Rajalakshmi Engineering College

Name: Prajeet V
Email: 240701389@rajalakshmi.edu.in
Roll no: 240701389
Phone: 9363389322
Branch: REC
Department: I CSE FD
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 : 50

Section 1 : Coding

1. 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 25


typedef struct {
    int items[MAX];
    int front, rear;
} Queue;


void initQueue(Queue* q) {
    q->front = 0;
    q->rear = -1;
}


int isEmpty(Queue* q) {
    return q->front > q->rear;
}
```

```
void enqueue(Queue* q, int customerID) {
    if (q->rear < MAX - 1) {
        q->items[++(q->rear)] = customerID;
    }
}

void dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Underflow\n");
        printf("Queue is empty\n");
        return;
    }
    q->front++;
}


void displayQueue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initQueue(&q);

    int n, customerID;
    scanf("%d", &n);

    if (n == 0) {
        dequeue(&q);
        return 0;
    }

    for (int i = 0; i < n; i++) {
        scanf("%d", &customerID);
```

```
        enqueue(&q, customerID);
    }

    dequeue(&q);

    displayQueue(&q);

    return 0;
}
```

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

2. 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>

// Definition of Node structure for the queue
struct Node {
    int data;
    struct Node* next;
};

// Main function to run the queue operations
int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;
    struct Node* temp;
    int value;

    // Read and enqueue positive integers until -1 is entered
    while (1) {
        scanf("%d", &value);
        if (value == -1) break;

        // Create a new node and assign data
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = value;
        newNode->next = NULL;

        // If the queue is empty, new node becomes both front and rear
        if (rear == NULL) {
            front = rear = newNode;
        } else {
            rear->next = newNode;  // Link new node at the end
            rear = newNode;        // Update rear to point to new node
        }
    }

    // Print dequeued elements
```

```
    printf("Dequeued elements: ");

    // Dequeue and print elements until the queue is empty
    int first = 1;
    while (front != NULL) {
        value = front->data;    // Get the data of front element
        temp = front;           // Store front node to free it later
        front = front->next;    // Move front to the next node

        // Print the dequeued element
        if (first) {
            printf("%d", value);  // Print first element without leading space
            first = 0;
        } else {
            printf(" %d", value); // Print subsequent elements with space
        }

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

    printf("\n");
    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>
#include <stdlib.h>

#define MAX_SIZE 10

// Function to enqueue an element into the queue
void enqueue(int queue[], int *rear, int element) {
    if (*rear < MAX_SIZE - 1) {
        queue[++(*rear)] = element;
        printf("Enqueued: %d\n", element);
    } else {
        printf("Queue is full. Cannot enqueue %d.\n", element);
    }
}
```

```
// Function to find the Kth largest element in the queue
int findKthLargest(int queue[], int size, int k) {
    // Sort the array in descending order
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (queue[i] < queue[j]) {
                // Swap elements
                int temp = queue[i];
                queue[i] = queue[j];
                queue[j] = temp;
            }
        }
    }
    // Return the Kth largest element (1-based index, so we need to adjust for 0-based index)
    return queue[k - 1];
}

int main() {
    int N, K;
    int queue[MAX_SIZE];
    int rear = -1;

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

    // Input the elements and enqueue them
    for (int i = 0; i < N; i++) {
        int element;
        scanf("%d", &element);
        enqueue(queue, &rear, element);
    }

    // Input K for Kth largest element
    scanf("%d", &K);

    // Find and print the Kth largest element
    int kthLargest = findKthLargest(queue, N, K);
    printf("The %dth largest element: %d\n", K, kthLargest);

    return 0;
```

}

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

4. 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

```
#include <stdio.h>

#define MAX 20


typedef struct {
    int items[MAX];
    int front, rear;
} Queue;


void initQueue(Queue* q) {
    q->front = 0;
    q->rear = -1;
}


int isEmpty(Queue* q) {
    return q->front > q->rear;
}


void enqueue(Queue* q, int ticketID) {
    if (q->rear < MAX - 1) {
        q->items[++(q->rear)] = ticketID;
    }
}


void dequeue(Queue* q) {
    if (!isEmpty(q)) {
```

```
        q->front++;
    }
}

void displayQueue(Queue* q, const char* message) {
    printf("%s", message);
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}

int main() {
    Queue q;
    initQueue(&q);

    int n, ticketID;
    scanf("%d", &n);


    for (int i = 0; i < n; i++) {
        scanf("%d", &ticketID);
        enqueue(&q, ticketID);
    }

    displayQueue(&q, "Queue: ");


    dequeue(&q);


    displayQueue(&q, "Queue After Dequeue: ");

    return 0;
}
```

Status : Correct *Marks : 10/10*

5. 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 display the queue
void displayQueue(int queue[], int size) {
    for (int i = 0; i < size; i++) {
        if (i != 0) {
            printf(" ");
        }
        printf("%d", queue[i]);
    }
    printf("\n");
}

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

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

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

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

    // Display the original queue
    printf("Original Queue: ");
    displayQueue(queue, n);

    // Create a new array to store the queue after selective dequeue
    int newQueue[MAX_SIZE];
    int newSize = 0;

    // Perform the selective dequeue operation
    for (int i = 0; i < n; i++) {
        // Add elements that are not divisible by the multiple to the newQueue
        if (queue[i] % multiple != 0) {
            newQueue[newSize++] = queue[i];
        }
    }

    // Display the updated queue after selective dequeue
    printf("Queue after selective dequeue: ");
    displayQueue(newQueue, newSize);

    return 0;
}
```

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