



CANTHO UNIVERSITY 

Chapter 2: Common abstract data types

Lâm Hoài Bảo - FSE – CICT
Trương Minh Thái – FSE - CICT

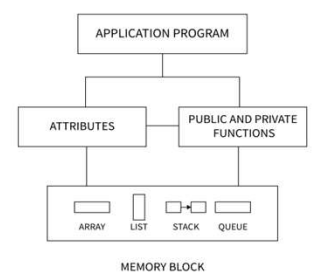
www.ctu.edu.vn

1

 CANTHO UNIVERSITY

Content

- List abstract data type (LIST)
- Stack abstract data type (STACK)
- Queue abstract data type (QUEUE)



```

graph TD
    AP[APPLICATION PROGRAM] --> A[ATTRIBUTES]
    AP --> PPF[PUBLIC AND PRIVATE FUNCTIONS]
    A --> MB[MEMORY BLOCK]
    PPF --> MB
    subgraph MB [MEMORY BLOCK]
        direction LR
        ARR[ARRAY]
        LIST[LIST]
        STACK[STACK]
        QUEUE[QUEUE]
    end
  
```

www.ctu.edu.vn

2



Agenda

- Introduction
- Array based Queue
- Linked List based Queue
- Summary

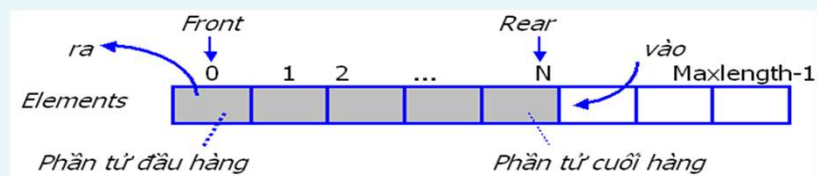
www.ctu.edu.vn

3

3



Counter queue



www.ctu.edu.vn

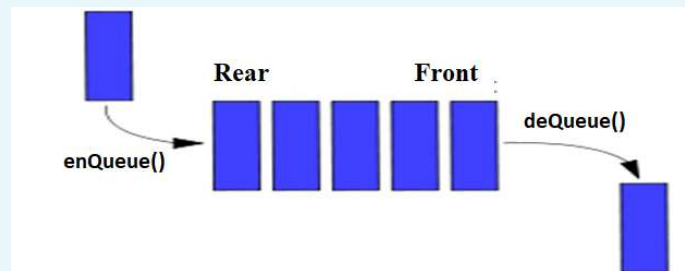
4

4



Definition

- A special kind of list that additions (**enqueue**) are done at one side (**Rear**), and deletions (**dequeue**) are done at other side (**Front**)
- **FIFO** (First In First Out) scheme



www.ctu.edu.vn

5

5



Operators

Operator	Ý nghĩa	List operator
<code>makeNull (&Q)</code>	Init an empty queue	<code>makeNull (&Q)</code>
<code>isEmpty (Q)</code>	Check a queue is empty or not?	<code>isEmpty (Q)</code>
<code>enqueue (x, &Q)</code>	Add a new item to the end of the queue	<code>append (x, &Q)</code>
<code>dequeue (&Q)</code>	Return the first item of the queue and remove it	<code>popFirst (&Q)</code>

www.ctu.edu.vn

6

6




Agenda

- Introduction
- Array based list
- Linked list based list
- Summary

7

www.ctu.edu.vn

7



Array based implementation

- Data structure
 - An array to store elements
 - `front` refers the first position
 - `rear` refers the position that can be added to
- Idea:
 - Removal: increase `front` by 1
 - Addition: decrease `rear` by 1
- Elements tend to **shift to the right**

2	3	5	7	11					
0	1				5				MaxSize-1
front				rear					

deQueue() 3 times

enQueue(13, &Q)

enQueue(17, &Q)

enQueue(19, &Q)

			7	11	13	17			
0	1								MaxSize-1


enQueue(19, &Q)

HOW???

8

www.ctu.edu.vn

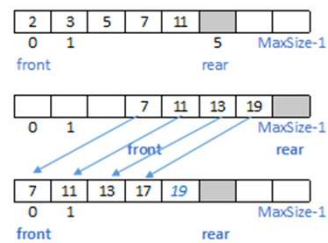
8



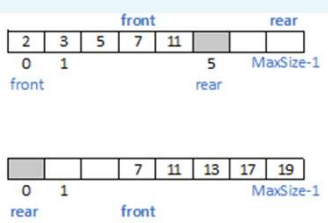
Solutions

- Shift all elements to the front

- Circular array, the next element is at the last position, rear is set to the beginning of the array



dequeue() 3 times
 enqueue(13, &Q)
 enqueue(17, &Q)
 enqueue(19, &Q)




dequeue() 3 times
 enqueue(13, &Q)
 enqueue(17, &Q)
 enqueue(19, &Q)

9

www.ctu.edu.vn

9

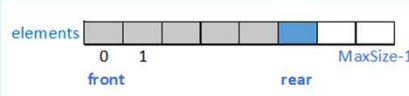


Circular queue

```

#define MaxSize ...
typedef ... ElementType;
typedef struct {
    ElementType elements[MaxSize];
    int front, rear;
}Queue;


```



10

www.ctu.edu.vn

10



Initialization - Empty checking

- Initialization
 - Set `front=rear=0`

elements

0	1						
front		MaxSize-1					
rear							

ALGORITHM `makeNull(*pQ)`:

`pQ->front` \leftarrow 0

`pQ->rear` \leftarrow `pQ->front`


- Check a queue is empty?
 - Check `front == rear`

ALGORITHM `isEmpty(Q)`:

return `Q.front == Q.rear`

www.ctu.edu.vn
11

11



Number of elements

0	1							
front		MaxSize-1						
rear								

0	1	2	3					
rear		MaxSize-1						
front								


- Pseudo code

ALGORITHM `size(Q)`:

return `(Q.rear + MaxSize - Q.front) % MaxSize`


www.ctu.edu.vn
12

12



Leave an empty cell

- If call `enqueue(x, &Q)`
 - Put `x` to rear position, increase rear
 - But `isEmpty() == TRUE`
- One solution: Leave an empty cell, check the queue is full?




```

ALGORITHM fullQueue(Q):
    return (Q.rear + 1) % MaxSize == Q.front
        
```

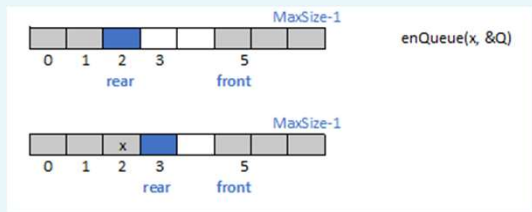
www.ctu.edu.vn

13



Add to queue - enqueue(x, &Q)

- Algorithm
 - If (Q is full)
 - raiseError "Error"
 - return
 - Put `x` to position rear
 - Increase rear by 1
- Pseudo code




```

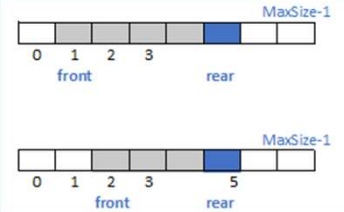
ALGORITHM enqueue(x, *pQ):
    if isFull(*pQ):
        raise "Error"
        return
    pQ->elements[pQ->rear] ← x
    pQ->rear ← (pQ->rear + 1) % MaxSize
        
```

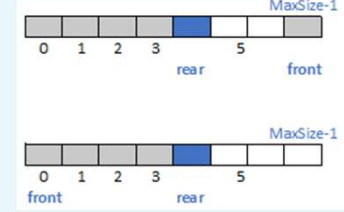
www.ctu.edu.vn

14



Dequeue





ALGORITHM dequeue(*pQ):

```

if (*pQ) is empty:
    return ERROR
x ← pQ->elements[pQ->front]
pQ->front ← (pQ->front + 1) % MaxSize
return x


```

$T(n) = O(1)$

www.ctu.edu.vn

15

15



Agenda

- Introduction
- Array based queue
- Linked List based queue
- Summary

www.ctu.edu.vn

16

16



Linked List implementation

- Recap: Queue only allows adding to the end and popping at the beginning.
- Linked list can be used to implement by adding a field to keep the last node address

17

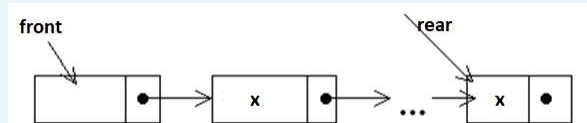
www.ctu.edu.vn

17



Declaration


```
typedef ... ElementType;
typedef struct NodeTag{
    ElementType data;
    struct NodeTag* next;
}Node;
typedef struct QueueTag{
    Node* front;
    Node* rear;
}Queue;
```



18

www.ctu.edu.vn

18

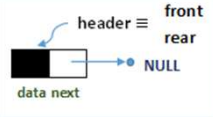


Initialization & empty checking

- Initialize an empty queue



```
void makeNull(Queue *pQ){
    Node* header = (Node*)malloc(sizeof(Node));
    header->next = NULL;
    pQ->front = header;
    pQ->rear = pQ->front;
}
```
- Check a queue is empty or not


```
int isEmpty(Queue Q){
    return Q.front == Q.rear;
}
```




www.ctu.edu.vn
19

19



enqueue(x, &Q)


- Add an element to the end of the queue pointed by pQ
 - Example: enqueue(x=11, &Q)


- Pseudo code

ALGORITHM enqueue(x, *pQ):


www.ctu.edu.vn
20

20



enQueue(x, &Q)

- Add an element to the end of the queue pointed by pQ
 - Example: enQueue(x=11, &Q)



- Pseudo code


```

ALGORITHM enQueue(x, *pQ):
    q ← malloc (|Node|)
    q->data ← x
  
```

11


21
www.ctu.edu.vn

21



enQueue(x, &Q)

- Add an element to the end of the queue pointed by pQ
 - Example: enQueue(x=11, &Q)



- Pseudo code


```

ALGORITHM enQueue(x, *pQ):
    q ← malloc (|Node|)
    q->data ← x
    q->next ← NULL
  
```

11

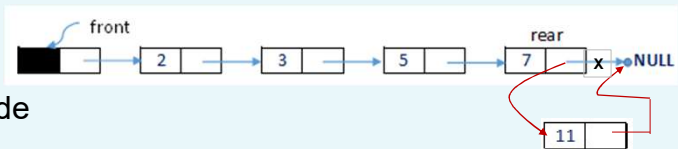
22
www.ctu.edu.vn

22



enQueue(x, &Q)

- Add an element to the end of the queue pointed by pQ
 - Example: enQueue(x=11, &Q)




■ Pseudo code

```

ALGORITHM enQueue(x, *pQ):
    q ← malloc (|Node|)
    q->data ← x
    q->next ← NULL
    pQ->rear->next ← q
      
```

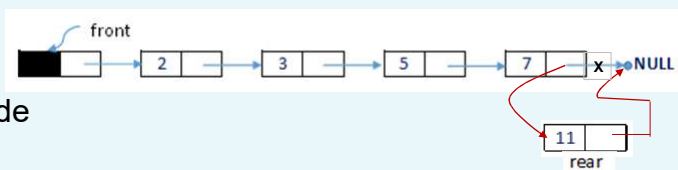
23
www.ctu.edu.vn

23



enQueue(x, &Q)

- Add an element to the end of the queue pointed by pQ
 - Example: enQueue(x=11, &Q)



■ Pseudo code


```

ALGORITHM enQueue(x, *pQ):
    q ← malloc (|Node|)
    q->data ← x
    q->next ← NULL
    pQ->rear->next ← q
    pQ->rear ← q
      
```

$T(n) = O(1)$

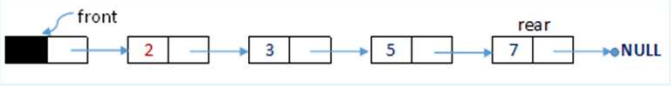
24
www.ctu.edu.vn

24



deQueue(&Q)

- Get the element at the front of queue and remove it.
 - Example: deQueue (&Q) → 2




- Pseudo code

```

ALGORITHM de_queue(*pQ):
    x ← pQ->front->next->data
        
```

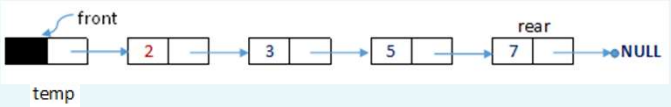
www.ctu.edu.vn
25

25



deQueue(&Q)

- Get the element at the front of queue and remove it.
 - Example: deQueue (&Q) → 2




- Pseudo code

```

ALGORITHM de_queue(*pQ):
    x ← pQ->front->next->data
    temp ← pQ->front
        
```

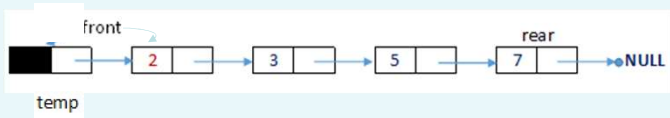
www.ctu.edu.vn
26

26



deQueue(&Q)

- Get the element at the front of queue and remove it.
 - Example: deQueue (&Q) → 2




■ Pseudo code

```

ALGORITHM de_queue(*pQ):
    x ← pQ->front->next->data
    temp ← pQ->front
    pQ->front ← temp->next
  
```

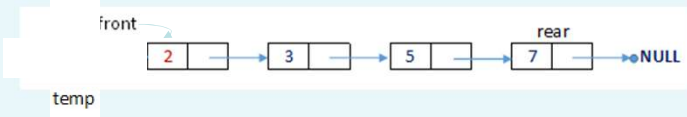
27
www.ctu.edu.vn

27



deQueue(&Q)

- Get the element at the front of queue and remove it.
 - Example: deQueue (&Q) → 2




■ Pseudo code

```

ALGORITHM de_queue(*pQ):
    x ← pQ->front->next->data
    temp ← pQ->front
    pQ->front ← temp->next
    free(temp)
    return x
  
```

$T(n) = O(1)$
 28
www.ctu.edu.vn

28



Queue traversal


- Visit each element of a queue.
- Algorithm
 - `while (Q != Φ):`
 - Get the element at the front of Q; process it
 - Remove that element
- After traversal, $Q \rightarrow \Phi$
- Example: Print all elements of a queue

```
ALGORITHM print(*pQ):
    while (!isEmpty(*pQ)):
        write(deQueue(pQ))
```

29

www.ctu.edu.vn

29




Agenda

- Introduction
- Array based Queue
- Linked List based Queue
- Summary

30

www.ctu.edu.vn

30




Summary

- Queue allows adding to one side and remove at other side (FIFO, LILO)
- A special case of sequence ADT
- Complexity

Data structures	Construction	Dynamic	
	makeNull ()	enqueue ()	dequeue ()
Array	$O(1)$	$O(1)$	$O(1)$
Linked List	$O(1)$	$O(1)$	$O(1)$

www.ctu.edu.vn 31

31



Exercise

Ex 6.1

- Write a program that inputs integers to a stack S.
- Then use a queue to reverse the order of the elements in stack S.

Ex 6.2:

- Write a program that inputs integers to queue Q.
- Then use a stack to reverse the order of the elements in queue Q.

www.ctu.edu.vn 32

32

