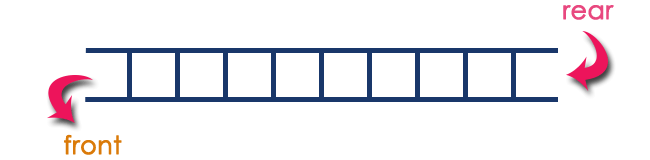
**Queues**

**Introduction**

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing of elements are performed at two different positions. The insertion is performed at one end and deletion is performed at other end. In a queue data structure, the insertion operation is performed at a position which is known as '**rear**' and the deletion operation is performed at a position which is known as '**front**'. In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



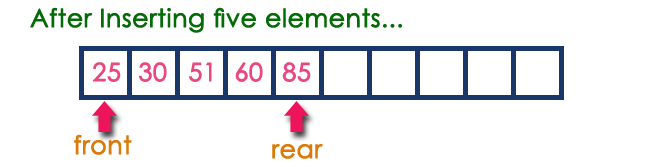
In a queue data structure, the insertion operation is performed using a function called "**enQueue()**" and deletion operation is performed using a function called "**deQueue()**".

Queue data structure can be defined as follows...

**"Queue data structure is a collection of similar data items in which insertion and deletion operations are performed based on FIFO principle".**

**Example**

Queue after inserting 25, 30, 51, 60 and 85.



**Operations on a Queue**

The following operations are performed on a queue data structure...

* **enQueue**(value) - (To insert an element into the queue)
* **deQueue()** - (To delete an element from the queue)
* **display()** - (To display the elements of the queue)

Queue data structure can be implemented in two ways. They are as follows...

* Using Array
* Using Linked List

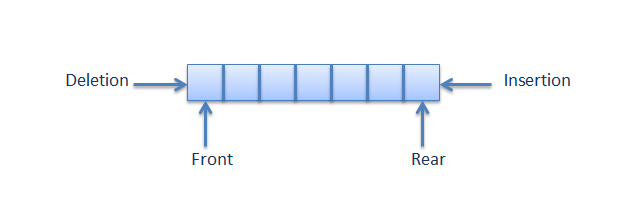
When a queue is implemented using array, that queue can organize only limited number of elements. When a queue is implemented using linked list, that queue can organize unlimited number of elements.

**Types of queue**

* Simple queue
* Circular queue
* Priority queue
* Double ended queue

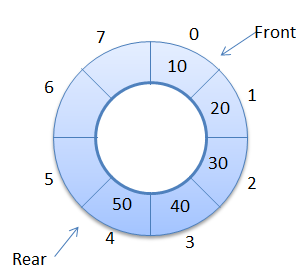
**Simple queue**

In Simple queue Insertion occurs at the rear of the list, and deletion occurs at the front of the list.



**Circular queue**

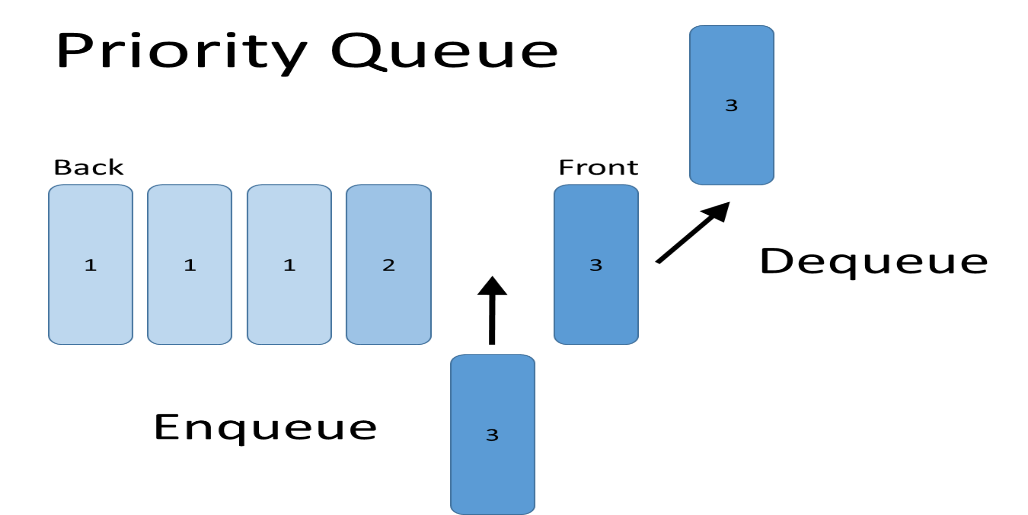
A circular queue is a queue in which all nodes are treated as circular such that the first node follows the last node.



**Priority queue**

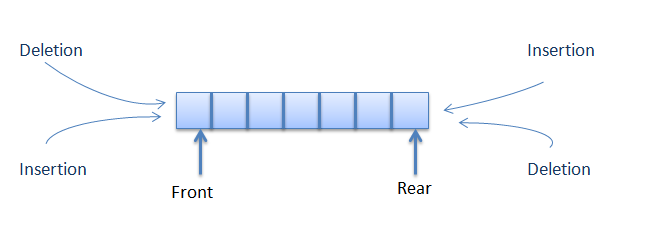
A priority queue is a collection of items in which each item is assigned a priority. The item having higher priority is processed prior to the item having lower priority

If more than one item has the same priority they are processed in the order in which they are inserted in the queue



**Double ended queue**

In dequeue (double ended queue) Insertion and Deletion occur at both the ends front and rear of the queue.

****

**Queue Using Array**

Queue implemented using array can store only fixed number of data values. 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 increment 'front' value by one and then display the value at 'front' position as deleted element.

**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**

The enQueue() function takes one integer value as 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**

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 is equal to **rear** (**i** <= **rear**)

**C Program to implement Queue using Array**

#include<stdio.h>

#include<conio.h>

#define SIZE 10

void enQueue(int);

void deQueue();

void display();

int queue[SIZE], front = -1, rear = -1;

void main()

{

int value, choice;

clrscr();

while(1){

printf("\n\n\*\*\*\*\* MENU \*\*\*\*\*\n");

printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d",&value);

enQueue(value);

break;

case 2: deQueue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void enQueue(int value){

if(rear == SIZE-1)

printf("\nQueue is Full!!! Insertion is not possible!!!");

else{

if(front == -1)

front = 0;

rear++;

queue[rear] = value;

printf("\nInsertion success!!!");

}

}

void deQueue(){

if(front == rear)

printf("\nQueue is Empty!!! Deletion is not possible!!!");

else{

printf("\nDeleted : %d", queue[front]);

front++;

if(front == rear)

front = rear = -1;

}

}

void display(){

if(rear == -1)

printf("\nQueue is Empty!!!");

else{

int i;

printf("\nQueue elements are:\n");

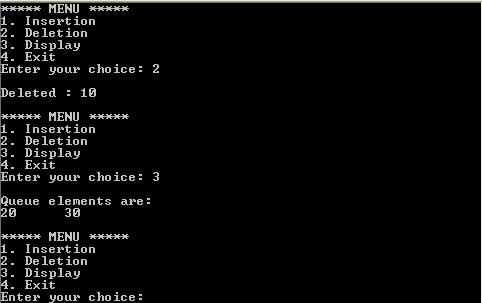
for(i=front; i<=rear; i++)

printf("%d\t",queue[i]);

}

}

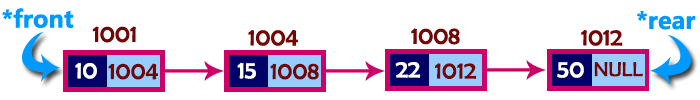
**Output**



**Queue using Linked List**

Queue using array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using linked list data structure. The queue which is implemented using linked list can work for unlimited number of values. That means, queue using linked list can work for variable size of data (No need to fix the size at beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want.   
In linked list implementation of a queue, the last inserted node is always pointed by 'rear' and the first node is always pointed by 'front'.

**Example**



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

**Operations**

To implement queue using linked list, we need to set the following things before implementing actual operations.

Step 1: Include all the **header files** which are used in the program. And declare all the **user defined functions**.

Step 2: Define a '**Node**' structure with two members **data** and **next**.

Step 3: Define two **Node** pointers '**front**' and '**rear**' and set both to **NULL**.

Step 4: Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

**enQueue(value) - Inserting an element into the Queue**

We can use the following steps to insert a new node into the queue...

Step 1: Create a newNode with given value and set 'newNode → next' to NULL.

Step 2: Check whether queue is Empty (rear == NULL)

Step 3: If it is Empty then, set front = newNode and rear = newNode.

Step 4: If it is Not Empty then, set rear → next = newNode and rear = newNode.

**deQueue() - Deleting an Element from Queue**

We can use the following steps to delete a node from the queue...

Step 1: Check whether queue is Empty (front == NULL).

Step 2: If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function

Step 3: If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.

Step 4: Then set 'front = front → next' and delete 'temp' (free(temp)).

**display() - Displaying the elements of Queue**

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

Step 1: Check whether queue is Empty (front == NULL).

Step 2: If it is Empty then, display 'Queue is Empty!!!' and terminate the function.

Step 3: If it is Not Empty then, define a Node pointer 'temp' and initialize with front.

Step 4: Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next !=NULL).

Step 4: Finally! Display 'temp → data ---> NULL'.

**C Program for Queue Using Linked List**

#include<stdio.h>

#include<conio.h>

struct Node

{

int data;

struct Node \*next;

}\*front = NULL,\*rear = NULL;

void insert(int);

void delete();

void display();

void main()

{

int choice, value;

clrscr();

printf("\n:: Queue Implementation using Linked List ::\n");

while(1){

printf("\n\*\*\*\*\*\* MENU \*\*\*\*\*\*\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("Enter the value to be insert: ");

scanf("%d", &value);

insert(value);

break;

case 2: delete(); break;

case 3: display(); break;

case 4: exit(0);

default: printf("\nWrong selection!!! Please try again!!!\n");

}

}

}

void insert(int value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode -> next = NULL;

if(front == NULL)

front = rear = newNode;

else{

rear -> next = newNode;

rear = newNode;

}

printf("\nInsertion is Success!!!\n");

}

void delete()

{

if(front == NULL)

printf("\nQueue is Empty!!!\n");

else{

struct Node \*temp = front;

front = front -> next;

printf("\nDeleted element: %d\n", temp->data);

free(temp);

}

}

void display()

{

if(front == NULL)

printf("\nQueue is Empty!!!\n");

else{

struct Node \*temp = front;

while(temp->next != NULL){

printf("%d--->",temp->data);

temp = temp -> next;

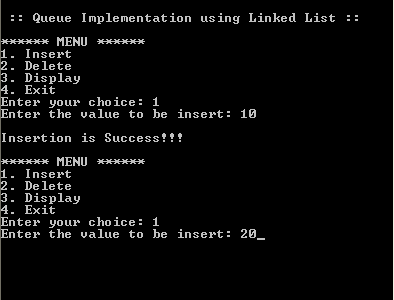
}

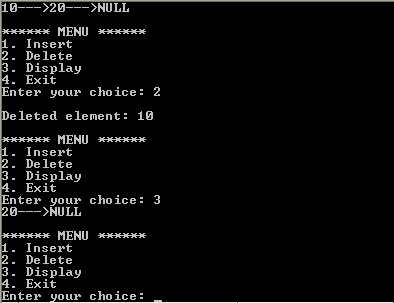
printf("%d--->NULL\n",temp->data);

}

}

**Output**



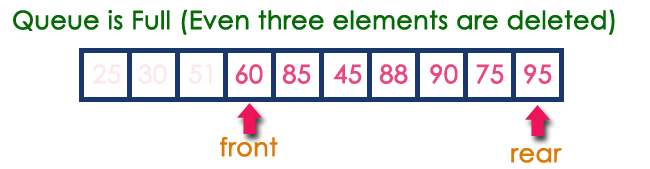


**Circular Queue**

In a normal Queue Data Structure, we can insert elements until queue becomes full. But once if queue becomes full, we can not insert the next element until all the elements are deleted from the queue. For example consider the queue below...  
  
After inserting all the elements into the queue.



Now consider the following situation after deleting three elements from the queue...



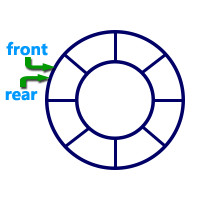
This situation also says that Queue is Full and we can not insert the new element because, '**rear**' is still at last position. In above situation, even though we have empty positions in the queue we can not make use of them to insert new element. This is the major problem in normal queue data structure. To overcome this problem we use circular queue data structure.

**What is Circular Queue?**

A Circular Queue can be defined as follows...

**Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.**

Graphical representation of a circular queue is as follows...



To implement a circular queue data structure using array, we first perform the following steps before we implement actual operations.

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 **user defined functions** used in circular queue implementation.

Step 3: Create a one dimensional array with above defined SIZE (**int cQueue[SIZE]**)

Step 4: Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)

Step 5: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

**enQueue(value) - Inserting value into the Circular Queue**

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

Step 1: Check whether queue is FULL. ((rear == SIZE-1 && front == 0) || (front == rear+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 check rear == SIZE - 1 && front != 0 if it is TRUE, then set rear = -1.

Step 4: Increment rear value by one (rear++), set queue[rear] = value and check 'front == -1' if it is TRUE, then set front = 0.

**deQueue() - Deleting a value from the Circular Queue**

In a circular queue, the element is always deleted from **front** position. The deQueue() function doesn't take any value as parameter. We can use the following steps to delete an element from the circular queue...

Step 1: Check whether queue is EMPTY. (front == -1 && rear == -1)

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 display queue[front] as deleted element and increment the front value by one (front ++). Then check whether front == SIZE, if it is TRUE, then set front = 0. Then check whether both front - 1 and rear are equal (front -1 ==rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

**display() - Displays the elements of a Circular Queue**

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

Step 1: Check whether queue is EMPTY. (front == -1)

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'.

Step 4: Check whether 'front <= rear', if it is TRUE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.

Step 5: If 'front <= rear' is FALSE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until'i <= SIZE - 1' becomes FALSE.

Step 6: Set i to 0.

Step 7: Again display 'cQueue[i]' value and increment i value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.

**Program to implement Queue using Array**

#include<stdio.h>

#include<conio.h>

#define SIZE 5

void enQueue(int);

void deQueue();

void display();

int cQueue[SIZE], front = -1, rear = -1;

void main()

{

int choice, value;

clrscr();

while(1){

printf("\n\*\*\*\*\*\* MENU \*\*\*\*\*\*\n");

printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: printf("\nEnter the value to be insert: ");

scanf("%d",&value);

enQueue(value);

break;

case 2: deQueue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nPlease select the correct choice!!!\n");

}

}

}

void enQueue(int value)

{

if((front == 0 && rear == SIZE - 1) || (front == rear+1))

printf("\nCircular Queue is Full! Insertion not possible!!!\n");

else{

if(rear == SIZE-1 && front != 0)

rear = -1;

cQueue[++rear] = value;

printf("\nInsertion Success!!!\n");

if(front == -1)

front = 0;

}

}

void deQueue()

{

if(front == -1 && rear == -1)

printf("\nCircular Queue is Empty! Deletion is not possible!!!\n");

else{

printf("\nDeleted element : %d\n",cQueue[front++]);

if(front == SIZE)

front = 0;

if(front-1 == rear)

front = rear = -1;

}

}

void display()

{

if(front == -1)

printf("\nCircular Queue is Empty!!!\n");

else{

int i = front;

printf("\nCircular Queue Elements are : \n");

if(front <= rear){

while(i <= rear)

printf("%d\t",cQueue[i++]);

}

else{

while(i <= SIZE - 1)

printf("%d\t", cQueue[i++]);

i = 0;

while(i <= rear)

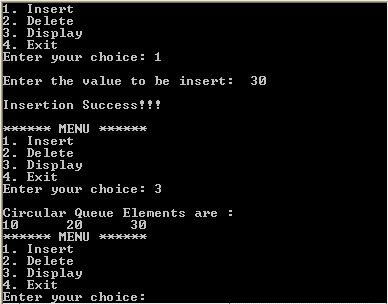
printf("%d\t",cQueue[i++]);

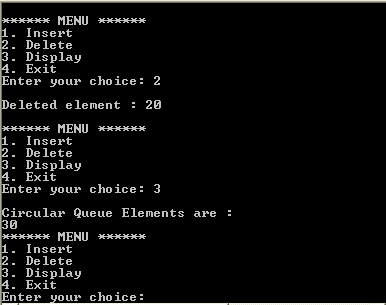
}

}

}

**Output**





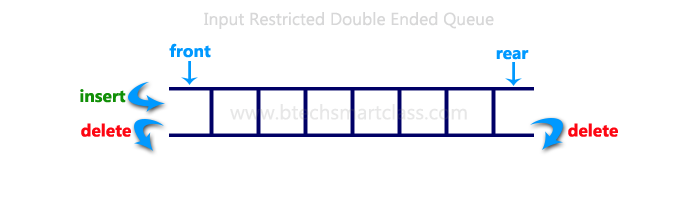
**Double Ended Queue**  
Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (**front** and **rear**). That means, we can insert at both front and rear positions and can delete from both front and rear positions.



Double Ended Queue can be represented in TWO ways, those are as follows...

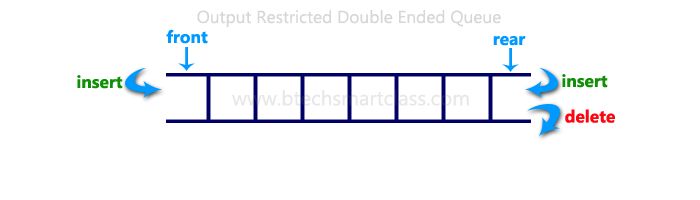
* Input Restricted Double Ended Queue
* Output Restricted Double Ended Queue

**Input Restricted Double Ended Queue**

In input restricted double ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.

**Output Restricted Double Ended Queue**

In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.



**Program to implement Double Ended Queue**

#include<stdio.h>

#include<conio.h>

#define SIZE 100

void enQueue(int);

int deQueueFront();

int deQueueRear();

void enQueueRear(int);

void enQueueFront(int);

void display();

int queue[SIZE];

int rear = 0, front = 0;

int main()

{

char ch;

int choice1, choice2, value;

printf("\n\*\*\*\*\*\*\* Type of Double Ended Queue \*\*\*\*\*\*\*\n");

do

{

printf("\n1.Input-restricted deque \n");

printf("2.output-restricted deque \n");

printf("\nEnter your choice of Queue Type : ");

scanf("%d",&choice1);

switch(choice1)

{

case 1:

printf("\nSelect the Operation\n");

printf("1.Insert\n2.Delete from Rear\n3.Delete from Front\n4. Display");

do

{

printf("\nEnter your choice for the operation in c deque: ");

scanf("%d",&choice2);

switch(choice2)

{

case 1: enQueueRear(value);

display();

break;

case 2: value = deQueueRear();

printf("\nThe value deleted is %d",value);

display();

break;

case 3: value=deQueueFront();

printf("\nThe value deleted is %d",value);

display();

break;

case 4: display();

break;

default:printf("Wrong choice");

}

printf("\nDo you want to perform another operation (Y/N): ");

ch=getch();

}while(ch=='y'||ch=='Y');

getch();

break;

case 2 :

printf("\n---- Select the Operation ----\n");

printf("1. Insert at Rear\n2. Insert at Front\n3. Delete\n4. Display");

do

{

printf("\nEnter your choice for the operation: ");

scanf("%d",&choice2);

switch(choice2)

{

case 1: enQueueRear(value);

display();

break;

case 2: enQueueFront(value);

display();

break;

case 3: value = deQueueFront();

printf("\nThe value deleted is %d",value);

display();

break;

case 4: display();

break;

default:printf("Wrong choice");

}

printf("\nDo you want to perform another operation (Y/N): ");

ch=getch();

} while(ch=='y'||ch=='Y');

getch();

break ;

}

printf("\nDo you want to continue(y/n):");

ch=getch();

}while(ch=='y'||ch=='Y');

}

void enQueueRear(int value)

{

char ch;

if(front == SIZE/2)

{

printf("\nQueue is full!!! Insertion is not possible!!! ");

return;

}

do

{

printf("\nEnter the value to be inserted:");

scanf("%d",&value);

queue[front] = value;

front++;

printf("Do you want to continue insertion Y/N");

ch=getch();

}while(ch=='y');

}

void enQueueFront(int value)

{

char ch;

if(front==SIZE/2)

{

printf("\nQueue is full!!! Insertion is not possible!!!");

return;

}

do

{

printf("\nEnter the value to be inserted:");

scanf("%d",&value);

rear--;

queue[rear] = value;

printf("Do you want to continue insertion Y/N");

ch = getch();

}

while(ch == 'y');

}

int deQueueRear()

{

int deleted;

if(front == rear)

{

printf("\nQueue is Empty!!! Deletion is not possible!!!");

return 0;

}

front--;

deleted = queue[front+1];

return deleted;

}

int deQueueFront()

{

int deleted;

if(front == rear)

{

printf("\nQueue is Empty!!! Deletion is not possible!!!");

return 0;

}

rear++;

deleted = queue[rear-1];

return deleted;

}

void display()

{

int i;

if(front == rear)

printf("\nQueue is Empty!!! Deletion is not possible!!!")

else{

printf("\nThe Queue elements are:");

for(i=rear; i < front; i++)

{

printf("%d\t ",queue[i]);

}

}

}