Name	**Said Rasool**
Sap	**5691**
sec	**Se3-2**

Lab-8:

Code.1

Input:

```
#include <iostream>

#include <string>

using namespace std;


// Student class

class Student {

private:

    int reg_no;

    string st_name;

    float cgpa;


public:

    // Constructor

    Student() : reg_no(0), st_name(""), cgpa(0.0) {}


    // Function to input student details

    void input() {
```

```
        cout << "Enter Registration Number: ";
        cin >> reg_no;
        cin.ignore();  // Clear the newline character from the input buffer
        cout << "Enter Student Name: ";
        getline(cin, st_name);
        cout << "Enter CGPA: ";
        cin >> cgpa;
    }

    // Function to output student details
    void output() const {
        cout << "----------------------------------" << endl;
        cout << "Student Details:" << endl;
        cout << "Registration Number: " << reg_no << endl;
        cout << "Name: " << st_name << endl;
        cout << "CGPA: " << cgpa << endl;
        cout << "----------------------------------" << endl;
    }
};

// Stack class to implement stack operations
class Stack {
```

```
private:
    Student* arr; // Pointer to dynamic array of Student objects
    int top; // Index of top of stack
    int capacity; // Total capacity of stack

public:
    // Constructor to initialize stack with user-defined size
    Stack(int size) {
        arr = new Student[size];
        capacity = size;
        top = -1; // Stack is initially empty
    }

    // Destructor to free dynamically allocated memory
    ~Stack() {
        delete[] arr;
    }

    // Function to add an object to the stack (push)
    void push(const Student& s) {
        if (top == capacity - 1) {
            cout << "Error: Stack Overflow! Cannot add more students." << endl;
```

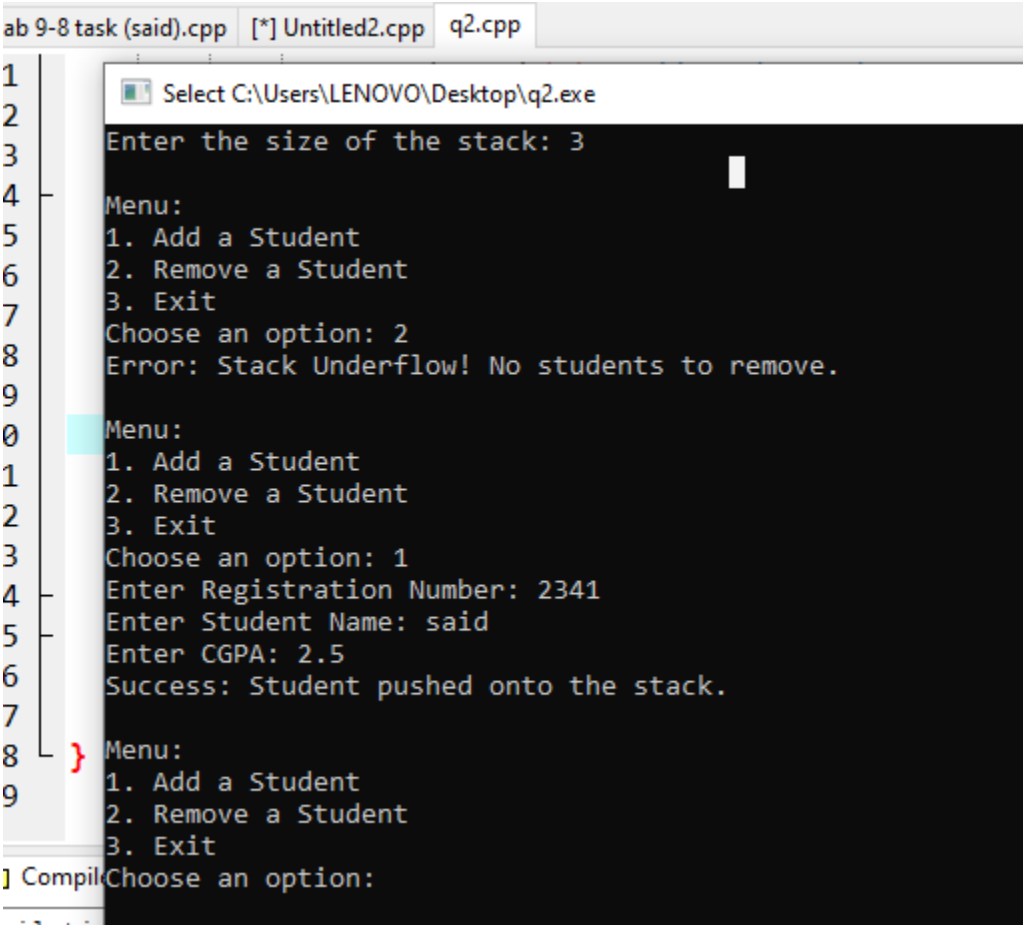
```
        return;
    }
    arr[++top] = s; // Add student to stack
    cout << "Success: Student pushed onto the stack." << endl;
}

// Function to remove an object from the stack (pop)
void pop() {
    if (top == -1) {
        cout << "Error: Stack Underflow! No students to remove." << endl;
        return;
    }
    cout << "Success: Student removed from the stack." << endl;
    arr[top--].output(); // Display and remove student
}

// Function to check if the stack is empty
bool isEmpty() const {
    return top == -1;
}

// Function to check if the stack is full
```

```
    bool isFull() const {
        return top == capacity - 1;
    }
};

int main() {
    int size;
    cout << "Enter the size of the stack: ";
    cin >> size;
    cin.ignore();  // Clear newline character after reading size

    Stack stack(size); // Create stack of user-defined size
    Student s;
    int choice;

    do {
        cout << "\nMenu:" << endl;
        cout << "1. Add a Student" << endl;
        cout << "2. Remove a Student" << endl;
        cout << "3. Exit" << endl;
        cout << "Choose an option: ";
        cin >> choice;
```

```
    switch (choice) {
    case 1:
        if (!stack.isFull()) {
            s.input();  // Input student details
            stack.push(s);  // Push student to stack
        } else {
            cout << "Error: Stack is full! Cannot add more students." << endl;
        }
        break;
    case 2:
        stack.pop();  // Pop student from stack
        break;
    case 3:
        cout << "Exiting the program. Thank you!" << endl;
        break;
    default:
        cout << "Error: Invalid choice. Please try again!" << endl;
    }
} while (choice != 3);


return 0;
```

}

Output;



Code.2:

Input:

```
#include <iostream>

#include <string>

using namespace std;


// Class for Person
```

```
class Person {
private:
    int per_id;
    string per_name;
    int per_age;
public:
    // Constructor
    Person() : per_id(0), per_name(""), per_age(0) {}

    // Input function to take user input
    void inputPerson() {
        cout << "Please enter the following details:\n";
        cout << "Person ID: ";
        cin >> per_id;
        cin.ignore(); // To clear the newline from the buffer
        cout << "Person Name: ";
        getline(cin, per_name);
        cout << "Person Age: ";
        cin >> per_age;
    }

    // Output function to display person details
```

```
    void outputPerson() const {
        cout << "======== Person Details ========\n";
        cout << "ID: " << per_id << "\n";
        cout << "Name: " << per_name << "\n";
        cout << "Age: " << per_age << "\n";
        cout << "==============================\n";
    }
};

// Node structure for the linked list
struct Node {
    Person person;
    Node* next;
};

// Queue class to manage linked list-based queue
class Queue {
private:
    Node* front; // Points to the front of the queue
    Node* rear;  // Points to the rear of the queue
public:
    // Constructor to initialize an empty queue
```

```
    Queue() : front(NULL), rear(NULL) {}


    // Function to check if the queue is empty
    bool isEmpty() const {
        return front == NULL;
    }


    // Function to add a person to the queue (enqueue)
    void addQueue(const Person& newPerson) {
        Node* newNode = new Node;
        newNode->person = newPerson;
        newNode->next = NULL;
        if (isEmpty()) {
            front = rear = newNode; // If queue is empty, front and rear both point to
the new node
        } else {
            rear->next = newNode;   // Link the new node at the end of the queue
            rear = newNode;         // Update the rear pointer to the new node
        }
        cout << "? Successfully added a person to the queue.\n";
    }
```

```
// Function to remove a person from the queue (dequeue)
void removeQueue() {
    if (isEmpty()) {
        cout << "? Queue is empty. Unable to remove a person.\n";
    } else {
        Node* temp = front; // Store the front node temporarily
        front = front->next; // Move front pointer to the next node

        if (front == NULL) { // If the queue becomes empty, set rear to NULL
            rear = NULL;
        }

        cout << "? Successfully removed a person from the queue:\n";
        temp->person.outputPerson();
        delete temp; // Delete the old front node
    }
}

// Function to display the front person in the queue without removing
void displayFrontPerson() const {
    if (!isEmpty()) {
        cout << "?? Person at the front of the queue:\n";
```

```
            front->person.outputPerson();
        } else {
            cout << "? Queue is currently empty. No person to display.\n";
        }
    }

    // Destructor to clean up the memory used by the queue
    ~Queue() {
        while (!isEmpty()) {
            removeQueue();
        }
    }
};

// Main function
int main() {
    Queue personQueue;
    Person person;
    int choice;

    do {
        cout << "\n===== Queue Menu =====\n";
```

```
cout << "1. Add Person to Queue\n";
cout << "2. Remove Person from Queue\n";
cout << "3. Display Front Person\n";
cout << "4. Check if Queue is Empty\n";
cout << "5. Exit\n";
cout << "Enter your choice: ";
cin >> choice;

switch (choice) {
case 1:
    person.inputPerson();
    personQueue.addQueue(person);
    break;

case 2:
    personQueue.removeQueue();
    break;

case 3:
    personQueue.displayFrontPerson();
    break;
```

```
        case 4:
            if (personQueue.isEmpty()) {
                cout << "? The queue is empty.\n";
            } else {
                cout << "? The queue is not empty.\n";
            }
            break;

        case 5:
            cout << "?? Exiting the program. Thank you for using our service!\n";
            break;

        default:
            cout << "?? Invalid choice. Please try again.\n";
        }
    } while (choice != 5);

    return 0;
}
```

Output:

said 8 lab.cpp

```
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152        } wh
153
154        retu
155    }
156
```

Compile Log

```
Compilation resul
--------
- Errors: 0
- Warnings: 0
```

C:\Users\LENOVO\Desktop\said 8 lab.exe

```
===== Queue Menu =====
1. Add Person to Queue
2. Remove Person from Queue
3. Display Front Person
4. Check if Queue is Empty
5. Exit
Enter your choice: 3
? Queue is currently empty. No person to

===== Queue Menu =====
1. Add Person to Queue
2. Remove Person from Queue
3. Display Front Person
4. Check if Queue is Empty
5. Exit
Enter your choice: 1
Please enter the following details:
Person ID: 5564
Person Name: said
Person Age: 19
? Successfully added a person to the queu

===== Queue Menu =====
1. Add Person to Queue
2. Remove Person from Queue
3. Display Front Person
4. Check if Queue is Empty
5. Exit
Enter your choice:
```

Code.3:

Input:

```
#include <iostream>

using namespace std;


// Node structure for linked list

struct Node {
```

```
    int data;
    Node* next;
};

// Queue class using linked list
class Queue {
private:
    Node* front;
    Node* rear;
    int size;
    static const int MAX_SIZE = 5;

public:
    Queue() : front(NULL), rear(NULL), size(0) {}
    ~Queue() {
        while (front) {
            Node* temp = front;
            front = front->next;
            delete temp;
        }
    }
```

```
bool isEmpty() { return front == NULL; }
bool isFull() { return size == MAX_SIZE; }

void enqueue(int value) {
    if (isFull()) {
        cout << "Queue Overflow!" << endl;
        return;
    }
    Node* newNode = new Node{value, NULL};
    if (rear) rear->next = newNode;
    else front = newNode;
    rear = newNode;
    size++;
}

int dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow!" << endl;
        return -1;
    }
    Node* temp = front;
    int data = front->data;
```

```
        front = front->next;
        if (!front) rear = NULL;
        delete temp;
        size--;
        return data;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return;
        }
        cout << "Queue elements: ";
        for (Node* temp = front; temp; temp = temp->next)
            cout << temp->data << " ";
        cout << endl;
    }
};

int main() {
    Queue queue;
    int choice, value;
```

```
do {
    cout << "\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
    case 1:
        cout << "Enter value to enqueue: ";
        cin >> value;
        queue.enqueue(value);
        break;
    case 2:
        value = queue.dequeue();
        if (value != -1) cout << "Dequeued: " << value << endl;
        break;
    case 3:
        queue.display();
        break;
    case 4:
        cout << "Exiting..." << endl;
        break;
```

```
        default:
            cout << "Invalid choice!" << endl;
        }
    } while (choice != 4);


    return 0;
}
```

Output: