Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 17

Section 1 : MCQ

1. 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***

2. 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***

3. What are the applications of dequeue?

Answer

All the mentioned options

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

4. 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

20 30 40 15

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

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

Answer

Overflow

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

6. 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***

7. 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***

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

Answer

First In First Out

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

9. 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 5

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

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

Answer

Dequeue

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

11. 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***

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

Answer

Enqueue and Dequeue

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

13. 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***

14. 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***

15. 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***

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. 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***

18. 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***

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

Answer

All of the mentioned options

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: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

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

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

Implement a program that provides the following functionalities:

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

Input Format

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

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

Choice 2: Dequeue a ticket from the queue.

Choice 3: Display the ticket IDs in the queue.

Choice 4: Exit the program.

Output Format

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

If the choice is 1:

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

If the choice is 2:

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

If the choice is 3:

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

If the choice is 4:

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

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

Refer to the sample output for formatting specifications.

Sample Test Case

```
Input: 1 101
1 202
1 203
1 204
1 205
1 206
3
2
3
4
Output: Helpdesk Ticket ID 101 is enqueued.
Helpdesk Ticket ID 202 is enqueued.
Helpdesk Ticket ID 203 is enqueued.
Helpdesk Ticket ID 204 is enqueued.
Helpdesk Ticket ID 205 is enqueued.
Queue is full. Cannot enqueue.
Helpdesk Ticket IDs in the queue are: 101 202 203 204 205
Dequeued Helpdesk Ticket ID: 101
Helpdesk Ticket IDs in the queue are: 202 203 204 205
Exiting the program
```

Answer

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

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

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

```
// You are using GCC
int isEmpty() {
    return front == -1 || front>rear;
}

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

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

int dequeue() {
    if(isEmpty()){
        return 0;

    }
    lastDequeued = ticketIDs[front];
    front ++;
    if(front>rear){
        initializeQueue();
    }
    return 1;
}

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

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

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

Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_PAH

Attempt : 1
Total Mark : 50
Marks Obtained : 0

Section 1 : Coding

1. 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

-

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

2. 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

-

Status : - **Marks : 0/10**

3. 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

-

Status : - *Marks : 0/10*

4. 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

-

Status : **-** ***Marks : 0/10***

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

-

Status : **-** ***Marks : 0/10***

Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

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

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

// Define the structure for a 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));
   if (!newNode) {
       printf("Memory error\n");
       exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

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

// Function to remove erroneous (negative) tasks
void removeErroneousTasks(struct Node** front) {
   while (*front != NULL && (*front)->data < 0) {
       struct Node* temp = *front;
       *front = (*front)->next;
       free(temp);
   }

   struct Node* current = *front;
   while (current != NULL && current->next != NULL) {
       if (current->next->data < 0) {
           struct Node* temp = current->next;
           current->next = current->next->next;
           free(temp);
       } else {
           current = current->next;
       }
   }
}
```

```
// Function to print remaining tasks in queue
void printQueue(struct Node* front) {
   printf("Queue Elements after Dequeue:");
   while (front != NULL) {
      printf(" %d", front->data);
      front = front->next;
   }
   printf("\n");
}

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

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

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

   removeErroneousTasks(&front);
   printQueue(front);

   return 0;
}
```

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

2. Problem Statement

Fathima has been tasked with developing a program to manage a queue of customers waiting in line at a service center. Help her write a program simulating a queue data structure using a linked list.

Here is a description of the scenario and the required operations:

Enqueue: Add a customer to the end of the queue.Dequeue: Remove and discard a customer from the front of the queue.Display waiting customers: Display the front and rear customer IDs in the queue.

Write a program that enqueues all the customers into the queue, performs a dequeue operation, and prints the front and rear elements.

Input Format

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

The second line consists of N space-separated integers, representing the customer IDs.

Output Format

The output prints "Front: X, Rear: Y" where X is the front element and Y is the rear element, after performing the dequeue operation.

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

Sample Test Case

Input: 5
112 104 107 116 109
Output: Front: 104, Rear: 109

Answer

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

// Define structure for a 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 customer
void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
}

// Function to dequeue a customer
void dequeue(struct Node** front) {
    if (*front == NULL) return;
    struct Node* temp = *front;
    *front = (*front)->next;
    free(temp);
}

// Function to get front element's data
int getFront(struct Node* front) {
    return front ? front->data : -1;
}

// Function to get rear element's data
int getRear(struct Node* rear) {
    return rear ? rear->data : -1;
}

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

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

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

    // Perform one dequeue operation
    dequeue(&front);

    printf("Front: %d, Rear: %d\n", getFront(front), getRear(rear));

    return 0;
}
```

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

3. 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>

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

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

// Enqueue function
void enqueue(struct Node** rear, struct Node** front, int data) {
   struct Node* newNode = createNode(data);
   if (*rear == NULL) {
      *rear = *front = newNode;
   } else {
      (*rear)->next = newNode;
      *rear = newNode;
   }
}

// Function to get Kth element from end
int getKthFromEnd(struct Node* front, int K) {
   struct Node* mainPtr = front;
   struct Node* refPtr = front;

   // Move refPtr K nodes ahead
   for (int i = 0; i < K; i++) {
      if (refPtr == NULL) return -1; // K > length (invalid)
      refPtr = refPtr->next;
   }

   // Move both pointers until refPtr reaches the end
   while (refPtr != NULL) {
      mainPtr = mainPtr->next;
      refPtr = refPtr->next;
   }

   return mainPtr->data;
}

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

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

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

   int K;
   scanf("%d", &K);

   int result = getKthFromEnd(front, K);
   printf("%d\n", result);

   return 0;
}
```

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