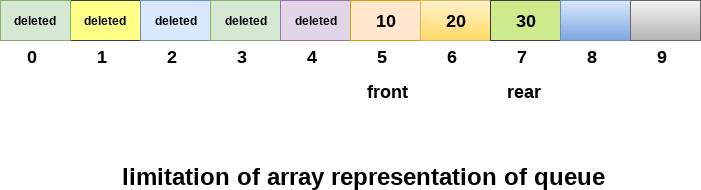
**Drawbacks of Array Implementation of queue**

Although, the technique of creating a queue is easy, but there are some drawbacks of using this technique to implement a queue.

* **Memory wastage:** The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.



The above figure shows how the memory space is wasted in the array representation of queue. In the above figure, a queue of size 10 having 3 elements, is shown. The value of the front variable is 5, therefore, we cannot reinsert the values in the place of already deleted element before the position of front. That much space of the array is wasted and cannot be used in the future (for this queue).

* **Deciding the array size**

One of the most common problems with array implementation is the size of the array which requires to be declared in advance. Due to the fact that, the queue can be extended at runtime depending upon the problem, the extension in the array size is a time taking process and almost impossible to be performed at runtime since a lot of reallocations take place. Due to this reason, we can declare the array large enough so that we can store queue elements as enough as possible but the main problem with this declaration is that, most of the array slots (nearly half) can never be reused. It will again lead to memory wastage.

Due to these drawbacks associated with array implementation of queue, thus the array implementation cannot be used for the large-scale applications where the queues are implemented. One of the alternatives of array implementation is linked list implementation of queue.

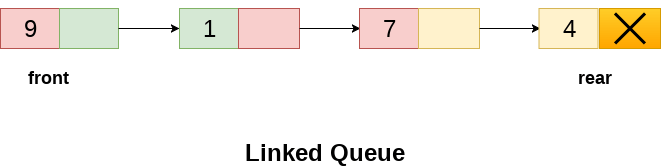
The storage requirement of linked representation of a queue with n elements is o(n) while the time requirement for operations is o (1).

In a linked queue, each node of the queue consists of two parts i.e., data part and the link part. Each element of the queue points to its immediate next element in the memory.

In the linked queue, there are two pointers maintained in the memory i.e., front pointer and rear pointer. The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.

Insertion and deletions are performed at rear and front end respectively. If front and rear both are NULL, it indicates that the queue is empty.

The linked representation of queue is shown in the following figure.



**Operations on Linked Queue**

There are two basic operations which can be implemented on the linked queues. The operations are Insertion and Deletion.

**Insert operation**

The insert operation appends the queue by adding an element to the end of the queue. The new element will be the last element of the queue.

Firstly, allocate the memory for the new node ptr by using the following statement.

Ptr = (struct node \*) malloc (sizeof(struct node));

There can be the two scenarios of inserting this new node ptr into the linked queue.

In the **first scenario**, we insert element into an empty queue. In this case, the condition **front = NULL** becomes true. Now, the new element will be added as the only element of the queue and the next pointer of front and rear pointer both, will point to NULL.

ptr -> data = item;

**if**(front == NULL)

        {

            front = ptr;

            rear = ptr;

            front -> next = NULL;

            rear -> next = NULL;

       }

**In the second case**, the queue contains more than one element. The condition front = NULL becomes false. In this scenario, we need to update the end pointer rear so that the next pointer of rear will point to the new node ptr. Since, this is a linked queue, hence we also need to make the rear pointer point to the newly added node **ptr**. We also need to make the next pointer of rear point to NULL.

rear -> next = ptr;

            rear = ptr;

            rear->next = NULL;

In this way, the element is inserted into the queue. The algorithm and the C implementation is given as follows.

**Algorithm**

* **Step 1:** Allocate the space for the new node PTR
* **Step 2:** SET PTR -> DATA = VAL
* **Step 3:** IF FRONT = NULL

SET FRONT = REAR = PTR

SET FRONT -> NEXT = REAR -> NEXT = NULL

ELSE  
SET REAR -> NEXT = PTR

SET REAR = PTR

SET REAR -> NEXT = NULL

[END OF IF]

* **Step 4:** END

**C-Function**

**void** insert(struct node \*ptr, **int** item; )

{

    ptr = (struct node \*) malloc (sizeof(struct node));

**if**(ptr == NULL)

    {

        printf("\nOVERFLOW\n");

**return**;

    }

**else**

    {

        ptr -> data = item;

**if**(front == NULL)

        {

           front = ptr;

           rear = ptr;

            front -> next = NULL;

            rear -> next = NULL;

        }

**else**

        {

            rear -> next = ptr;

            rear = ptr;

            rear->next = NULL;

        }

    }

}

## Deletion

Deletion operation removes the element that is first inserted among all the queue elements. Firstly, we need to check either the list is empty or not. The condition front == NULL becomes true if the list is empty, in this case, we simply write underflow on the console and make exit.

Otherwise, we will delete the element that is pointed by the pointer front. For this purpose, copy the node pointed by the front pointer into the pointer ptr. Now, shift the front pointer, point to its next node and free the node pointed by the node ptr. This is done by using the following statements.

ptr = front;

        front = front -> next;

        free(ptr);

The algorithm and C function is given as follows.

**Algorithm**

* **Step 1:** IF FRONT = NULL

Write " Underflow "

Go to Step 5

[END OF IF]

* **Step 2:** SET PTR = FRONT
* **Step 3:** SET FRONT = FRONT -> NEXT
* **Step 4:** FREE PTR
* **Step 5:** END

**C-Function**

**void** delete (struct node \*ptr)

{

**if**(front == NULL)

    {

        printf("\nUNDERFLOW\n");

**return**;

    }

**else**

    {

        ptr = front;

        front = front -> next;

        free(ptr);

    }

}

**Implementation of Queue using Linked List**

#include<stdio.h>

#include<stdlib.h>

struct node

{

**int** data;

    struct node \*next;

};

struct node \*front;

struct node \*rear;

**void** insert();

**void** delete();

**void** display();

**void** main ()

{

**int** choice;

**while**(choice != 4)

    {

        printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

        printf("\n=================================================================\n");

   printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

        printf("\nEnter your choice ?");

        scanf("%d",& choice);

**switch**(choice)

        {

**case** 1:

            insert();

**break**;

**case** 2:

            delete();

**break**;

**case** 3:

            display();

**break**;

**case** 4:

            exit(0);

**break**;

**default**:

            printf("\nEnter valid choice??\n");

        }

    }

}

**void** insert()

{

    struct node \*ptr;

**int** item;

    ptr = (struct node \*) malloc (sizeof(struct node));

**if**(ptr == NULL)

    {

        printf("\nOVERFLOW\n");

**return**;

    }

**else**

    {

        printf("\nEnter value?\n");

        scanf("%d",&item);

        ptr -> data = item;

**if**(front == NULL)

        {

            front = ptr;

            rear = ptr;

            front -> next = NULL;

            rear -> next = NULL;

        }

**else**

        {

            rear -> next = ptr;

            rear = ptr;

            rear->next = NULL;

        }

    }

}

**void** delete ()

{

    struct node \*ptr;

**if**(front == NULL)

    {

        printf("\nUNDERFLOW\n");

**return**;

    }

**else**

    {

        ptr = front;

        front = front -> next;

        free(ptr);

    }

}

**void** display()

{

    struct node \*ptr;

    ptr = front;

**if**(front == NULL)

    {

        printf("\nEmpty queue\n");

    }

**else**

    {   printf("\nprinting values .....\n");

**while**(ptr != NULL)

        {

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

            ptr = ptr -> next;

        }

    }

}

**Output:**

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

123

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

123

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?4