Queue

– a queue data structure

- Definition: Q*ueue* is a "first in first out" data structure. Which means elements inserted are popped in the same order.
- Thus what goes first will come out first.

- It is called a FIFO data structure: First-In, First-Out

  - » Insert A: insert A in the Queue
  - » Insert B: Insert B in the Queue

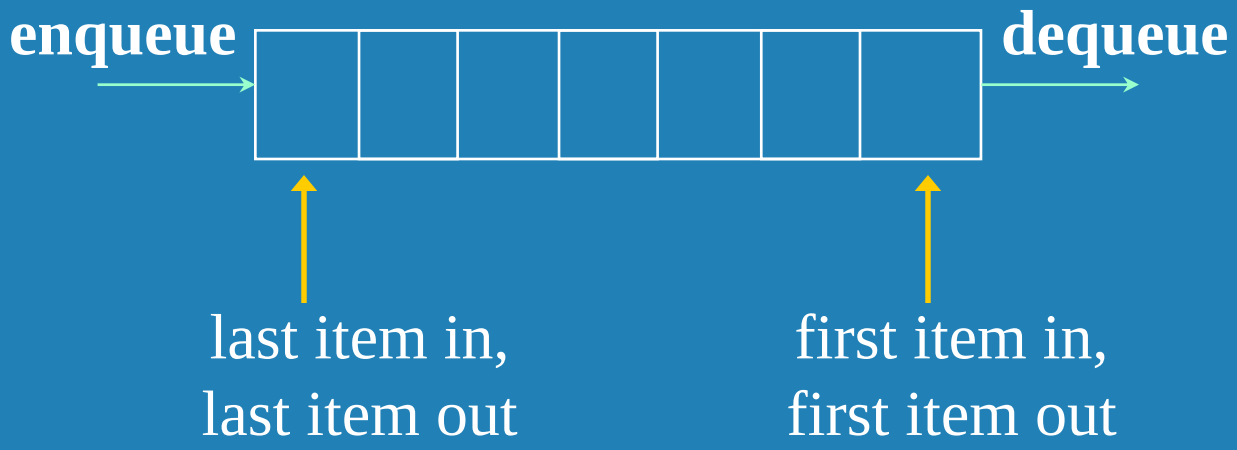
  enqueue

  - » Remove: Remove A
  - » Remove : Remove B

  dequeue

– a queue data structure

- A *queue* is similar to a list but adds items only to the rear of the list and removes them only from the front
- It is called a FIFO data structure: First-In, First-Out
- Analogy: a line of people at a movie ticket window



Queues

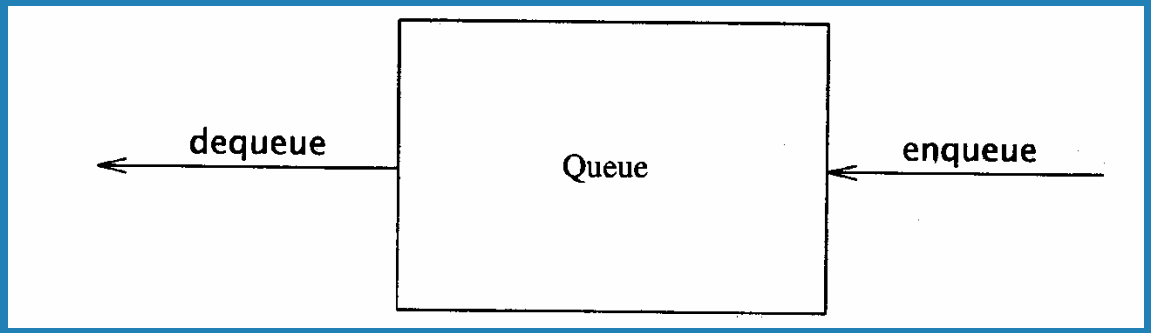
- We can define the operations for a queue
  - enqueue - add an item to the rear of the queue
  - dequeue (or serve) - remove an item from the front of the queue
  - empty - returns true if the queue is empty

- As with our linked list example, by storing generic `Object` references, any object can be stored in the queue

- Queues often are helpful in simulations or any situation in which items get "backed up" while awaiting processing
  - ✓ `(Jobs waiting their turn to be processed.)`
    - ✓ Like customers standing in a check-out line in a store, the first customer in is the first customer served.

The Queue ADT

- Like a stack, a *queue* is also a list. However, with a queue, insertion is done at one end, while deletion is performed at the other end.
- Another form of restricted list
  - Insertion is done at one end, whereas deletion is performed at the other end
- Basic operations:
  - enqueue: insert an element at the rear of the list
  - dequeue: delete the element at the front of the list

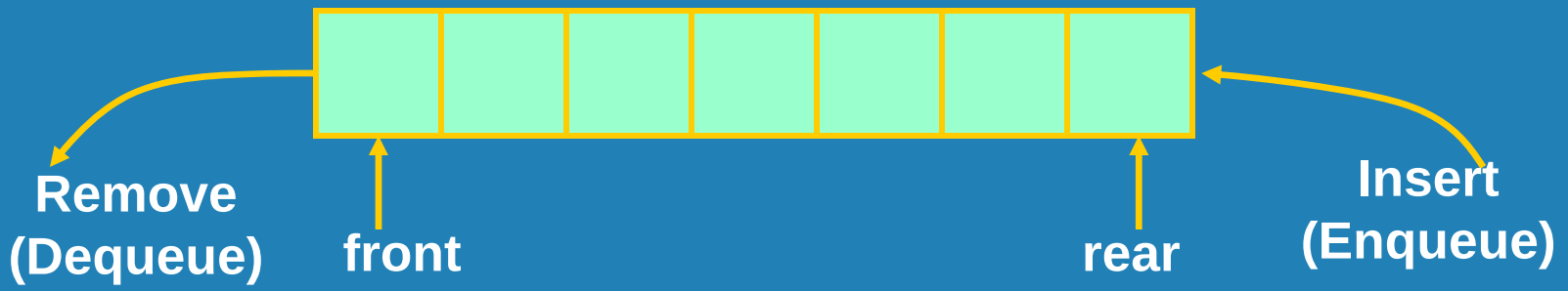


- First-in First-out (FIFO) list

Operation in the Queue

Enqueue and Dequeue

- Primary queue operations: Enqueue and Dequeue
- Like check-out lines in a store, a queue has a front and a rear.
- Enqueue
  - Insert an element at the rear of the queue
- Dequeue
  - Remove an element from the front of the queue



Operations on Queue

- isSize(): Return the number of elements in the queue at any time
- isempty(): Return a Boolean identification of the queue
- Empty (0,1) 1 means true there is no element in the Queue , 0 means false the Queue has elements
-
- Front(): return the front element of the Queue without removing it, if the queue is empty and error is return

Implementation of Queue

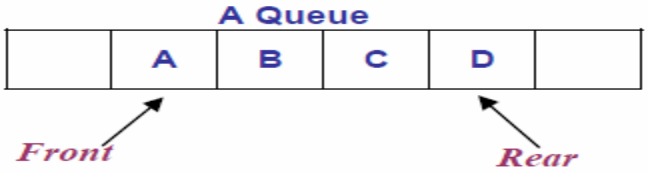
- Just as stacks can be implemented as arrays or linked lists, so with queues.
- (1) if it use array there are limited number of elements to be inserted
- (2) if it use linked list there in no such limited number
-
- Dynamic queues have the same advantages over static queues as dynamic stacks have over static stacks

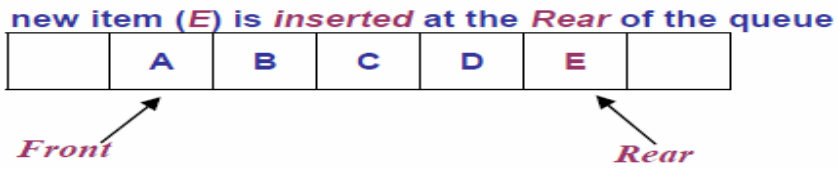
Outline

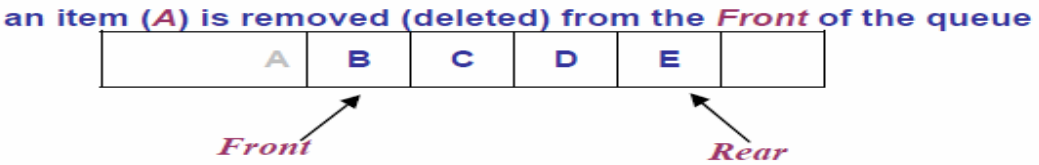
- Queues
  - Definition of Queue
  - Queue Operations
    - Insertion (Enqueue)
    - Removing (Dequeue)
  - Applications of the Queues

Definition of Queue

- A *queue* is an ordered collection of items from which items may be deleted at one end (*front* of the queue) and into which items may be inserted at the other end (*rear* of the queue).
- The first element inserted into the queue is the first element to be removed. For this reason a queue is sometimes called a *fifo* (first-in first-out) list as opposed to the stack, which is a *lifo* (last-in first-out).





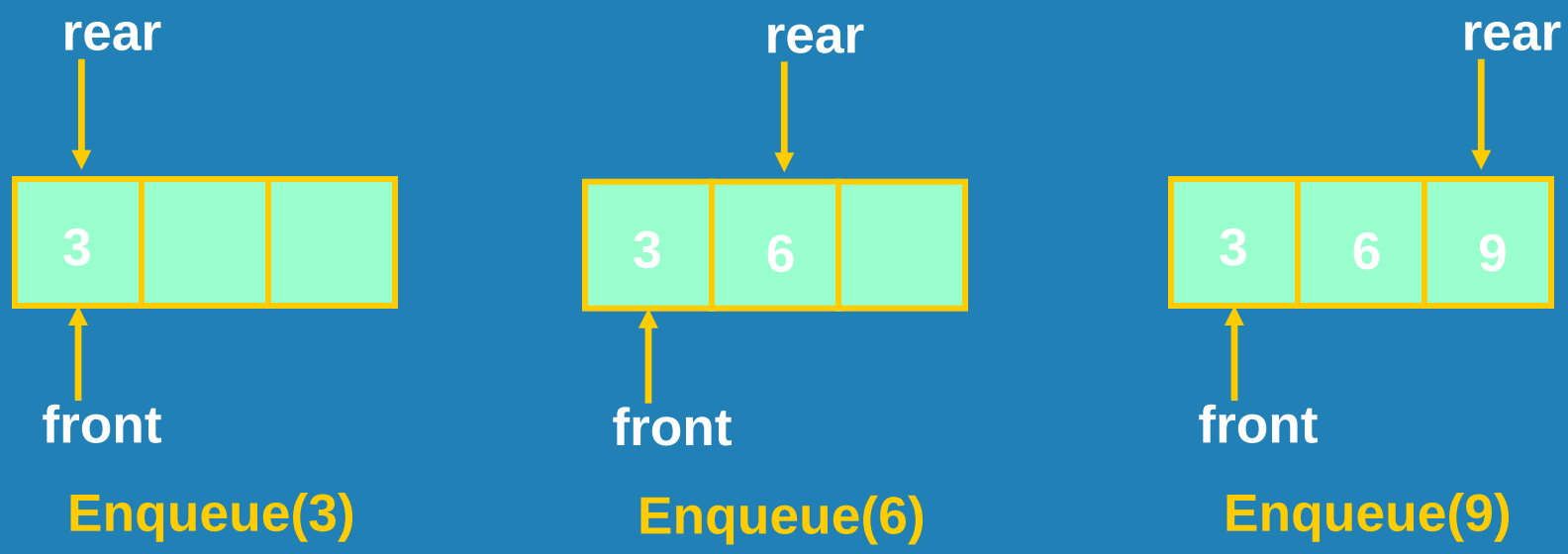


Queue Operations

- Initialize the queue
- *Insert* to the rear of the queue (also called as Enqueue)
- *Remove* (Delete) from the front of the queue (also called as Dequeue)
- Is the Queue Empty
- Is the Queue Full
- What is the size of the Queue

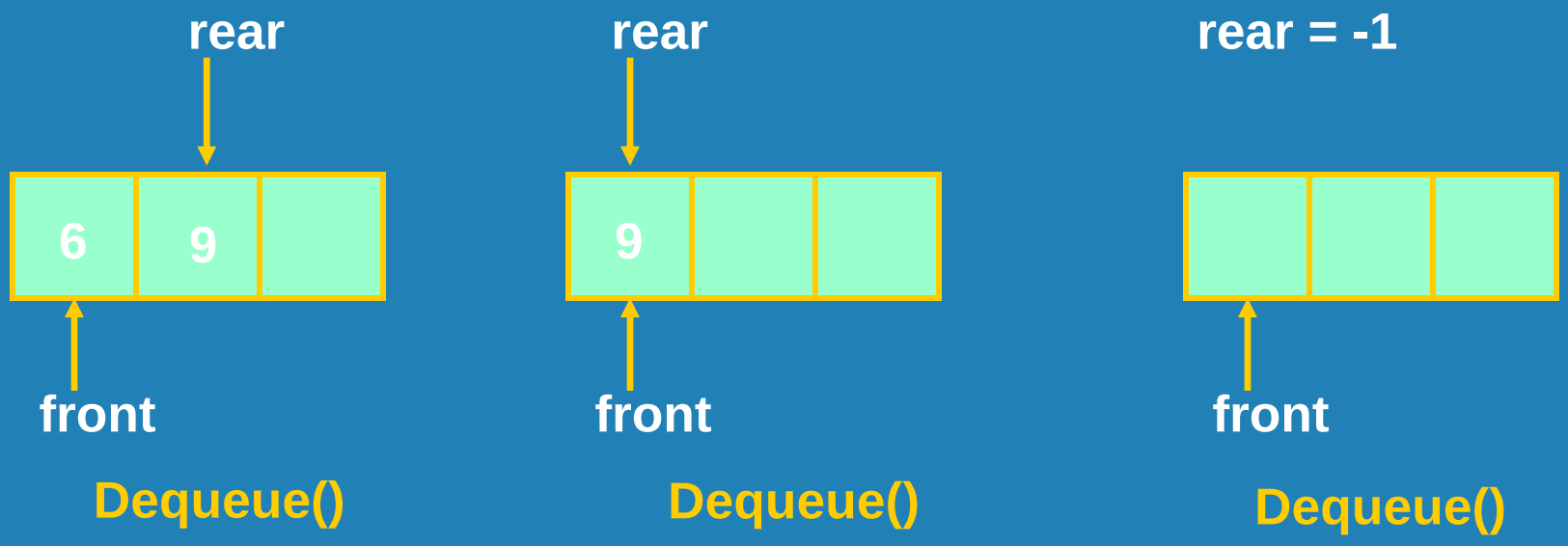
Queue Implementation of Array

- There are several different algorithms to implement Enqueue and Dequeue
- Naïve way
  - When enqueuing, the front index is always fixed and the rear index moves forward in the array.

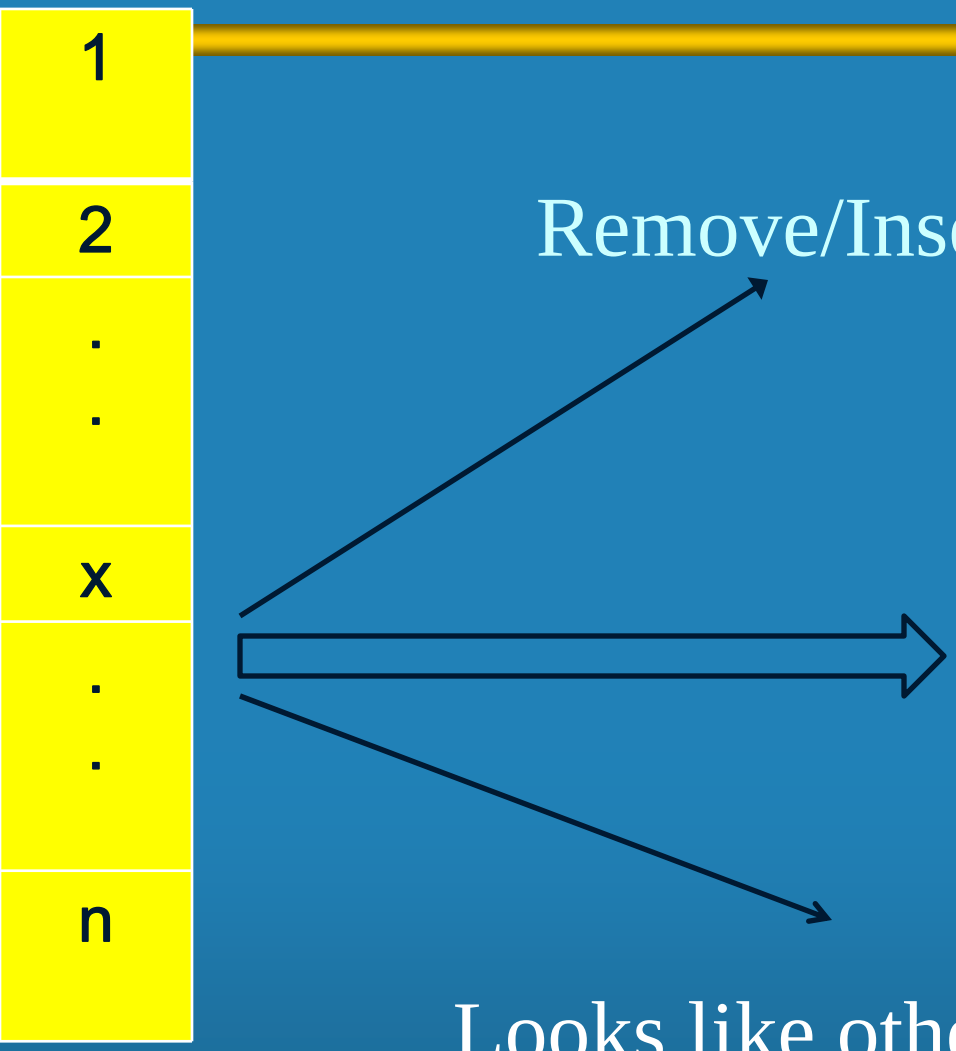


Queue Implementation of Array

- Naïve way
  - When enqueuing, the front index is always fixed and the rear index moves forward in the array.
  - When dequeuing, the element at the front the queue is removed. Move all the elements after it by one position. (Inefficient!!!)



Selective Removal Operation



Looks like other data structure as array

Queue Operations implemented by Array of size 3

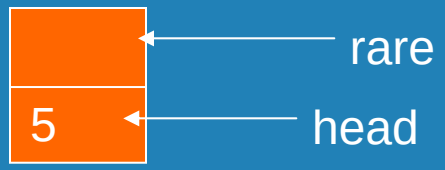
Size()=0

Isempty()=1

front()=Null

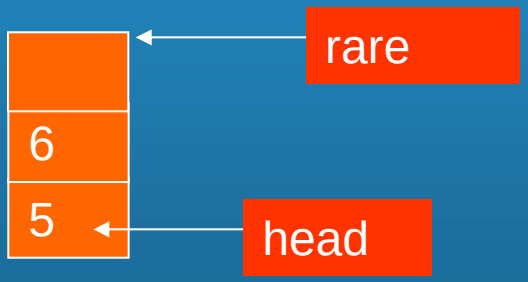
enqueue(5)

Size()=1
Isempty()=0
front()=5

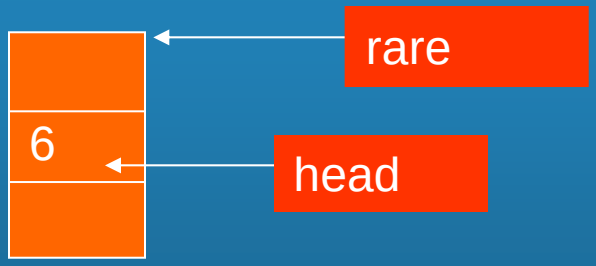


enqueue(6)

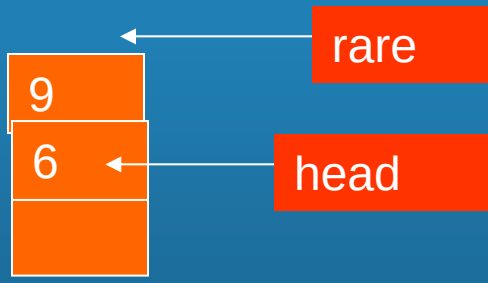
Size()=2
Isempty()=0
front()=6



Dequeue()

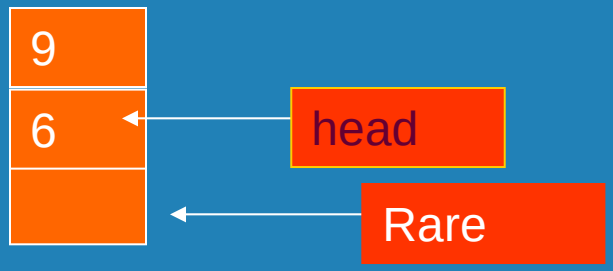


Size()=2
Isempty()=0
front()=9



Cont. Queue Operations

Size()=2
Isempty()=0
front()=9



Enqueue (2)

Now the queue is full there is one place is empty and the rare will not return error, And it will go to the first place again

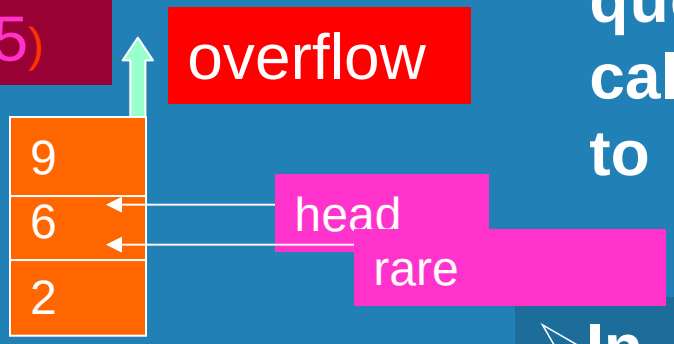
Size()=3
Isempty()=0
front()=2

- In that case rare and head point the same place as head and this is the limitation in the queue when using array.
- Now we can say the queue is empty when both rare and head point to the same place, but when the both are equal but the queue is full too, but the difference between both is when size()=0, the queue is empty and when size()=max, the queue is full.

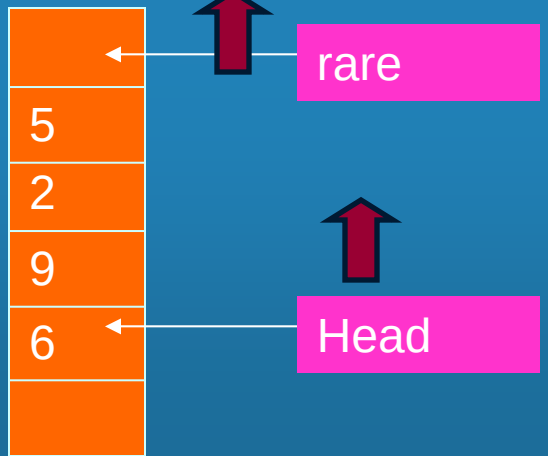
Cont. Queue Operations

Size()=3
Isempty()=0
front()=2

Enqueue(5)



Size()=4
Isempty()=0
front()=5



Notice that , if we want to add more element the size of the queue is MAX, and we have to call the Realloc() library function to increase the space of the array

- In that case call all element and the top element reserve the front
- The drawback of this method is the consuming time , and losing time and wastage of space

There is other way to implement the stack using Linked list

Queue Operations implemented by linked list

Size()=0

Isempty()=1

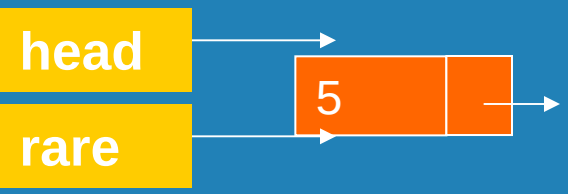
Front()=Head= Null

Size()=1

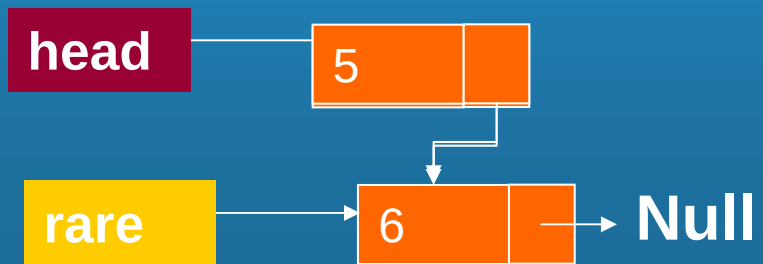
Isempty()=0

front()=5

enqueue(5)



enqueue6)



Size()=2

Isempty()=0

front()=6

- **Advantage:**
  - No wastage of space
  - No upper limit size
- **Disadvantage**
  - The cost of the node creation
  - Push() and Pop() take time as node creation and deletion take more list

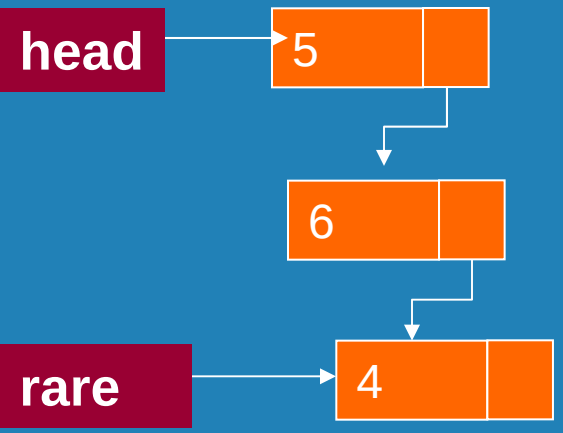
Stack Operations implemented by linked list

Size()=3

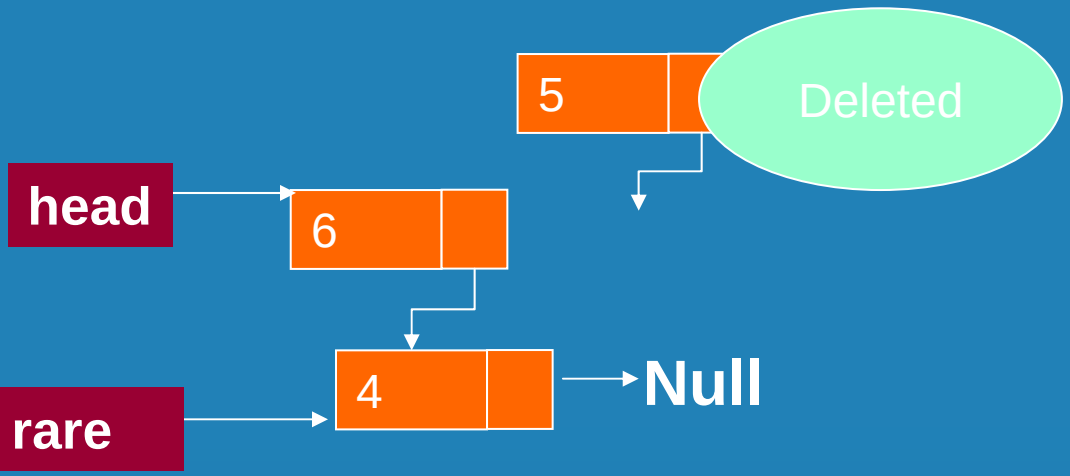
Isempty()=0

front()=4

Enqueue((4))



Deque()



Size()=2

Isempty()=0

front()=4

```
/******    Program to Implement Queue using Array    ******/

#include <stdio.h>

#define MAX 50

void insert();
void delet();
void display();

int queue[MAX], rear=-1, front=-1, item;

main()
{
     int ch;
     do
     {
          printf("\nEnter your choice: ");
          scanf("%d", &ch);

          switch(ch)
          {
               case 1:
                    insert();
                    break;

               case 2:
                    delet();
                    break;

               case 3:
                    display();
                    break;
```

```
                case 4:
                        exit(0);

                default:
                        printf("\n\nInvalid entry. Please try again...\n");
            }
        } while(1);
        getch();
}
 void insert()
 {
        if(rear == MAX-1)
                printf("\n\nQueue is full.");
        else
        {
                printf("\n\nEnter ITEM: ");
                scanf("%d", &item);

                if (rear == -1 && front == -1)
                {
                        rear = 0;
                        front = 0;
                }
                else
                        rear++;
```

```
                queue[rear] = item;
                printf("\n\nItem inserted: %d", item);
        }
}
```

```
void delet()
{
     if(front == -1)
          printf("\n\nQueue is empty.");
     else
     {
          item = queue[front];

          if (front == rear)
          {
               front = -1;
               rear = -1;
          }
          else
               front++;

          printf("\n\nItem deleted: %d", item);
                                                  }

               }
```

```
void display()
{
     int i;

     if(front == -1)
          printf("\n\nQueue is empty.");
     else
     {
          printf("\n\n");

          for(i=front; i<=rear; i++)
               printf("   %d", queue[i]);
     }
}
```