Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Imagine a bustling coffee shop, where customers are placing their orders for their favorite coffee drinks. The cafe owner Sheeren wants to efficiently manage the queue of coffee orders using a digital system. She needs a program to handle this queue of orders.

You are tasked with creating a program that implements a queue for coffee orders. Each character in the queue represents a customer's coffee order, with 'L' indicating a latte, 'E' indicating an espresso, 'M' indicating a macchiato, 'O' indicating an iced coffee, and 'N' indicating a nabob.

Customers can place orders and enjoy their delicious coffee drinks.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the coffee order into the queue. If the choice is 1, the following input is a space-separated character ('L', 'E', 'M', 'O', 'N').

Choice 2: Dequeue a coffee order from the queue.

Choice 3: Display the orders in the queue.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given order into the queue and display "Order for [order] is enqueued." where [order] is the coffee order that is inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue more orders."

If the choice is 2:

1. Dequeue a character from the queue and display "Dequeued Order: " followed by the corresponding order that is dequeued.
2. If the queue is empty without any orders, print "No orders in the queue."

If the choice is 3:

1. The output prints "Orders in the queue are: " followed by the space-separated orders present in the queue.
2. If there are no orders in the queue, print "Queue is empty. No orders available."

If the choice is 4:

1. Exit the program and print "Exiting program"

If any other choice is entered, the output prints "Invalid option."

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

Sample Test Case

Input: 1 L
1 E
1 M
1 O
1 N
1 O
3
2
3
4

Output: Order for L is enqueued.
Order for E is enqueued.
Order for M is enqueued.
Order for O is enqueued.
Order for N is enqueued.
Queue is full. Cannot enqueue more orders.
Orders in the queue are: L E M O N
Dequeued Order: L
Orders in the queue are: E M O N
Exiting program

Answer

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

#define MAX_SIZE 5
int isFull(int front, int rear) {
    return (rear + 1) % MAX_SIZE == front;
}


int isEmpty(int front, int rear) {
    return front == -1;
}
```

```
void enqueue(char queue[], int *rear, int *front, char order) {
    if (isFull(*front, *rear)) {
        printf("Queue is full. Cannot enqueue more orders.\n");
    } else {
        if (isEmpty(*front, *rear)) {
            *front = 0;
        }
        *rear = (*rear + 1) % MAX_SIZE;
        queue[*rear] = order;
        printf("Order for %c is enqueued.\n", order);
    }
}


char dequeue(char queue[], int *front, int *rear) {
    if (isEmpty(*front, *rear)) {
        printf("No orders in the queue.\n");
        return '\0';
    } else {
        char order = queue[*front];
        if (*front == *rear) {
            *front = -1;
            *rear = -1;
        } else {
            *front = (*front + 1) % MAX_SIZE;
        }
        return order;
    }
}


void display(char queue[], int front, int rear) {
    if (isEmpty(front, rear)) {
        printf("Queue is empty. No orders available.\n");
    } else {
        printf("Orders in the queue are: ");
        int i = front;
        do {
            printf("%c ", queue[i]);
            i = (i + 1) % MAX_SIZE;
        } while (i != (rear + 1) % MAX_SIZE);
```

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

int main() {
    char queue[MAX_SIZE];
    int front = -1, rear = -1;
    int choice;
    char order;

    do {
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                scanf(" %c", &order);
                enqueue(queue, &rear, &front, order);
                break;
            case 2:
                order = dequeue(queue, &front, &rear);
                if (order != '\0')
                    printf("Dequeued Order: %c\n", order);
                break;
            case 3:
                display(queue, front, rear);
                break;
            case 4:
                printf("Exiting program\n");
                break;
            default:
                printf("Invalid option.\n");
        }
    } while (choice != 4);

return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 0

Section 1 : Coding

1. Problem Statement

In a bustling IT department, staff regularly submit helpdesk tickets to request technical assistance. Managing these tickets efficiently is vital for providing quality support.

Your task is to develop a program that uses an array-based queue to handle and prioritize helpdesk tickets based on their unique IDs.

Implement a program that provides the following functionalities:

Enqueue Helpdesk Ticket: Add a new helpdesk ticket to the end of the queue. Provide a positive integer representing the ticket ID for the new ticket.Dequeue Helpdesk Ticket: Remove and process the next helpdesk ticket from the front of the queue. The program will display the ticket ID of the processed ticket.Display Queue: Display the ticket IDs of all the

helpdesk tickets currently in the queue.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the ticket ID into the queue. If the choice is 1, the following input is a space-separated integer, representing the ticket ID to be enqueued into the queue.

Choice 2: Dequeue a ticket from the queue.

Choice 3: Display the ticket IDs in the queue.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given ticket ID into the queue and display "Helpdesk Ticket ID [id] is enqueued." where [id] is the ticket ID that is inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue."

If the choice is 2:

1. Dequeue a ticket ID from the queue and display "Dequeued Helpdesk Ticket ID: " followed by the corresponding ID that is dequeued.
2. If the queue is empty without any elements, print "Queue is empty."

If the choice is 3:

1. The output prints "Helpdesk Ticket IDs in the queue are: " followed by the space-separated ticket IDs present in the queue.
2. If there are no elements in the queue, print "Queue is empty."

If the choice is 4:

1. Exit the program and print "Exiting the program"

If any other choice is entered, print "Invalid option."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1 101
1 202
1 203
1 204
1 205
1 206
3
2
3
4

Output: Helpdesk Ticket ID 101 is enqueued.
Helpdesk Ticket ID 202 is enqueued.
Helpdesk Ticket ID 203 is enqueued.
Helpdesk Ticket ID 204 is enqueued.
Helpdesk Ticket ID 205 is enqueued.
Queue is full. Cannot enqueue.
Helpdesk Ticket IDs in the queue are: 101 202 203 204 205
Dequeued Helpdesk Ticket ID: 101
Helpdesk Ticket IDs in the queue are: 202 203 204 205
Exiting the program

Answer

-

Status : Skipped ***Marks : 0/10***

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 0

Section 1 : Coding

1. Problem Statement

Write a program to implement a queue using an array and pointers. The program should provide the following functionalities:

Insert an element into the queue.Delete an element from the queue.Display the elements in the queue.

The queue has a maximum capacity of 5 elements. If the queue is full and an insertion is attempted, a "Queue is full" message should be displayed. If the queue is empty and a deletion is attempted, a "Queue is empty" message should be displayed.

Input Format

Each line contains an integer representing the chosen option from 1 to 3.

Option 1: Insert an element into the queue followed by an integer representing the element to be inserted, separated by a space.

Option 2: Delete an element from the queue.

Option 3: Display the elements in the queue.

Output Format

For option 1 (insertion):-

1. The program outputs: "<data> is inserted in the queue." if the data is successfully inserted.
2. "Queue is full." if the queue is already full and cannot accept more elements.

For option 2 (deletion):-

1. The program outputs: "Deleted number is: <data>" if an element is successfully deleted and returns the value of the deleted element.
2. "Queue is empty." if the queue is empty no elements can be deleted.

For option 3 (display):-

1. The program outputs: "Elements in the queue are: <element1> <element2> ... <elementN>" where <element1>, <element2>, ..., <elementN> represent the elements present in the queue.
2. "Queue is empty." if the queue is empty no elements can be displayed.

For invalid options, the program outputs: "Invalid option."

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 1 10

3
5
Output: 10 is inserted in the queue.
Elements in the queue are: 10
Invalid option.

Answer

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

#define MAX 5 // Maximum size of the queue

int queue[MAX];
int front = -1;
int rear = -1;

// Function to insert an element into the queue
void insert(int data) {
    if (rear == MAX - 1) {
        printf("Queue is full.\n");
    } else {
        if (front == -1)
            front = 0;
        queue[++rear] = data;
        printf("%d is inserted in the queue.\n", data);
    }
}

// Function to delete an element from the queue
int delete_element() {
    if (front == -1) {
        printf("Queue is empty.\n");
        return -1; // Return -1 to indicate an empty queue
    } else {
        int data = queue[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front++;
        }
```

```
        printf("Deleted number is: %d\n", data);
        return data;
    }
}

// Function to display the elements in the queue
void display() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Elements in the queue are: ");
        for (int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int choice, data;

    while (1) {
        printf("Enter your choice (1: Insert, 2: Delete, 3: Display): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert: ");
                scanf("%d", &data);
                insert(data);
                break;
            case 2:
                delete_element();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid option.\n");
        }
    }
```

```
return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In an office setting, a print job management system is used to efficiently handle and process print jobs. The system is implemented using a queue data structure with an array.

The program provides the following operations:

Enqueue Print Job: Add a print job with a specified number of pages to the end of the queue.Dequeue Print Job: Remove and process the next print job in the queue.Display Queue: Display the print jobs in the queue

The program should ensure that print jobs are processed in the order they are received.

Input Format

The input consists of integers corresponding to the operation that needs to be performed:

Choice 1: Enqueue the print job into the queue. If the choice is 1, the following input is a space-separated integer, representing the pages to be enqueued into the queue.

Choice 2: Dequeue a print job from the queue.

Choice 3: Display the print jobs in the queue.

Choice 4: Exit the program.

Output Format

The output displays messages according to the choice and the status of the queue:

If the choice is 1:

1. Insert the given page into the queue and display "Print job with [page] pages is enqueued." where [page] is the number of pages that are inserted.
2. If the queue is full, print "Queue is full. Cannot enqueue."

If the choice is 2:

1. Dequeue a page from the queue and display "Processing print job: [page] pages" where [page] is the corresponding page that is dequeued.
2. If the queue is empty without any elements, print "Queue is empty."

If the choice is 3:

1. The output prints "Print jobs in the queue: " followed by the space-separated pages present in the queue.
2. If there are no elements in the queue, print "Queue is empty."

If the choice is 4:

1. Exit the program and print "Exiting program"

If any other choice is entered, the output prints "Invalid option."

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 1
10
1
20
1
30
1
40
1
50
1
60
3
2
3
4

Output: Print job with 10 pages is enqueued.
Print job with 20 pages is enqueued.
Print job with 30 pages is enqueued.
Print job with 40 pages is enqueued.
Print job with 50 pages is enqueued.
Queue is full. Cannot enqueue.
Print jobs in the queue: 10 20 30 40 50
Processing print job: 10 pages
Print jobs in the queue: 20 30 40 50
Exiting program

Answer

```
void enqueue(int page)
{
   if ((rear + 1) % MAX_SIZE == front)
   {
      printf("Queue is full. Cannot enqueue.\n");
      return;
```

```
    }

    if (front == -1)
    {
        front = rear = 0;
    }
    else
    {
        rear = (rear + 1) % MAX_SIZE;
    }

    queue[rear] = page;
    printf("Print job with %d pages is enqueued.\n", page);
}

void dequeue()
{
    if (front == -1)
    {
        printf("Queue is empty.\n");
        return;
    }

    int page = queue[front];
    printf("Processing print job: %d pages\n", page);

    if (front == rear)
    {
        front = rear = -1;
    }
    else
    {
        front = (front + 1) % MAX_SIZE;
    }
}


void display()
{
    if (front == -1)
    {
```

```
        printf("Queue is empty.\n");
        return;
    }

    printf("Print jobs in the queue: ");
    int i = front;
    while (1)
    {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % MAX_SIZE;
    }
    printf("\n");
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are tasked with implementing basic operations on a queue data structure using a linked list.

You need to write a program that performs the following operations on a queue:

Enqueue Operation: Implement a function that inserts an integer element at the rear end of the queue.Print Front and Rear: Implement a function that prints the front and rear elements of the queue. Dequeue Operation: Implement a function that removes the front element from the queue.

Input Format

The first line of input consists of an integer N, representing the number of elements to be inserted into the queue.

The second line consists of N space-separated integers, representing the queue elements.

Output Format

The first line prints "Front: X, Rear: Y" where X is the front and Y is the rear elements of the queue.

The second line prints the message indicating that the dequeue operation (front element removed) is performed: "Performing Dequeue Operation:".

The last line prints "Front: M, Rear: N" where M is the front and N is the rear elements after the dequeue operation.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 5
12 56 87 23 45

Output: Front: 12, Rear: 45
Performing Dequeue Operation:
Front: 56, Rear: 45

Answer

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int d)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = d;
```

```
    newNode->next = NULL;
    if (rear == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}

void printFrontRear()
{
    if (front != NULL && rear != NULL)
    {
        printf("Front: %d, Rear: %d\n", front->data, rear->data);
    }
}

void dequeue()
{
    if (front != NULL)
    {
        struct Node* temp = front;
        front = front->next;
        if (front == NULL)
        {
            rear = NULL;
        }
        free(temp);
    }
}

int main() {
    int n, data;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        enqueue(data);
    }
    printFrontRear();
    printf("Performing Dequeue Operation:\n");
```

```
    dequeue();
    printFrontRear();
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_CY

Attempt : 1
Total Mark : 30
Marks Obtained : 20

Section 1 : Coding

1. 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>

// Structure to represent a node in the linked list
struct Node {
    int data;
    struct Node* next;
};

// Structure to represent the queue
struct Queue {
    struct Node* front;
```

```
    struct Node* rear;
};

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    if (queue == NULL) {
        printf("Memory allocation failed for queue.\n");
        exit(1); // Exit if memory allocation fails
    }
    queue->front = NULL;
    queue->rear = NULL;
    return queue;
}

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

// Function to enqueue an element into the queue
void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed for new node.\n");
        exit(1); // Exit if memory allocation fails
    }
    newNode->data = data;
    newNode->next = NULL;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    printf("Enqueued: %d\n", data);
}

// Function to dequeue an element from the queue
int dequeue(struct Queue* queue) {
```

```
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1; // Return a sentinel value to indicate error
    } else {
        struct Node* temp = queue->front;
        int data = temp->data;
        queue->front = queue->front->next;
        if (queue->front == NULL) {
            queue->rear = NULL; // Update rear if the last element is dequeued
        }
        free(temp);
        return data;
    }
}

// Function to display the elements in the queue after removing negative
numbers
void displayQueue(struct Queue* queue) {
    printf("Queue Elements after Dequeue: ");
    struct Queue* tempQueue = createQueue(); //use a temporary queue.
    while (!isEmpty(queue))
    {
        int data = dequeue(queue);
        if(data >= 0)
        {

            enqueue(tempQueue, data); //push the positive number into tempQueue
        }
    }

    //restore the original queue.
    while(!isEmpty(tempQueue))
    {
        enqueue(queue, dequeue(tempQueue));
    }
    free(tempQueue);

}

int main() {
    int n, task;
    struct Queue* queue = createQueue();
```

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

    displayQueue(queue);

    // Free the remaining nodes in the queue.  Important for memory cleanup.
    while (!isEmpty(queue)) {
        dequeue(queue);
    }
    free(queue); // Free the queue structure itself.

    return 0;
}
```

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

2. Problem Statement

Manoj is learning data structures and practising queues using linked lists. His professor gave him a problem to solve. Manoj started solving the program but could not finish it. So, he is seeking your assistance in solving it.

The problem is as follows: Implement a queue with a function to find the Kth element from the end of the queue.

Help Manoj with 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, representing the queue elements.

The third line consists of an integer K.

Output Format

The output prints an integer representing the Kth element from the end of the queue.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
2 4 6 7 5
3
Output: 6

Answer

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
struct Queue
{
   struct Node* front;
   struct Node* rear;
};
struct Queue* createQueue()
{
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   if (queue == NULL)
   {
      printf("Memory allocation failed for queue.\n");
      exit(1);
   }
   queue->front = NULL;
   queue->rear = NULL;
   return queue;
}
```

```
int isEmpty(struct Queue* queue)
{
    return (queue->front == NULL);
}
void enqueue(struct Queue* queue, int data)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL)
    {
        printf("Memory allocation failed for new node.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;

    if (isEmpty(queue))
    {
        queue->front = newNode;
        queue->rear = newNode;
    }
    else
    {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    else
    {
        struct Node* temp = queue->front;
        int data = temp->data;
        queue->front = queue->front->next;
        if (queue->front == NULL)
        {
            queue->rear = NULL;
        }
```

```
        free(temp);
        return data;
    }
}
int findKthFromEnd(struct Queue* queue, int k) {
    if (isEmpty(queue) || k <= 0) {
        return -1;
    }

    struct Node* current = queue->front;
    struct Node* kthNode = queue->front;
    int count = 0;
    while (current != NULL && count < k)
    {
        current = current->next;
        count++;
    }

    if (count < k || current == NULL)
    {
        return -1;
    }
    while (current != NULL)
    {
        current = current->next;
        kthNode = kthNode->next;
    }

    return kthNode->data;
}

int main()
{
    int n, k, element;
    struct Queue* queue = createQueue();

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

```
    scanf("%d", &k);

    int result = findKthFromEnd(queue, k);
    printf("%d\n", result);


    while (!isEmpty(queue))
    {
        dequeue(queue);
    }
    free(queue);

    return 0;
}
```

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

3. Problem Statement

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
struct Queue
{
   struct Node* front;
   struct Node* rear;
};
struct Queue* createQueue()
{
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   if (queue == NULL)
   {
      printf("Memory allocation failed for queue.\n");
      exit(1);
   }
   queue->front = NULL;
   queue->rear = NULL;
   return queue;
}
int isEmpty(struct Queue* queue)
{
   return (queue->front == NULL);
}
void enqueue(struct Queue* queue, int data)
```

```
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL)
    {
        printf("Memory allocation failed for new node.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;

    if (isEmpty(queue))
    {
        queue->front = newNode;
        queue->rear = newNode;
    }
    else
    {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
    {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    else
    {
        struct Node* temp = queue->front;
        int data = temp->data;
        queue->front = queue->front->next;
        if (queue->front == NULL)
        {
            queue->rear = NULL;
        }
        free(temp);
        return data;
    }
}
int calculateSum(struct Queue* queue)
```

```
{
    int sum = 0;
    struct Node* current = queue->front;
    while (current != NULL)
    {
        sum += current->data;
        current = current->next;
    }
    return sum;
}

int main()
{
    int n, ticketNumber;
    struct Queue* queue = createQueue();

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

    int totalSum = calculateSum(queue);
    printf("%d\n", totalSum);
    while (!isEmpty(queue))
    {
        dequeue(queue);
    }
    free(queue);

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Hruthika Lakshmi K
Email: 240801116@rajalakshmi.edu.in
Roll no: 240801116
Phone: 9445752530
Branch: REC
Department: I ECE AE
Batch: 2028
Degree: B.E - ECE

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node* next;
};
struct Node* front = NULL;
struct Node* rear = NULL;
void enqueue(int d)
```

```
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = d;
    newNode->next = NULL;
    if (front == NULL)
    {
        front = newNode;
        rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}
void dequeue()
{
    if (front == NULL)
    {
        return;
    }
    struct Node* temp = front;
    front = front->next;
    if (front == NULL)
    {
        rear = NULL;
    }
    free(temp);
}
void displayQueue()
{
    struct Node* temp = front;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
```

```
    int n, data;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &data);
        enqueue(data);
    }
    printf("Queue: ");
    displayQueue();
    dequeue();
    printf("Queue After Dequeue: ");
    displayQueue();
    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

```
#include <stdio.h>

#define MAX_SIZE 25

int queue[MAX_SIZE];
int front = 0, rear = -1, size = 0;
int isFull()
{
   return size == MAX_SIZE;
}
int isEmpty()
{
   return size == 0;
}
void enqueue(int customerID)
{
   if (isFull())
   {
      printf("Queue is full. Cannot add customer %d.\n", customerID);
      return;
   }
   rear = (rear + 1) % MAX_SIZE;
   queue[rear] = customerID;
   size++;
}
void dequeue()
{
   if (isEmpty())
```

```
    {
        printf("Underflow\n");
        return;
    }
    front = (front + 1) % MAX_SIZE;
    size--;
}
void displayQueue()
{
    if (isEmpty())
    {
        printf("Queue is empty\n");
        return;
    }
    int tempFront = front;
    for (int i = 0; i < size; i++)
    {
        printf("%d ", queue[tempFront]);
        tempFront = (tempFront + 1) % MAX_SIZE;
    }
    printf("\n");
}

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

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

```
#include <stdlib.h>

#define MAX_SIZE 10

int queue[MAX_SIZE];
int size = 0;
void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void heapify(int i, int n)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && queue[left] > queue[largest])
        largest = left;

    if (right < n && queue[right] > queue[largest])
        largest = right;

    if (largest != i)
    {
        swap(&queue[i], &queue[largest]);
        heapify(largest, n);
    }
}
void buildMaxHeap(int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(i, n);
}
void enqueue(int element)
{
    if (size < MAX_SIZE)
    {
        queue[size] = element;
        printf("Enqueued: %d\n", element);
        size++;
```

```
        buildMaxHeap(size);
    }
}
int findKthLargest(int k)
{
    buildMaxHeap(size);
    for (int i = 0; i < k - 1; i++) {
        queue[0] = queue[size - i - 1];
        heapify(0, size - i - 1);
    }
    return queue[0];
}

int main()
{
    int n, element, k;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &element);
        enqueue(element);
    }
    scanf("%d", &k);
    printf("The %dth largest element: %d\n", k, findKthLargest(k));
    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

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
   int data;
   struct Node* next;
} Node;
Node* front = NULL;
Node* rear = NULL;
void enqueue(int d)
{
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = d;
   newNode->next = NULL;
   if (front == NULL)
   {
      front = newNode;
      rear = newNode;
   }
   else
   {
```

```
        rear->next = newNode;
        rear = newNode;
    }
}
int dequeue()
{
    if (front == NULL)
    {
        return -1;
    }
    int data = front->data;
    Node* temp = front;
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    free(temp);
    return data;
}
void displayDequeuedElements()
{
    printf("Dequeued elements: ");
    while (front != NULL)
    {
        printf("%d ", dequeue());
    }
    printf("\n");
}

int main()
{
    int data;
    while (1)
    {
        scanf("%d", &data);
        if (data == -1)
        {
            break;
        }
        if (data >= 0)
        {
            enqueue(data);
```

```
    }
  }
  displayDequeuedElements();
  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

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

struct Node
{
   int data;
   struct Node* next;
};
struct Node* front = NULL;
struct Node* rear = NULL;
void enqueue(int d)
{
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = d;
    newNode->next = NULL;
    if (front == NULL)
    {
        front = newNode;
        rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
}
void displayQueue(char* message)
{
    printf("%s", message);
    struct Node* temp = front;
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
void selectiveDequeue(int multiple)
{
    struct Node* temp = front;
    struct Node* prev = NULL;
    while (temp != NULL)
    {
        struct Node* nextNode = temp->next;
        if (temp->data % multiple == 0)
        {
            if (prev == NULL)
            {
                front = nextNode;
            }
            else
            {
                prev->next = nextNode;
            }
```

```
            if (nextNode == NULL)
            {
                rear = prev;
            }
            free(temp);
        }
        else
        {
            prev = temp;
        }
        temp = nextNode;
    }
}

int main()
{
    int n, data, multiple;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &data);
        enqueue(data);
    }
    displayQueue("Original Queue: ");
    scanf("%d", &multiple);
    selectiveDequeue(multiple);
    displayQueue("Queue after selective dequeue: ");
    return 0;
}
```

Status : Correct *Marks : 10/10*