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_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

240701215

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

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

// Define the structure for a queue node
typedef struct Node {
   int requestID;
   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 request
void enqueue(Queue* queue, int requestID) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->requestID = requestID;
```

240701215

```
    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 remove duplicates from the queue and print unique requests
void printUniqueRequests(Queue* queue) {
    Node* current = queue->front;
    int seen[101] = {0}; // Array to track seen requests (1-100)
    int first = 1; // To handle spacing correctly

    while (current != NULL) {
        if (!seen[current->requestID]) {
            seen[current->requestID] = 1; // Mark this request as seen
            if (!first) {
                printf(" ");
            }
            printf("%d", current->requestID);
            first = 0; // After the first print, set first to 0
        }
        current = current->next;
    }
    printf("\n");
}

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

    Queue* queue = createQueue();
    int requestID;

    // Read requests and enqueue them
```

240701215 240701215 240701215 240701215

```
    for (int i = 0; i < N; i++) {
        scanf("%d", &requestID);
        enqueue(queue, requestID);
    }

    // Print unique requests
    printUniqueRequests(queue);

    // Free the queue memory
    Node* current = queue->front;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        free(temp);
    }
    free(queue);

    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.

240701215

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

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

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

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

240701215

```
    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 task
void enqueue(Queue* queue, int taskID) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->taskID = taskID;
    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;
    }

    printf("Enqueued: %d\n", taskID);
}

// Function to remove erroneous tasks and print valid tasks
void printValidTasks(Queue* queue) {
    Node* current = queue->front;
    int first = 1; // To handle spacing correctly

    printf("Queue Elements after Dequeue: ");
    while (current != NULL) {
        if (current->taskID >= 0) {
            if (!first) {
                printf(" ");
            }
            printf("%d", current->taskID);
```

240701215

```
            first = 0; // After the first print, set first to 0
        }
        current = current->next;
    }
    printf("\n");
}

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

    Queue* queue = createQueue();
    int taskID;

    // Read tasks and enqueue them
    for (int i = 0; i < N; i++) {
        scanf("%d", &taskID);
        enqueue(queue, taskID);
    }

    // Print valid tasks after removing erroneous ones
    printValidTasks(queue);

    // Free the queue memory
    Node* current = queue->front;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        free(temp);
    }
    free(queue);

    return 0;
}
```

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

3. Problem Statement

240701215

Saran is developing a simulation for a theme park where people wait in a queue for a popular ride.

Each person has a unique ticket number, and he needs to manage the queue using a linked list implementation.

Your task is to write a program for Saran that reads the number of people in the queue and their respective ticket numbers, enqueue them, and then calculate the sum of all ticket numbers to determine the total ticket value present in the queue.

Input Format

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

The second line consists of N space-separated integers, representing the ticket numbers.

Output Format

The output prints an integer representing the sum of all ticket numbers.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
2 4 6 7 5

Output: 24

Answer

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

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

240701215

```
} 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 number
void enqueue(Queue* queue, int ticketNumber) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->ticketNumber = ticketNumber;
   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 calculate the sum of all ticket numbers in the queue
int calculateTotal(Queue* queue) {
   Node* current = queue->front;
   int total = 0;

   while (current != NULL) {
      total += current->ticketNumber;
      current = current->next;
   }
```
240701215

```
    return total;
}

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

    Queue* queue = createQueue();
    int ticketNumber;

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

    // Calculate and print the total ticket value
    int total = calculateTotal(queue);
    printf("%d\n", total);

    // Free the queue memory
    Node* current = queue->front;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        free(temp);
    }
    free(queue);

    return 0;
}
```

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

240701215