Data Structures Lab

Experiment 7: Circular Queue

Project By:

Mohammed Rabeeh
Roll No: 35
TVE18CS036

Contents

1 Aim

To write a menu driven program to implement Circular Queue.

2 Problem Description

Circular Queue is similar to Queue data structure. However it stores data in a circular manner. The advantages of Circular Queue over Normal Queue are

1. The entire list can be traversed starting from any node (traverse means visit every node just once).
2. Some problems are circular and a circular data structure would be more natural when used to represent it.

3 Algorithm

3.1 Insert

1. If queue is full, return
2. If front = -1 and rear = -1, assign both of them to zero.
3. Assign rear = (rear + 1) % size;
4. Assign cq[rear] = n

3.2 Pop

1. If queue is empty, return
2. If front is equal to rear, assign front and rear both to -1
3. Else, front = (front + 1) % size

3.3 Display

1. If queue is empty, return
2. If front ¡ rear, print elements from left to right
3. if front ¿ rear, print elements from front to size - 1 and from 0 index to rear

4 Program Code

```
#include<stdio.h>
#include<stdlib.h>

int* cq;
int front = -1, rear = -1, size, option, n;

int isFull() {
   if((front == 0 && rear == size-1) || (rear == front - 1))
      return 1;
   return 0;
}
void push(int n) {
   if (isFull()) {
      printf("\nQueue Full.\n");
      return ;
   }
   if(front == -1 && rear == -1) {
      front = 0;
      rear = 0;
   } else {
      rear = (rear + 1) % size;
   }
   cq[rear] = n;
   printf("\n%d inserted successfully!\n", n);
}

int isEmpty() {
   return (front == -1 && rear == -1) ? 1 : 0;
```

```
}

void pop() {
   if(isEmpty()) {
      printf("\nQueue empty.\n");
      return ;
   }
   printf("\n%d popped!\n", cq[front]);
   if(front == rear) {
      front = -1;
      rear = -1;
   } else {
      front = (front + 1) % size;
   }
}

void display() {
   int i;
   if(isEmpty()) {
      printf("\nQueue empty.\n");
      return ;
   }
   printf("\n");
   if(front <= rear) {
      for(i = front; i <= rear; i ++) {
         printf("%d ", cq[i]);
      }
   } else {
      for(i = front; i < size; i++) {
         printf("%d ", cq[i]);
      }

      for(i = 0; i <= rear; i++) {
         printf("%d ", cq[i]);
      }
   }
   printf("\n");
}

void main() {
```

```
   printf("Enter size: ");
   scanf("%d", &size);
   cq = (int *) malloc(size * sizeof(int));
   printf("\n1.Push\n2.Pop\n3.Display\n");
   while(1) {
      printf("Choose an option: ");
      scanf("%d", &option);
      switch(option) {
         case 1:
            printf("Enter an element: ");
            scanf("%d", &n);
            push(n);
            break;
         case 2:
            pop();
            break;
         case 3:
            display();
            break;
         default:
            printf("Wrong option!\n");
      }
   }
}
```

5 Output

```
Enter size: 3

1.Push
2.Pop
3.Display
Choose an option: 1
Enter an element: 5

5 inserted successfully!
Choose an option: 1
Enter an element: 6

6 inserted successfully!
Choose an option: 1
Enter an element: 7

7 inserted successfully!
Choose an option: 3

5 6 7
Choose an option: 2

5 popped!
Choose an option: 3

6 7
Choose an option:
```

6 Result

Circular Queue was implemented in the C languages and the basic functionalities of the queue were tested and the output was verified.