

Queue Implementations

Circular Queues

buffer of fixed capacity
improvements and cost estimates

Dequeues

the double ended queue
queue as double linked circular list

- 1 **Circular Queues**
buffer of fixed capacity
improvements and cost estimates
- 2 **Dequeues**
the double ended queue
queue as double linked circular list

MCS 360 Lecture 17
Introduction to Data Structures
Jan Verschelde, 1 October 2010

Queue Implementations

Circular Queues

buffer of fixed capacity

improvements and cost estimates

Dequeues

the double ended queue

queue as double linked circular list

- 1 **Circular Queues**
buffer of fixed capacity
improvements and cost estimates
- 2 **Dequeues**
the double ended queue
queue as double linked circular list

a circular queue

Circular Queues

buffer of fixed
capacity

improvements and
cost estimates

Dequeues

the double ended
queue

queue as double
linked circular list

A queue can be linear or circular.

Applications for circular queues:

- waiting room with fixed #seats,
- buffer to process data.

We first use an array to implement a queue

- 1 two indices, to front and back element,
- 2 update: $(\text{index} + 1) \bmod \text{array size}$.
→ modulo capacity, indices only increase

private data members

Circular Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
#ifndef MCS360_CIRCULAR_FIXED_BUFFER_H
#define MCS360_CIRCULAR_FIXED_BUFFER_H

namespace mcs360_circular_fixed_buffer
{
    template <typename T>
    class Queue
    {
    private:

        T *data;
        size_t capacity; // capacity of buffer
        int current;     // index to front of queue
        int back;       // index to end of queue
        size_t number;  // number of elements
    };
};
```

public methods

Circular
Queues

buffer of fixed
capacity

improvements and
cost estimates

Dequeues

the double ended
queue

queue as double
linked circular list

```
public:
    Queue( int c );
    // creates an empty queue
    void push( T item );
    // pushes the item at the end
    bool empty();
    // return true if queue is empty
    T front();
    // returns the front element
    void pop();
    // removes the front element
};
}
#include "mcs360_circular_fixed_buffer.tc"
#endif
```

constructor and push

Circular Queues

buffer of fixed capacity

improvements and cost estimates

Dequeues

the double ended queue

queue as double linked circular list

```
#include "mcs360_circular_fixed_buffer.h"
#include <iostream>
namespace mcs360_circular_fixed_buffer
{
    template <typename T>
    Queue<T>::Queue( int c )
    {
        capacity = c; data = new T[c];
        current = -1; back = -1; number = 0;
    }
    template <typename T>
    void Queue<T>::push( T item )
    {
        this->back
            = (this->back + 1) % this->capacity;
        this->data[this->back] = item;
        this->number = this->number + 1;
        if(this->current < 0) this->current = 0;
    }
}
```

rest of methods

Circular
Queues

buffer of fixed
capacity

improvements and
cost estimates

Dequeues

the double ended
queue

queue as double
linked circular list

```
template <typename T>
bool Queue<T>::empty()
{
    return (this->number == 0);
}
template <typename T>
T Queue<T>::front()
{
    return data[this->current];
}
template <typename T>
void Queue<T>::pop()
{
    this->current
        = (this->current + 1) % this->capacity;
    this->number = this->number - 1;
}
```

testing the buffer

Circular
Queuesbuffer of fixed
capacityimprovements and
cost estimates

Dequeues

the double ended
queuequeue as double
linked circular list

```
#include <iostream>
#include "mcs360_circular_fixed_buffer.h"
using namespace mcs360_circular_fixed_buffer;
using namespace std;

int main()
{
    Queue<int> q(10);

    for(int i=1; i<6; i++) q.push(i);
    for(; !q.empty(); q.pop())
        cout << q.front() << endl;
    for(int i=6; i<12; i++) q.push(i);
    for(; !q.empty(); q.pop())
        cout << q.front() << endl;
    for(int i=12; i<18; i++) q.push(i);
    for(; !q.empty(); q.pop())
        cout << q.front() << endl;
```

Queue Implementations

Circular Queues

buffer of fixed capacity

improvements and cost estimates

Dequeues

the double ended queue

queue as double linked circular list

1 Circular Queues

buffer of fixed capacity

improvements and cost estimates

2 Deques

the double ended queue

queue as double linked circular list

Circular
Queues

buffer of fixed
capacity

improvements and
cost estimates

Dequeues

the double ended
queue

queue as double
linked circular list

improving...

The implementation is simple and efficient:

- management of indices straightforward,
- efficient if queue size \approx capacity.

All operations have cost $O(1)$.

Suggestions for improvement:

- throw exceptions for when pop empty or push to full queue;
- enlarge capacity when full.

Queue Implementations

Circular Queues

buffer of fixed capacity
improvements and cost estimates

Dequeues

the double ended queue
queue as double linked circular list

- 1 Circular Queues
buffer of fixed capacity
improvements and cost estimates
- 2 Deques
the double ended queue
queue as double linked circular list

the Deque

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

A deque is a double ended queue:
we can pop from the front or the end,
and push to front or to the end.

Methods of STL deque, on `deque<T> q`:

- `q.push_back(t)`: append to queue at end
- `q.push_front(t)`: insert to queue at front
- `t = q.back()`: return last element of queue
- `t = q.front()`: return first element of queue
- `q.pop_back()`: remove last element of queue
- `q.pop_front()`: remove first element of queue

Queue Implementations

Circular Queues

buffer of fixed capacity
improvements and cost estimates

Dequeues

the double ended queue
queue as double linked circular list

- 1 Circular Queues
buffer of fixed capacity
improvements and cost estimates
- 2 Deques
the double ended queue
queue as double linked circular list

implementing a deque

Circular Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

We adapt our `mcs360_double_list::List`,
using the `Node` definition of `mcs360_double_node.h`.

The goal is to implement a deque:

- circular: we can circulate
- doubly linked: move forward & backward.

If we allow to push and pop to the front and end of the queue, the queue is double ended, or a deque.

the data members

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
#ifndef MCS360_CIRCULAR_DOUBLE_RING_H
#define MCS360_CIRCULAR_DOUBLE_RING_H
```

```
#define NULL 0
```

```
namespace mcs360_circular_double_ring
```

```
{
```

```
    template <typename T>
```

```
    class Queue
```

```
    {
```

```
        private:
```

```
            #include "mcs360_double_node.h"
```

```
            Node *current; // pointer to current node
```

```
            int number; // number of elements
```

the Node definition

From lecture 12: mcs360_double_node.h:

```
#ifndef DNODE_H
#define DNODE_H

struct Node
{
    T data; // T is template parameter
    Node *next; // pointer to next node
    Node *prev; // pointer to previous node

    Node(const T& item,
         Node* next_ptr = NULL,
         Node* prev_ptr = NULL) :
        data(item), next(next_ptr), prev(prev_ptr) {}
};

#endif
```

public methods

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

public:

```
Queue(); // returns an empty queue

bool empty();
// true if queue empty, false otherwise
int size();
// returns the size of the queue
T front();
// returns the item at front of queue

void move_front_forward();
// makes next element front of queue
void move_front_backward();
// makes previous item front of queue
```

pushing and popping

Circular Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
void push_front(T item);  
// inserts item in front of queue  
void push_back(T item);  
// appends item to the end  
  
void pop_front();  
// removes the element at front  
void pop_back();  
// removes the element at end  
  
};  
}  
#include "mcs360_circular_double_ring.tc"  
#endif
```

Notice the grouping of the methods.

first group of methods

Circular
Queues

buffer of fixed
capacity

improvements and
cost estimates

Dequeues

the double ended
queue

queue as double
linked circular list

```
namespace mcs360_circular_double_ring
{
    template <typename T>
    Queue<T>::Queue()
    {
        current = NULL;
        number = 0;
    }
    template <typename T>
    bool Queue<T>::empty()
    {
        return (current == NULL);
    }
    template <typename T>
    int Queue<T>::size()
    {
        return number;
    }
}
```

first group continued

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
template <typename T>
T Queue<T>::front()
{
    return current->data;
}
template <typename T>
void Queue<T>::move_front_forward()
{
    current = current->next;
}
template <typename T>
void Queue<T>::move_front_backward()
{
    current = current->prev;
}
```

appending to the queue

Circular Queues

buffer of fixed capacity

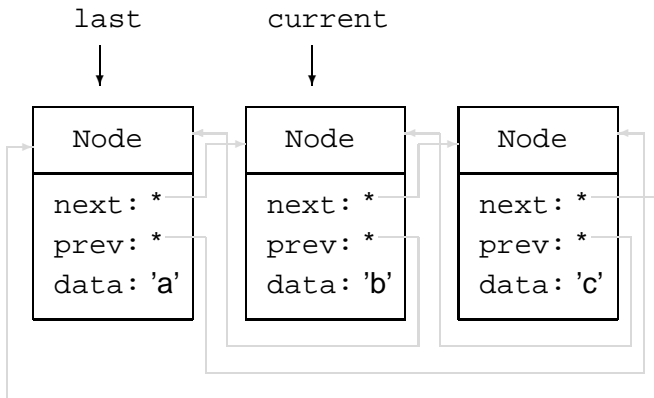
improvements and cost estimates

Dequeues

the double ended queue

queue as double linked circular list

Appending 'd' to a queue that stores 'a', 'b', 'c':



1 Oct 2010

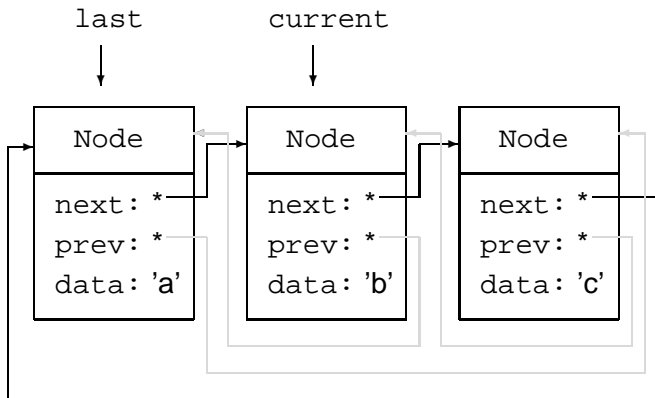
appending to the queue

Circular
Queuesbuffer of fixed
capacityimprovements and
cost estimates

Dequeues

the double ended
queuequeue as double
linked circular list

Appending 'd' to a queue that stores 'a', 'b', 'c':



1 Oct 2010

appending to the queue

Circular Queues

buffer of fixed capacity

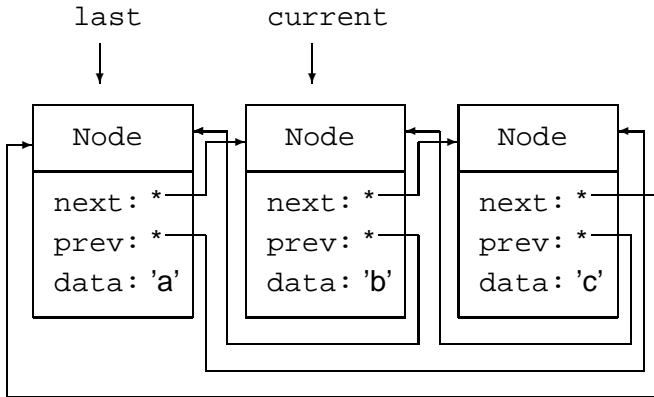
improvements and cost estimates

Dequeues

the double ended queue

queue as double linked circular list

Appending 'd' to a queue that stores 'a', 'b', 'c':



appending to the queue

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
template <typename T>
void Queue<T>::push_back(T item)
{
    if(current == NULL)
    {
        current = new Node(item);
        current->next = current;
        current->prev = current;
    }
    else
    {
        Node *last = current->prev;
        current->prev
            = new Node(item, current, current->prev);
        last->next = current->prev;
    }
    number = number + 1;
}
```

removing the front

Circular
Queues

buffer of fixed
capacity
improvements and
cost estimates

Dequeues

the double ended
queue
queue as double
linked circular list

```
template <typename T>
void Queue<T>::pop_front()
{
    if(current != NULL) {
        if(current->prev == current->next)
        {
            delete current;
            current = NULL; number = 0;
        }
        else {
            Node *last = current->prev;
            last->next = current->next;
            current->next->prev = last;
            delete current;
            current = last->next;
            number = number - 1;
        }
    }
}
```

Summary + Assignments

Circular Queues

buffer of fixed capacity
improvements and cost estimates

Dequeues

the double ended queue
queue as double linked circular list

More on Chapter 6 on queue implementations.

Assignments:

- 1 Use the STL list in a templated class to implement a queue. Define an exception class `Queue_Empty` and illustrate how to operate your queue implementation with a test program.
- 2 Give code for `push_front` on our circular doubly linked list implementation for a deque. Make a drawing to illustrate the logic of the code.
- 3 Give code for `pop_back` on our circular doubly linked list implementation for a deque. Make a drawing to illustrate the logic of the code.