Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 : 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

-

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

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 : 2
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;
}

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() {
    if (isEmpty()) {
        return 0;
    }

    lastDequeued = 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;
                }
                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: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 : 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

-

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

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 5

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

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

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

void enqueue(int page) {
    if (isFull()) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty()) {
        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 (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }
    int page = queue[front];
    if (front == rear) {
        front = rear = -1; // Queue becomes empty
    } else {
        front = (front + 1) % MAX_SIZE;
    }
    printf("Processing print job: %d pages\n", page);
}

void display() {
    if (isEmpty()) {
        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;
        printf(" ");
        i = (i + 1) % MAX_SIZE;
    }
    printf("\n");
}

int main() {
    int choice, pages;
    while (1) {
        if (scanf("%d", &choice) != 1) break;

        switch (choice) {
            case 1:
                if (scanf("%d", &pages) == 1)
                    enqueue(pages);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program\n");
                return 0;
            default:
                printf("Invalid option.\n");
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 : 0

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

-

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

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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>

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

// Function to create a new node
Node* createNode(int task) {
   Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->task = task;
    newNode->next = NULL;
    return newNode;
}

// Function to enqueue a task
void enqueue(Node** front, Node** rear, int task) {
    Node* newNode = createNode(task);
    if (*rear == NULL) {
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
    printf("Enqueued: %d\n", task);
}

// Function to dequeue and remove erroneous tasks (negative integers)
void dequeueErroneousTasks(Node** front, Node** rear) {
    Node* temp = *front;
    Node* prev = NULL;

    while (temp != NULL) {
        if (temp->task < 0) {
            // Remove the erroneous task (negative)
            if (prev == NULL) {
                // The first node is erroneous
                *front = temp->next;
            } else {
                prev->next = temp->next;
            }

            if (temp == *rear) {
                *rear = prev;  // Adjust the rear pointer if needed
            }
            free(temp);
            temp = (prev == NULL) ? *front : prev->next;  // Move to next node
        } else {
            prev = temp;
            temp = temp->next;
        }
    }
```

```
}

// Function to display the queue
void displayQueue(Node* front) {
    if (front == NULL) {
        printf("Queue is empty.\n");
    } else {
        Node* temp = front;
        printf("Queue Elements after Dequeue: ");
        while (temp != NULL) {
            printf("%d ", temp->task);
            temp = temp->next;
        }
        printf("\n");
    }
}

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

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

    // Enqueue the tasks
    for (int i = 0; i < N; i++) {
        int task;
        scanf("%d", &task);
        enqueue(&front, &rear, task);
    }

    // Remove erroneous tasks
    dequeueErroneousTasks(&front, &rear);

    // Display the remaining tasks
    displayQueue(front);

    return 0;
}
```

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

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>
```

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

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to enqueue an element to the queue
void enqueue(Node** front, Node** rear, int data) {
   Node* newNode = createNode(data);
   if (*rear == NULL) {
      *front = *rear = newNode;
      return;
   }
   (*rear)->next = newNode;
   *rear = newNode;
}

// Function to find the Kth element from the end
int findKthFromEnd(Node* front, int K) {
   Node* first = front;
   Node* second = front;

   // Move first pointer K nodes ahead
   for (int i = 0; i < K; i++) {
      if (first == NULL) {
         return -1; // Invalid K, but according to the problem K is always valid.
      }
      first = first->next;
   }

   // Move both pointers one step at a time
   while (first != NULL) {
      first = first->next;
      second = second->next;
```

```
    }

    return second->data;  // second is now K nodes from the end
}

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

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

    // Read elements into the queue
    for (int i = 0; i < N; i++) {
        int data;
        scanf("%d", &data);
        enqueue(&front, &rear, data);
    }

    // Read K
    scanf("%d", &K);

    // Find the Kth element from the end
    int result = findKthFromEnd(front, K);

    // Output the result
    printf("%d\n", result);

    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>

// Define a structure for a node in the linked list
typedef struct Node {
   int ticket_number;
   struct Node* next;
} Node;

// Function to create a new node with the given ticket number
Node* createNode(int ticket_number) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->ticket_number = ticket_number;
   newNode->next = NULL;
```

```
    return newNode;
}

// Function to enqueue a ticket number into the queue
void enqueue(Node** front, Node** rear, int ticket_number) {
    Node* newNode = createNode(ticket_number);
    if (*rear == NULL) {
        *front = *rear = newNode;  // Queue is empty
        return;
    }
    (*rear)->next = newNode;  // Add to the end of the queue
    *rear = newNode;
}

// Function to calculate the sum of ticket numbers in the queue
int calculateSum(Node* front) {
    int sum = 0;
    Node* temp = front;
    while (temp != NULL) {
        sum += temp->ticket_number;
        temp = temp->next;
    }
    return sum;
}

int main() {
    int N;
    scanf("%d", &N);  // Number of people in the queue

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

    // Enqueue the ticket numbers
    for (int i = 0; i < N; i++) {
        int ticket_number;
        scanf("%d", &ticket_number);
        enqueue(&front, &rear, ticket_number);
    }

    // Calculate and print the sum of the ticket numbers in the queue
    int totalSum = calculateSum(front);
    printf("%d\n", totalSum);
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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 : 16

Section 1 : MCQ

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

Answer

None of these

Status : Wrong ***Marks : 0/1***

2. 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***

3. 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***

4. 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***

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

Answer

First In First Out

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

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

Answer

Enqueue and Dequeue

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

7. 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***

8. 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***

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

Answer

All of the mentioned options

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

10. 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***

11. 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? 0

Status : Wrong ***Marks : 0/1***

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

Answer

Stack

Status : Wrong ***Marks : 0/1***

13. 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***

14. 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***

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

Answer

Overflow

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. 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***

18. What are the applications of dequeue?

Answer

All the mentioned options

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

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

Answer

Dequeue

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

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

Answer

FRONT==REAR-1

Status : Wrong ***Marks : 0/1***

Rajalakshmi Engineering College

Name: Naren Kartic B
Email: 241901065@rajalakshmi.edu.in
Roll no:
Phone: 6374678252
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_SIZE 25

int main() {
    int queue[MAX_SIZE];
    int front = 0, rear = -1;
    int N, i;

    // Read the number of customers
    scanf("%d", &N);

    // If N is greater than 0, read customer IDs
    if (N > 0) {
        for (i = 0; i < N; i++) {
            scanf("%d", &queue[i]);
            rear++;
        }
    } else {
        // Read the dummy 0 input if N == 0, as per sample input format
        int dummy;
        scanf("%d", &dummy);
    }

    // Dequeue (serve the first customer)
    if (front > rear) {
        printf("Underflow\n");
    } else {
        front++; // Move front to the next customer
    }

    // Display remaining queue
    if (front > rear) {
        printf("Queue is empty\n");
    } else {
        for (i = front; i <= rear; i++) {
            printf("%d", queue[i]);
            if (i < rear) {
                printf(" ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

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>

#define MAX_SIZE 20

int main() {
   int queue[MAX_SIZE];
   int front = 0, rear = -1;
   int n;
```

```
    // Read number of tickets
    scanf("%d", &n);

    // Enqueue ticket IDs
    for (int i = 0; i < n; i++) {
        scanf("%d", &queue[i]);
        rear++;
    }

    // Display initial queue
    printf("Queue: ");
    for (int i = front; i <= rear; i++) {
        printf("%d", queue[i]);
        if (i < rear) {
            printf(" ");
        }
    }
    printf("\n");

    // Dequeue the front ticket
    front++;

    // Display queue after dequeue
    printf("Queue After Dequeue: ");
    for (int i = front; i <= rear; i++) {
        printf("%d", queue[i]);
        if (i < rear) {
            printf(" ");
        }
    }
    printf("\n");

    return 0;
}
```

Status : Correct **_Marks : 10/10_**

3. 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>

int main() {
   int n, multiple;

   // Read the number of elements in the queue
   scanf("%d", &n);

   int queue[n];

   // Read the elements of the queue
   for (int i = 0; i < n; i++) {
      scanf("%d", &queue[i]);
   }

   // Read the divisor for selective dequeue
   scanf("%d", &multiple);

   // Display original queue
   printf("Original Queue: ");
   for (int i = 0; i < n; i++) {
      printf("%d", queue[i]);
      if (i < n - 1) {
         printf(" ");
```

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

    // Process the queue to remove elements that are divisible by 'multiple'
    printf("Queue after selective dequeue: ");
    int first = 1;
    for (int i = 0; i < n; i++) {
        if (queue[i] % multiple != 0) {
            if (!first) {
                printf(" ");
            }
            printf("%d", queue[i]);
            first = 0;
        }
    }
    printf("\n");

    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>

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

// Function to create a new node
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to enqueue a new element into the queue
void enqueue(struct Node** front, struct Node** rear, int data) {
   struct Node* newNode = createNode(data);

   // If the queue is empty, both front and rear will point to the new node
   if (*rear == NULL) {
      *front = *rear = newNode;
      return;
   }

   // Add the new node to the end of the queue and move rear
```

```
    (*rear)->next = newNode;
    *rear = newNode;
}

// Function to dequeue an element from the queue
int dequeue(struct Node** front) {
    // If the queue is empty, return -1
    if (*front == NULL) {
        return -1;
    }

    // Get the node at the front of the queue
    struct Node* temp = *front;
    int data = temp->data;

    // Move the front pointer to the next node
    *front = (*front)->next;

    // Free the memory of the dequeued node
    free(temp);

    return data;
}

// Function to print the dequeued elements
void printDequeuedElements(struct Node** front) {
    printf("Dequeued elements: ");

    // Dequeue all elements and print them
    int data;
    int first = 1;
    while ((data = dequeue(front)) != -1) {
        if (!first) {
            printf(" ");
        }
        printf("%d", data);
        first = 0;
    }
    printf("\n");
}

int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;
    int num;

    // Read the integers and enqueue them
    while (1) {
        scanf("%d", &num);
        if (num == -1) {
            break;
        }
        enqueue(&front, &rear, num);
    }

    // Print the dequeued elements
    printDequeuedElements(&front);

    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>

#define MAX_SIZE 10

// Function to print the queue elements
void printEnqueued(int arr[], int n) {
   for (int i = 0; i < n; i++) {
      printf("Enqueued: %d\n", arr[i]);
   }
}

// Function to find the Kth largest element in the queue
int findKthLargest(int arr[], int n, int K) {
   // Sort the array in descending order
   for (int i = 0; i < n-1; i++) {
      for (int j = i+1; j < n; j++) {
         if (arr[i] < arr[j]) {
               // Swap arr[i] and arr[j]
               int temp = arr[i];
               arr[i] = arr[j];
               arr[j] = temp;
         }
      }
   }
   return arr[K-1];  // Kth largest is at index K-1 in a zero-indexed array
}

int main() {
   int N, K;
   int queue[MAX_SIZE];

   // Input the number of elements
   scanf("%d", &N);

   // Input the elements and enqueue them
   for (int i = 0; i < N; i++) {
      scanf("%d", &queue[i]);
      printf("Enqueued: %d\n", queue[i]);
   }

   // Input K for finding the Kth largest element
   scanf("%d", &K);

   // Find the Kth largest element and display it
   int result = findKthLargest(queue, N, K);
   printf("The %dth largest element: %d\n", K, result);

   return 0;
}
```

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