**Batch: A1 Roll No.: 1911004**

**Experiment No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:**  Implementation of Queue Data Structure |

**Objective:** To understand different types of queues and their implementation.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| CO1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. Data Structures APsedocode Approach with C, Richard F.Gilberg & Behrouz A.Forouzan,2nd edition, CENGAGE learning.
2. Data Structures Using C & C++, Rajesh K. Shukla, Wiley- india.

**Abstract**:-

**Queue is an ADT (abstract data type , linear data structure), it follows a particular order in which the operations can be performed. The order is First In First Out (FIFO), in which the first element is inserted from one end called the Rear (Tail), and the removal of existing element takes place from the other end called as Front (Head).**

**Double Ended Queue (Dequeue) is a more advance form of queue data structure. It is an ordered collection of elements similar to the queue. It has two ends, a front and a rear, and the elements remain positioned in the collection. Dequeue is different due to its concept of adding and removing items at both ends . New items can be added at either the front or the rear. Thus, existing items can be removed from either end.**

**Related Theory: -**

List different queue types, explain insertion and deletions operations on them with example. Draw/paste diagram wherever necessary. Do cite credits if the images are adopted from internet sources.

**There are four types of Queues:**

1. **Simple Queue**
2. **Circular Queue**
3. **Priority Queue**
4. **Dequeue (Double Ended Queue)**

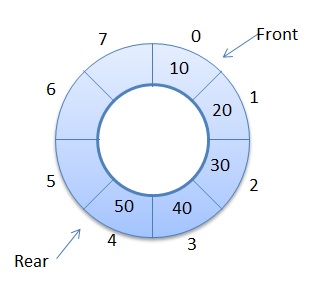
**Simple Queue**

**It is defines the Simple Queue Operation Like Insertion at Rear & Deletion at Front of the data structure Queue. It represents FIFO(First In First Out) .**

**Eg . In the Queue(Line ) when we are waiting for anything hear people from front get access first and people join queue from rear side.**

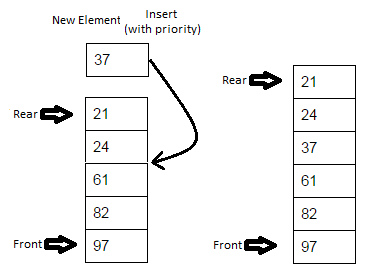
**Circular Queue**

**Circular Queue is a linear data structure, it implements Simple Queue, thus in it also the operations are performed by FIFO (First In First Out) principle and the rear is connected back to the front to make a circle. Thus this is  ‘Ring Buffer’(Circular Data). Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue.**



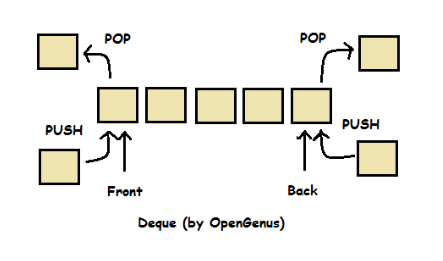
**Priority Queue**

**Priority Queue is more specialized data structure Queue. It implements simple queue, it has same functions yet these methods have some difference. In Priority queue we order elements by key value so that element with the lowest value of key is at front & element with the highest value of key is at rear or vice versa. Thus we assigned priority to elements based on their key value. Lower the value, higher the priority & vice versa.**



**Dequeue**

**Double Ended Queue (Dequeue) is a more advance form of queue data structure. It is an ordered collection of elements similar to the queue. It has two ends, a front and a rear, and the elements remain positioned in the collection. . Dequeue is different due to its concept of adding and removing items at both ends . New items can be added at either the front or the rear. Thus, existing items can be removed from either end.**



Images of all Queues are from Google

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored**

* **Create a dequeue struct for implementation.**
* **Initialize the front & rear of dequeue .**
* **Create a empty() to check if queue is empty**
* **Create a full() to check if queue is full.**
* **Dequeue enables us to insert element at rear or front side of queue.**
* **We create Renqueue() to insert element at Rear of Queue**
* **We create enqueueF() to insert element at Front of Queue**
* **Dequeue also facilitates us to delete elements from rear /front ends of the queue.** 
  + **We create Renqueue() to delete element at Rear of Queue**
  + **We create Fenqueue() to delete element at Front of Queue**
* **We create print() to display all elements present in queue**
* **In main() we create menu driven approach to ask user various operation he wants to perform thus we call all above functions according to choice given.**

1. **Explain your program logic, classes and methods used.**

* **Functions used :**

void initialize(dequeue \*P) //initial the queue of elements

int empty(dequeue \*P) //check if queue is empty

int full(dequeue \*P) //check if queue is full

void Renqueue(dequeue \*P, int x) //insert element at Rear of Queue

void enqueueF(dequeue \*P, int x) //insert elements at Front of Queue

int Fdequeue(dequeue \*P) //delete element from Front of Queue

int Rdequeue(dequeue \*P) //delete element from Rear of Queue

void print(dequeue \*P) //print elements of queue

int main()//main function to call & implement rest functions

1. **Explain the Importance of the approach followed by you**

* **By implementing dequeue the program gives the us the ability to insert elements either at rear / front side of queue .**
* **It gives us option to also delete the element from rear / front ends of queue .**
* **Thus we can operate on queue more easily and efficiently.**
* **We can use Dequeue as a simple queue also by inserting & deleting elements from a single end thus serving as a simple queue .**
* **Thus Dequeue is very flexible for implementation.**

**Program Code:**

**#include<stdio.h>**

**#define MAX 50**

**typedef struct dequeue //dequeue struct**

**{**

**int data[MAX];**

**int rear,front;**

**}dequeue;**

**void initialize(dequeue \*P)**

**{**

**P->rear=-1;**

**P->front=-1;**

**}**

**int empty(dequeue \*P)**

**{**

**if(P->rear==-1)**

**return(1);**

**return(0);**

**}**

**int full(dequeue \*P)**

**{**

**if((P->rear+1)%MAX==P->front)**

**return(1);**

**return(0);**

**}**

**void Renqueue(dequeue \*P,int x) //insert at Rear**

**{**

**if(empty(P))**

**{**

**P->rear=0;**

**P->front=0;**

**P->data[0]=x;**

**}**

**else**

**{**

**P->rear=(P->rear+1)%MAX;**

**P->data[P->rear]=x;**

**}**

**}**

**void enqueueF(dequeue \*P,int x) //insert at Front**

**{**

**if(empty(P))**

**{**

**P->rear=0;**

**P->front=0;**

**P->data[0]=x;**

**}**

**else**

**{**

**P->front=(P->front-1+MAX)%MAX;**

**P->data[P->front]=x;**

**}**

**}**

**int Fdequeue(dequeue \*P) //delete element**

**{**

**int x;**

**x=P->data[P->front];**

**if(P->rear==P->front)**

**initialize(P);**

**else**

**P->front=(P->front+1)%MAX;**

**return(x);**

**}**

**int Rdequeue(dequeue \*P) //delete element**

**{**

**int x;**

**x=P->data[P->rear];**

**if(P->rear==P->front)**

**initialize(P);**

**else**

**P->rear=(P->rear-1+MAX)%MAX;**

**return(x);**

**}**

**void print(dequeue \*P)**

**{**

**if(empty(P))**

**{**

**printf("\nQueue is empty!!");**

**exit(0);**

**}**

**int i;**

**i=P->front;**

**while(i!=P->rear)**

**{**

**printf("\n%d",P->data[i]);**

**i=(i+1)%MAX;**

**}**

**printf("\n%d\n",P->data[P->rear]);**

**}**

**int main()**

**{**

**int i,x,choice,n;**

**dequeue q;**

**initialize(&q);**

**do**

**{**

**printf("OPERATIONS ARE: \n");**

**printf("\n1.Create\n2.Insert(R)\n3.Insert(F)\n4.Delete(R)\n5.Delete(F)\n6.Print\n0.Exit\n\nEnter your choice : ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1: printf("\nEnter number of elements : ");**

**scanf("%d",&n);**

**initialize(&q);**

**printf("\nEnter the data : ");**

**for(i=0;i<n;i++)**

**{**

**scanf("%d",&x);**

**if(full(&q))**

**{**

**printf("\nQueue is full!!");**

**exit(0);**

**}**

**Renqueue(&q,x);**

**}**

**break;**

**case 2: printf("\nEnter element to be inserted : ");**

**scanf("%d",&x);**

**if(full(&q))**

**{**

**printf("\nQueue is full!!");**

**exit(0);**

**}**

**Renqueue(&q,x);**

**break;**

**case 3: printf("\nEnter the element to be inserted : ");**

**scanf("%d",&x);**

**if(full(&q))**

**{**

**printf("\nQueue is full!!");**

**exit(0);**

**}**

**enqueueF(&q,x);**

**break;**

**case 4: if(empty(&q))**

**{**

**printf("\nQueue is empty!!");**

**exit(0);**

**}**

**x=Rdequeue(&q);**

**printf("\nElement deleted is %d\n",x);**

**break;**

**case 5: if(empty(&q))**

**{**

**printf("\nQueue is empty!!");**

**exit(0);**

**}**

**x=Fdequeue(&q);**

**printf("\nElement deleted is %d\n",x);**

**break;**

**case 6: print(&q);**

**break;**

**case 0:printf("EXITING");**

**break;**

**default: break;**

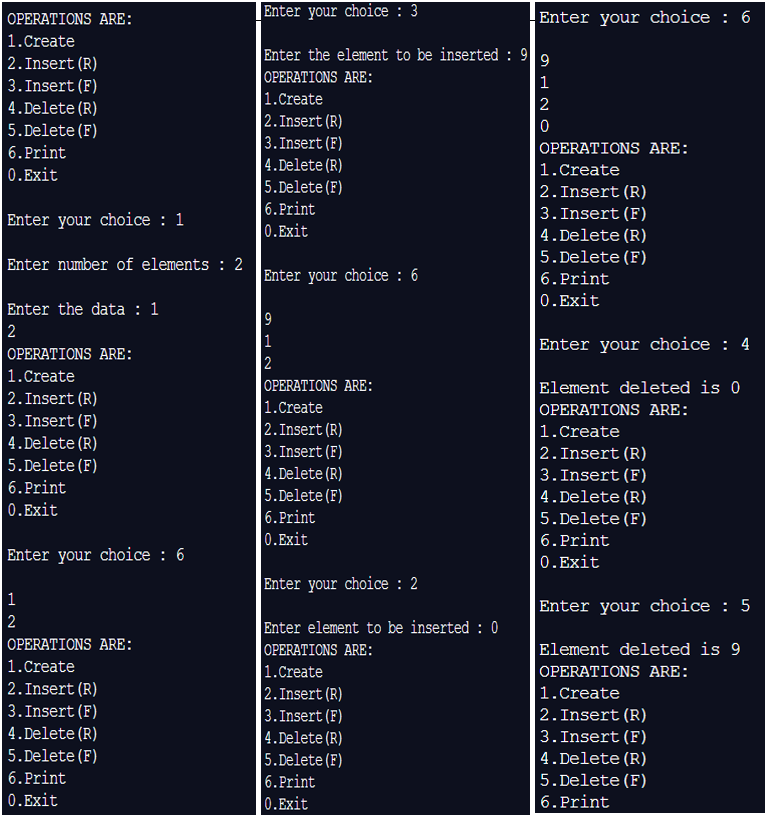
**}**

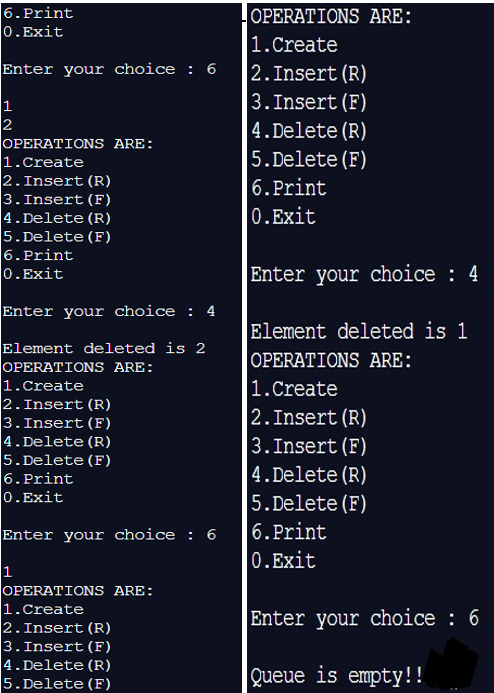
**}while(choice!=0);**

**return 0;**

**}**

**Output Screenshots:**





**Conclusion:-**

**We understood the concept of QUEUES & implemented it in C ; successfully obtained correct necessary output.**