BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

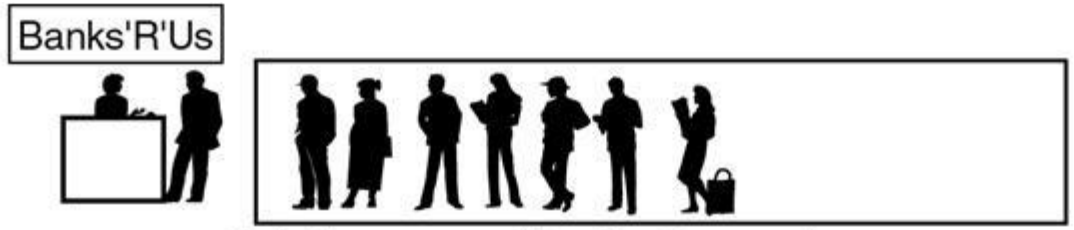
Programming and Computer Applications-2

Queues

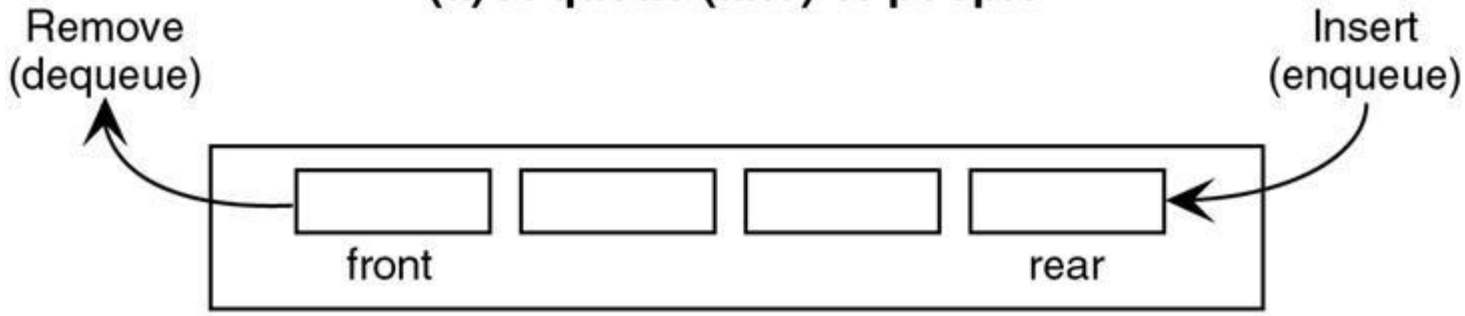
Instructor : PhD, Associate Professor Leyla Muradkhanli

Introduction to the Queue

- Queue: a FIFO (first in, first out) data structure.
- Examples:
  - people in line at the theatre box office
  - print jobs sent to a printer
- Implementation:
  - static: fixed size, implemented as array
  - dynamic: variable size, implemented as linked list



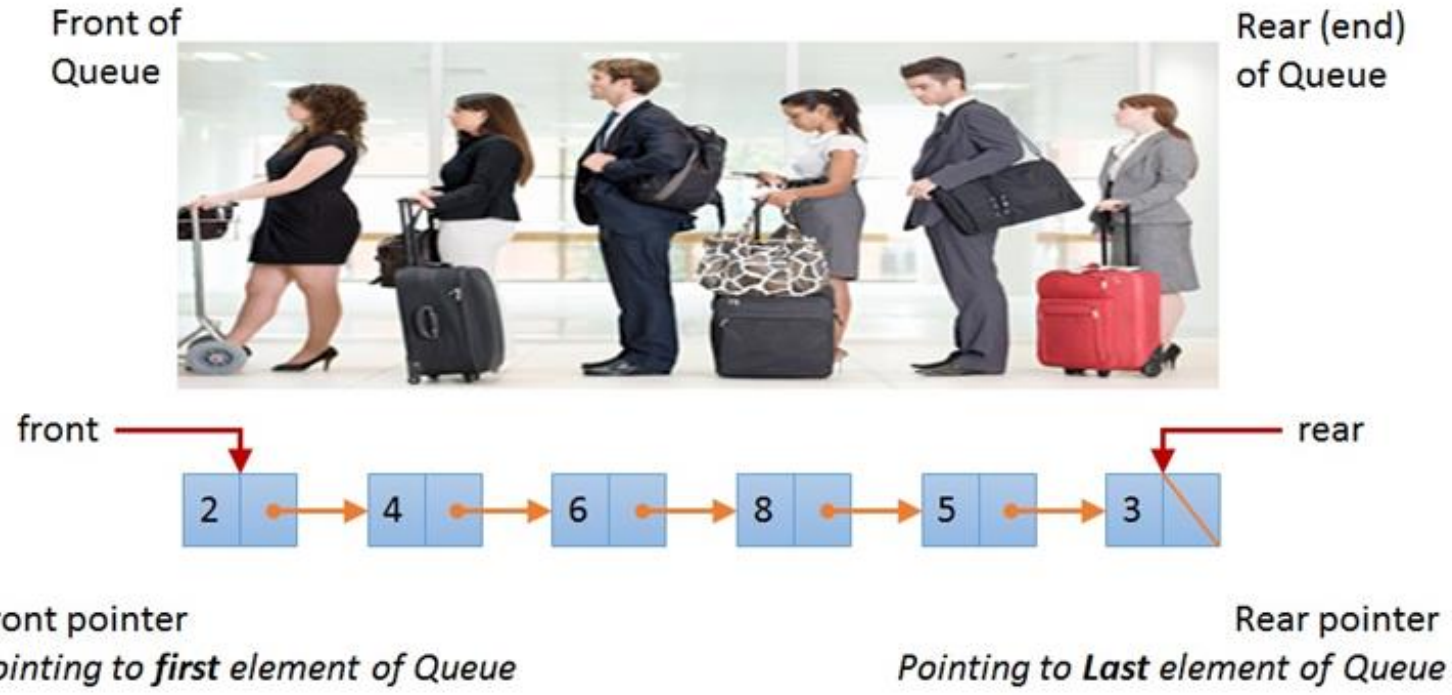
(a) A queue (line) of people



(b) A computer queue

Queue Locations and Operations

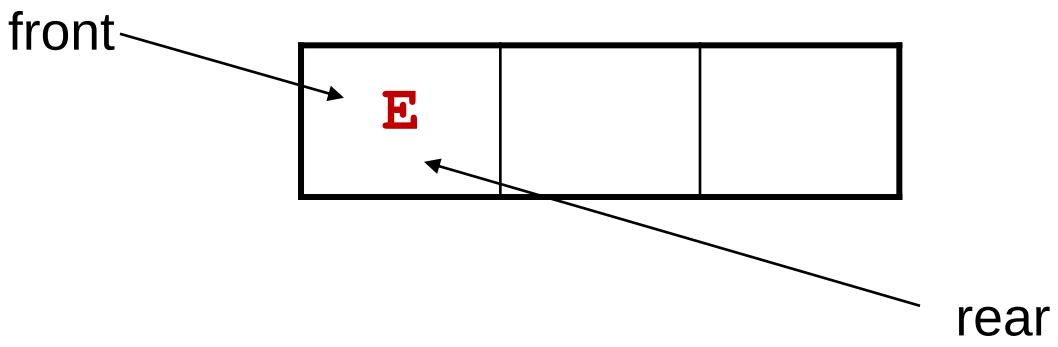
- **rear:** position where elements are added
- **front:** position from which elements are removed
- **enqueue:** add an element to the rear of the queue
- **dequeue:** remove an element from the front of a queue



Queue Operations - Example

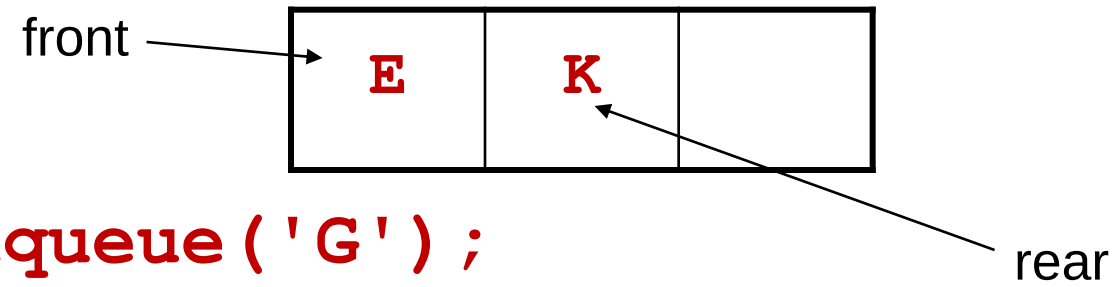
- A currently empty queue that can hold **`char`** values:

- **`enqueue('E');`**

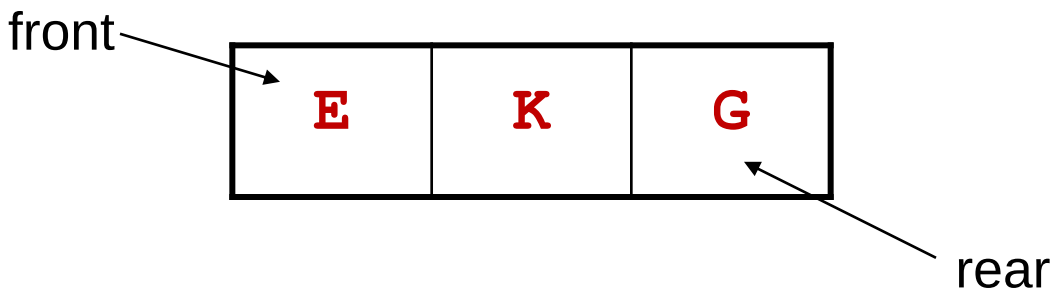


Queue Operations - Example

- `enqueue('K');`

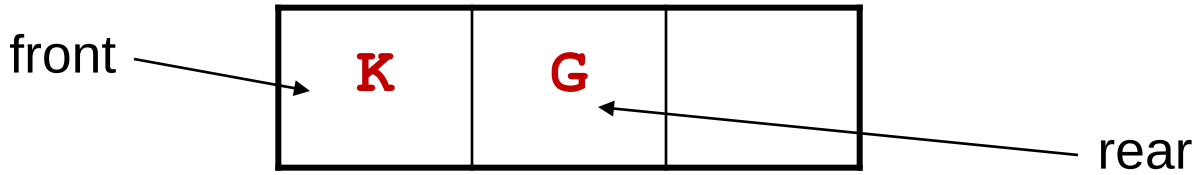


- `enqueue('G');`

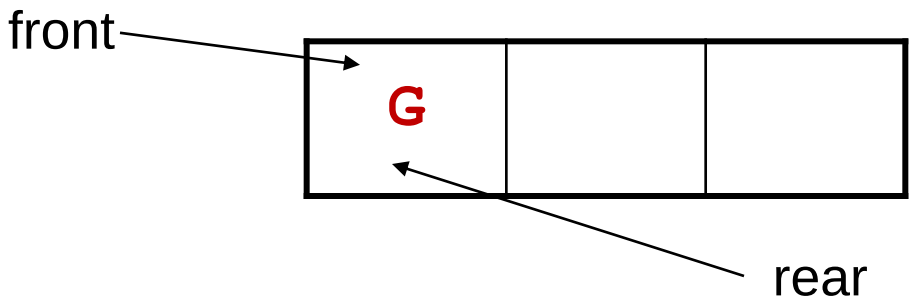


Queue Operations - Example

- `dequeue(); // remove E`



- `dequeue(); // remove K`



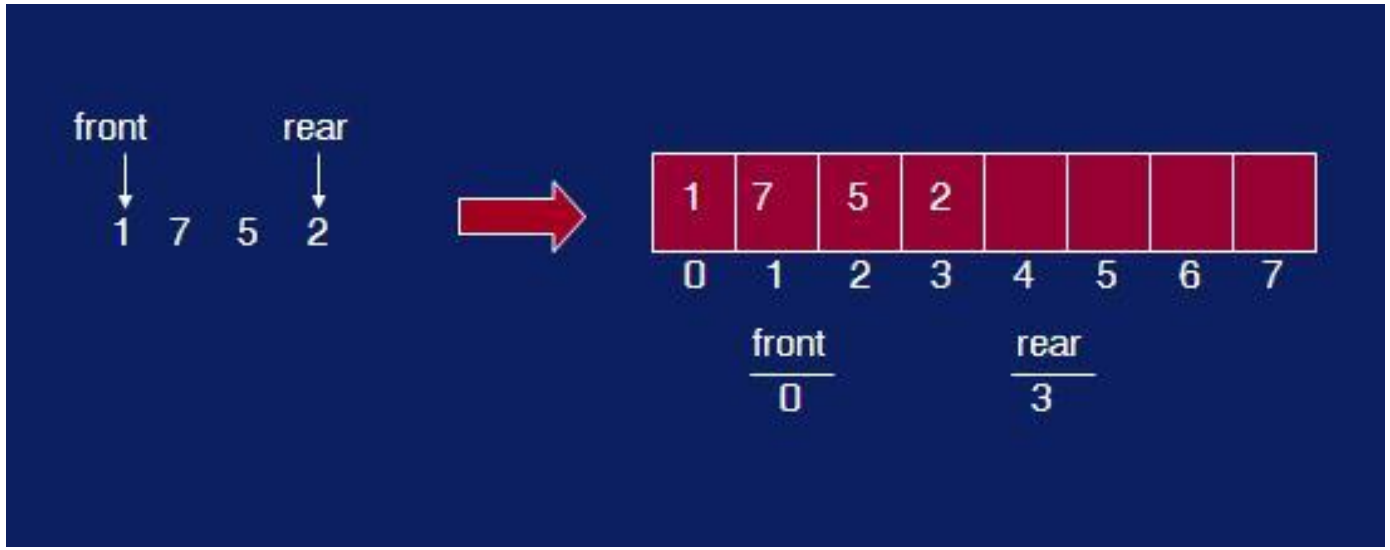
dequeue Issue, Solutions

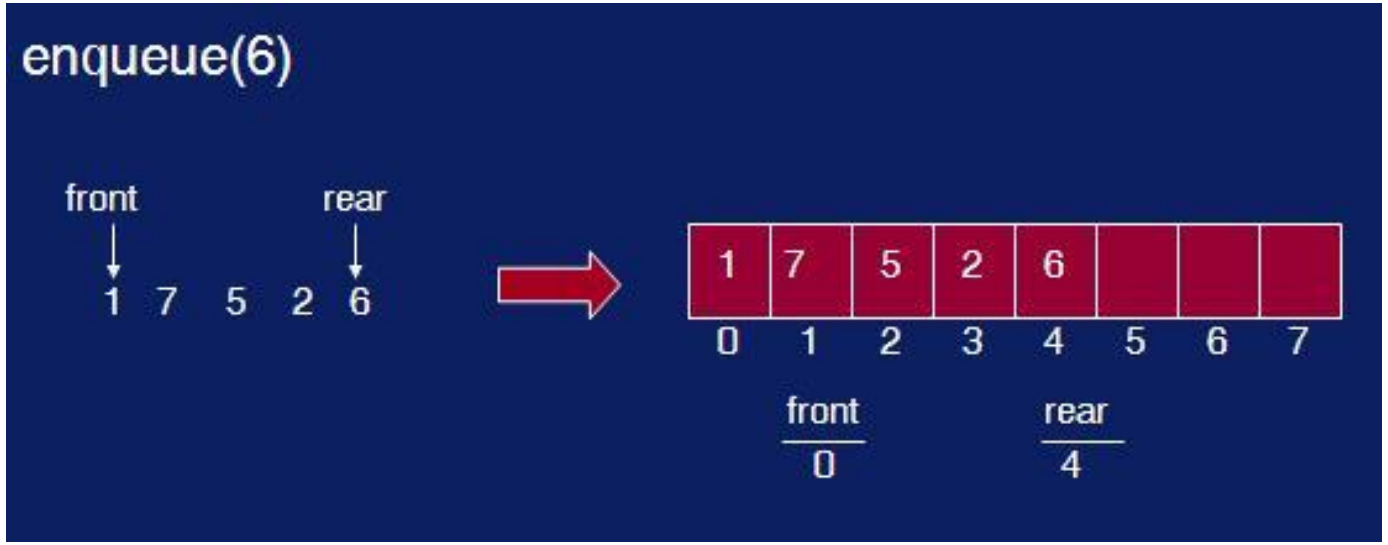
- When removing an element from a queue, remaining elements must shift to front
- **Solutions:**
  - Let front index move as elements are removed (works as long as rear index is not at end of array)
  - Use above solution, and also let rear index "wrap around" to front of array, treating array as circular instead of linear (more complex enqueue, dequeue code)

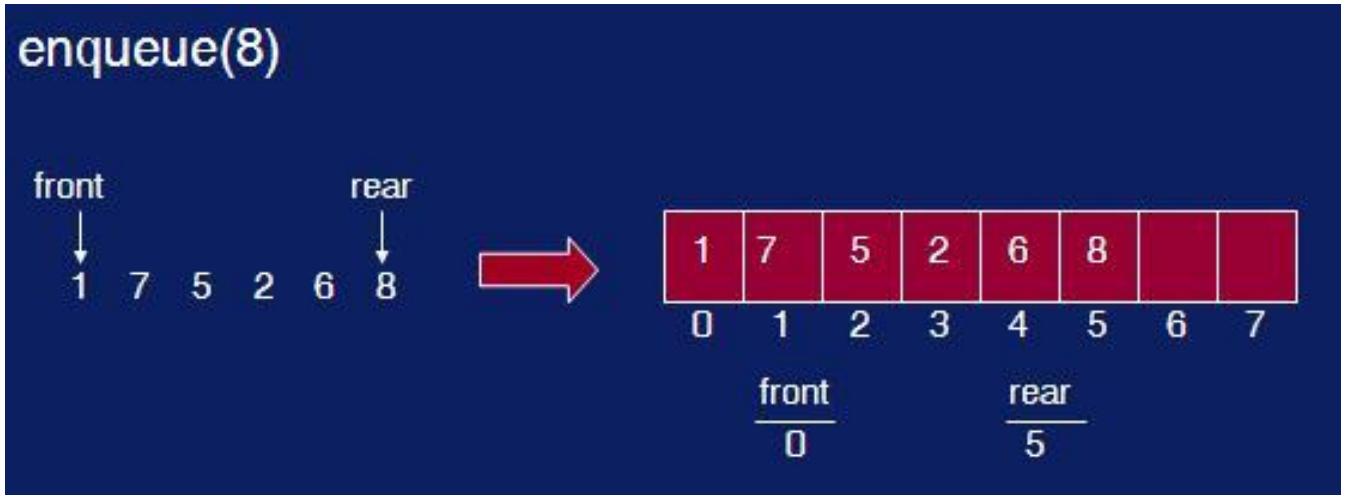
Implementation using Array

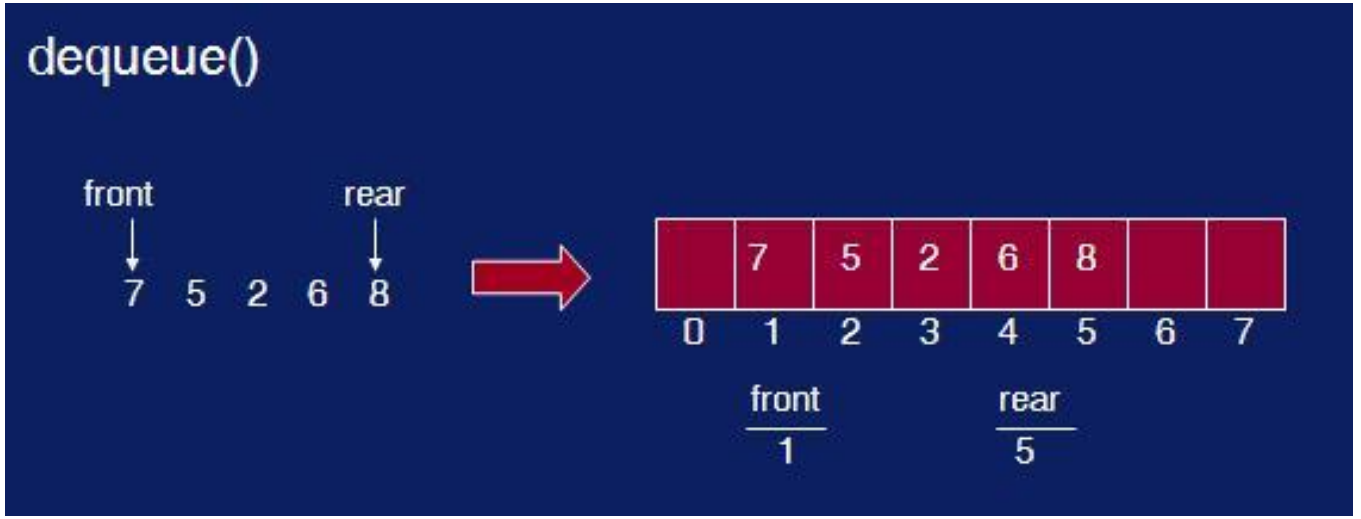
If we use an array to hold queue elements, both insertions and removal at the front (start) of the array are expensive. This is because we may have to shift up to “n” elements.

For the stack, we needed only one end; for queue we need both. To get around this, we will not shift upon removal of an element.

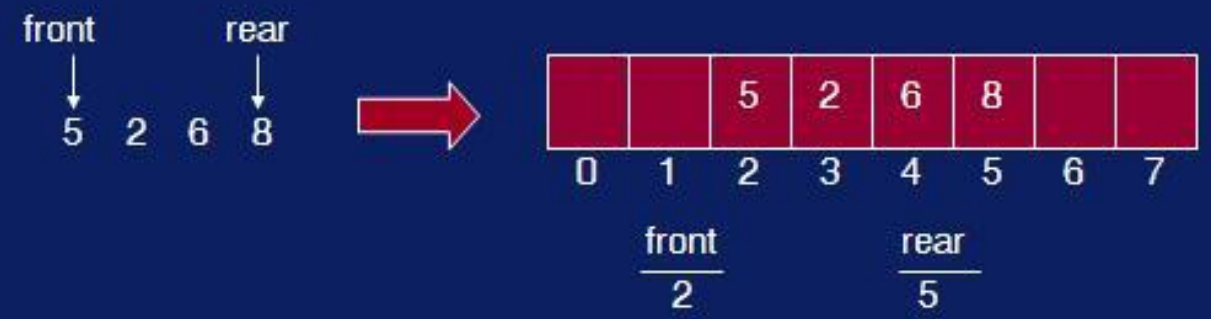

front
rear
1 7 5 2
1 7 5 2
0 1 2 3 4 5 6 7
front
0
rear
3


enqueue(6)
front
rear
1 7 5 2 6
1 7 5 2 6
0 1 2 3 4 5 6 7
front
0
rear
4

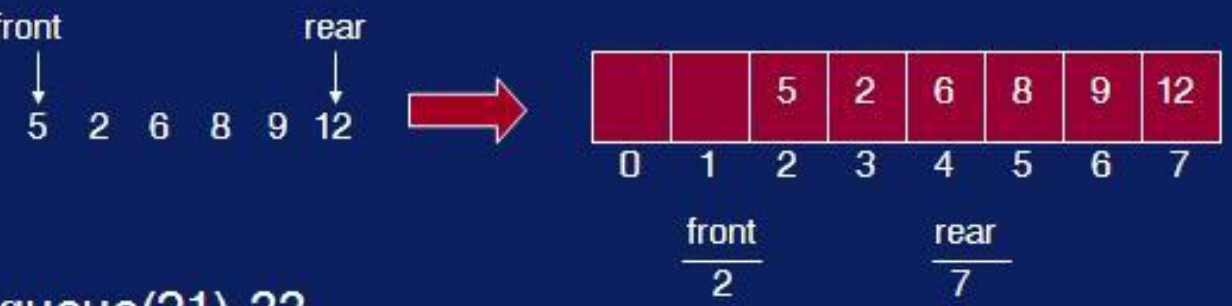

enqueue(8)
front
rear
1 7 5 2 6 8
1 7 5 2 6 8
0 1 2 3 4 5 6 7
front
0
rear
5


dequeue()
front
rear
7 5 2 6 8
7 5 2 6 8
0 1 2 3 4 5 6 7
front
1
rear
5

dequeue()

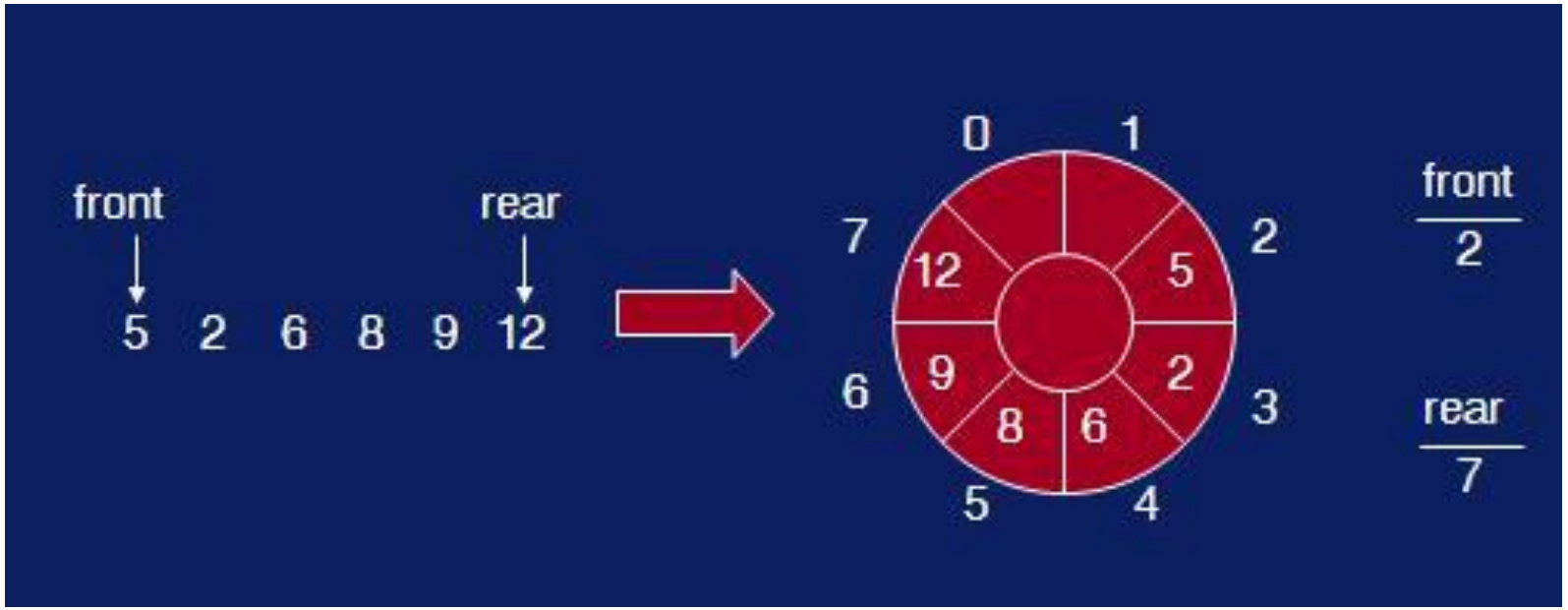


enqueue(9)
enqueue(12)



enqueue(21) ??

Solution: allow the queue to “wrap around”. Basic idea is to picture the array as a **circular** array as show below.



```
#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
   int val;
   if (rear == n - 1)
      cout<<"Queue Overflow"<<endl;
   else {
      if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      rear++;
      queue[rear] = val;
   }
}
```

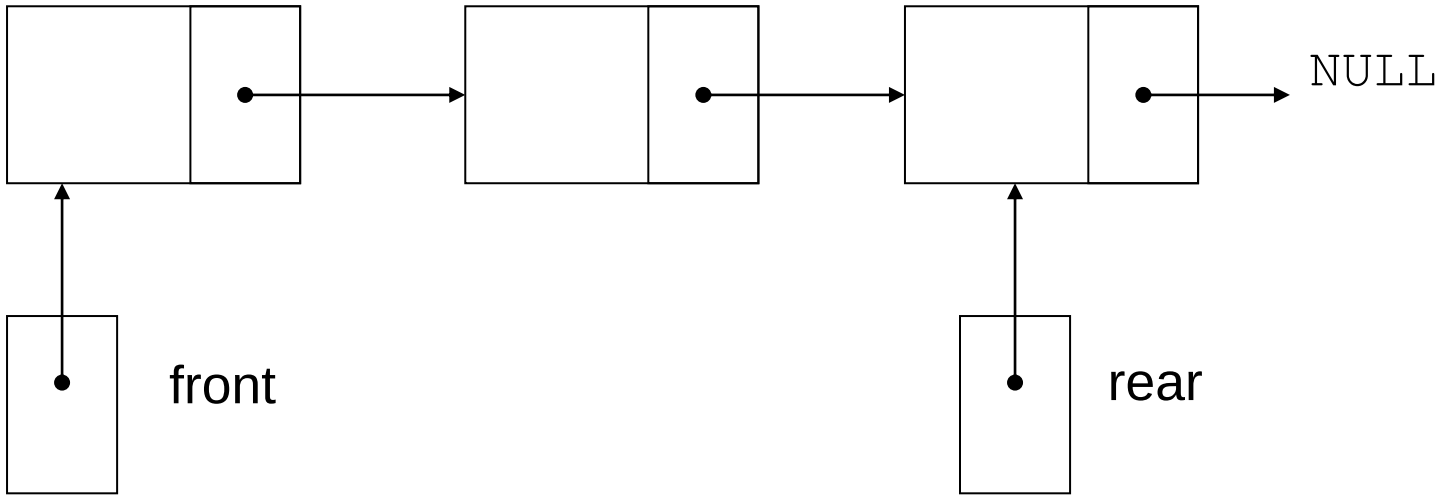
```
void Delete() {
    if (front == - 1 || front > rear) {
        cout<<"Queue Underflow ";
    return ;
    } else {
        cout<<"Element deleted from queue is :
"<< queue[front] <<endl;
        front++;
    }
}
```

```
void Display() {
    if (front == - 1)
    cout<<"Queue is empty"<<endl;
    else {
        cout<<"Queue elements are : ";
        for (int i = front; i <= rear; i++)
            cout<<queue[i]<<" ";
        cout<<endl;
    }
}
```

```
int main() {
    Insert();
    Insert();
    cout<<"Display all the elements of queue :
"<<endl;
    Display();
    cout<<"Delete element from queue : "<<endl;
    Delete();
    cout<<"Display all the elements of queue :
"<<endl;
    Display();
    return 0;
}
```

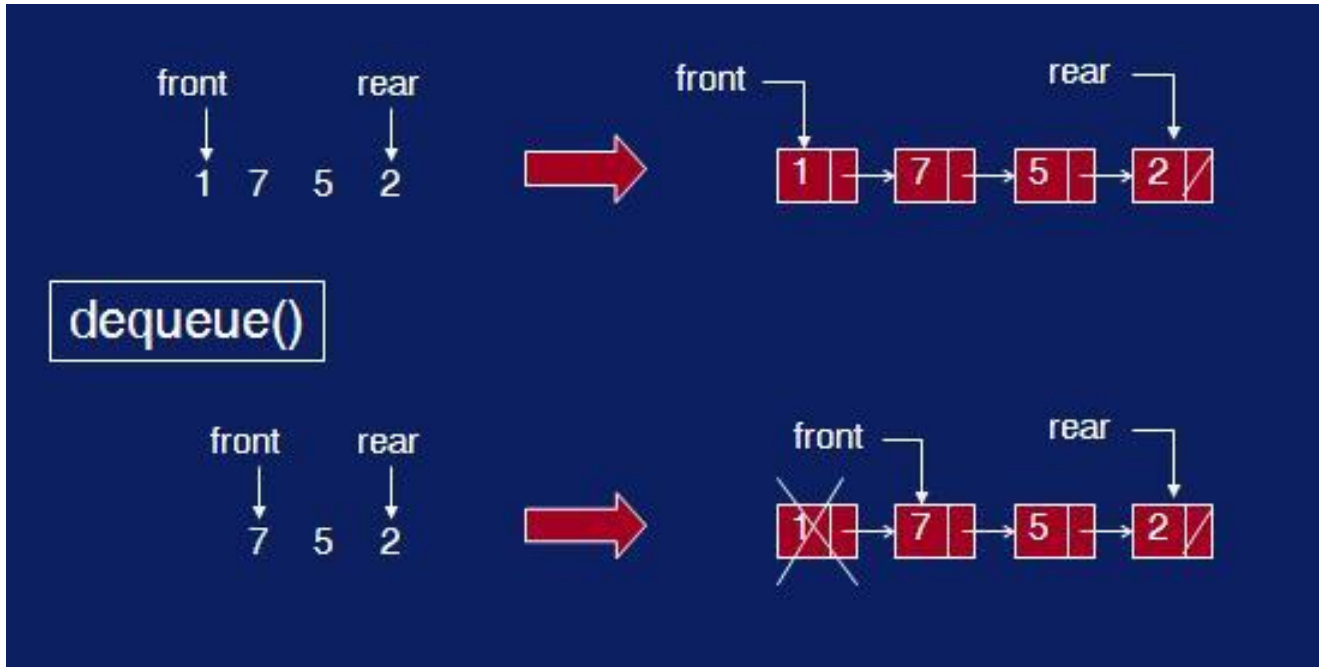
Dynamic Queues

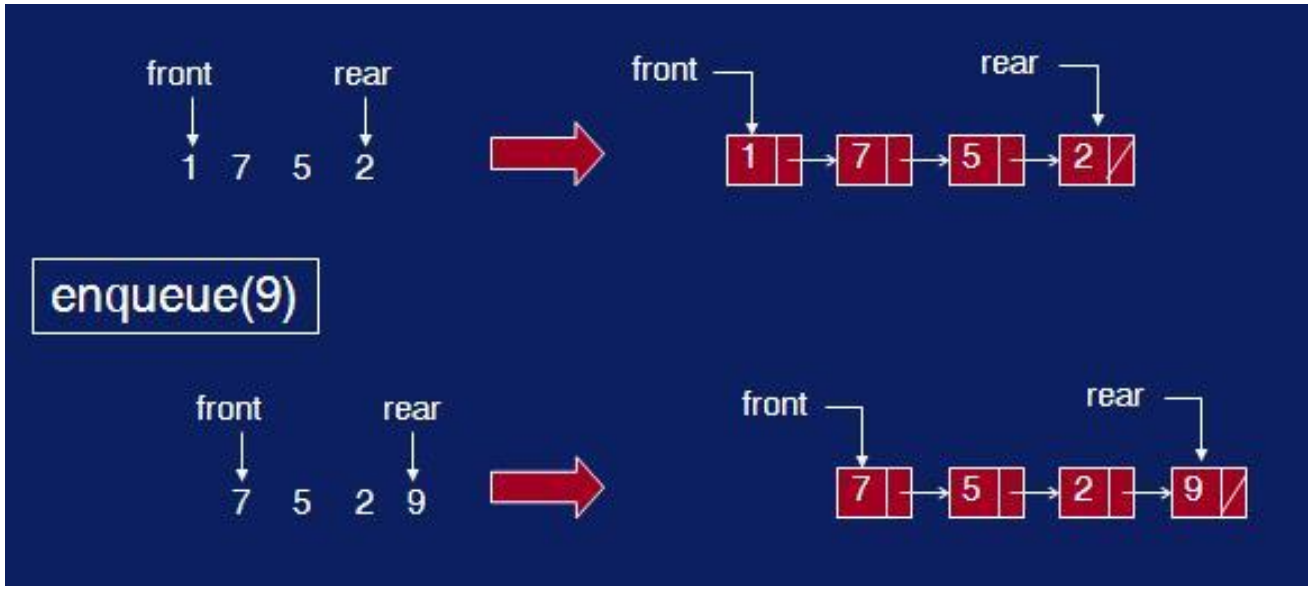
- Like a stack, a queue can be implemented using a linked list
- Allows dynamic sizing, avoids issue of shifting elements or wrapping indices



Implementation using Linked List

Usually Queues are implemented using linked List. Insert works in constant time for either end of a linked list. Remove works in constant time only. Seems best that head of the linked list be the front of the queue so that all removes will be from the front. Inserts will be at the end of the list. Consider the following diagram for implementation of Queue using linked list.


front
rear
1 7 5 2
front
rear
1
7
5
2
dequeue()
front
rear
7 5 2
front
rear
1
7
5
2


front
rear
1 7 5 2
front
rear
1
7
5
2
enqueue(9)
front
rear
7 5 2 9
front
rear
7
5
2
9

```
#include <iostream>
using namespace std;
struct node {
    int data;
    struct node *next;
};
struct node* front = NULL;
struct node* rear = NULL;
struct node* temp;

void Insert() {
    int val;
    cout<<"Insert the element in queue : "<<endl;
    cin>>val;
```

```
if (rear == NULL) {
        rear=new node;
        rear->next = NULL;
        rear->data = val;
        front = rear;
    } else {

   temp=new node;
        rear->next = temp;
        temp->data = val;
        temp->next = NULL;
        rear = temp;
    }
}
```

```
void Delete() {
    temp = front;
    if (front == NULL) {
        cout<<"Underflow"<<endl;
        return;
    }
    else
    if (temp->next != NULL) {
        temp = temp->next;
        cout<<"Element deleted from queue is : "<<front->data<<endl;
        free(front);
        front = temp;
    } else {
        cout<<"Element deleted from queue is : "<<front->data<<endl;
        free(front);
        front = NULL;
        rear = NULL;
    }
}
```

```
void Display() {
    temp = front;
    if ((front == NULL) && (rear == NULL)) {
        cout<<"Queue is empty"<<endl;
        return;
    }
    cout<<"Queue elements are: ";
    while (temp != NULL) {
        cout<<temp->data<<" ";
        temp = temp->next;
    }
    cout<<endl;
}
```

```
int main() {
    Insert();
    Insert();
    Insert();
    cout<<"Display all the elements of queue : "<<endl;
    Display();
    cout<<"Delete element from queue : "<<endl;
    Delete();
    cout<<"Display all the elements of queue : "<<endl;
    Display();

    return 0;
}
```