LAB TASK(week 5)

TASK : 1

```
#include <iostream>
using namespace std;

class Queue {
private:
    static const int SIZE = 100; // Define the maximum size of the queue
    int arr[SIZE];             // Array to store queue elements
    int front;                 // Index pointing to the front of the queue
    int rear;                  // Index pointing to the rear of the queue

public:
    // Constructor to create an empty queue
    Queue() {
        front = -1;   // Front points to -1 indicating the queue is initially empty
        rear = -1;    // Rear also points to -1 when the queue is empty
    }

    // Check if the queue is empty
    bool isEmpty() {
        return front == -1;
    }
```

```
// Insert an element at the rear of the queue
void enqueue(int item) {
    if (rear == SIZE - 1) {
        cout << "Queue is full!" << endl;
    } else {
        if (front == -1) {
            front = 0; // Set front to 0 if adding the first element
        }
        rear++;
        arr[rear] = item; // Insert the element at the rear
        cout << item << " enqueued to queue" << endl;
    }
}

// Remove the element from the front of the queue
int dequeue() {
    if (isEmpty()) {
        cout << "Queue is empty!" << endl;
        return -1;
    } else {
        int item = arr[front]; // Store the front element
        if (front >= rear) {
            // If front meets or surpasses rear, reset the queue
            front = -1;
            rear = -1;
        } else {
            front++;
```

```
        }
        cout << item << " dequeued from queue" << endl;
        return item;
    }
  }

  // Display all elements of the queue
  void display() {
    if (isEmpty()) {
        cout << "Queue is empty!" << endl;
    } else {
        cout << "Queue elements: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
  }
};

int main() {
  Queue q;

  q.enqueue(10);
  q.enqueue(20);
  q.enqueue(30);
```

```
    q.display();  // Output: Queue elements: 10 20 30

    q.dequeue();  // Output: 10 dequeued from queue
    q.display();  // Output: Queue elements: 20 30

    cout << (q.isEmpty() ? "Queue is empty!" : "Queue is not empty!") << endl;  // Output: Queue is not empty!

    return 0;
}
```

TASK : 2

```
#include <iostream>
#include <cstring>  // For strtok() and strlen()
using namespace std;

#define MAX 100  // Maximum size of queue

// Function to display a character array (queue)
void displayQueue(char queue[], int size) {
    for (int i = 0; i < size; i++) {
        cout << queue[i] << " ";
    }
    cout << endl;
```

```
}

// Function to concatenate two character queues
void concatenateQueues(char queue1[], int &size1, char queue2[], int size2) {
    for (int i = 0; i < size2; i++) {
        queue1[size1++] = queue2[i];
    }
}

int main() {
    char input[] = "Data Structure and Algorithms";  // Input string
    char *words[MAX];  // Array to store words
    int wordCount = 0; // Count of words in the input string

    // Split the input string into words using strtok()
    char *token = strtok(input, " ");
    while (token != NULL) {
        words[wordCount++] = token;  // Store each word in the words array
        token = strtok(NULL, " ");
    }

    // Create character arrays for each word (simulate queues)
    char queues[wordCount][MAX];  // Array of character arrays to hold each queue
    int sizes[wordCount];  // Array to store sizes of each queue
```

```
    // Fill the queues with characters from each word
    for (int i = 0; i < wordCount; i++) {
        int length = strlen(words[i]);  // Get the length of the word
        sizes[i] = length;  // Store the size of each word
        for (int j = 0; j < length; j++) {
            queues[i][j] = words[i][j];  // Store each character in the queue
        }
    }

    // Display each queue (for visualization)
    for (int i = 0; i < wordCount; i++) {
        cout << "Q" << i + 1 << ": ";
        displayQueue(queues[i], sizes[i]);
    }

    // Concatenate all the queues into one single queue
    char concatenatedQueue[MAX];  // Array to hold the concatenated result
    int concatenatedSize = 0;

    for (int i = 0; i < wordCount; i++) {
        concatenateQueues(concatenatedQueue, concatenatedSize, queues[i],
sizes[i]);
    }
```

```
    // Display the final concatenated queue
    cout << "Concatenated Queue: ";
    displayQueue(concatenatedQueue, concatenatedSize);

    return 0;
}
```