Rajalakshmi Engineering College

Name: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 20

Section 1 : MCQ

1. What will be the output of the following code?

```
#include <stdio.h>
#define MAX_SIZE 5
typedef struct {
   int arr[MAX_SIZE];
   int front;
   int rear;
   int size;
} Queue;

void enqueue(Queue* queue, int data) {
   if (queue->size == MAX_SIZE) {
      return;
   }
   queue->rear = (queue->rear + 1) % MAX_SIZE;
```

```
    queue->arr[queue->rear] = data;
    queue->size++;
}
int dequeue(Queue* queue) {
    if (queue->size == 0) {
        return -1;
    }
    int data = queue->arr[queue->front];
    queue->front = (queue->front + 1) % MAX_SIZE;
    queue->size--;
    return data;
}
int main() {
    Queue queue;
    queue.front = 0;
    queue.rear = -1;
    queue.size = 0;
    enqueue(&queue, 1);
    enqueue(&queue, 2);
    enqueue(&queue, 3);
    printf("%d ", dequeue(&queue));
    printf("%d ", dequeue(&queue));
    enqueue(&queue, 4);
    enqueue(&queue, 5);
    printf("%d ", dequeue(&queue));
    printf("%d ", dequeue(&queue));
    return 0;
}
```

Answer

1 2 3 4

Status : Correct ***Marks : 1/1***

2. Which operations are performed when deleting an element from an array-based queue?

Answer

Dequeue

Status : Correct ***Marks : 1/1***

3. The essential condition that is checked before insertion in a queue is?

Answer

Overflow

Status : Correct ***Marks : 1/1***

4. Front and rear pointers are tracked in the linked list implementation of a queue. Which of these pointers will change during an insertion into the EMPTY queue?

Answer

Both front and rear pointer

Status : Correct ***Marks : 1/1***

5. What does the front pointer in a linked list implementation of a queue contain?

Answer

The address of the first element

Status : Correct ***Marks : 1/1***

6. In a linked list implementation of a queue, front and rear pointers are tracked. Which of these pointers will change during an insertion into a non-empty queue?

Answer

Only rear pointer

Status : Correct ***Marks : 1/1***

7. Which of the following properties is associated with a queue?

Answer

First In First Out

Status : Correct ***Marks : 1/1***

8. What are the applications of dequeue?

Answer

All the mentioned options

Status : Correct ***Marks : 1/1***

9. The process of accessing data stored in a serial access memory is similar to manipulating data on a

Answer

Queue

Status : Correct ***Marks : 1/1***

10. In what order will they be removed If the elements “A”, “B”, “C” and “D” are placed in a queue and are deleted one at a time

Answer

ABCD

Status : Correct ***Marks : 1/1***

11. In linked list implementation of a queue, the important condition for a queue to be empty is?

Answer

FRONT is null

Status : Correct ***Marks : 1/1***

12. What will be the output of the following code?

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5
typedef struct {
   int* arr;
   int front;
   int rear;
   int size;
} Queue;
Queue* createQueue() {
   Queue* queue = (Queue*)malloc(sizeof(Queue));
   queue->arr = (int*)malloc(MAX_SIZE * sizeof(int));
   queue->front = -1;
   queue->rear = -1;
   queue->size = 0;
   return queue;
}
int isEmpty(Queue* queue) {
   return (queue->size == 0);
}
int main() {
   Queue* queue = createQueue();
   printf("Is the queue empty? %d", isEmpty(queue));
   return 0;
}
```

Answer

Is the queue empty? 1

Status : Correct ***Marks : 1/1***

13. Which one of the following is an application of Queue Data Structure?

Answer

All of the mentioned options

Status : Correct ***Marks : 1/1***

14. Which of the following can be used to delete an element from the front end of the queue?

Answer

public Object deleteFront() throws emptyDEQException{if(isEmpty())throw new emptyDEQException("Empty");else{Node temp = head.getNext();Node cur = temp.getNext();Object e = temp.getEle();head.setNext(cur);size--;return e;}}

Status : Correct ***Marks : 1/1***

15. After performing this set of operations, what does the final list look to contain?

InsertFront(10);
InsertFront(20);
InsertRear(30);
DeleteFront();
InsertRear(40);
InsertRear(10);
DeleteRear();
InsertRear(15);
display();

Answer

10 30 40 15

Status : Correct ***Marks : 1/1***

16. When new data has to be inserted into a stack or queue, but there is no available space. This is known as

Answer

overflow

Status : Correct ***Marks : 1/1***

17. What is the functionality of the following piece of code?

```
public void function(Object item)
{
   Node temp=new Node(item,trail);
   if(isEmpty())
   {
      head.setNext(temp);
      temp.setNext(trail);
   }
   else
   {
      Node cur=head.getNext();
      while(cur.getNext()!=trail)
      {
         cur=cur.getNext();
      }
      cur.setNext(temp);
   }
   size++;
}
```

Answer

Insert at the rear end of the dequeue

Status : Correct ***Marks : 1/1***

18. Insertion and deletion operation in the queue is known as

Answer

Enqueue and Dequeue

Status : Correct ***Marks : 1/1***

19. A normal queue, if implemented using an array of size MAX_SIZE, gets full when

Answer

Rear = MAX_SIZE – 1

Status : Correct ***Marks : 1/1***

20. What will the output of the following code?

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
   int* arr;
   int front;
   int rear;
   int size;
} Queue;
Queue* createQueue() {
   Queue* queue = (Queue*)malloc(sizeof(Queue));
   queue->arr = (int*)malloc(5 * sizeof(int));
   queue->front = 0;
   queue->rear = -1;
   queue->size = 0;
   return queue;
}
int main() {
   Queue* queue = createQueue();
   printf("%d", queue->size);
   return 0;
}
```

Answer

0

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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* new_n = (struct Node*)malloc(sizeof(struct Node));
   new_n->data = d;
   new_n->next = NULL;
```

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

void printFrontRear() {
    printf("Front: %d, ", front->data);
    printf("Rear: %d\n", rear->data);

}

void dequeue() {
    struct Node* temp;
    temp = front;
    front = front->next;
    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: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

```
#include <iostream>
#define MAX_SIZE 5
using namespace std;

int pages[MAX_SIZE];
int front = -1;
int rear = -1;
void initializeQueue() {
   front = -1;
   rear = -1;
}

bool isEmpty() {
   return front == -1;
}

bool isFull() {
   return (rear + 1) % MAX_SIZE == front;
}

void enqueue(int p) {
   if (isFull()) {
      cout << "Queue is full. Cannot enqueue." << endl;
      return;
   }

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

   pages[rear] = p;
   cout << "Print job with " << p << " pages is enqueued." << endl;
}

bool dequeue(int& p) {
   if (isEmpty()) {
      return false;
   }

   p = pages[front];

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

```
    }
    return true;
}

void display() {
    if (isEmpty()) {
        cout << "Queue is empty." << endl;
    } else {
        cout << "Print jobs in the queue: ";
        int i = front;
        while (i != rear) {
            cout << pages[i] << " ";
            i = (i + 1) % MAX_SIZE;
        }
        cout << pages[rear] << " " << endl;
    }
}
int main() {
    int option;
    int p;
    initializeQueue();
    while (true) {
        if (!(cin >> option)) {
            break;
        }
        switch (option) {
            case 1:
                if (!(cin >> p)) {
                    break;
                }
                enqueue(p);
                break;
            case 2:
                if (dequeue(p)) {
                    cout << "Processing print job: " << p << " pages" << endl;
                } else {
                    cout << "Queue is empty." << endl;
                }
                break;
            case 3:
                display();
                break;
            case 4:
                cout << "Exiting program";
                return 0;
            default:
                cout << "Invalid option." << endl;
                break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

int delq(int queue[max], int *front, int *rear, int *data)
{
   if (*front == *rear)
   {
      *data = -1;
      return 0;
   }
   else
   {
      (*front)++;
      *data = queue[*front];
      if (*front > *rear)
      {
         *front = -1;
         *rear = -1;
      }
      return 1;
   }
}
```

```
void display(int queue[max], int front, int rear)
{
   if (front == rear)
   {
      printf("Queue is empty.\n");
   }
   else
   {
      printf("Elements in the queue are: ");
      for (int i = front + 1; i <= rear; i++)
      {
         printf("%d ", queue[i]);
      }
      printf("\n");
   }
}
int main()
{
   int queue[max], data;
   int front, rear, reply, option;
   front = rear = -1;
   while (1)
   {
      if (scanf("%d", &option) != 1)
         break;
      switch (option)
      {
         case 1:
            if (scanf("%d", &data) != 1)
               break;
            reply = insertq(queue, &rear, &data);
            if (reply == 0)
               printf("Queue is full.\n");
            else
               printf("%d is inserted in the queue.\n", data);
            break;
         case 2:
            reply = delq(queue, &front, &rear, &data);
            if (reply == 0)
            {
               printf("Queue is empty.\n");
            }
            else
            {
                printf("Deleted number is: %d\n", data);
            }
            break;
        case 3:
            display(queue, front, rear);
            break;
        default:
            printf("Invalid option.\n");
            break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Daniel Joshua C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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;

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

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

int isFull() {
    return (rear + 1) % MAX_SIZE == front;
}

int enqueue(int ticketID) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue.\n");
        return 0;
    }

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

    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];

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

    return 1;
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
    } else {
        printf("Helpdesk Ticket IDs in the queue are: ");
        int i = front;
        while (i != rear) {
            printf("%d ", ticketIDs[i]);
            i = (i + 1) % MAX_SIZE;
        }
        printf("%d\n", ticketIDs[rear]);
    }
}

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;
                }
                if (enqueue(ticketID)) {
                }
                break;
            case 2:
                if (dequeue(&ticketID)) {
                    printf("Dequeued Helpdesk Ticket ID: %d\n", ticketID);
                } 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: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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>
#define MAX_SIZE 5

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

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

```
}

int isFull() {
   return (rear + 1) % MAX_SIZE == front;
}

int enqueue(char order) {
   if (isFull()) {
      printf("Queue is full. Cannot enqueue more orders.\n");
      return 0;
   }

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

   orders[rear] = order;
   printf("Order for %c is enqueued.\n", order);
   return 1;
}

int dequeue(char* order) {
   if (isEmpty()) {
      return 0;
   }

   *order = orders[front];

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

   return 1;
}

void display() {
   if (isEmpty()) {
      printf("Queue is empty. No orders available.\n");
```

```
    } else {
        printf("Orders in the queue are: ");
        int i = front;
        while (i != rear) {
            printf("%c ", orders[i]);
            i = (i + 1) % MAX_SIZE;
        }
        printf("%c\n", orders[rear]);
    }
}
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:
                if (dequeue(&order)) {
                    printf("Dequeued Order: %c\n", order);
                } else {
                    printf("No orders in the queue.\n");
                }
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program");
                return 0;
            default:
                printf("Invalid option.\n");
            break;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

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();  //    Called without arguments

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

    return 0;
}
```

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

2. 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>
#include <stdlib.h>

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

Node* rear = NULL;  // Global rear pointer

Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void enqueue(int data) {
   Node* newNode = createNode(data);
   if (rear == NULL) {
      rear = newNode;
      newNode->next = newNode;
   } else {
      newNode->next = rear->next;
      rear->next = newNode;
      rear = newNode;
   }
}

int dequeue() {
   if (rear == NULL) {
        printf("Queue is empty\n");
        return -1;
    }

    Node* front = rear->next;
    int data = front->data;

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

    return data;
}

void printAverages(int size) {
    if (rear == NULL || size < 2) return;

    Node* temp = rear->next;
    for (int i = 0; i < size; i++) {
        int first = temp->data;
        temp = temp->next;
        int second = temp->data;
        printf("%.1f ", (first + second) / 2.0);
    }
    printf("\n");
}

void printSum() {
    if (rear == NULL) return;
    Node* temp = rear->next;
    int sum = 0;
    do {
        sum += temp->data;
        temp = temp->next;
    } while (temp != rear->next);
    printf("Sum of all elements: %d\n", sum);
}
```

```
void printEvenOddCount() {
   if (rear == NULL) return;
   Node* temp = rear->next;
   int evenCount = 0, oddCount = 0;
   do {
      if (temp->data % 2 == 0) evenCount++;
      else oddCount++;
      temp = temp->next;
   } while (temp != rear->next);
   printf("Number of even elements: %d\n", evenCount);
   printf("Number of odd elements: %d\n", oddCount);
}

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

   printf("Averages of pairs:\n");
   printAverages(n);
   printSum();
   printEvenOddCount();

   return 0;
}
```

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

3. Problem Statement

Pathirana is a medical lab specialist who is responsible for managing blood count data for a group of patients. The lab uses a queue-based system to track the blood cell count of each patient. The queue structure helps in processing the data in a first-in-first-out (FIFO) manner.

However, Pathirana needs to remove the blood cell count that is positive even numbers from the queue using array implementation of queue, as they are not relevant to the specific analysis he is performing. The remaining data will then be used for further medical evaluations and reporting.

Input Format

The first line consists of an integer n, representing the number of a patient's blood cell count.

The second line consists of n space-separated integers, representing a blood cell count value.

Output Format

The output displays space-separated integers, representing the remaining blood cell count after removing the positive even numbers.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
1 2 3 4 5

Output: 1 3 5

Answer

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

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

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

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

struct Queue* createQueue() {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = queue->rear = NULL;
   return queue;
}

void enqueue(struct Queue* queue, int data) {
   struct Node* newNode = createNode(data);
   if (queue->rear == NULL) {
       queue->front = queue->rear = newNode;
       return;
   }
   queue->rear->next = newNode;
   queue->rear = newNode;
}

int dequeue(struct Queue* queue) {
   if (queue->front == NULL) {
       return -1;
   }
   int data = queue->front->data;
   struct Node* temp = queue->front;
   queue->front = queue->front->next;
   free(temp);
   if (queue->front == NULL) {
       queue->rear = NULL;
   }
   return data;
}

void dequeueEvenPositive(struct Queue* queue) {
   struct Node* current = queue->front;
   struct Node* prev = NULL;

   while (current != NULL) {
       if (current->data % 2 == 0 && current->data > 0) {
           if (current == queue->front) {
               queue->front = current->next;
               free(current);
               current = queue->front;
               if (queue->front == NULL) {
                   queue->rear = NULL;
               }
           } else {
               prev->next = current->next;
               if (current == queue->rear) {
                   queue->rear = prev;
               }
               free(current);
               current = prev->next;
           }
       } else {
           prev = current;
           current = current->next;
       }
   }
}

int main() {
   int capacity, num, i;
   scanf("%d", &capacity);

   struct Queue* queue = createQueue();

   for (i = 0; i < capacity; ++i) {
       scanf("%d", &num);
       enqueue(queue, num);
   }

   dequeueEvenPositive(queue);

   while (queue->front != NULL) {
       printf("%d ", dequeue(queue));
   }
   printf("\n");

   free(queue);
   return 0;
```

```
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Daniel Joshua   C S
Email: 241901020@rajalakshmi.edu.in
Roll no: 241901020
Phone: 7358493188
Branch: REC
Department: I CSE (CS) FA
Batch: 2028
Degree: B.E - CSE (CS)

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

```
#include <stdio.h>
 #define MAX 25

int queue[MAX];
int rear = - 1;
int front = - 1;
void Enqueue(int data) {
   if (rear == MAX - 1)
      return;
   else {
      if (front == - 1)
         front = 0;
      rear = rear + 1;
      queue[rear] = data;
   }
}

void Dequeue() {
   if (front == - 1 || front > rear) {
      printf("Underflow\n");
      return ;
   }
```

```
    else {
        front = front + 1;
    }
}

void display() {
    int i;
    if (front == - 1)
        printf("Queue is empty\n");
    else    {
        for (i = front; i <= rear; i++)
            printf("%d ", queue[i]);
    }
}
int main () {
    int n,i,e;
    scanf("%d",&n);
    for(i=0;i<n;i++) {
        scanf("%d",&e);
        Enqueue(e);
    }
    Dequeue();
    display();
}
```

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

2. 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;

// Function to insert a node in the queue
void enqueue(int d) {
    struct Node* new_n = (struct Node*)malloc(sizeof(struct Node));
    new_n->data = d;
    new_n->next = NULL;
    if (front == NULL && rear == NULL) {
        front = rear = new_n;
    } else {
        rear->next = new_n;
        rear = new_n;
    }
}

// Function to display the queue
void display() {
    struct Node* temp = front;
    while (temp) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to delete an element from the queue
void dequeue() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node* temp = front;
    front = front->next;
    free(temp);
    if (front == NULL) {
        rear = NULL;
    }
}

int main() {
    int a, data;
    scanf("%d", &a);
    for (int i = 0; i < a; i++) {
        scanf("%d", &data);
        enqueue(data);
    }

    printf("Queue: ");
    display();

    printf("Queue After Dequeue: ");
    dequeue();
    display();

    return 0;
}
```

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

3. 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>

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

// Define the queue structure
struct Queue {
   struct Node* front;
   struct Node* rear;
};

// Declare the queue as a global variable
struct Queue myQueue;

// Initialize an empty queue
void initializeQueue() {
   myQueue.front = NULL;
   myQueue.rear = NULL;
}

// Enqueue (add to the back) operation
void enqueue(int d) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = d;
   newNode->next = NULL;

   if (myQueue.rear == NULL) {
      myQueue.front = newNode;
```

```
        myQueue.rear = newNode;
    } else {
        myQueue.rear->next = newNode;
        myQueue.rear = newNode;
    }
}

// Dequeue (remove from the front) operation without arguments
int dequeue() {
    if (myQueue.front == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }

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

    // If the queue becomes empty after dequeue, update the rear pointer
    if (myQueue.front == NULL) {
        myQueue.rear = NULL;
    }

    free(temp);
    return data;
}

// Display the elements of the queue
void display() {
    struct Node* current = myQueue.front;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    initializeQueue();

    int d;
    do {
        scanf("%d", &d);
        if (d > 0) {
            enqueue(d);
        }
    } while (d > -1);

    // Dequeue and display elements
    printf("Dequeued elements: ");
    while (myQueue.front != NULL) {
        int element = dequeue();
        printf("%d ", element);
    }
    printf("\n");

    return 0;
}
```

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

4. 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 Queue {
   struct Node* front;
   struct Node* rear;
};

// Declare global variables for the queue and 'multiple'
struct Queue* queue;
int multiple;

// Create a new queue
struct Queue* createQueue() {
   struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
   q->front = NULL;
   q->rear = NULL;
   return q;
}

// Enqueue operation (add to the rear of the queue)
void enqueue(struct Queue* q, int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;

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

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

// Selectively dequeue based on the global 'multiple' variable
void selectiveDequeue() {
   // Remove elements at the front if they are divisible by 'multiple'
   while (queue->front != NULL && (queue->front->data % multiple == 0)) {
        struct Node* temp = queue->front;
        queue->front = queue->front->next;
        free(temp);
    }

    struct Node* current = queue->front;
    struct Node* previous = NULL;

    // Traverse and remove all nodes divisible by 'multiple'
    while (current != NULL) {
        if (current->data % multiple == 0) {
            previous->next = current->next;
            free(current);
            current = previous->next;
        } else {
            previous = current;
            current = current->next;
        }
    }
}

// Display the elements of the queue
void displayQueue() {
    struct Node* current = queue->front;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    queue = createQueue();  // Initialize the global queue
    int n, value;

    scanf("%d", &n);

    // Enqueue elements into the queue
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        enqueue(queue, value);
    }
    // Get the 'multiple' value
    scanf("%d", &multiple);

    printf("Original Queue: ");
    displayQueue();

    // Call selectiveDequeue without arguments
    selectiveDequeue();

    printf("Queue after selective dequeue: ");
    displayQueue();

    return 0;
}
```

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

5. 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>

typedef struct {
   int* arr;
   int capacity;
   int front;
   int rear;
   int size;
} Queue;

Queue* createQueue(int cap) {
   Queue* queue = (Queue*)malloc(sizeof(Queue));
   queue->capacity = cap;
   queue->arr = (int*)malloc(cap * sizeof(int));
   queue->front = 0;
   queue->rear = -1;
   queue->size = 0;
   return queue;
}

int isFull(Queue* queue) {
   return queue->size == queue->capacity;
}
```

```
int isEmpty(Queue* queue) {
    return queue->size == 0;
}

void enqueue(Queue* queue, int data) {
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->arr[queue->rear] = data;
    queue->size++;
    printf("Enqueued: %d\n", data);
}

int compare(const void* a, const void* b) {
    return (*(int*)b - *(int*)a);
}

int findKthLargest(Queue* queue, int k) {
    int* tempArr = (int*)malloc(queue->size * sizeof(int));
    int count = queue->size;
    int idx = queue->front;

    for (int i = 0; i < queue->size; ++i) {
        tempArr[i] = queue->arr[idx];
        idx = (idx + 1) % queue->capacity;
    }

    qsort(tempArr, queue->size, sizeof(int), compare);

    int kthLargest = tempArr[k - 1];
    free(tempArr);
    return kthLargest;
}

int main() {
    int capacity = 10, n, k, value;
    Queue* q = createQueue(capacity);

    scanf("%d", &n);

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

    scanf("%d", &k);

    int kthLargest = findKthLargest(q, k);
    if (kthLargest != -1) {
        printf("The %dth largest element: %d", k, kthLargest);
    }

    free(q->arr);
    free(q);
    return 0;
}
```

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