
        cerr << "Error: Stack is empty. Cannot remove an element.\n";
        return;
    }
    --topIndex;
}

T getTop() const {
    if (isEmpty()) {
        cerr << "Error: Stack is empty. Cannot get the top element.\n";
        return T();
    }
    return elements[topIndex];
}



void clearAll() {
    topIndex = -1;
}
```

OUTPUT:



TASK:02.

INPUT: #include<iostream>

using namespace std;

class Node {

public:

Node* next;

int data;

Node(int value) {

next = NULL;

```
        data = value;
    }
};

class Queue {
    Node* front;
    Node* rear;
    int size;
    int maxSize;

public:
    Queue(int capacity) {
        front = rear = NULL;
        size = 0;
        maxSize = capacity;
    }

    bool isFull() {
        return size == maxSize;
    }

    bool isEmpty() {
        return size == 0;
    }
```

```
void enqueue(int value) {
    if (isFull()) {
        cout << "Queue Overflow! Cannot enqueue " << value << endl;
        return;
    }

    Node* newNode = new Node(value);
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    size++;
}

int dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow! Cannot dequeue." << endl;
        return -1;
    }

    Node* temp = front;
    int value = front->data;
    front = front->next;
```

```
    if (front == NULL) {
        rear = NULL;
    }

    delete temp;
    size--;
    return value;
}

int count() {
    return size;
}

void clear() {
    while (!isEmpty()) {
        dequeue();
    }
    cout << "Queue cleared." << endl;
}

void display() {
    if (isEmpty()) {
        cout << "Queue is empty!" << endl;
        return;
```

```
        }

        Node* temp = front;
        cout << "Queue contents: ";
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    Queue q(5);

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);

    q.display();

    cout << "Queue size: " << q.count() << endl;
```

```
    q.enqueue(60);


    cout << "Dequeuing: " << q.dequeue() << endl;
    q.display();


    q.clear();
    q.display();


    return 0;
}
```

OUTPUT: