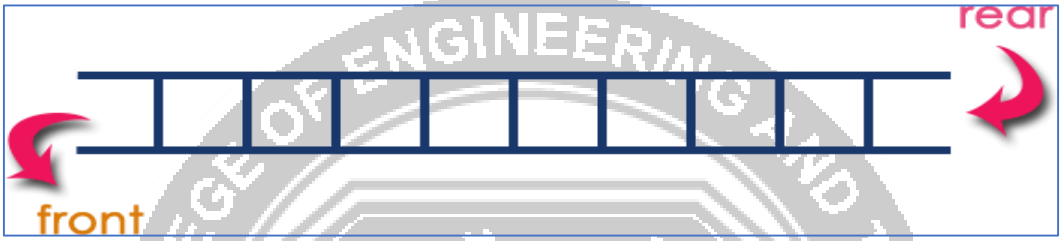
QUEUE

Queue data structure is a linear data structure in which the operations are performed based on FIFO principle. In a queue data structure, adding and removing of elements are performed at two different positions. 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 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 using array can be implemented as follows

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

Step1: 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.

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 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.

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 frontposition. 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).

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)

Program:

```
#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

        {

        inti;
```

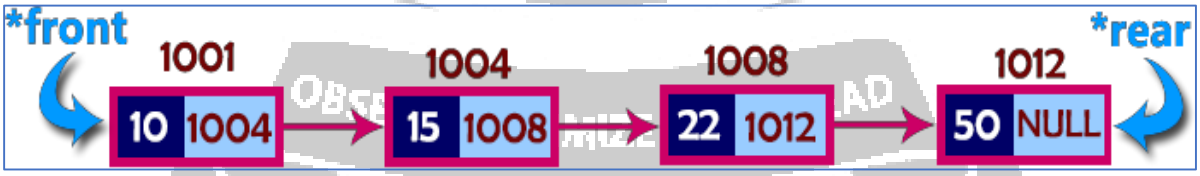
```
        printf("\nQueue elements are:\n");
        for(i=front; i<=rear; i++)
        printf("%d\t",queue[i]);
        }
}
```

QUEUE USING LINKED LIST

The major problem with the queue implemented using array is, It will work for only fixed number of data. That means, the amount of data must be specified in the beginning itself. 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.

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'.

Program:

```
#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);

        }

}
```