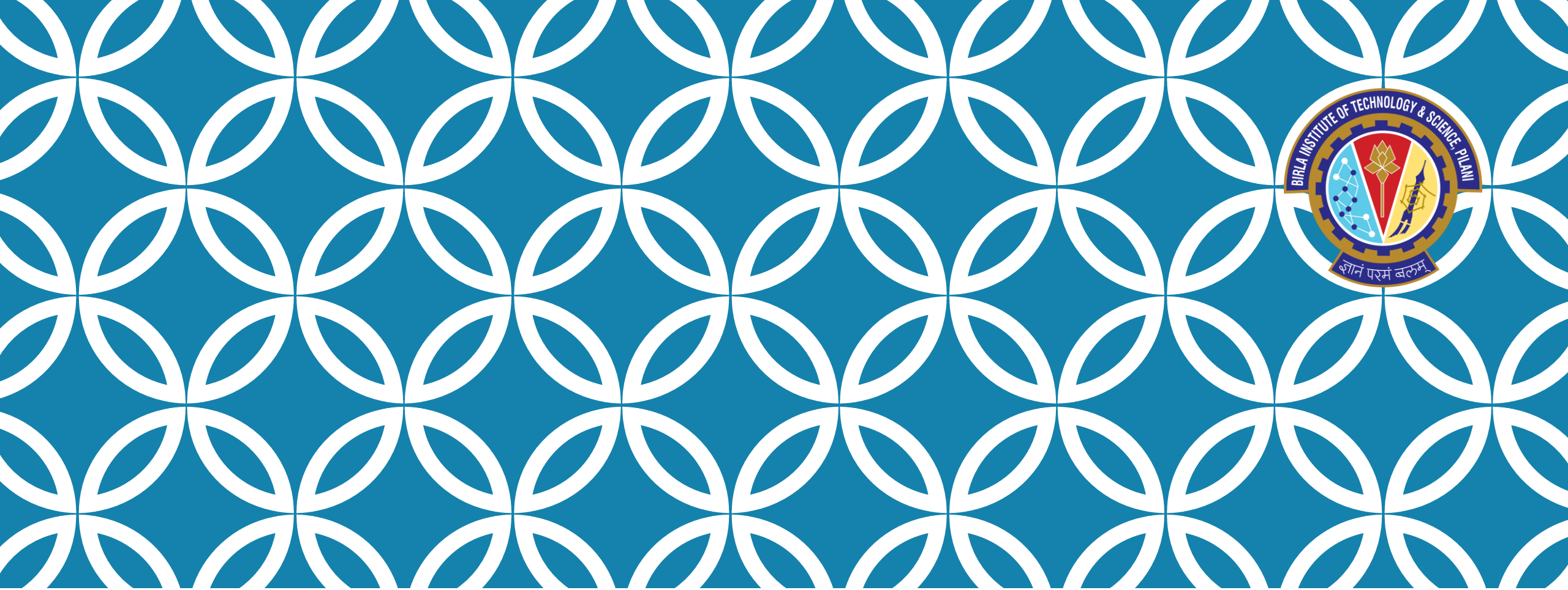


BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) QUEUE ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

QUEUE ADT USING CIRCULAR LINKED-LIST

```
typedef string Elem;
class LinkedQueue {
public:
  LinkedQueue();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(QueueEmpty);
  void enqueue(const Elem& e);
  void dequeue() throw(QueueEmpty);
private:
  CircleList C;
  int n;
};
```

(Class structure for Linked queue)

QUEUE IMPLEMENTATION USING CIRCULAR LINKED-LIST IN C++

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

class CircleList;

typedef string Elem;
class CNode {
private:
    Elem elem;
    CNode* next;
    friend class CircleList;
};

class CircleList {
public:
    CircleList();
    ~CircleList();
    bool empty() const;
    const Elem& front() const;
    const Elem& back() const;
    void advance();
    void add(const Elem& e);
    void remove();
    void traverse();
private:
    CNode* cursor;
};
```

```cpp
CircleList::CircleList()
    : cursor(NULL) { }
CircleList::~CircleList()
    { while (!empty()) remove(); }

bool CircleList::empty() const
    { return cursor == NULL; }
const Elem& CircleList::back() const
    { return cursor->elem; }
const Elem& CircleList::front() const
    { return cursor->next->elem; }
void CircleList::advance()
    { cursor = cursor->next; }

void CircleList::add(const Elem& e) {
    CNode* v = new CNode;
    v->elem = e;
    if (cursor == NULL) {
        v->next = v;
        cursor = v;
    }
    else {
        v->next = cursor->next;
        cursor->next = v;
    }
}
```

```
1
23
Enqueing 23
1
56
Enqueing 56
1
78
Enqueing 78
3
Getting front element
23
4
Getting size
3
5
Queue is not empty
2
Dequeing
3
Getting front element
56
```

```cpp
LinkedQueue::LinkedQueue()
: C(), n(0) { }

int LinkedQueue::size() const
{ return n; }

bool LinkedQueue::empty() const
{ return n == 0; }

const Elem& LinkedQueue::front() {
    if (empty())
        cout<<"front of empty queue\n";
    return C.front();
}

void LinkedQueue::enqueue(const Elem& e) {
    C.add(e);
    C.advance();
    n++;
}

void LinkedQueue::dequeue() {
    if (empty())
        cout<<"dequeue of empty queue\n";
    C.remove();
    n--;
}
```

Lab 7: next week's lab

STACK USING TWO QUEUES: MAKING PUSH COSTLY

Algorithm 1: Push

```
Data: two queues: q1, q2
      element to push to stack: E
Result: element E is now at the head of queue q1
if q1.isEmpty() then
    q1.enqueue(E);
else
    q1Size:= q1.size();
    for i=0...q1Size do
        q2.enqueue(q1.dequeue());
    end
    q1.enqueue(E);
    for j=0...q1Size do
        q1.enqueue(q2.dequeue());
    end
end
```

Algorithm 2: Pop

```
Data: two queues: q1, q2
Result: the head element of the queue q1 is removed
element:= q1.dequeue();
return element;
```

What is the time complexity?

The STL Queue:

```
#include <queue>
using std::queue;
queue<float> myQueue;
```

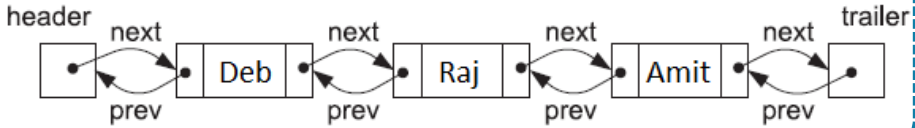
size(), empty(), push(e), pop(), front(), back()

Lab 7: next week's lab

Alternate way: making pop() costly…

Is it possible to do using only one queue?

DEQUE IMPLEMENTATION



(A Doubly linked-list with Sentinels)

```
typedef string Elem;
class LinkedDeque {
public:
  LinkedDeque();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(DequeEmpty);
  const Elem& back() const throw(DequeEmpty);
  void insertFront(const Elem& e);
  void insertBack(const Elem& e);
  void removeFront() throw(DequeEmpty);
  void removeBack() throw(DequeEmpty);
private:
  DLinkedList D;
  int n;
};
```

```
void LinkedDeque::insertFront(const Elem& e) {
  D.addFront(e);
  n++;
}

void LinkedDeque::insertBack(const Elem& e) {
  D.addBack(e);
  n++;
}

void LinkedDeque::removeFront() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeFront of empty deque");
  D.removeFront();
  n--;
}

void LinkedDeque::removeBack() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeBack of empty deque");
  D.removeBack();
  n--;
}
```

```
1
10
Inserting 10 to the front
1
30
Inserting 30 to the front
1
60
Inserting 60 to the front
2
20
Inserting 20 to the end
2
40
Inserting 40 to the end
5
Front element
60
3
Removing from front
5
Front element
30
7
Queue is not empty
8
Size of queue : 4
```

Complexity of Operations?

A QUEUE USING TWO STACKS

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Queue {
    stack<int> s1, s2;
    void enQueue(int x)
    {
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
        s1.push(x);
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
    }
    int deQueue() {
        if (s1.empty()) {
            cout << "Q is Empty";
            exit(0);
        }
        int x = s1.top();
        s1.pop();
        return x;
    }
};
```

```cpp
int main() {
    Queue q;
    q.enQueue(10);
    q.enQueue(20);
    cout << q.deQueue() << '\n';
    q.enQueue(30);
    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';
    q.enQueue(40);
    cout << q.deQueue() << '\n';
    return 0;
}
```

```
10
20
30
40


...Program finished with exit code 0
Press ENTER to exit console.
```

DOUBLE-ENDED QUEUE ADT: DEQUE

- A queue-like data structure that supports insertion and deletion at both the front and the rear of the queue.
- Applications: Work-Stealing algorithm in Intel's parallel programming, etc.
- insertFront(), front(), eraseFront(), insertBack(), back(), eraseBack()

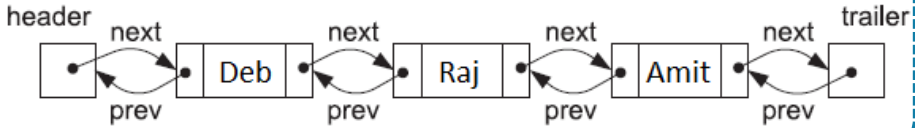
```
#include <deque>
using std::deque;
deque<string> myDeque;
```

(The STL deque)

size(), empty(),
push_front(e),
push_back(e), pop_front(),
pop_back(), front(), back()

What are some of the scenarios where Deque operations might be applicable?

DEQUE IMPLEMENTATION



(A Doubly linked-list with Sentinels)

```
typedef string Elem;
class LinkedDeque {
public:
  LinkedDeque();
  int size() const;
  bool empty() const;
  const Elem& front() const throw(DequeEmpty);
  const Elem& back() const throw(DequeEmpty);
  void insertFront(const Elem& e);
  void insertBack(const Elem& e);
  void removeFront() throw(DequeEmpty);
  void removeBack() throw(DequeEmpty);
private:
  DLinkedList D;
  int n;
};
```

```
void LinkedDeque::insertFront(const Elem& e) {
  D.addFront(e);
  n++;
}

void LinkedDeque::insertBack(const Elem& e) {
  D.addBack(e);
  n++;
}

void LinkedDeque::removeFront() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeFront of empty deque");
  D.removeFront();
  n--;
}

void LinkedDeque::removeBack() throw(DequeEmpty) {
  if (empty())
    throw DequeEmpty("removeBack of empty deque");
  D.removeBack();
  n--;
}
```

```
1
10
Inserting 10 to the front
1
30
Inserting 30 to the front
1
60
Inserting 60 to the front
2
20
Inserting 20 to the end
2
40
Inserting 40 to the end
5
Front element
60
3
Removing from front
5
Front element
30
7
Queue is not empty
8
Size of queue : 4
```

Complexity of Operations?