

Name	Shubhan Singh
UID no.	2022300118
Experiment No.	2

AIM:	Study of queue data structure
-------------	-------------------------------

Program 1

PROBLEM STATEMENT :	Implement a Queue ADT with a circular array
----------------------------	---

Code:	<pre> /* * File: queue.c * Author: Siddhartha Chandra * Email: siddhartha_chandra@spit.ac.in * Created: September 1, 2023 * Description: This program implements a Queue ADT with a circular array */ #include<stdio.h> #include<stdlib.h> #include<stdbool.h> // Build a Queue Abstract Data structure and perform operations on it // Operations: // 1. display // 2. isFull // 3. isEmpty // 4. enqueue // 5. dequeue // 6. front // 7. rear typedef struct Queue { int front; int rear; unsigned size; char *array; bool prev;//this stores the previous operation performed(push or pop), which helps in ascertaining whether the queue is full or empty //true=pop,false=push }que; int isEmpty(struct Queue* queue); char rear(struct Queue* queue); // 0 -> Initialize struct Queue* initialize_queue(unsigned size){ </pre>
--------------	---

```

    que* queue=malloc(sizeof(que));
    queue->size=size;
    queue->array=calloc(size,sizeof(char));
    queue->rear=0;
    queue->front=0;
    queue->prev=true;
}

// 0 -> display
void display(struct Queue* queue){
    if(isEmpty(queue)){
        printf("Queue is empty!\n");
        return;
    }
    int f=queue->front;
    int r=queue->rear;
    printf("The elements in the queue(from front to rear) are:\n");
    while(true){
        printf("%c ",queue->array[f]);
        f++;
        f%=queue->size;
        if(f==r){
            break;
        }
    }
    printf("\n");
}

// 1 -> isEmpty
int isEmpty(struct Queue* queue){
    if(queue->front==queue->rear){
        if(queue->prev){
            return 1;
        }
    }
    return 0;
}

// 3 -> isFull
int isFull(struct Queue* queue){
    if(queue->front==queue->rear){
        if(!(queue->prev)){
            return 1;
        }
    }
    return 0;
}

// 4 -> enqueue
void enqueue(struct Queue *queue, char item){
    if(!isFull(queue)){
        queue->array[queue->rear]=item;
        queue->rear++;
        queue->rear%=queue->size;
        queue->prev=false;
    }
}

```

```

    }
    else{
        printf("Queue is full!\n");
    }
}

// 5 -> dequeue
char dequeue(struct Queue* queue){
    if(!isEmpty(queue)){
        queue->prev=true;
        char retval=queue->array[queue->front];
        queue->front++;
        queue->front%=queue->size;
        return retval;
    }
    else{
        printf("Queue is empty!\n");
        return 0;
    }
}

// 6 -> front
char front(struct Queue* queue){
    if(!isEmpty(queue)){
        return queue->array[queue->front];
    }
    else{
        printf("Queue is empty!\n");
        return 0;
    }
}

// 7 -> rear
char rear(struct Queue* queue){
    if(!isEmpty(queue)){
        int i=queue->rear-1;
        if(i<0){
            i=queue->size-1;
        }
        return queue->array[i];
    }
    else{
        printf("Queue is empty!\n");
        return 0;
    }
}

void destruct_q(queue* qu){
    free(qu);
}

//driver code

void operate(queue* operand, int operator){

```

```

    if(operator==0){
        return;
    }
    else if(operator==1){
        int no;
        printf("Enter number of elements to enqueue: ");
        scanf("%d",&no);
        char temp;
        char str[no+1];
        while((getchar())!='\n');
        printf("Enter elements to enqueue (in a string):\n");
        fgets(str,no+1,stdin);
        while((getchar())!='\n');
        for(int i=0;i<no;i++){
            temp=str[i];
            enqueue(operand,temp);
        }
    }
    else if(operator==2){
        int temp=dequeue(operand);
    }
    else if(operator==3){
        printf("The element at the front of the stack is:
%c\n",front(operand));
    }
    else if(operator==4){
        printf("The element at the rear of the stack is: %c\n",rear(operand));
    }
    else if(operator==5){
        display(operand);
    }
    else if(operator==6){
        if(isEmpty(operand)){printf("The queue is empty\n");}
        else{
            printf("The queue is not empty\n");
        }
    }
    else if(operator==7){
        if(isFull(operand)){printf("The queue is full\n");}
        else{
            printf("The queue is not full\n");
        }
    }
    else{
        printf("Invalid input\n");
    }
}

int main(){
    unsigned size;
    printf("Enter size of queue\n");
    scanf("%u",&size);
    que* queue=malloc(sizeof(que));
    queue=initialize_queue(size);
    int input=-1;

```

```

        while(input){
            printf("Enter\n1 to enqueue\n2 to dequeue\n3 to see front\n4 to see
rear\n5 to display queue\n6 to check if empty\n7 to check if full\n0 to end
program\n");
            scanf("%d",&input);
            operate(queue,input);
        }
        destruct_q(queue);
    return 0;
}

```

```

PS C:\Users\shubh\OneDrive - Bharatiya Vidya Bhavans Sardar Patel Institute Of
echnology\C programs\data structures\" ; if ($?) { gcc queue_char.c -o queue_
Enter size of queue
8
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
1
Enter number of elements to enqueue: 6
Enter elements to enqueue (in a string):
shubha
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
1
Enter number of elements to enqueue: 4
Enter elements to enqueue (in a string):
shub
Queue if full!
Queue if full!
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program

```

RESULT:

```
0 to end program
5
The elements in the queue(from front to rear) are:
s h u b h a s h
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
2
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
5
The elements in the queue(from front to rear) are:
h u b h a s h
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
6
The queue is not empty
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
```

```

The element at the front of the stack is: h
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
4
The element at the rear of the stack is: h
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
5
The elements in the queue(from front to rear) are:
h u b h a s h
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
1
Enter number of elements to enqueue: 1
Enter elements to enqueue (in a string):
d
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
7
The queue is full
Enter
1 to enqueue
2 to dequeue
3 to see front
4 to see rear
5 to display queue
6 to check if empty
7 to check if full
0 to end program
0
PS C:\Users\shubh\OneDrive - Bha

```

Program 2

PROBLEM STATEMENT :	<p>Digital circle of destiny- Problem statement:</p> <p>In the realm of computational challenges, we encounter an intriguing scenario involving 'n' tech-savvy individuals engrossed in a virtual gaming experience. These friends are seated in a virtual circle, each bearing a unique numeric identifier from 1 to n. In this digital circle, navigating clockwise from the ith individual takes you to the $(i+1)$th individual, where $1 \leq i < n$. Furthermore, if you venture clockwise from the nth individual, you'll seamlessly return to the 1st individual.</p> <p>Now, let's delve into the intricate rules governing this immersive gaming environment:</p> <ol style="list-style-type: none"> 1. The adventure begins with the 1st friend. 2. The next k friends in the clockwise direction, including the starting friend, are meticulously counted. It's important to note that this counting wraps around the virtual circle, which means you may end up counting the same friend more than once. 3. The friend who is counted last in this process must bid farewell to the circle and is, unfortunately, declared out of the game. 4. If there are still more than one friend remaining in the circle, the gaming saga continues. We return to step 2, starting from the friend immediately clockwise of the
----------------------------	--

	<p>individual who just left the circle, and the counting ritual repeats.</p> <p>5. The game continues until only one friend remains inside the circle. At this point, the last remaining friend is crowned as the ultimate victor of the virtual gaming contest.</p> <p>Your mission, as an aspiring computer engineer, is to develop a computational solution that, given the number of friends represented by 'n' and an integer 'k', can efficiently determine and declare the triumphant friend who emerges victorious from this captivating digital circle.</p> <p>The challenge awaits your algorithmic prowess! Craft a program to unveil the winner of this virtual gaming extravaganza, and may the code be ever in your favor.</p>
ALGORITHM:	<p>To find the winner of the digital circle of destiny, we use the following algorithm:</p> <ol style="list-style-type: none"> 1. Initialize a queue of size n and load numbers from 1 to n in it. 2. Repeat steps 3-4 n-1 times. 3. perform the dequeue and enqueue operation(immediately after dequeue) k-1 times on the queue. 4. perform the dequeue operation once 5. return the value at the front(the only remaining value in the array) as the answer
CODE:	<pre> #include "queue.c" #include<stdio.h> #include<stdlib.h> int findDCODChampion(int n, int k){ que* dcod=initialize_queue(n); for(int i=1;i<=n;i++){ enqueue(dcod,i); } } </pre>

```

    }
    int temp;
    for(int c=0;c<n-1;c++){
        for(int i=0;i<k-1;i++){
            temp=dequeue(dcod);
            enqueue(dcod,temp);
        }
        temp=dequeue(dcod);
    }
    return front(dcod);
}
int main(){
    int n,k,ans;
    printf("Enter value of n and k for DCOD\n");
    scanf("%d %d",&n,&k);
    ans=findDCODChampion(n,k);
    printf("The winner of the game is %d\n",ans);
    return 0;
}

```

RESULT:

```

technology\c programs\data struc
Enter value of n and k for DCOD
5 2
The winner of the game is 3
PS C:\Users\shubh\OneDrive\Pl

```

```

technology\c programs\data struc
Enter value of n and k for DCOD
6 5
The winner of the game is 1
PS C:\Users\shubh\OneDrive\Pl

```

```

technology\c programs\data struc
Enter value of n and k for DCOD
8 3
The winner of the game is 7

```

Solution on paper:

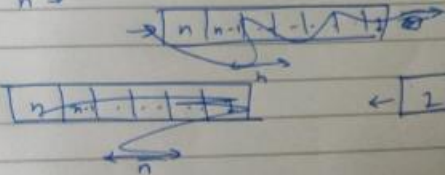
Shubhan Singh / 2022300118 / B3

Experiment 2 - queues Date - 11-9-23


2-A - Implementation of queue ADT using circular array
→ Solved using method given in PPT.

2-B - Digital circle of destiny

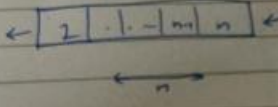
→ enter the contestants' numbers in a queue of size n



→

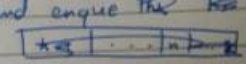


→



→

dequeue and enqueue the first $k-1$ times



→ now dequeue the k^{th} element

→ k → no. of elements in queue dec. by 2

→ repeat this $(n-1)$ times

→ Only one element in the queue will remain at the end → That is the winner

Explanation of a test case:

Input: $n = 5, k = 2$

Output: 3

Explanation: Here are the steps of the game:

- 1) Start at friend 1.
- 2) Count 2 friends clockwise, which are friends 1 and 2.
- 3) Friend 2 leaves the circle. Next start is friend 3.
- 4) Count 2 friends clockwise, which are friends 3 and 4.
- 5) Friend 4 leaves the circle. Next start is friend 5.

	<p>6) Count 2 friends clockwise, which are friends 5 and 1.</p> <p>7) Friend 1 leaves the circle. Next start is friend 3.</p> <p>8) Count 2 friends clockwise, which are friends 3 and 5.</p> <p>9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.</p> <p>This is implemented in our code by dequeuing the kth person, while maintaining the other participants in the game, and since our queue is circular, we wrap around the queue again and again.</p>
Theory:	<p>Queue Data Structure:</p> <p>A queue is a fundamental data structure in computer science that follows the First-In-First-Out (FIFO) principle. It represents a linear collection of elements in which elements are added at one end (the rear or enqueue operation) and removed from the other end (the front or dequeue operation). Queues are commonly used to manage tasks or data that must be processed in the order they are received.</p> <p>Key characteristics of a queue data structure:</p> <ol style="list-style-type: none"> 1. FIFO Order: The element that has been in the queue the longest is the first one to be removed (dequeued). 2. Operations: <ul style="list-style-type: none"> • Enqueue: Add an element to the rear of the queue. • Dequeue: Remove and return the element from the front of the queue. • Front: Get the element at the front of the queue without removing it. • Rear: Get the element at the rear of the queue without removing it. 3. Applications: Queues are used in various applications such as scheduling tasks in operating systems, managing print jobs in printers, implementing breadth-first search algorithms in graph traversal, and more. <p>Circular Queues:</p> <p>A circular queue (also known as a circular buffer or a ring buffer) is a variant of the standard queue data structure with some unique features. In a circular queue, the last element is connected to the first element, creating a circular arrangement. This design offers several advantages:</p> <ol style="list-style-type: none"> 1. Efficient Use of Space: In a regular queue implemented using an array, when elements are dequeued, the space at the front becomes unusable. In contrast, a circular queue reuses the space at the front

	<p>when elements are dequeued, maximizing space utilization.</p> <ol style="list-style-type: none"> 2. Constant Time Enqueue and Dequeue: In a circular queue, enqueue and dequeue operations can be performed in constant time, $O(1)$, because the front and rear pointers wrap around when they reach the end of the array. 3. Applications: Circular queues are particularly useful in situations where a fixed-size buffer is needed, such as in data streaming applications, input and output buffers in operating systems, and managing a fixed number of resources in real-time systems.
Conclusion:	<ol style="list-style-type: none"> 1. We learnt about the queue data structure and the implementation of the same using an abstract circular array. 2. Queues are useful data structures, especially in the form of circular arrays, when we want our values to stay in a range and wrap around.