Rajalakshmi Engineering College

Name: Jhanani shree
Email: 240701215@rajalakshmi.edu.in
Roll no: 240701215
Phone: 7373333511
Branch: REC
Department: I CSE AH
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 40

Section 1 : Coding

1. 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>
#include <stdlib.h>

#define MAX_SIZE 10

// Function to enqueue elements and find the Kth largest element
void enqueueAndFindKthLargest(int queue[], int *size, int N, int K) {
    for (int i = 0; i < N; i++) {
        if (*size < MAX_SIZE) {
            queue[*size] = queue[i]; // Add to the queue
            printf("Enqueued: %d\n", queue[i]);
            (*size)++;
        }
    }

    // Sort the queue to find the Kth largest element
```

```
    for (int i = 0; i < *size - 1; i++) {
        for (int j = 0; j < *size - i - 1; j++) {
            if (queue[j] < queue[j + 1]) {
                // Swap
                int temp = queue[j];
                queue[j] = queue[j + 1];
                queue[j + 1] = temp;
            }
        }
    }

    // Print the Kth largest element
    if (K <= *size) {
        printf("The %dth largest element: %d\n", K, queue[K - 1]);
    } else {
        printf("Not enough elements in the queue to find the %dth largest element.
\n", K);
    }
}

// Main function
int main() {
    int N, K;
    int queue[MAX_SIZE];
    int size = 0;

    // Read number of elements
    scanf("%d", &N);

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

    // Read K
    scanf("%d", &K);

    // Enqueue elements and find the Kth largest element
    enqueueAndFindKthLargest(queue, &size, N, K);

    return 0;
}
```

Status : Correct *Marks : 10/10*

2. 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>

#define MAX_SIZE 50

// Function to manage the selective queue
void selectiveDequeue(int queue[], int *size, int multiple) {
    int newQueue[MAX_SIZE];
    int newSize = 0;

    // Iterate over the original queue and retain only elements not divisible by the
multiple
    for (int i = 0; i < *size; i++) {
        if (queue[i] % multiple != 0) {
            newQueue[newSize++] = queue[i];
        }
    }
```

```
    // Update the original queue and its size
    for (int i = 0; i < newSize; i++) {
        queue[i] = newQueue[i];
    }
    *size = newSize;
}

// Main function
int main() {
    int n, multiple;
    int queue[MAX_SIZE];

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

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

    // Read the multiple for selective dequeue
    scanf("%d", &multiple);

    // Print the original queue
    printf("Original Queue: ");
    for (int i = 0; i < n; i++) {
        printf("%d", queue[i]);
        if (i < n - 1) {
            printf(" ");
        }
    }
    printf("\n");

    // Perform selective dequeue
    selectiveDequeue(queue, &n, multiple);

    // Print the updated queue
    printf("Queue after selective dequeue: ");
    for (int i = 0; i < n; i++) {
        printf("%d", queue[i]);
        if (i < n - 1) {
```

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

    return 0;
}
```

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

3. 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_CUSTOMERS 25

// Function to display the queue
void displayQueue(int queue[], int size) {
    if (size == 0) {
        printf("Queue is empty\n");
    } else {
        for (int i = 0; i < size; i++) {
            printf("%d", queue[i]);
            if (i < size - 1) {
                printf(" ");
            }
        }
        printf("\n");
    }
}

// Function to serve (dequeue) a customer
void serveCustomer(int queue[], int *size) {
    if (*size == 0) {
        printf("Underflow\n");
    } else {
        // Remove the first customer
        for (int i = 1; i < *size; i++) {
            queue[i - 1] = queue[i];
        }
        (*size)--; // Decrease the size of the queue
    }
}

// Main function
int main() {
```

```
    int n;
    int queue[MAX_CUSTOMERS];
    int size = 0;

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

    // Read customer IDs into the queue
    if (n > 0 && n <= MAX_CUSTOMERS) {
        for (int i = 0; i < n; i++) {
            scanf("%d", &queue[i]);
        }
        size = n;
    }

    // Serve the first customer
    serveCustomer(queue, &size);

    // Display the remaining customers in the queue
    displayQueue(queue, size);

    return 0;
}
```

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

4. 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>

// Define the structure for a queue node
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Define the structure for the queue
typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

// Function to create a new queue
Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->front = NULL;
    queue->rear = NULL;
    return queue;
}
```

```
// Function to enqueue an element
void enqueue(Queue* queue, int value) {
    if (value <= 0) return; // Only enqueue positive integers

    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;

    if (queue->rear == NULL) {
        // If the queue is empty, both front and rear are the new node
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        // Add the new node at the end and update rear
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}

// Function to dequeue an element
int dequeue(Queue* queue) {
    if (queue->front == NULL) {
        return -1; // Queue is empty
    }

    Node* temp = queue->front;
    int dequeuedValue = temp->data;
    queue->front = queue->front->next;

    // If the front is NULL, then also set rear to NULL
    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    free(temp);
    return dequeuedValue;
}

// Function to display the dequeued elements
void displayDequeuedElements(Queue* queue) {
    Node* current = queue->front;
    printf("Dequeued elements: ");
```

```
    while (current != NULL) {
        printf("%d", current->data);
        current = current->next;
        if (current != NULL) {
            printf(" ");
        }
    }
    printf("\n");
}

// Main function
int main() {
    Queue* queue = createQueue();
    int input;

    // Read integers until -1 is entered
    while (1) {
        scanf("%d", &input);
        if (input == -1) {
            break;
        }
        enqueue(queue, input);
    }

    // Dequeue all elements and display them
    int value;
    while ((value = dequeue(queue)) != -1) {
        printf("%d ", value);
    }
    printf("\n");

    // Display the dequeued elements
    displayDequeuedElements(queue);

    // Free the queue memory
    free(queue);

    return 0;
}
```

Status : Wrong **Marks : 0/10**

5. 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 the structure for a queue node
typedef struct Node {
    int ticketID;
    struct Node* next;
} Node;

// Define the structure for the queue
typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

// Function to create a new queue
Queue* createQueue() {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->front = NULL;
    queue->rear = NULL;
    return queue;
}

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

    if (queue->rear == NULL) {
        // If the queue is empty, both front and rear are the new node
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        // Add the new node at the end and update rear
```

```
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}

// Function to dequeue a ticket
int dequeue(Queue* queue) {
    if (queue->front == NULL) {
        return -1; // Queue is empty
    }

    Node* temp = queue->front;
    int ticketID = temp->ticketID;
    queue->front = queue->front->next;

    // If the front is NULL, then also set rear to NULL
    if (queue->front == NULL) {
        queue->rear = NULL;
    }

    free(temp);
    return ticketID;
}

// Function to display the current state of the ticket queue
void displayQueue(Queue* queue) {
    Node* current = queue->front;
    while (current != NULL) {
        printf("%d", current->ticketID);
        current = current->next;
        if (current != NULL) {
            printf(" ");
        }
    }
}

// Main function
int main() {
    int n;
    scanf("%d", &n);

    Queue* queue = createQueue();
```

```
    int ticketID;

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

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

    // Dequeue the first ticket
    dequeue(queue);

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

    // Free the queue memory
    free(queue);

    return 0;
}
```

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