Lecture#8

Data Structures

Dr. Abu Nowshed Chy

Department of Computer Science and Engineering

University of Chittagong

February 13, 2025

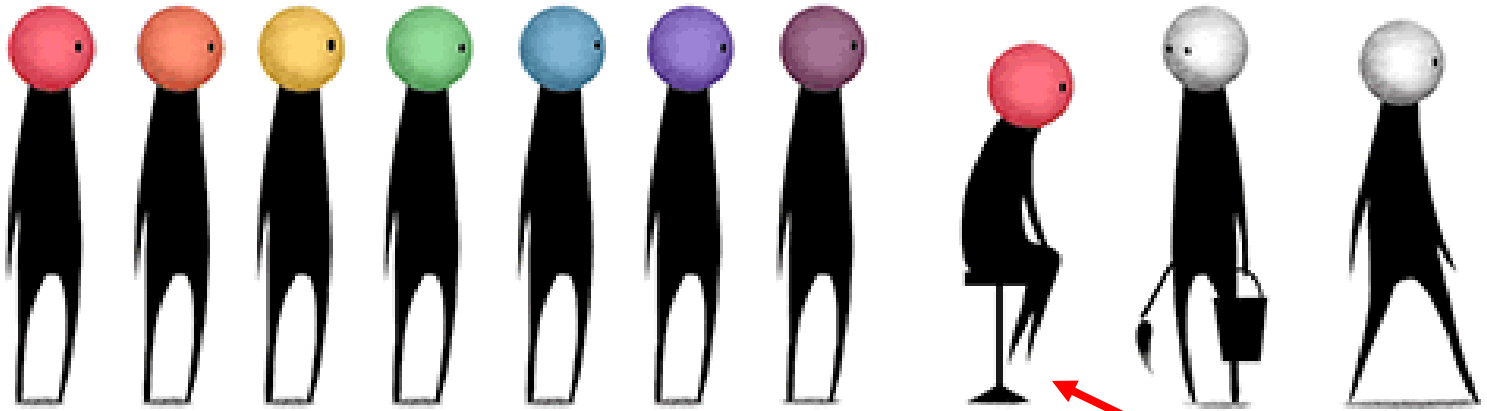
Faculty Profile

Queue

Queue

- A queue is a two ended data structure where insertion is done at one end and deletion is performed at the other end
- Queues are a special form of collection with FIFO semantics
- The insertion end is called rear and the deletion end is called front

Queue

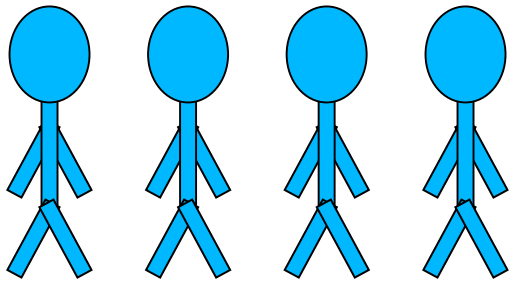


Insertion End → Rear

Deletion End → Front

Queue

- A queue is like a line of people waiting in front of a immigration clearance desk. The queue has a **front** and a **rear**.



Queue


DEPARTS
GATE
TRAVEL THE WORLD
ARRIVAL FLIGHT
BAGGAGE CLAIM B-01
18

Application of Queues

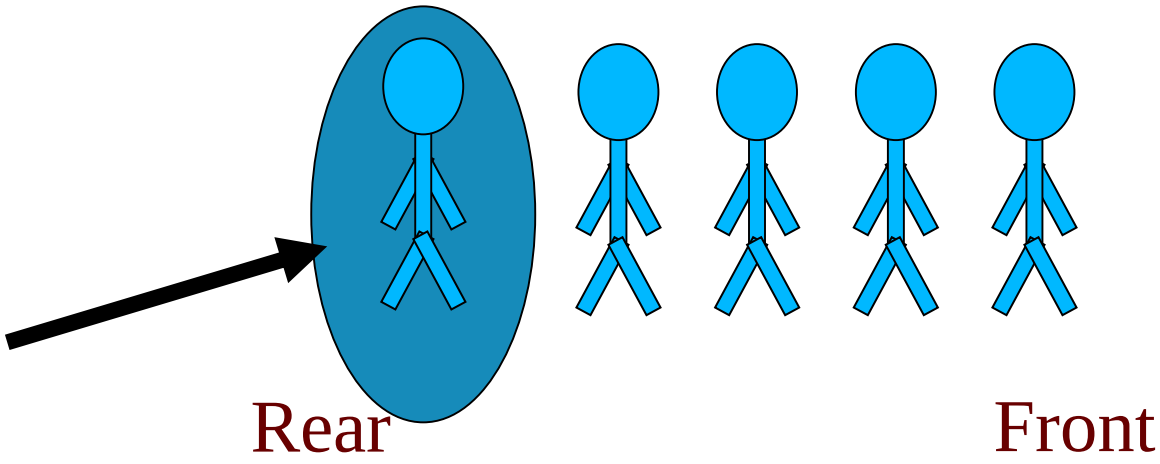
- Operating systems:
  - queue of print jobs to send to the printer
  - queue of programs / processes to be run
  - queue of network data packets to send
- Programming:
  - modeling a line of customers or clients
  - storing a queue of computations to be performed in order
- Real world examples:
  - people on an escalator or waiting in a line
  - cars at a gas station (or on an assembly line)

Enqueue

- *Function*: Adds new Item to the rear of the queue.
- *Preconditions*: Queue has been initialized and is not full.
- *Postconditions*: new Item is at rear of queue.

Enqueue

- New people must enter the queue at the rear. It is usually called an **enqueue** operation.

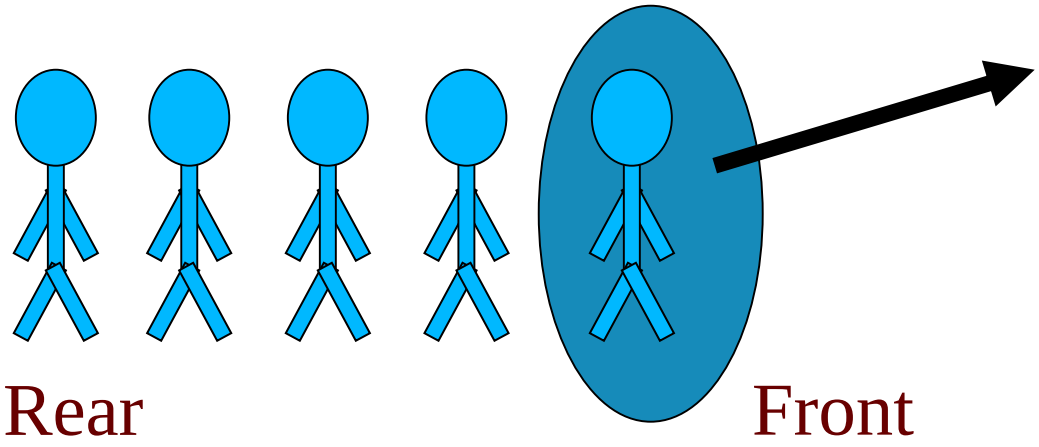


Dequeue

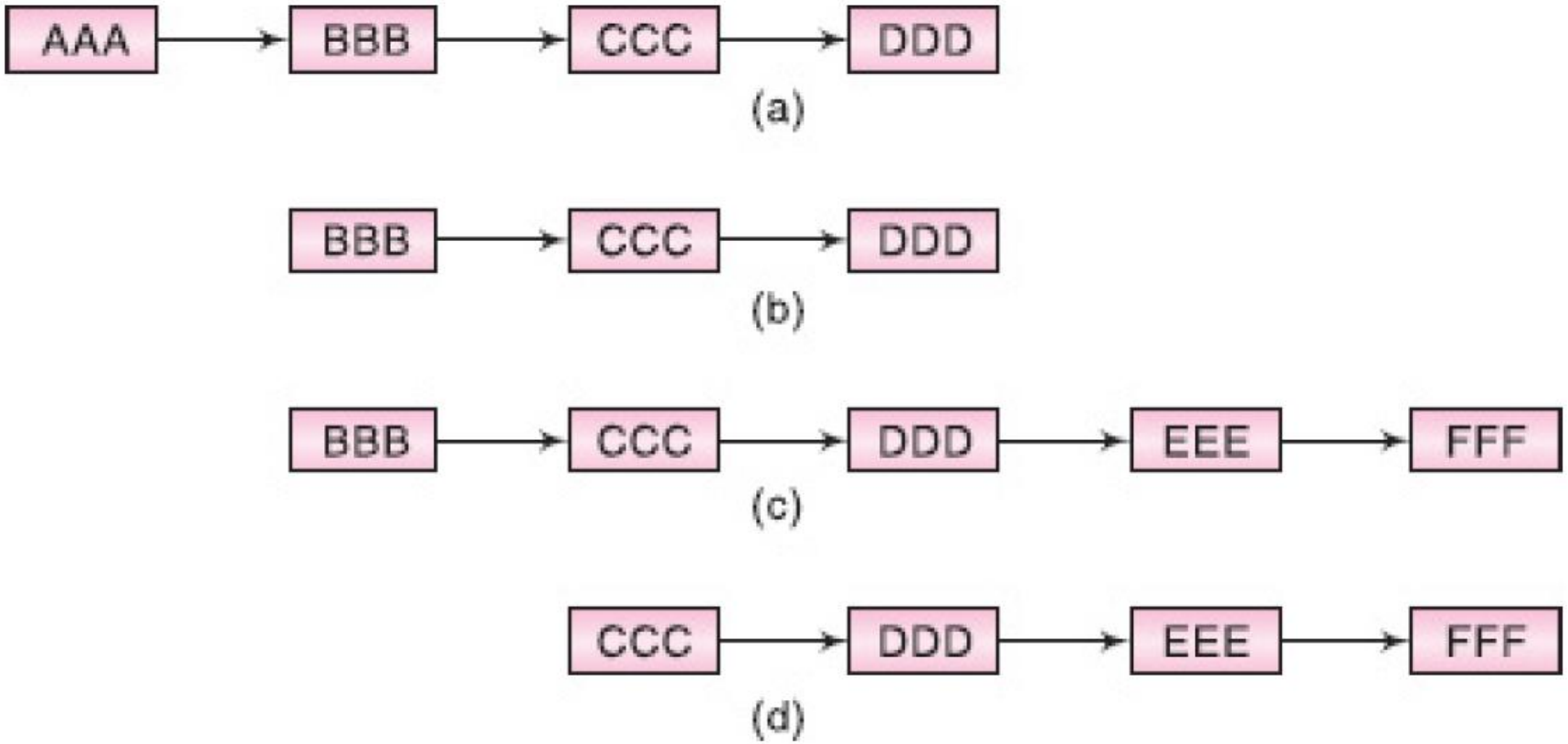
- *Function*: Removes front item from queue and returns it in item.
- *Preconditions*: Queue has been initialized and is not empty.
- *Postconditions*: Front element has been removed from queue and item is a copy of removed element.

Dequeue

- When an item is taken from the queue, it always comes from the front. It is usually called a **dequeue** operation.

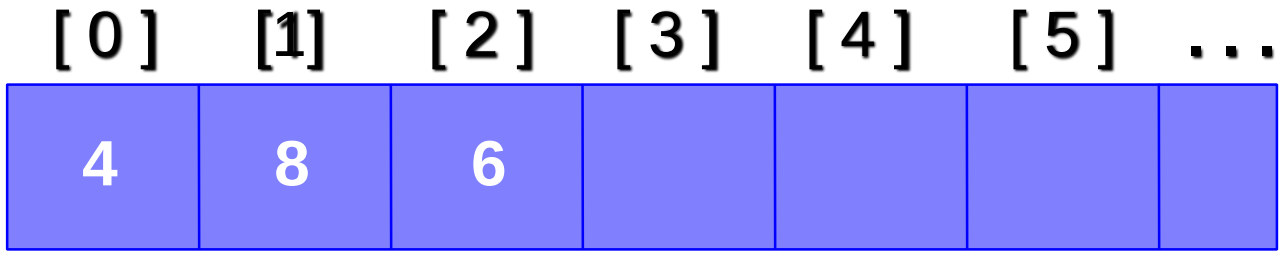


Queue Operations



Array Implementation

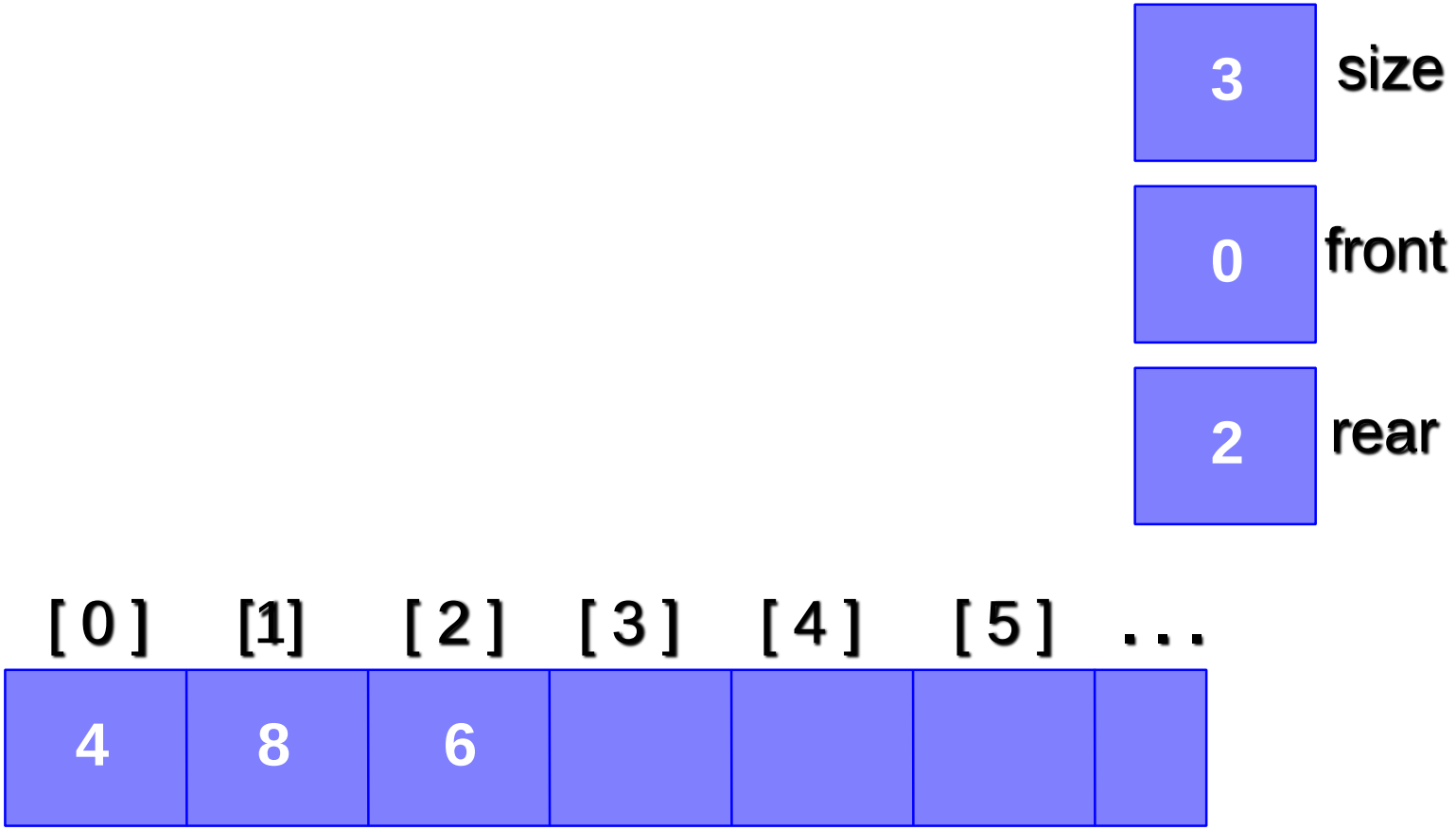
- A queue can be implemented with an array, as shown here. For example, this queue contains the integers 4 (at the front), 8 and 6 (at the rear).



An array of integers to implement a queue of integers

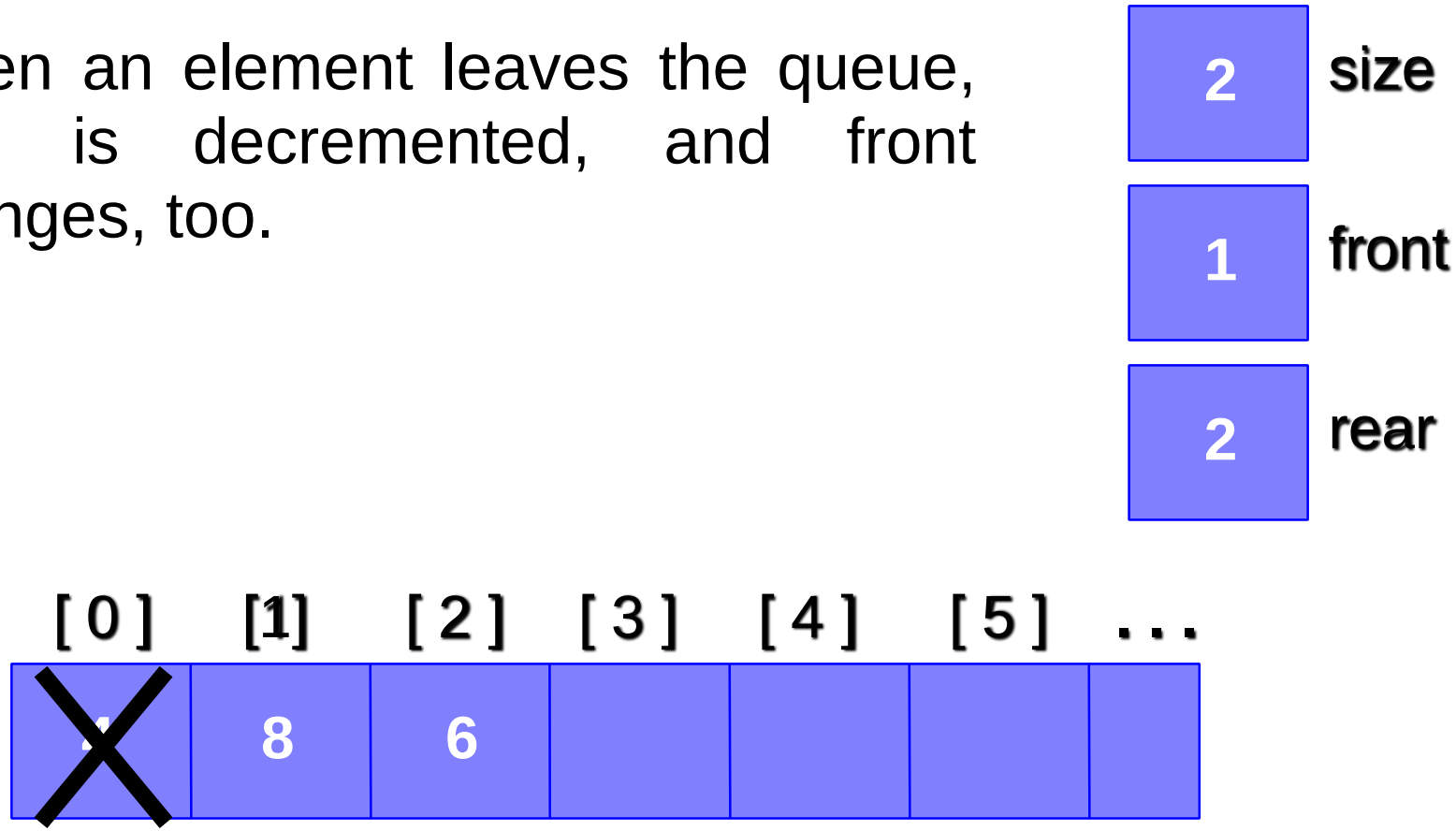
We don't care what's in this part of the array.

Array Implementation



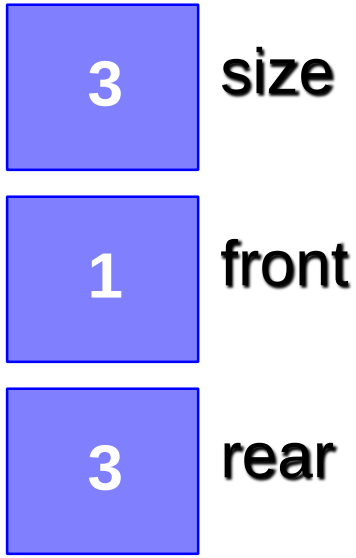
Array Implementation

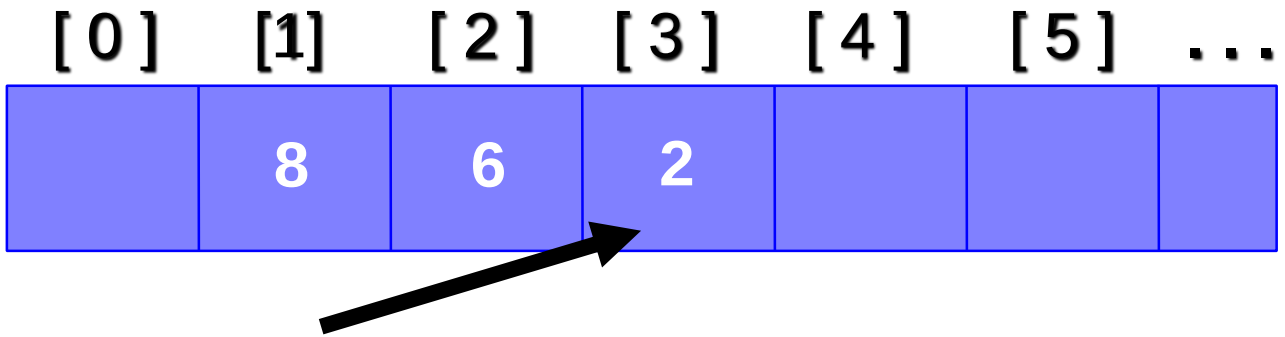
- When an element leaves the queue, size is decremented, and front changes, too.



Array Implementation

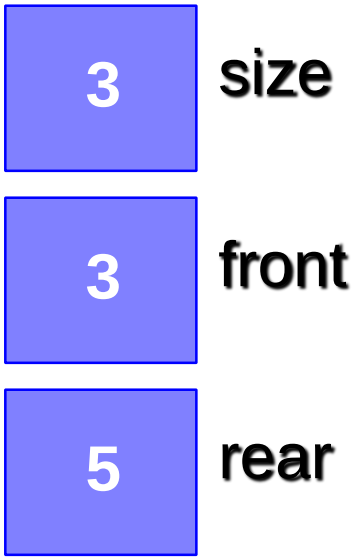
- When an element enters the queue, size is incremented, and rear changes, too.

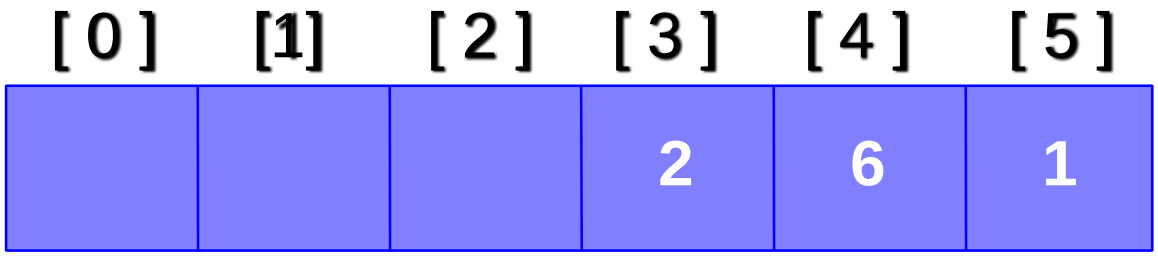




Array Implementation

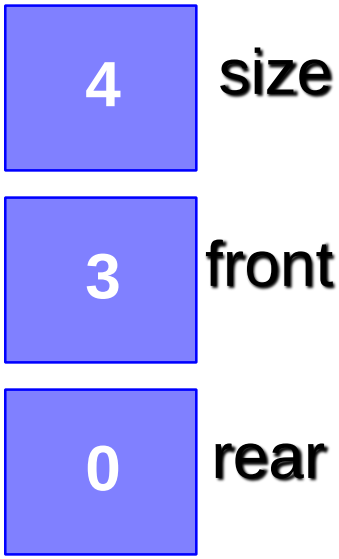
- There is special behaviour at the end of the array. For example, suppose we want to add a new element to this queue, where the last index is [5]:

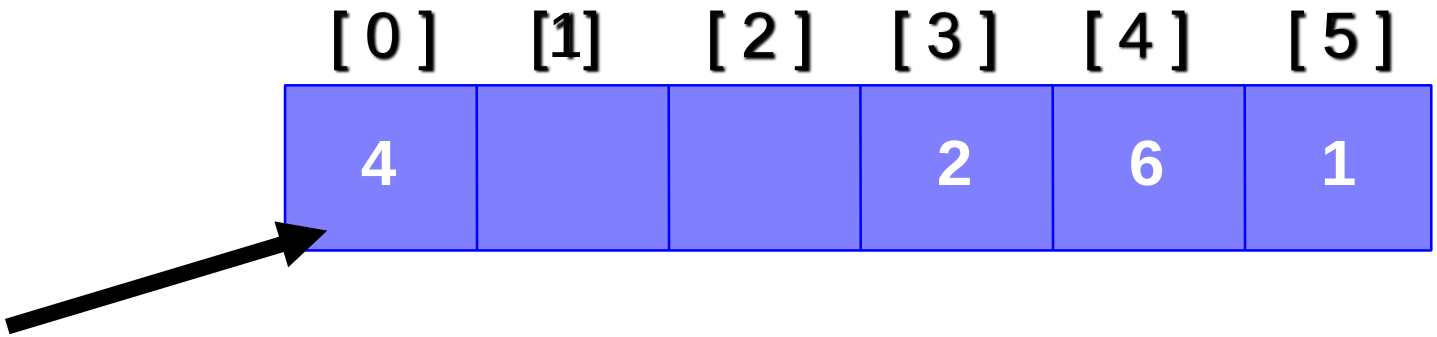




Array Implementation

- The new element goes at the front of the array (if that spot isn't already used):

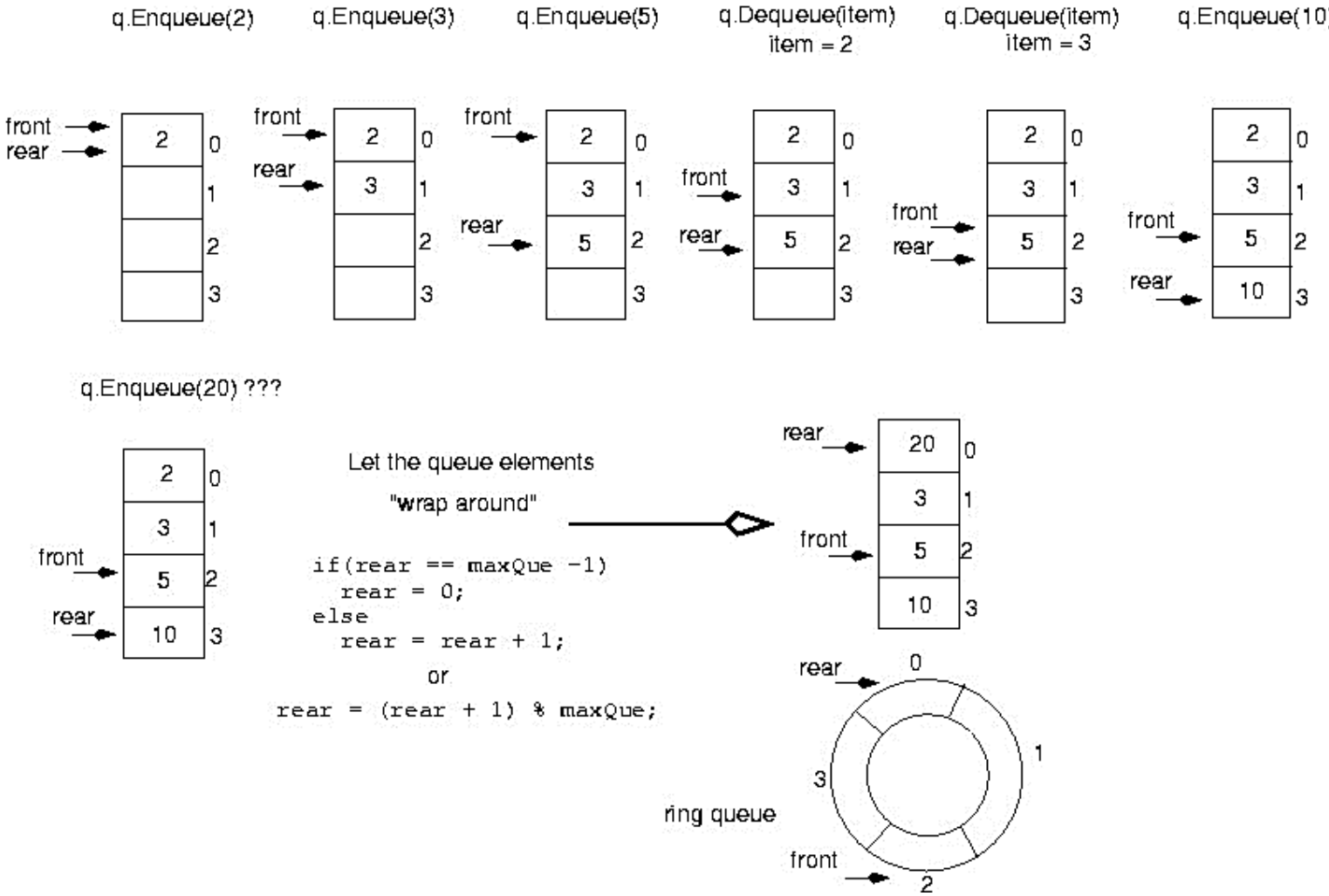




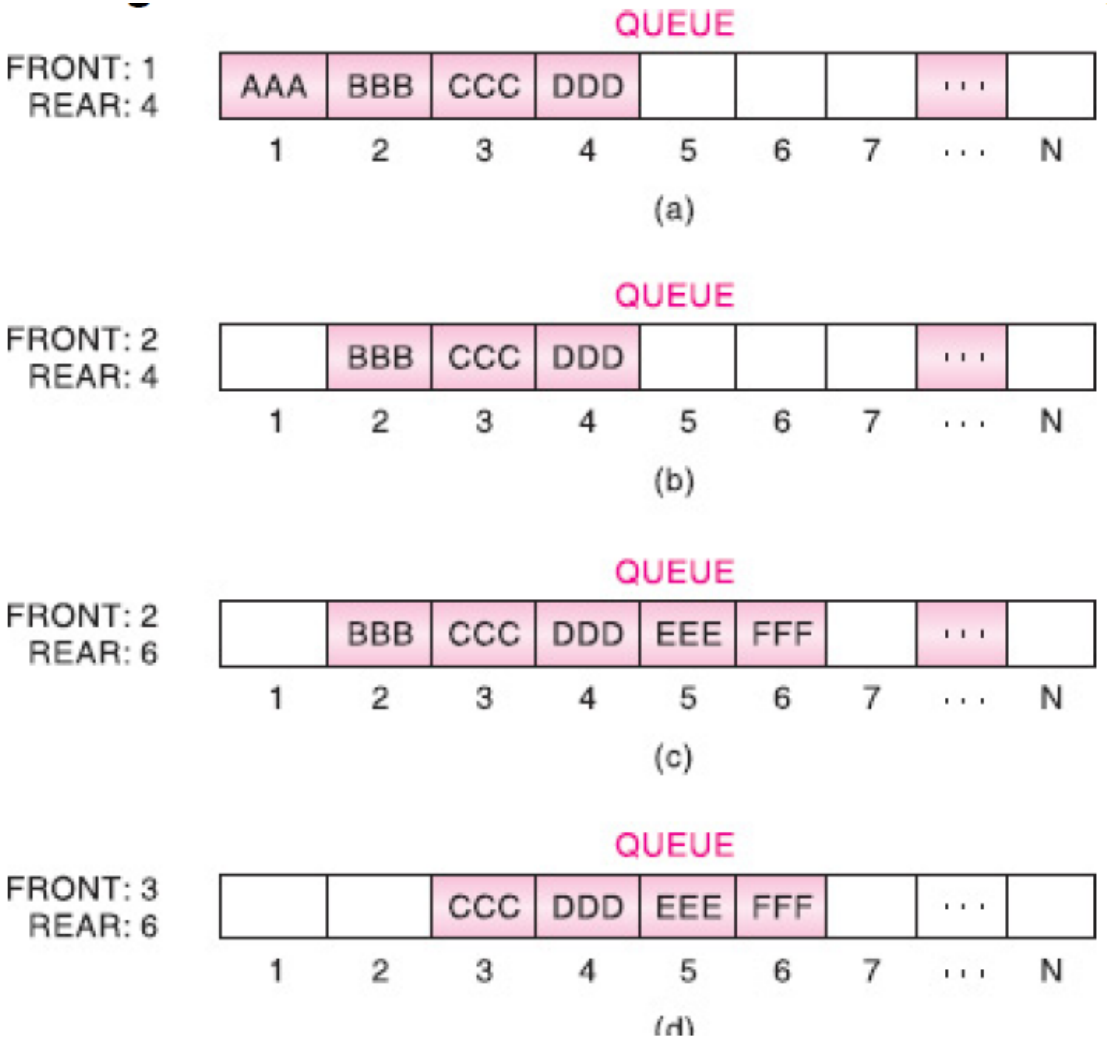
Array Implementation

- Easy to implement
- But it has a limited capacity with a fixed array
- Or you must use a dynamic array for an unbounded capacity
- Special behaviour is needed when the rear reaches the end of the array.

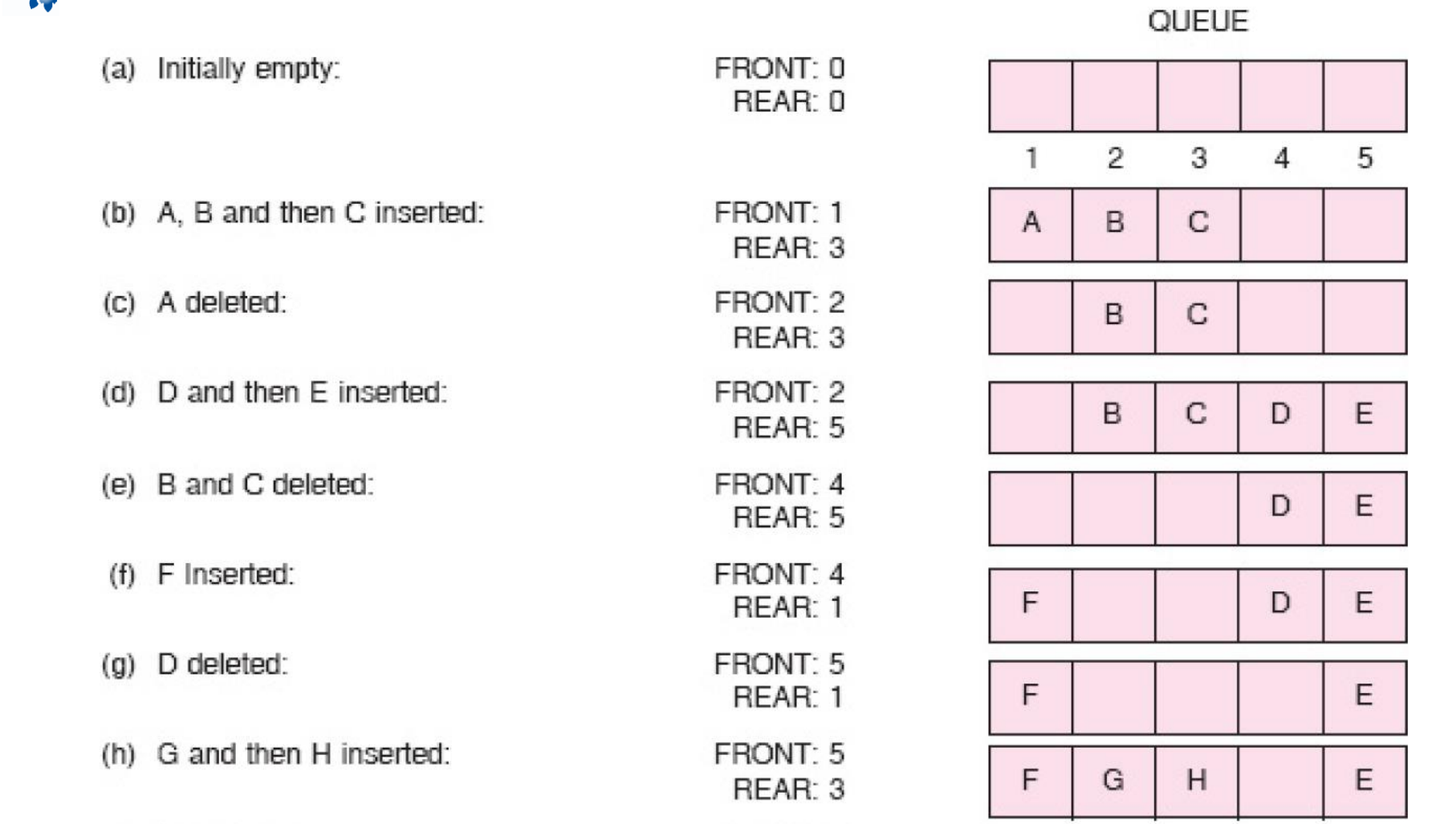
Queue Operations



Queue Operations

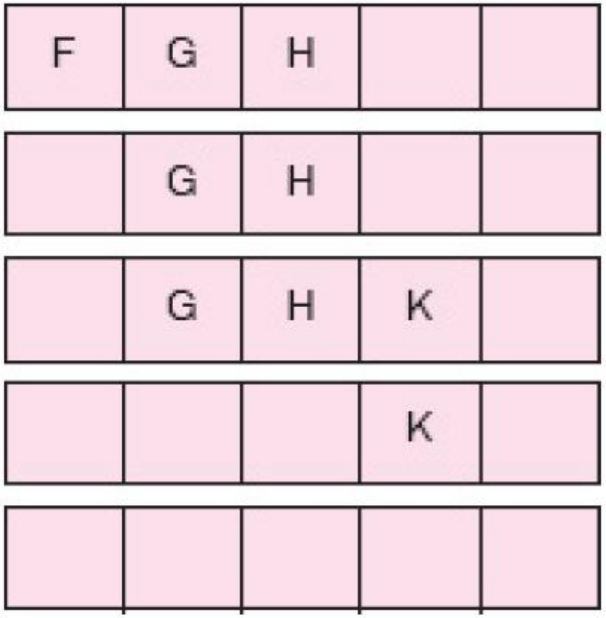


Sample Problems on Queue



Sample Problems on Queue

(i) E deleted:	FRONT: 1 REAR: 3	
(j) F deleted:	FRONT: 2 REAR: 3	
(k) K inserted:	FRONT: 2 REAR: 4	
(l) G and H deleted:	FRONT: 4 REAR: 4	
(m) K deleted, QUEUE empty:	FRONT: 0 REAR: 0	



Sample Problems on Queue

6.22 Consider the following queue of characters, where QUEUE is a circular array which is allocated six memory cells: FRONT = 2, REAR = 4 QUEUE: ___, A, C, D, ___, ___ (For notational convenience, we use "___" to denote an empty memory cell.) Describe the queue as the following operations take place: **(a)** F is added to the queue.

(b) two letters are deleted.

(c) K, L and M are added to the queue.

(d) two letters are deleted.

(e) R is added to the queue.

(f) two letters are deleted.

(g) S is added to the queue.

(h) two letters are deleted.

(i) one letter is deleted.

(j) one letter is deleted.

Sample Problems on Queue

(a) F is added to the rear of the queue, yielding

FRONT = 2, REAR = 5 QUEUE: ___, A, C, D, F, ___

(b) The two letters, A and C, are deleted, leaving

FRONT = 4, REAR = 5 QUEUE: ___, ___, ___, D, F, ___

(c) K, L and M are added to the rear of the queue. Since K is placed in the last memory cell of QUEUE, L and M are placed in the first two memory cells. This yields FRONT = 4, REAR = 2 QUEUE: L, M, ___, D, F, K

(d) The two front letters, D and F are deleted, leaving

FRONT = 6, REAR = 2 QUEUE: L, M, ___, ___, ___, K

Sample Problems on Queue

(e) R is added to the rear of the queue, yielding

FRONT = 6, REAR = 3 QUEUE: L, M, R, ___, ___, K

(f) The two front letters, K and L, are deleted, leaving

FRONT = 2, REAR = 3 QUEUE: ___, M, R, ___, ___, ___

(g) S is added to the rear of the queue, yielding

FRONT = 2, REAR = 4 QUEUE: ___, M, R, S, ___, ___

(h) The two front letters, M and R, are deleted, leaving

FRONT = 4, REAR = 4 QUEUE: ___, ___, ___, S, ___, ___

(i) The front letter S is deleted. Since FRONT = REAR, this means that the queue is empty; hence we assign NULL to FRONT and REAR. Thus FRONT = 0, REAR = 0 QUEUE: ___, ___, ___, ___, ___, ___

(j) Since FRONT = NULL, no deletion can take place. That is, underflow has occurred.

Queue Container

- Queue
  - Allows access only at the front and rear of the sequence
  - Items enter at the rear and exit from the front
  - Example: waiting line at a grocery store
  - FIFO ordering (first-in first-out)
  - *push(add* object to a queue)
  - *pop* (remove object from queue)

Queue Container

Method	Definition
empty()	Returns whether the queue is empty. It return true if the queue is empty otherwise returns false.
size()	Returns the size of the queue.
swap()	Exchange the contents of two queues but the queues must be of the same data type, although sizes may differ.
emplace()	Insert a new element into the queue container, the new element is added to the end of the queue.
front()	Returns a reference to the first element of the queue.
back()	Returns a reference to the last element of the queue.
push(g)	Adds the element ‘g’ at the end of the queue.
pop()	Deletes the first element of the queue.

Queue Container

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    queue<int> qu;
    qu.push(21);
    qu.push(22);
    qu.push(24);
    qu.push(25);

    int num=5;
    qu.push(num);

    qu.pop();
    qu.pop();
    qu.pop();

    while (!qu.empty()) {
        cout << qu.front() <<" ";
        qu.pop();
    }
}
```

Example Problems Practice using Queue

- A *palindrome* is a string that reads the same forward and backward.

Able was I ere I saw Elba

- We will read the line of text into both a stack and a queue.
- Compare the contents of the stack and the queue character-by-character to see if they would produce the same string of characters.

Example Problems Practice using Queue

```
#include <iostream.h>
#include <ctype.h>
#include "stack.h"
#include "queue.h“
int main()
{
 StackType<char> s;
 QueType<char> q;
 char ch;
 char sItem, qItem;
 int mismatches = 0;
```

```
 cout << "Enter string: " << endl;

 while(cin.peek() != '\\n') {

   cin >> ch;
   if(isalpha(ch)) {

     if(!s.IsFull())
       s.Push(toupper(ch));

     if(!q.IsFull())
       q.Enqueue(toupper(ch));
   }
}
```

Example Problems Practice using Queue

```
while( (!q.IsEmpty()) && (!s.IsEmpty()) ) {

  s.Pop(sItem);
  q.Dequeue(qItem);

  if(sItem != qItem)
    ++mismatches;
}
if (mismatches == 0)
  cout << "That is a palindrome" << endl;
else
 cout << That is not a palindrome" << endl;

return 0;
}
```

Queue Swap

```
#include <queue>
#include <iostream>
using namespace std;

int main(){
    queue<int> qu1;
    queue<int> qu2;

    // pushing elements into first queue
    qu1.push(1);
    qu1.push(2);
    qu1.push(3);
    qu1.push(4);

    // pushing elements into 2nd queue
    qu2.push(3);
    qu2.push(5);
    qu2.push(7);
    qu2.push(9);

// swap elements of queue
 qu1.swap(qu2);

 // printing the first queue
    cout<<"Queue1 = ";
    while (!qu1.empty()) {
        cout<<qu1.front()<<" ";
        qu1.pop();
    }

 // printing the second queue
    cout<<endl<<"Queue2 = ";
    while (!qu2.empty()) {
        cout<<qu2.front()<<" ";
        qu2.pop();
    }
    return 0;
}
```

Priority Queues

- Retrieve the most “interesting” element
  - Elements are given **priorities**
  - Retrieve the element with the **highest priority**
  - Several elements may have the same priority

- Examples:
  - Emergency room
    - Highest priority = most severe condition
  - Processes in an OS
    - Highest priority = *well, it’s complicated*
  - Homework due
    - Highest priority = …

Priority Queues

```
#include <iostream>
#include <queue>
using namespace std;
int main() {
    priority_queue<int> qu;
    qu.push(21);
    qu.push(22);
    qu.push(24);
    qu.push(25);

    qu.pop();

    while (!qu.empty()) {
        cout << qu.top() <<" ";
        qu.pop();
    }
}
```

Queue Problem Practice

Given a line of text, reverse the text without reversing the individual words.

For example,

Input: Technical Interview Preparation

Output: Preparation Interview Technical

Queue Problem Practice

Given a string, str, the task is to remove all the duplicate adjacent characters from the given string.

Input: str= “azxxzy”
Output: ay
Removal of “xx” modifies the string to “azzy”.
Now, the removal of “zz” modifies the string to “ay”.
Since the string “ay” doesn’t contain duplicates, output→ ay

Input: “aaccdd”
Output: Empty String

Queue Problem Practice

Given two strings s1 and s2, let us assume that while typing the strings there were some backspaces encountered which are represented by #. The task is to determine whether the resultant strings after processing the backspace character would be equal or not.

Input: s1= geee#e#ks, s2 = gee##eeks
Output: True

Input: s1 = equ#ual, s2 = ee#quaal#
Output: False


Thank You!