Rajalakshmi Engineering College

Name: Swethaa V A
Email: 241501258@rajalakshmi.edu.in
Roll no: 241501258
Phone: 7200023170
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 1

Attempt : 2
Total Mark : 10
Marks Obtained : 0

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>
#define MAX_SIZE 5

char orders[MAX_SIZE];
int front = -1;
int rear = -1;

void initializeQueue() {
    front = -1;
    rear = -1;
}
// You are using GCC
```

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

#define MAX_SIZE 100

int ticketIDs[MAX_SIZE];
int front = -1, rear = -1;

void initializeQueue() {
   front = -1;
   rear = -1;
}

typedef struct node {
   char data;
   struct node* next;
} node;

int len(node* head){
   node* t = head;
   if (head == NULL) {
      return 0;
   }
   int i = 0;
   while(t != NULL) {
      t = t->next;
      i++;
   }
   return i;
}

node* enq(node* head, char value){
   node* newnode = (node*)malloc(sizeof(node));
   newnode->data = value;
   newnode->next = NULL;
   int l = len(head);
   if (l == 5) {
      printf("Queue is full. Cannot enqueue more orders.\n");
      return head;
   }
   if (head == NULL) {
      printf("Order for %c is enqueued.\n", value);
```

```
        return newnode;
    }
    node* t = head;
    while(t->next != NULL) {
        t = t->next;
    }
    t->next = newnode;
    printf("Order for %c is enqueued.\n", value);
    return head;
}

node* deq(node* head){
    if (head == NULL) {
        printf("No orders in the queue.\n");
        return head;
    }
    node* t = head;
    head = head->next;
    printf("Dequeued Order: %c\n", t->data);
    free(t);
    return head;
}

void display(node* head){
    if (head == NULL) {
        printf("Queue is empty. No orders available.\n");
        return;
    }
    printf("Orders in the queue are: ");
    node* t = head;
    while(t != NULL) {
        printf("%c ", t->data);
        t = t->next;
    }
    printf("\n");
}

int isempty() {
    return front == -1 || front > rear;
}

int isfull() {
```

```
    return rear == MAX_SIZE - 1;
}

int enqueue(int ticketID) {
    if (isfull()) {
        printf("Queue is full. Cannot enqueue.\n");
        return 0;
    }
    if (isempty()) front = 0;
    rear++;
    ticketIDs[rear] = ticketID;
    printf("Helpdesk Ticket ID %d is enqueued.\n", ticketID);
    return 1;
}

int dequeue(int* ticketID) {
    if (isempty()) return 0;
    *ticketID = ticketIDs[front];
    front++;
    if (front > rear) {
        initializeQueue();
    }
    return 1;
}

int main() {
    node* head = NULL;
    while(1) {
        int c;
        scanf("%d", &c);
        if (c == 1) {
            char t;
            scanf(" %c", &t);
            head = enq(head, t);
        }
        else if (c == 2) {
            head = deq(head);
        }
        else if (c == 3) {
            display(head);
        }
        else if (c == 4) {
```

```
            printf("Exiting program\n");
            break;
        }
        else {
            printf("Invalid option.\n");
        }
    }
}

int main() {
    char order;
    int option;
    initializeQueue();
    while (1) {
        if (scanf("%d", &option) != 1) {
            break;
        }
        switch (option) {
            case 1:
                if (scanf(" %c", &order) != 1) {
                    break;
                }
                if (enqueue(order)) {
                }
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program");
                return 0;
            default:
                printf("Invalid option.\n");
                break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Swethaa V A
Email: 241501258@rajalakshmi.edu.in
Roll no: 241501258
Phone: 7200023170
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

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 : 10

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

```
#include <stdio.h>
#define MAX_SIZE 5

int ticketIDs[MAX_SIZE];
int front = -1;
int rear = -1;
int lastDequeued;

void initializeQueue() {
    front = -1;
    rear = -1;
}
```

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

int isFull() {
    return rear == MAX_SIZE - 1;
}

int enqueue(int ticketID) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue.\n");
        return 0;
    }
    if (front == -1) front = 0;
    rear++;
    ticketIDs[rear] = ticketID;
    printf("Helpdesk Ticket ID %d is enqueued. \n", ticketID);
    return 1;
}

int dequeue() {
    if (isEmpty()) {
        return 0;
    }
    lastDequeued = ticketIDs[front];
    front++;
    if (front > rear) {
        front = rear = -1;
    }
    return 1;
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Helpdesk Ticket IDs in the queue are: ");
    for (int i = front; i <= rear; i++)
        printf("%d ", ticketIDs[i]);
    printf("\n");
}
```

```
int main() {
    int ticketID;
    int option;
    initializeQueue();
    while (1) {
        if (scanf("%d", &option) == EOF) {
            break;
        }
        switch (option) {
            case 1:
                if (scanf("%d", &ticketID) == EOF) {
                    break;
                }
                enqueue(ticketID);
                break;
            case 2:
                if (dequeue()) {
                    printf("Dequeued Helpdesk Ticket ID: %d\n", lastDequeued);
                } else {
                    printf("Queue is empty.\n");
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting the program\n");
                return 0;
            default:
                printf("Invalid option.\n");
                break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Swethaa V A
Email: 241501258@rajalakshmi.edu.in
Roll no: 241501258
Phone: 7200023170
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

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 : 10

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

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

#define max 5

int queue[max];
int front = -1, rear = -1;

int insertq(int *data) {
    if (rear == max - 1) {
        return 0;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = *data;
    return 1;
}

int delq() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
        return 0;
    }
    printf("Deleted number is: %d\n", queue[front]);
    front++;
    if (front > rear) {
        front = rear = -1;
    }
    return 1;
}

void display() {
    if (front == -1 || front > rear) {
```

```
        printf("Queue is empty.\n");
        return;
    }
    printf("Elements in the queue are: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main()
{
    int data, reply, option;
    while (1)
    {
        if (scanf("%d", &option) != 1)
            break;
        switch (option)
        {
            case 1:
                if (scanf("%d", &data) != 1)
                    break;
                reply = insertq(&data);
                if (reply == 0)
                    printf("Queue is full.\n");
                else
                    printf("%d is inserted in the queue.\n", data);
                break;
            case 2:
                delq();  //    Called without arguments
                break;
            case 3:
                display();
                break;
            default:
                printf("Invalid option.\n");
                break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Swethaa V A
Email: 241501258@rajalakshmi.edu.in
Roll no: 241501258
Phone: 7200023170
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

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 pages) {
    if (rear == MAX_SIZE - 1) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
  if (front == -1) front = 0;
```

```
    rear++;
    queue[rear] = pages;
    printf("Print job with %d pages is enqueued. \n", pages);
}
    void dequeue() {
        if (front == -1 || front > rear) {
            printf("Queue is empty.\n");
            return;

    }printf("Processing print job: %d pages\n", queue[front]);
    front++;
    if (front > rear) {
        front = -1;
        rear=-1;
    }
    }
    void display(){
        if (front == -1 || front > rear) {
            printf("Queue is empty.\n");
            return;
        }
            printf("Print jobs in the queue: ");
            for (int i = front; i <= rear; i++) {
           printf("%d ", queue[i]);
            }

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

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

Rajalakshmi Engineering College

Name: Swethaa V A
Email: 241501258@rajalakshmi.edu.in
Roll no: 241501258
Phone: 7200023170
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 5

Attempt : 2
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;

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

```
// Node structure for the linked list
struct Node {
    int data;
    struct Node* next;
};

// Queue structure with front and rear pointers
struct Queue {
    struct Node* front;
    struct Node* rear;
};

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

// Enqueue operation
void enqueue(struct Queue* q, int value) {
    struct Node* temp = (struct Node*) malloc(sizeof(struct Node));
    temp->data = value;
    temp->next = NULL;

    if (q->rear == NULL) {
        q->front = q->rear = temp;
        return;
    }

    q->rear->next = temp;
    q->rear = temp;
}

// Dequeue operation
void dequeue(struct Queue* q) {
    if (q->front == NULL) {
        return;
    }

    struct Node* temp = q->front;
    q->front = q->front->next;
```

```
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
}

// Function to print front and rear
void printFrontAndRear(struct Queue* q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Front: %d, Rear: %d\n", q->front->data, q->rear->data);
}

int main() {
    int N, value;
    scanf("%d", &N);

    struct Queue* queue = createQueue();

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

    // Print front and rear before dequeue
    printFrontAndRear(queue);

    // Perform dequeue
    printf("Performing Dequeue Operation:\n");
    dequeue(queue);

    // Print front and rear after dequeue
    printFrontAndRear(queue);

    return 0;
}
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***