Vidyavardhini's College of Engineering & Technology
Department of Computer Engineering

Experiment No 4: Linear Queue

Name : Vaibhav Tatkare
Branch / Div : Comps 3
Roll No. : 52

A queue data structure can be implemented using one dimensional array. The queue implemented using array stores only fixed number of data values. The implementation of queue data structure using array is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then delete the element which is at 'front' position and increment 'front' value by one.

Queue Operations using Array

Queue data structure using array can be implemented as follows...

Before we implement actual operations, first follow the below steps to create an empty queue.

- Step 1 - Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
- Step 2 - Declare all the user defined functions which are used in queue implementation.
- Step 3 - Create a one dimensional array with above defined SIZE (int queue[SIZE])
- Step 4 - Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
- Step 5 - Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

enQueue(value) - Inserting value into the queue

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at rear position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

- Step 1 - Check whether queue is FULL. (rear == SIZE-1)

- Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value.

deQueue - Deleting a value from the Queue

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from front position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

- Step 1 - Check whether queue is EMPTY. (front == rear)
- Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...

- Step 1 - Check whether queue is EMPTY. (front == rear)
- Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function.
- Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'.
- Step 4 - Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

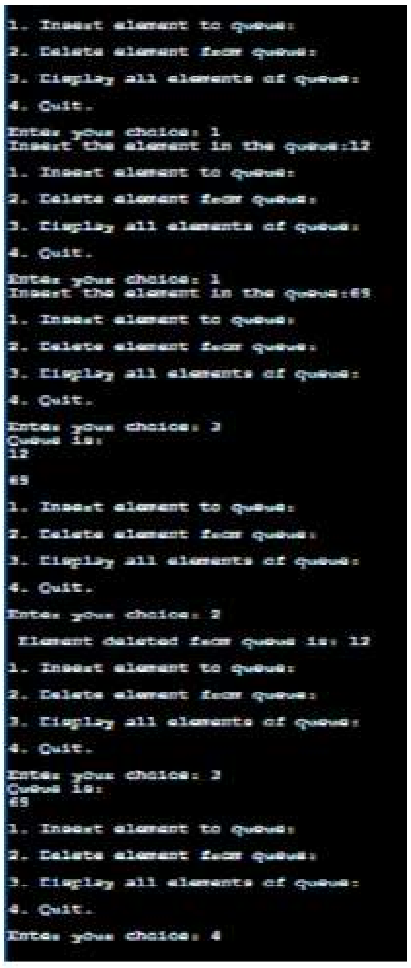
Code :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
void enqueue () ;
void dequeue () ;
void display () ;
int queue_array[MAX] ;
int rear = -1 ;
int front = -1 ;

void main () {
  int choice ;
  clrscr () ;
  while (1) {
    printf ("\n1. Insert element to queue:\n") ;
    printf ("\n2. Delete element from queue:\n") ;
    printf ("\n3. Display all elements of queue:\n") ;
    printf ("\n4. Quit.\n") ;
    printf ("\nEnter your choice: ") ;
    scanf ("%d",&choice) ;
    switch (choice) {
      case 1 :
      enqueue () ;
      break ;
      case 2 :
      dequeue () ;
      break ;
      case 3 :
      display () ;
      break ;
      case 4 :
      exit (1) ;
      default:
      printf ("\nWrong choice\n") ;
    }
  }
}
void enqueue () {
  int item ;
  if (rear == MAX - 1)
    printf ("\nQueue Overflow\n") ;
  else {
    if (front == -1)
      front = 0 ;
    printf ("Insert the element in the queue:") ;
    scanf ("%d", &item) ;
    rear++ ;
    queue_array [rear] = item ;
  }
}
void dequeue () {
  if (front == -1 || front > rear) {
    printf ("\nQueue Underflow\n") ;

  }
  else {
    printf ("\n Element deleted from queue is: %d\n", queue_array[front]) ;
    front++ ;
  }
}
void display () {
  int i ;
  if (front == -1)
    printf ("\nQueue is Empty.\n") ;
  else {
    printf ("Queue is: ") ;
    for (i=front ; i<=rear ; i++) {
      printf ("\n%d\t", queue_array[i]) ;
      printf ("\n") ;
    }
  }
}
```

Output:



```
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 1
Insert the element in the queue:12
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 1
Insert the element in the queue:65
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 3
Queue is:
12

65
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 2
 Element deleted from queue is: 12
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 3
Queue is:
65
1. Insert element to queue:
2. Delete element from queue:
3. Display all elements of queue:
4. Quit.
Enter your choice: 4
```