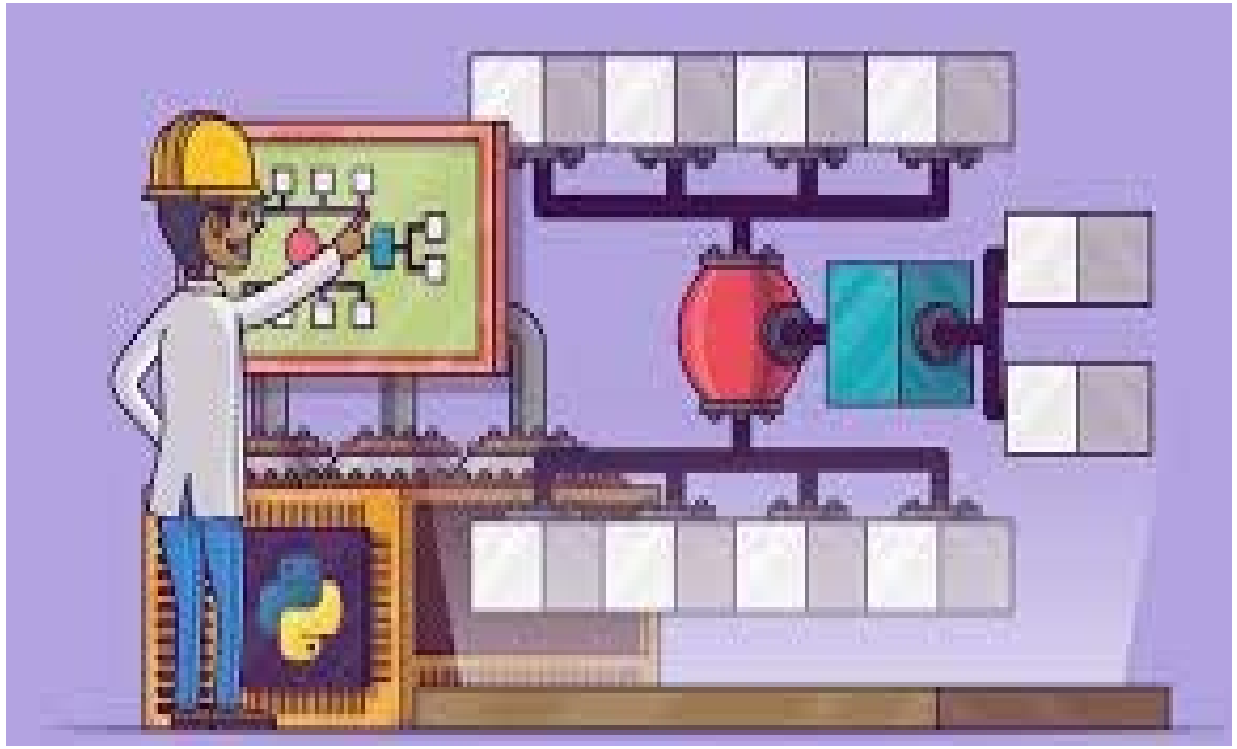
ساختمان داده ها



مدرس:
سمانه حسینی سمنانی

دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

Evaluating Postfix Expressions

- Evaluation process
- Make a single left-to-right scan of the expression.
- Place the operands on a stack until an operator is found.
- Remove, from the stack, the correct numbers of operands for the operator, perform the operation, and place the result back on the stack.

Evaluating Postfix Expressions

```
void Eval(Expression e)
{// Evaluate the postfix expression e. It is assumed that the last token (a token
// is either an operator, operand, or '#') in e is '#'. A function NextToken is
// used to get the next token from e. The function uses the stack stack
      Stack<Token> stack;      //initialize stack
      for (Token x = NextToken (e) ; x != '#'; x = NextToken (e))
          if (x is an operand) stack.Push(x) // add to stack
          else {// operator
                remove the correct number of operands for operator x from stack;
                perform the operation x and store the result (if any) onto the stack;
          }
}
```

Infix to Postfix

(1) Fully parenthesize expression

a / b - c + d * e - a * c -->

((((a / b) - c) + (d * e)) - a * c))

(2) All operators replace their corresponding right parentheses.

((((a / b) - c) + (d * e)) - a * c))

(3) Delete all parentheses. ab/c-de*+ac*-

Infix to Postfix

Infix Expression: **A+ (B*C-(D/E^F)*G)*H**, where ^ is an exponential operator.

عملگر با اولویت پایین تر یا مساوی نمی تواند روی عملگر اولویت بالا در پشته قرار گیر.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack , that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack , that's why '^' Come first
19.	*	(+*	ABC*DEF^/G*-	
20.	H	(+*	ABC*DEF^/G*-H	
21.	)	Empty	ABC*DEF^/G*-H*+	END

Rules

1. Push "(" onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
    [End of If]
    [End of If]
7. END.

Rules

isp: in-stack priority: اولویت در پشته
icp: in-coming priority: اولویت در عبارت

- The left parenthesis is placed in the stack whenever it is found in the expression, but it is unstacked only when its matching right parenthesis is found.
- We remove an operator from the stack only if its isp is greater than or equal to the icp of the new operator.

عملگر با اولویت پایین تر یا مساوی نمی تواند روی عملگر اولویت بالا در پشته قرار گیر.

Infix to Postfix Function

```
void Postfix (Expression e)
{// Output the postfix from of the infix expression e. NextToken
// is as in function Eval (Program 3.18). It is assumed that
// the last token in e is '#'. Also. '#' is used at the bottom of the stack
    Stack<Token>stack; // initialize stack
    stack.Push('#');
    for (Token x = NextToken(e); x != '#'; x = NextToken(e))
        if (x is an operand) cout<<x;
        else if (x == ')')
            {// unstack until '('
                for (; stack.Top () != '('; stack.Pop())
                    cout << stack.Top ();
                stack.Pop (); // unstack '('
            }
        else { // x is an operator
            for (; isp (stack.Top()) <= icp (x); stack.Pop())
                cout << stack.Top();
            stack.Push(x);
        }

    // end of expression; empty the stack
    for (; !stack.IsEmpty(); cout << stak.Top(), stack.Pop());
    cout << endl;
}
```

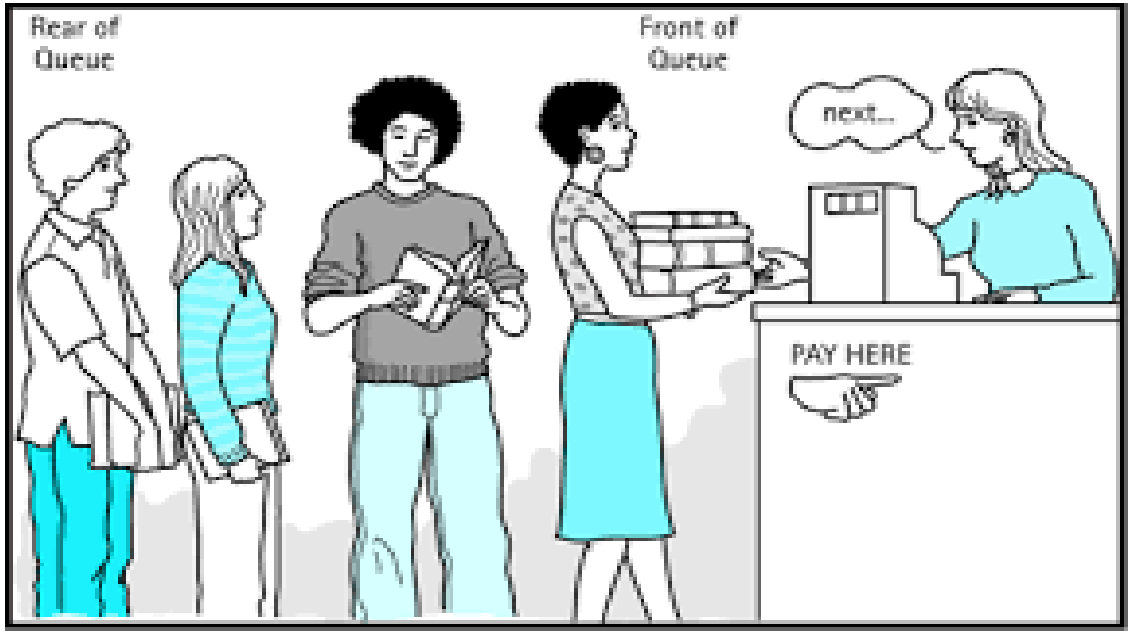
Analysis of postfix()

- Let n be the number of tokens in the expression.
- $\theta(n)$ time is spent extracting tokens and outputting them.
- for operands: $\theta(1)$
- for operators: $\theta(2)$
- Time is spent in stacking and unstacking.
- The total time spent here is θ(n) as the number of tokens that get stacked and unstacked is linear in n.

So, the complexity of function postfix() is $\theta(n)$.

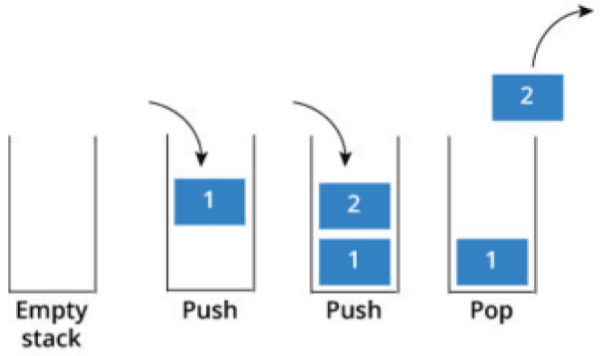
Queue ADT

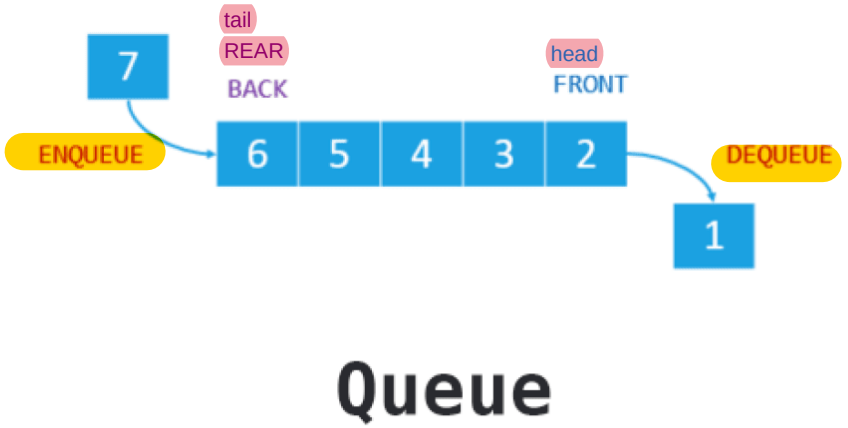
- **Queue:** an ordered list in which all insertions take place at one end and all deletions take place at the opposite end.
- First-in, last-out: **FIFO**



Queue ADT

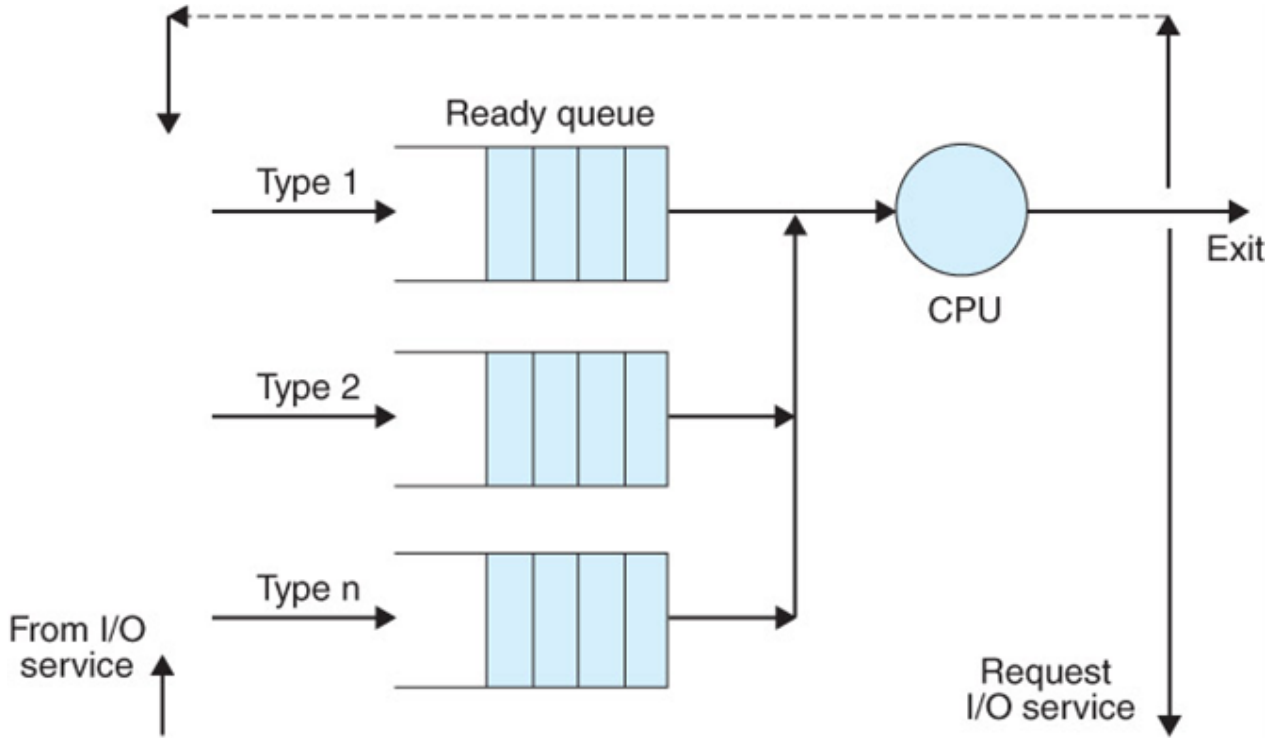
- **Queue:** an ordered list in which all insertions take place at one end and all deletions take place at the opposite end.
- First-in, last-out: **FIFO**





Queue Applications

- **Queue:** an ordered list in which all insertions take place at one end and all deletions take place at the opposite end.
- First-in, last-out: **FIFO**

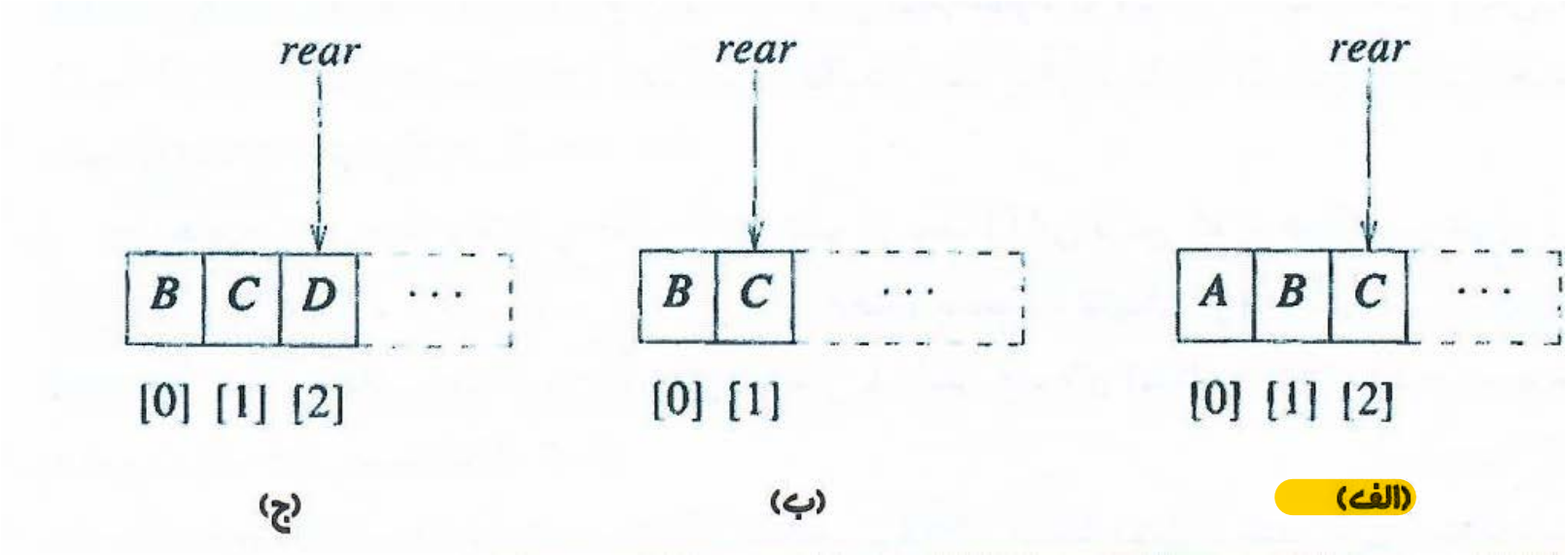


Queue ADT

```
template <class T>
class Queue
{ // A finite ordered list with zero or more elements.
public:
    Queue(int queueCapacity = 10);
    // Create an empty queue whose initial capacity is queueCapacity

    bool IsEmpty( ) const;
    // If number of elements in the queue is 0, return true else return false.

    T& Front() const;
    // Return the element at the front of the queue.

    T& Rear() const;
    // Return the element at the rear of the queue.

    void Push (const T& item);
    // Insert item at the rear of the queue.

    void Pop();
    // Delete the front element of the queue.
};
```

Queue implementation using array

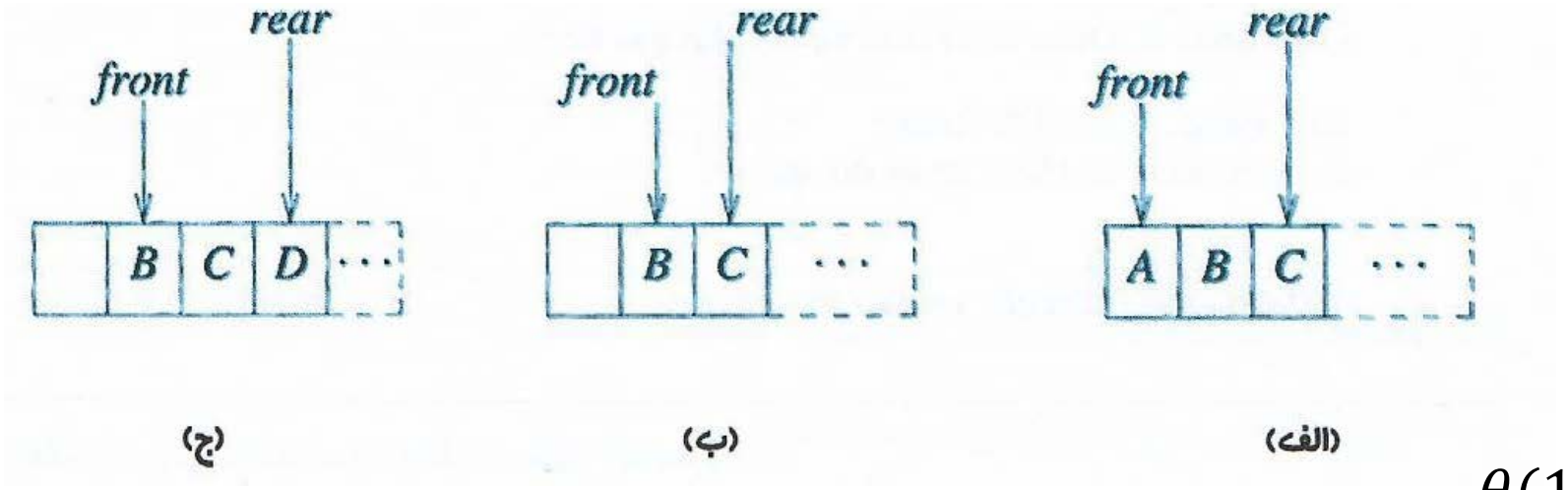
- پیاده سازی صف تنها با متغیر rear



- هزینه درج: $\theta(1)$
- بعد از حذف یک عنصر باید عناصر را به چپ شیفت دهیم.
- برای صفی با n عنصر، هزینه حذف: $\theta(n)$
- به استثنای زمان مربوط به تغییر اندازه مورد نیاز آرایه

Queue implementation using array

- پیاده سازی صف با دو متغیر rear و front

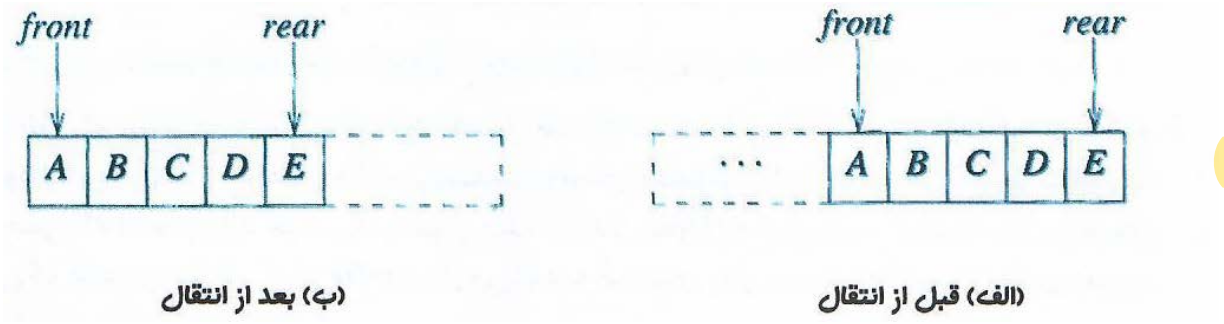


- هزینه درج: $\theta(1)$
- هزینه حذف: $\theta(1)$
- به استثنای زمان مربوط به تغییر اندازه مورد نیاز آرایه

سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

صف حلقوی

- مشکل این روش:

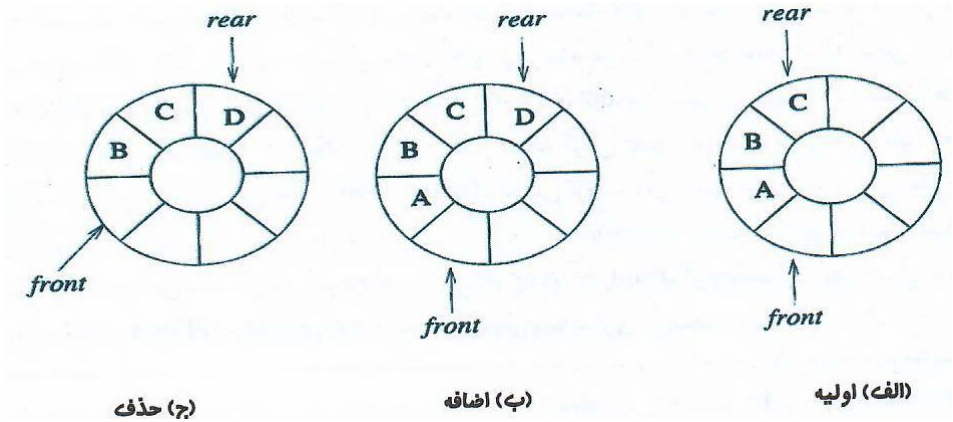


rear = capacity-1

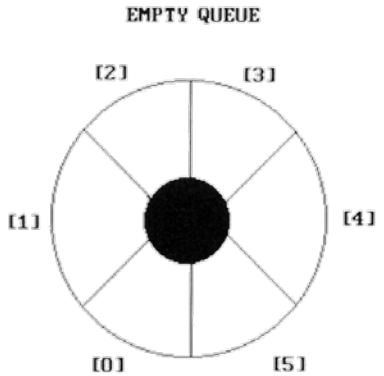
```
if (rear == capacity – 1) rear = 0;
else rear++;
```

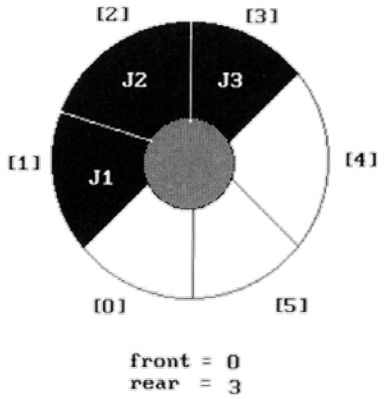
یا

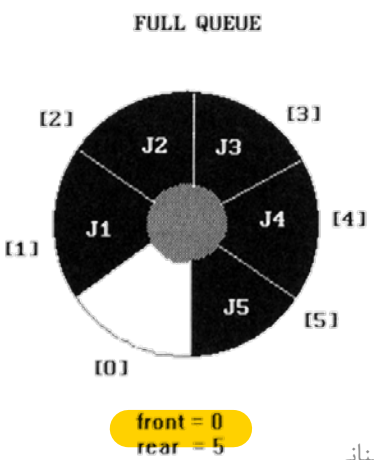
```
(rear + 1) % capacity
```

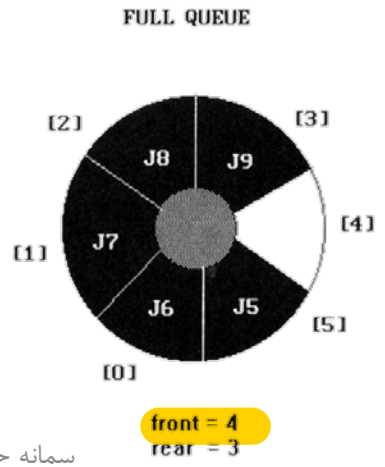


صف حلقوی


EMPTY QUEUE
[2]
[3]
[1]
[4]
[0]
[5]
front = 0
rear = 0

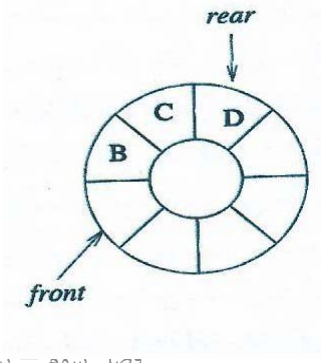

[2]
[3]
J2
J3
[1]
J1
[4]
[0]
[5]
front = 0
rear = 3


FULL QUEUE
[2]
[3]
J2
J3
J1
J4
[1]
[4]
J5
[5]
[0]
front = 0
rear = 5


FULL QUEUE
[2]
[3]
J8
J9
[1]
J7
[4]
J6
J5
[5]
[0]
front = 4
rear = 3

Queue implementation using array

```
private:
    T *queue;       // array for queue elements
    int front,      // one counterclockwise from front
        rear,       // array position of rear element
        capacity;   // capacity of queue array

template <class T>
Queue<T>::Queue (int queueeCapacity) : capacity (queueCapacity)
{
    if (capacity < 1) throw "Queue capacity must be > 0";
    queue = new T [capacity];
    front = rear = 0;
}
```



چالش پر یا خالی بودن صف

پر یا خالی بودن صف را چطور تشخیص دهیم؟ برای هر دو حالت front = rear

راه حل برای حل مشکل:

۱- ذخیره تعداد عناصر موجود در صف

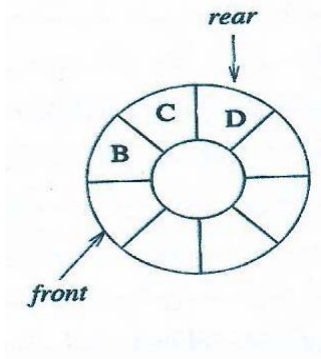
۲- اجازه ندهیم صف پر شود. همیشه یک عنصر صف را خالی نگه داریم

۳- نگه داشتن آخرین عمل

Queue ADT

```
template <class T>
inline bool Queue<T>::IsEmpty() {return front == rear;}

template <class T>
inline T& Queue<T>::Front()
{
        if (IsEmpty ()) throw "Queue is empty. No front element";
        return queue [(front + 1) % capacity];
}

template <class T>
inline T& Queue<T>::Rear()
{
        if (IsEmpty()) throw "Queue is empty. No rear element";
        return queue [rear];
}
```

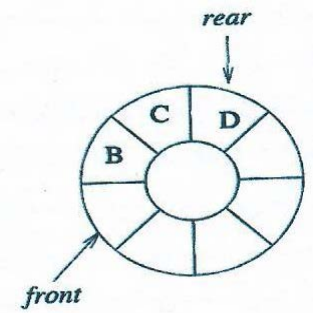
rear == front



Push in the Queue

```
emplate <class T>
oid Queue<T>::Push(const& x)
// Add x at rear of queue.
    if ((rear + 1) % capacity == front)
    {// queue full, double capacity
        // code to double queue capacity comes here
    }

    rear = (rear + 1) % capacity;
    queue[rear] = x;
}
```

rear + 1 == front



front = 0
rear = 7
(rear+1) % capacity =
8 % 8 = 0
Front = 0 ->
queue is full