Rajalakshmi Engineering College

Name: RAKSHITHA R
Email: 240701418@rajalakshmi.edu.in
Roll no: 240701418
Phone: 7305274265
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_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

A customer support system is designed to handle incoming requests using a queue. Implement a linked list-based queue where each request is represented by an integer. After processing the requests, remove any duplicate requests to ensure that each request is unique and print the remaining requests.

Input Format

The first line of input consists of an integer N, representing the number of requests to be enqueued.

The second line consists of N space-separated integers, each representing a request.

Output Format

The output prints space-separated integers after removing the duplicate requests.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
2 4 2 7 5

Output: 2 4 7 5

Answer

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 10000

// Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Global queue
typedef struct {
    Node* front;
    Node* rear;
} Queue;

Queue q; // make queue global

// Initialize the queue
void initQueue() {
    q.front = q.rear = NULL;
}

// Check if the queue is empty
int isEmpty() {
    return q.front == NULL;
}
```

```
// Enqueue operation
void enqueue(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;

    if (q.rear == NULL) {
        q.front = q.rear = newNode;
        return;
    }

    q.rear->next = newNode;
    q.rear = newNode;
}

// Dequeue operation
int dequeue() {
    if (isEmpty()) {
        return -1;
    }

    Node* temp = q.front;
    int data = temp->data;
    q.front = q.front->next;

    if (q.front == NULL) {
        q.rear = NULL;
    }

    free(temp);
    return data;
}

// Check if a value is in the seen array
int isSeen(int seen[], int value, int size) {
    for (int i = 0; i < size; i++) {
        if (seen[i] == value) {
            return 1;
        }
    }
    return 0;
```

```
}

// Remove duplicates from the queue
void removeDuplicates() {
    if (isEmpty()) {
        return;
    }

    int seen[MAX];
    int seenSize = 0;

    Node* current = q.front;
    Node* prev = NULL;

    while (current != NULL) {
        if (isSeen(seen, current->data, seenSize)) {
            Node* temp = current;
            if (prev != NULL) {
                prev->next = current->next;
            } else {
                q.front = current->next;
            }
            if (current == q.rear) {
                q.rear = prev;
            }
            current = current->next;
            free(temp);
        } else {
            seen[seenSize++] = current->data;
            prev = current;
            current = current->next;
        }
    }
}

// Print the queue
void printQueue() {
    Node* temp = q.front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
```

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

int main() {
    int numElements;
    initQueue();

    scanf("%d", &numElements);

    for (int i = 0; i < numElements; ++i) {
        int element;
        scanf("%d", &element);
        enqueue(element);
    }

    removeDuplicates();
    printQueue();

    return 0;
}
```

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

2. Problem Statement

Imagine you are developing a basic task management system for a small team of software developers. Each task is represented by an integer, where positive integers indicate valid tasks and negative integers indicate erroneous tasks that need to be removed from the queue before processing.

Write a program using the queue with a linked list that allows the team to add tasks to the queue, remove all erroneous tasks (negative integers), and then display the valid tasks that remain in the queue.

Input Format

The first line consists of an integer N, representing the number of tasks to be added to the queue.

The second line consists of N space-separated integers, representing the tasks. Tasks can be both positive (valid) and negative (erroneous).

Output Format

The output displays the following format:

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

The last line displays the "Queue Elements after Dequeue: " followed by removing all erroneous (negative) tasks and printing the valid tasks remaining in the queue in the order they were enqueued.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
12 -54 68 -79 53

Output: Enqueued: 12
Enqueued: -54
Enqueued: 68
Enqueued: -79
Enqueued: 53
Queue Elements after Dequeue: 12 68 53

Answer

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* next;
} Node;

typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;
```

```
Queue q;  // Global queue

void initQueue() {
    q.front = q.rear = NULL;
}

int isEmpty() {
    return q.front == NULL;
}

void enqueue(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    if (q.rear == NULL) {
        q.front = q.rear = newNode;
    } else {
        q.rear->next = newNode;
        q.rear = newNode;
    }
    printf("Enqueued: %d\n", data);
}

void dequeueAllNegative() {
    if (isEmpty()) {
        return;
    }

    while (q.front != NULL && q.front->data < 0) {
        Node* temp = q.front;
        q.front = q.front->next;
        free(temp);
    }

    if (q.front == NULL) {
        q.rear = NULL;
        return;
    }

    Node* current = q.front;
    while (current->next != NULL) {
        if (current->next->data < 0) {
```

```
            Node* temp = current->next;
            current->next = current->next->next;
            free(temp);
            if (current->next == NULL) {
                q.rear = current;
            }
        } else {
            current = current->next;
        }
    }
}

void display() {
    Node* temp = q.front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

int main() {
    int n, value;
    initQueue();

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

    dequeueAllNegative();

    printf("Queue Elements after Dequeue: ");
    display();

    return 0;
}
```

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

3. Problem Statement

John is working on a project to manage and analyze the data from various sensors in a manufacturing plant. Each sensor provides a sequence of integer readings, and John needs to process this data to get some insights. He wants to implement a queue to handle these sensor readings efficiently. The requirements are as follows:

Enqueue Operations: Each sensor reading needs to be added to the circular queue.Average Calculation: Calculate and print the average of every pair of consecutive sensor readings.Sum Calculation: Compute the sum of all sensor readings.Even and Odd Count: Count and print the number of even and odd sensor readings.

Assist John in implementing the program.

Input Format

The first input line contains an integer n, which represents the number of sensor readings.

The second line contains n space-separated integers, each representing a sensor reading.

Output Format

The first line should print "Averages of pairs:" followed by the averages of every pair of consecutive sensor readings, separated by spaces.

The second line should print "Sum of all elements: " followed by the sum of all sensor readings.

The third line should print "Number of even elements: " followed by the count of even sensor readings.

The fourth line should print "Number of odd elements: " followed by the count of odd sensor readings.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5
Output: Averages of pairs:
1.5 2.5 3.5 4.5 3.0
Sum of all elements: 15
Number of even elements: 2
Number of odd elements: 3

Answer

```
#include <stdio.h>

#define MAX_SIZE 10

typedef struct {
    int data[MAX_SIZE];
    int front, rear, size;
} Queue;

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

void enqueue(Queue *q, int value) {
    if (q->size == MAX_SIZE) return;
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->data[q->rear] = value;
    q->size++;
}

void processSensorData(Queue *q) {
    printf("Averages of pairs:\n");
    for (int i = 0; i < q->size - 1; i++)
        printf("%.1f ", (q->data[i] + q->data[i + 1]) / 2.0);
    printf("%.1f\n", (q->data[q->size - 1] + q->data[0]) / 2.0);

    int sum = 0, even_count = 0, odd_count = 0;
    for (int i = 0; i < q->size; i++) {
        sum += q->data[i];
```

```
        if (q->data[i] % 2 == 0) even_count++;
        else odd_count++;
    }

    printf("Sum of all elements: %d\n", sum);
    printf("Number of even elements: %d\n", even_count);
    printf("Number of odd elements: %d\n", odd_count);
}

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

    scanf("%d", &n);
    if (n < 1 || n > MAX_SIZE) return 1;

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

    processSensorData(&q);
    return 0;
}
```

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