Rajalakshmi Engineering College

Name: Seeralan .M
Email: 241501193@rajalakshmi.edu.in
Roll no: 241501193
Phone: 8610861705
Branch: REC
Department: I AIML AE
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

You are tasked with developing a simple ticket management system for a customer support department. In this system, customers submit support tickets, which are processed in a First-In-First-Out (FIFO) order. The system needs to handle the following operations:

Ticket Submission (Enqueue Operation): New tickets are submitted by customers. Each ticket is assigned a unique identifier (represented by an integer). When a new ticket arrives, it should be added to the end of the queue.

Ticket Processing (Dequeue Operation): The support team processes tickets in the order they are received. The ticket at the front of the queue is processed first. After processing, the ticket is removed from the queue.

Display Ticket Queue: The system should be able to display the current state of the ticket queue, showing the sequence of ticket identifiers from front to rear.

Input Format

The first input line contains an integer n, the number of tickets submitted by customers.

The second line consists of a single integer, representing the unique identifier of each submitted ticket, separated by a space.

Output Format

The first line displays the "Queue: " followed by the ticket identifiers in the queue after all tickets have been submitted.

The second line displays the "Queue After Dequeue: " followed by the ticket identifiers in the queue after processing (removing) the ticket at the front.

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 6
14 52 63 95 68 49
Output: Queue: 14 52 63 95 68 49
Queue After Dequeue: 52 63 95 68 49

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define a structure for a queue node
struct Node {
    int ticketID;
    struct Node* next;
};
```

```
// Define front and rear pointers for the queue
struct Node* front = NULL;
struct Node* rear = NULL;

// Function to enqueue a ticket (submit ticket)
void enqueue(int ticketID) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->ticketID = ticketID;
    newNode->next = NULL;

    // If queue is empty, both front and rear will point to the new node
    if (rear == NULL) {
        front = rear = newNode;
        return;
    }

    // Otherwise, add the new node at the end and move the rear pointer
    rear->next = newNode;
    rear = newNode;
}

// Function to dequeue a ticket (process ticket)
void dequeue() {
    if (front == NULL) {
        return; // Queue is empty, nothing to dequeue
    }

    struct Node* temp = front;
    front = front->next;

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

    free(temp); // Free the memory of the dequeued ticket
}

// Function to display the current ticket queue
void displayQueue() {
    struct Node* temp = front;
```

```
    if (temp == NULL) {
        printf("Queue is empty\n");
        return;
    }

    while (temp != NULL) {
        printf("%d ", temp->ticketID);
        temp = temp->next;
    }
}

int main() {
    int n, ticketID;

    // Read the number of tickets
    scanf("%d", &n);

    // Read the ticket IDs and enqueue them
    for (int i = 0; i < n; i++) {
        scanf("%d", &ticketID);
        enqueue(ticketID);
    }

    // Display the current state of the queue
    printf("Queue: ");
    displayQueue();
    printf("\n");

    // Perform a dequeue operation
    dequeue();

    // Display the state of the queue after dequeue
    printf("Queue After Dequeue: ");
    displayQueue();
    printf("\n");

    return 0;
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Guide Harish in developing a simple queue system for a customer service center. The customer service center can handle up to 25 customers at a time. The queue needs to support basic operations such as adding a customer to the queue, serving a customer (removing them from the queue), and displaying the current queue of customers.

Use an array for implementation.

Input Format

The first line of the input consists of an integer N, the number of customers arriving at the service center.

The second line consists of N space-separated integers, representing the customer IDs in the order they arrive.

Output Format

After serving the first customer in the queue, display the remaining customers in the queue.

If a dequeue operation is attempted on an empty queue, display "Underflow".

If the queue is empty, display "Queue is empty".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
101 102 103 104 105
Output: 102 103 104 105

Answer

```
// You are using GCC
#include <stdio.h>

#define MAX_SIZE 25 // Maximum size of the queue
```

```
// Define a structure for the queue
struct Queue {
    int arr[MAX_SIZE];
    int front, rear;
};

// Initialize the queue
void initQueue(struct Queue* queue) {
    queue->front = -1;
    queue->rear = -1;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return queue->front == -1;
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return queue->rear == MAX_SIZE - 1;
}

// Function to enqueue a customer to the queue
void enqueue(struct Queue* queue, int customerID) {
    if (isFull(queue)) {
        printf("Queue is full\n");
        return;
    }
    if (queue->front == -1) {
        queue->front = 0; // First customer arrives
    }
    queue->rear++;
    queue->arr[queue->rear] = customerID;
}

// Function to dequeue (serve) a customer
void dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Underflow\n");
        return;
    }
```

```
    for (int i = 0; i < queue->rear; i++) {
        queue->arr[i] = queue->arr[i + 1];
    }
    queue->rear--;
    if (queue->rear == -1) {
        queue->front = -1; // Reset to empty state
    }
}

// Function to display the current state of the queue
void displayQueue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->arr[i]);
    }
    printf("\n");
}

int main() {
    struct Queue queue;
    int n;

    // Initialize the queue
    initQueue(&queue);

    // Read the number of customers
    scanf("%d", &n);

    // Read the customer IDs and enqueue them
    for (int i = 0; i < n; i++) {
        int customerID;
        scanf("%d", &customerID);
        enqueue(&queue, customerID);
    }

    // Serve the first customer (dequeue operation)
    dequeue(&queue);

    // Display the remaining customers in the queue
```

```
    displayQueue(&queue);

    return 0;
}
```

Status : Correct **Marks : 10/10**

3. Problem Statement

Sharon is developing a queue using an array. She wants to provide the functionality to find the Kth largest element. The queue should support the addition and retrieval of the Kth largest element effectively. The maximum capacity of the queue is 10.

Assist her in the program.

Input Format

The first line of input consists of an integer N, representing the number of elements in the queue.

The second line consists of N space-separated integers.

The third line consists of an integer K.

Output Format

For each enqueued element, print a message: "Enqueued: " followed by the element.

The last line prints "The [K]th largest element: " followed by the Kth largest element.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
23 45 93 87 25
4

Output: Enqueued: 23
Enqueued: 45
Enqueued: 93
Enqueued: 87
Enqueued: 25
The 4th largest element: 25

Answer

```
// You are using GCC
#include <stdio.h>

#define MAX_SIZE 10 // Maximum size of the queue

// Function to sort the queue in descending order
void sortQueue(int queue[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (queue[i] < queue[j]) {
                // Swap elements if they are not in descending order
                int temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
}

int main() {
    int queue[MAX_SIZE];
    int n, k;

    // Read the number of elements in the queue
    scanf("%d", &n);

    // Read the elements and enqueue them
    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
        printf("Enqueued: %d\n", queue[i]);
    }

    // Read the value of K
    scanf("%d", &k);
```

```
    // Sort the queue in descending order
    sortQueue(queue, n);

    // The Kth largest element is the element at index K-1
    printf("The %dth largest element: %d\n", k, queue[k-1]);

    return 0;
}
```

Status : Correct **Marks : 10/10**

4. Problem Statement

Amar is working on a project where he needs to implement a special type of queue that allows selective dequeuing based on a given multiple. He wants to efficiently manage a queue of integers such that only elements not divisible by a given multiple are retained in the queue after a selective dequeue operation.

Implement a program to assist Amar in managing his selective queue.

Example

Input:

5

10 2 30 4 50

5

Output:

Original Queue: 10 2 30 4 50

Queue after selective dequeue: 2 4

Explanation:

After selective dequeue with a multiple of 5, the elements that are multiples of 5 should be removed. Therefore, only 10, 30, and 50 should be removed from the queue. The updated Queue is 2 4.

Input Format

The first line contains an integer n, representing the number of elements initially present in the queue.

The second line contains n space-separated integers, representing the elements of the queue.

The third line contains an integer multiple, representing the divisor for selective dequeue operation.

Output Format

The first line of output prints "Original Queue: " followed by the space-separated elements in the queue before the dequeue operation.

The second line prints "Queue after selective dequeue: " followed by the remaining space-separated elements in the queue, after deleting elements that are the multiples of the specified number.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
10 2 30 4 50
5
Output: Original Queue: 10 2 30 4 50
Queue after selective dequeue: 2 4

Answer

```
// You are using GCC
#include <stdio.h>

int main() {
    int n, multiple;

    // Read the number of elements in the queue
    scanf("%d", &n);

    int queue[n];
```

```
    // Read the elements into the queue
    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
    }

    // Read the divisor (multiple)
    scanf("%d", &multiple);

    // Print the original queue
    printf("Original Queue: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");

    // Perform selective dequeue (remove elements divisible by 'multiple')
    printf("Queue after selective dequeue: ");
    for (int i = 0; i < n; i++) {
        if (queue[i] % multiple != 0) {  // Only print elements that are not divisible by
the multiple
            printf("%d ", queue[i]);
        }
    }
    printf("\n");

    return 0;
}
```

Status : Correct **Marks : 10/10**

5. Problem Statement

You've been assigned the challenge of developing a queue data structure using a linked list.

The program should allow users to interact with the queue by enqueuing positive integers and subsequently dequeuing and displaying elements.

Input Format

The input consists of a series of integers, one per line. Enter positive integers into the queue.

Enter -1 to terminate input.

Output Format

The output prints the space-separated dequeued elements.

Refer to the sample output for the exact text and format.

Sample Test Case

Input: 1
2
3
4
-1
Output: Dequeued elements: 1 2 3 4

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Structure for a queue node
struct Node {
    int data;
    struct Node* next;
};

// Function to enqueue a new element
void enqueue(struct Node** front, struct Node** rear, int value) {
    // Create a new node
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    // If the queue is empty, the new node is both the front and rear
    if (*rear == NULL) {
        *front = *rear = newNode;
```

```
        return;
    }

    // Add the new node at the end and update rear
    (*rear)->next = newNode;
    *rear = newNode;
}

// Function to dequeue an element from the queue
int dequeue(struct Node** front) {
    // If the queue is empty, return a sentinel value (e.g., -1)
    if (*front == NULL) {
        return -1;
    }

    // Store the current front element
    struct Node* temp = *front;
    int value = temp->data;

    // Move the front pointer to the next node
    *front = (*front)->next;

    // Free the memory of the dequeued node
    free(temp);

    return value;
}

// Function to display the dequeued elements
void displayDequeued(struct Node* front) {
    printf("Dequeued elements: ");

    // If the queue is empty, print nothing
    if (front == NULL) {
        printf("\n");
        return;
    }

    // Dequeue all elements and print them
    while (front != NULL) {
        printf("%d ", front->data);
        front = front->next;
```

```
    }
    printf("\n");
}

int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;
    int value;

    // Read input values until -1 is encountered
    while (1) {
        scanf("%d", &value);
        if (value == -1) {
            break;
        }
        enqueue(&front, &rear, value);
    }

    // Display the dequeued elements
    displayDequeued(front);

    return 0;
}
```

Status : Correct ***Marks : 10/10***