Rajalakshmi Engineering College

Name: krithika narasimhan
Email: 240701277@rajalakshmi.edu.in
Roll no: 240701277
Phone: 9677451731
Branch: REC
Department: I CSE FC
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 4_PAH

Attempt : 2
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. 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***

2. 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***

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

```
#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***

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

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 <iostream>
using namespace std;

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

Node* front = nullptr;
```

```
Node* rear = nullptr;

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

// Function to display the queue
void display() {
    Node* temp;
    temp = front;
    while (temp) {
        cout <<  temp->data << " " ;
        temp = temp->next;
    }

    cout << endl;
}

// Function to delete an element from the queue
void dequeue() {
    Node* temp;
    temp = front;
    front = front->next;
    delete temp;

}

int main() {
    int a, data;
    cin >> a;
    for (int i = 0; i < a; i++) {
        cin >> data;
        enqueue(data);
    }
    cout << "Queue: ";
    display();
    cout << "Queue After Dequeue: ";
    dequeue();
    display();

    return 0;
}
```

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