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

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>

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

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

```
void enqueue(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;

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

void dequeue() {
    if (front == NULL) {
        return;
    }

    Node* temp = front;
    front = front->next;

    if (front == NULL) {
        rear = NULL;
    }

    free(temp);
}

void displayQueue() {
    Node* temp = front;
    printf("Queue: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void displayQueueAfterDequeue() {
    dequeue();
    Node* temp = front;
    printf("Queue After Dequeue: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

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

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

    displayQueue();
    displayQueueAfterDequeue();

    return 0;
}
```

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

2. 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

int findKthLargest(int arr[], int size, int k) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (arr[i] < arr[j]) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    return arr[k - 1];
}

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

   scanf("%d", &N);

   if (N > MAX_SIZE) {
      printf("Error: Maximum queue size exceeded.\n");
      return 1;
   }

   for (int i = 0; i < N; i++) {
      scanf("%d", &queue[i]);
      printf("Enqueued: %d\n", queue[i]);
   }

   scanf("%d", &K);

   if (K < 4 || K > N) {
      printf("Error: K is out of valid range.\n");
      return 1;
   }

   printf("The %dth largest element: %d\n", K, findKthLargest(queue, N, K));

   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>

#define MAX_SIZE 50

void selectiveDequeue(int queue[], int n, int multiple) {
   printf("Original Queue: ");
   for (int i = 0; i < n; i++) {
      printf("%d ", queue[i]);
   }
   printf("\n");

   printf("Queue after selective dequeue: ");
   for (int i = 0; i < n; i++) {
      if (queue[i] % multiple != 0) {
         printf("%d ", queue[i]);
      }
   }
   printf("\n");
}

int main() {
   int n, multiple;
   int queue[MAX_SIZE];

   scanf("%d", &n);

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

   scanf("%d", &multiple);
```

```
    selectiveDequeue(queue, n, multiple);

    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>

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

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

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

void enqueue(Queue* queue, int value) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = value;
   newNode->next = NULL;

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

void dequeueAll(Queue* queue) {
   printf("Dequeued elements: ");
   while (queue->front != NULL) {
      Node* temp = queue->front;
      printf("%d ", temp->data);
      queue->front = queue->front->next;
      free(temp);
   }
   printf("\n");
   queue->rear = NULL;
}
```

```
int main() {
    Queue* queue = createQueue();
    int value;

    while (1) {
        scanf("%d", &value);
        if (value == -1) break;
        enqueue(queue, value);
    }

    dequeueAll(queue);

    free(queue);

    return 0;
}
```

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

5. 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

// Function to serve (dequeue) the first customer and display the remaining
queue
void serveCustomer(int queue[], int size) {
   if (size == 0) {
      printf("Underflow\nQueue is empty\n");
      return;
   }

   // Printing the updated queue after serving the first customer
   for (int i = 1; i < size; i++) {
      printf("%d ", queue[i]);
   }
   printf("\n");
}

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

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

```
    // Check if N exceeds the maximum queue capacity
    if (N > MAX_SIZE) {
        printf("Error: Maximum queue size exceeded.\n");
        return 1;
    }

    // Read customer IDs
    for (int i = 0; i < N; i++) {
        scanf("%d", &queue[i]);
    }

    // Serve the first customer and display the updated queue
    serveCustomer(queue, N);

    return 0;
}
```

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