Queue

CSE 2020 Computer Science II

Learning Objectives

- define queue ADT
- implement queue ADT using array and linked structure
- analyze the time complexity of operations in different implementations
- apply queue class defined in STL

Queue ADT

- A queue stores a list of elements, which inserts an element at the back/rear of the list and deletes an element at the front of the list.
  - The first added element is at the front
  - The most recent added element is at the back
  - The add and remove happen at the different positions
  - The most recent added element is the last to be removed
  - First-In-First-Out (FIFO)

Operations of Queue

The operations are

- bool empty() const: return true if the queue is empty
- bool full() const: return true if the queue is full, only for array implementation
- void clear(): remove all elements in the queue
- void enqueue(const T & x): add x at the back of the queue
- void dequeue(): remove the front element from the queue
- const T& front_element() const: return the front element

Implementation of Queue

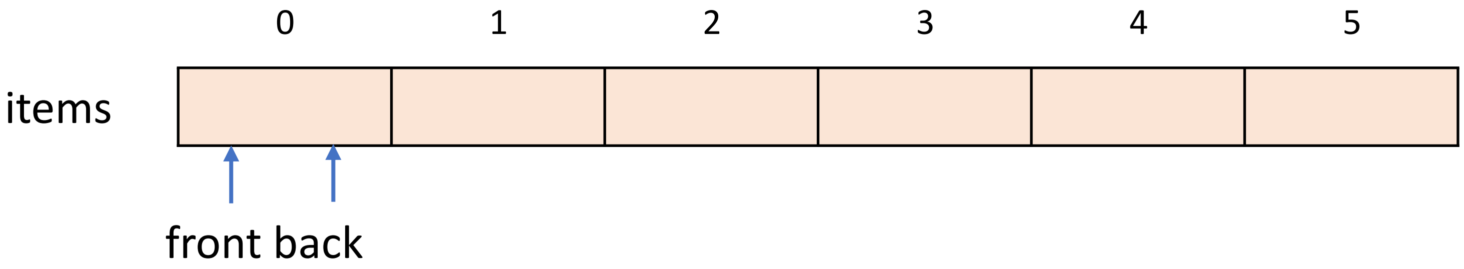
- Circular queue - array implementation of Queue
- Linked structure implementation of Queue
  - Linked queue - Singly linked structure implementation
  - Doubly linked queue - Doubly linked structure implementation

Circular Queue

- Array implementation of Queue is called circular queue, circular array is used to implement circular queue
- `Queue` class template defined in CircularQueue.cpp (in Queue.txt)
  - private attribute *`T* items,`* items points to a dynamic array
  - private attribute *`int capacity`* stores the capacity of the queue
  - private attribute *`int front`* is the index of the position **preceding** the front element
  - private attribute *`int back`* is the index of the rear/back element

Circular Queue Example

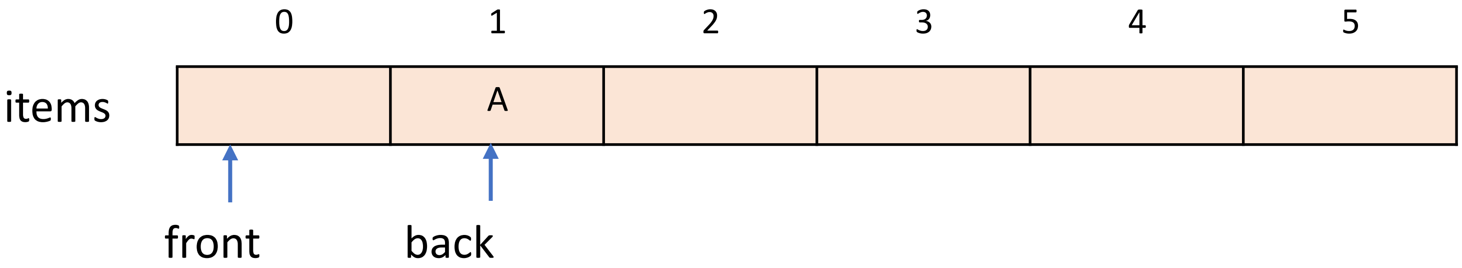
- q.capacity = 6
- initial empty state



	0	1	2	3	4	5
items						

front back

- q.enqueue(‘A’);

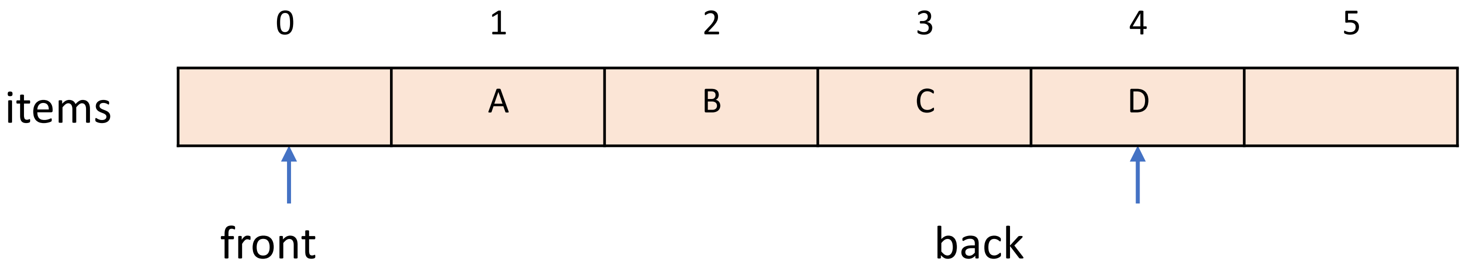


	0	1	2	3	4	5
items		A				

front back

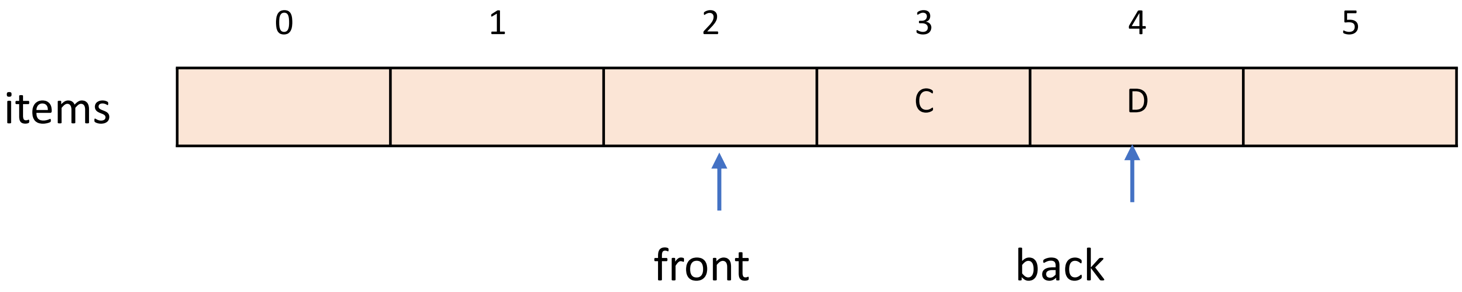
Circular Queue Example (cont.)

- q.enqueue(‘B’); q.enqueue(‘C’); q.enqueue(‘D’);



	0	1	2	3	4	5
items		A	B	C	D	

- q.dequeue(); q.dequeue();

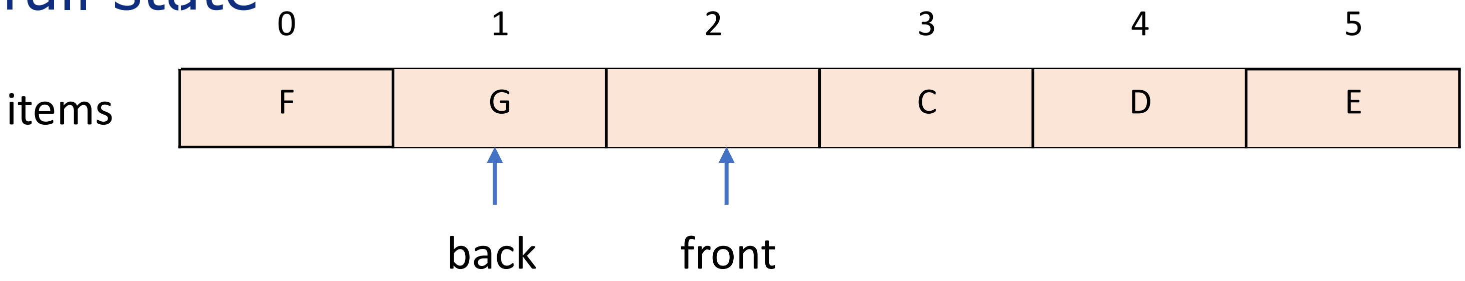


	0	1	2	3	4	5
items				C	D	

Circular Queue Example (cont.)

- q.enqueue(‘E’); q.enqueue(‘F’); q.enqueue(‘G’);

  full state



	0	1	2	3	4	5
items	F	G		C	D	E

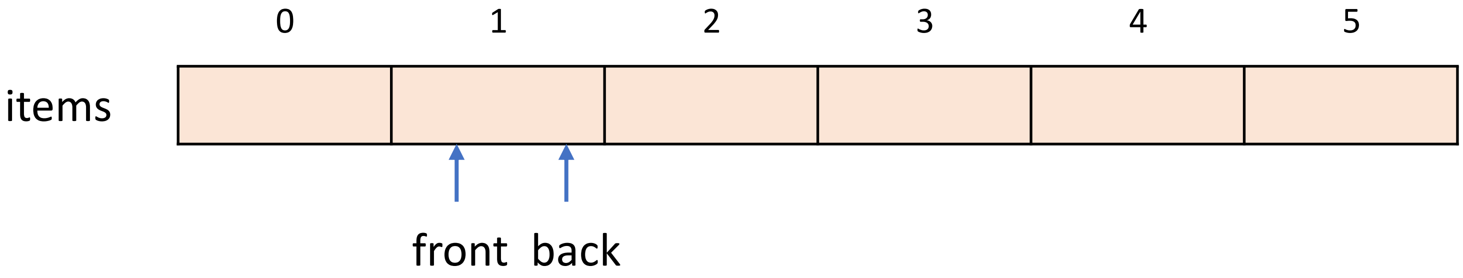
back

front

- add
  - `back = (back + 1) % capacity;`
  - `items[back] = x;`
- full state
  - `(back + 1) % capacity == front`

Circular Queue Example (cont.)

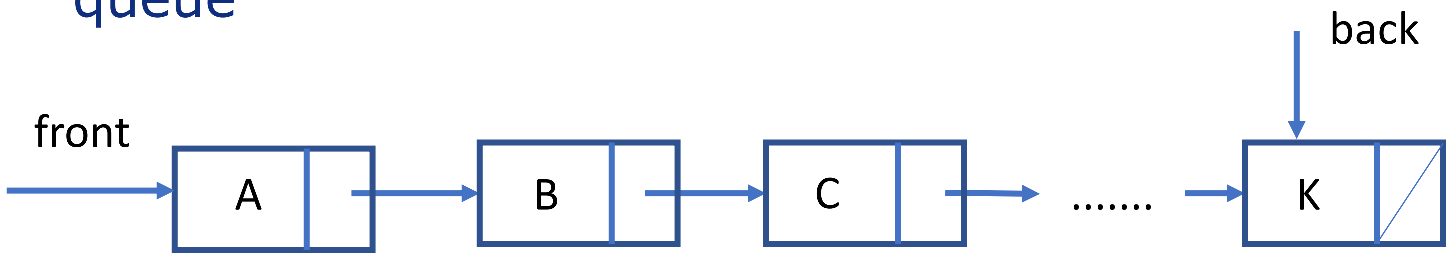
- call q.dequeue() five times, empty state



- delete
  - `front = (front + 1) % capacity;`
- empty state
  - `back == front`

Linked Queue

- `Queue` class template defined in LinkedQueue.cpp (in Queue.txt)
  - private struct template *NodeType,* contains *T data* and *NodeType* next* points to next node
  - private pointer *NodeType* front* points to the first node of queue
  - private pointer *NodeType* back* points to the last node of queue



Linked Queue enqueue

- empty state

```
front == nullptr && back == nullptr
```

- enqueue(const T& item) O(1)

```
NodeType* ptr = new NodeType(item);
back->next = ptr;
back = ptr;
```

  - Special case – add the first element
    - front = ptr; back = ptr;

Linked Queue dequeue

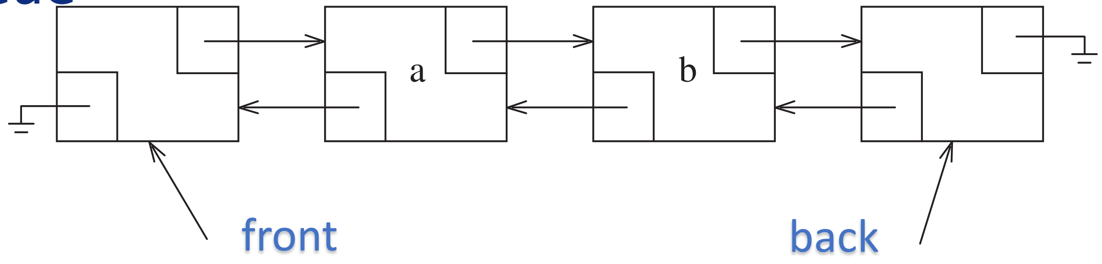
- dequeue() O(1)

  ```
  Nodetype* ptr = front;
  front = front->next;
  delete ptr;
  ```

  - Special case – delete the last element
    - back = nullptr;

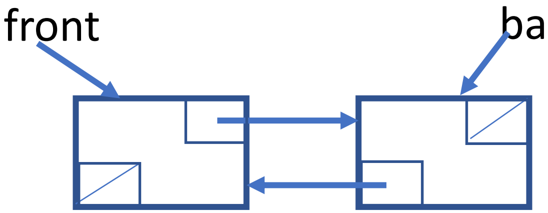
Doubly Linked Queue

- Queue class template defined in DoublyLinkedQueue.cpp (lab exercise)
  - private struct template *NodeType* contains *T data and NodeType* next* points to next node, *NodeType* prev* points to previous node
  - private pointer *NodeType* front* points to the header node of queue
  - private pointer *NodeType* back* points to the tail node of queue



Doubly Linked Queue Operations

- empty state (front->next == back)
- enqueue(const T& x) O(1)



```
NodeType* ptr = new NodeType(x)
ptr->prev = back->prev;
ptr->next = back;
back->prev->next = ptr;
back->prev = ptr;
```

- dequeue() O(1)

```
NodeType* ptr = front->next;
front->next = ptr->next;
ptr->next->prev = front;
delete ptr;
```

Access Queue Class Template

- Use class template to define Queue in a general way, the element type is `T`, which is bound to an actual data type in main() function, as shown in TestQueue.cpp (in Queue.txt)
  - #include “CircularQueue.cpp” or “LinkedQueue.cpp” or “DoublyLinkedQueue.cpp”
  - `Queue<int>`
  - `Queue<double>`
  - `Queue<string>`
  - `Queue<char>`
  - `Queue<Employee>`

Array vs Linked Structure Queue

- Array implementation of Queue has fixed capacity
- Linked structure of Queue has no fixed capacity

Operations	Circular Queue	Linked Queue	Doubly Linked Queue
empty()	O(1)	O(1)	O(1)
clear()	O(1)	O(n)	O(n)
enqueue(x)	O(1)	O(1)	O(1)
dequeue()	O(1)	O(1)	O(1)
front_element()	O(1)	O(1)	O(1)

STL queue

- queue uses a doubly linked structure to implement the Queue ADT

```
#include <queue>
queue<int> intque;
intque.enqueue(10);
intque.dequeue();
queue<double> dblque;
queue<string> strque;
```