COL100 Lecture 20

Note Title 11-10-2018

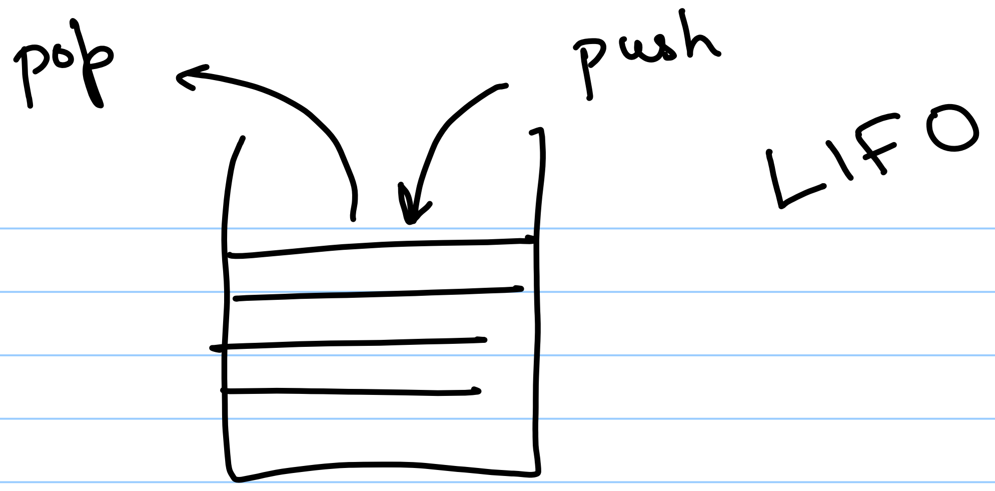
Review:

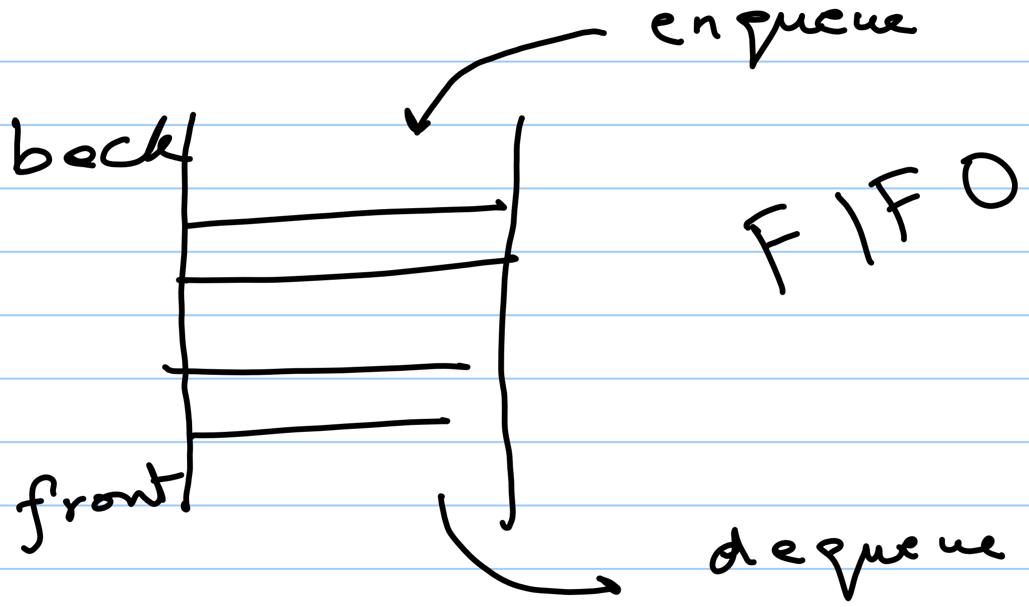
Stack<__>

Last-in First-out ← All O(1)
- s.push(value)
- s.pop() returns value and pops
- s.peek() returns value
- s.size()

Queue<__>

First in First out ← All O(1)
- q.enqueue(value)
- q.dequeue()
- q.peek()
- q.size()


pop
push
LIFO


enqueue
back
FIFO
front
dequeue

cout << q; ✓

```
Queue <int>    q;

for (int   i = 1;  i<= 6;  i++)
{
        q.enqueue (i);
 }
 int  qsize = q.size();
 for (int  i = 0;  i < ~~q.size()~~ qsize;  i++)
 {
        cout << q.dequeue() << " ";
 }
 cout << "q.size() = " << q.size() << endl;
```

Output : 1 2 3 q.size() = 3

Iteration 0 :

i = 0

q.size() : 6

i < q.size() : TRUE

q. f{ 1 2 3 4 5 6 }b

Iteration 1

i = 1

q.size() = 5

i < q.size() : TRUE

q: f{ 2 3 4 5 6 }b

Iteration 2

i = 2

q.size() = 4

i < q.size() : TRUE

q: f{ 3 4 5 6 }b

Iteration 3 :

i = 3

q.size() = 3

i < q.size() : FALSE

Another
solution:

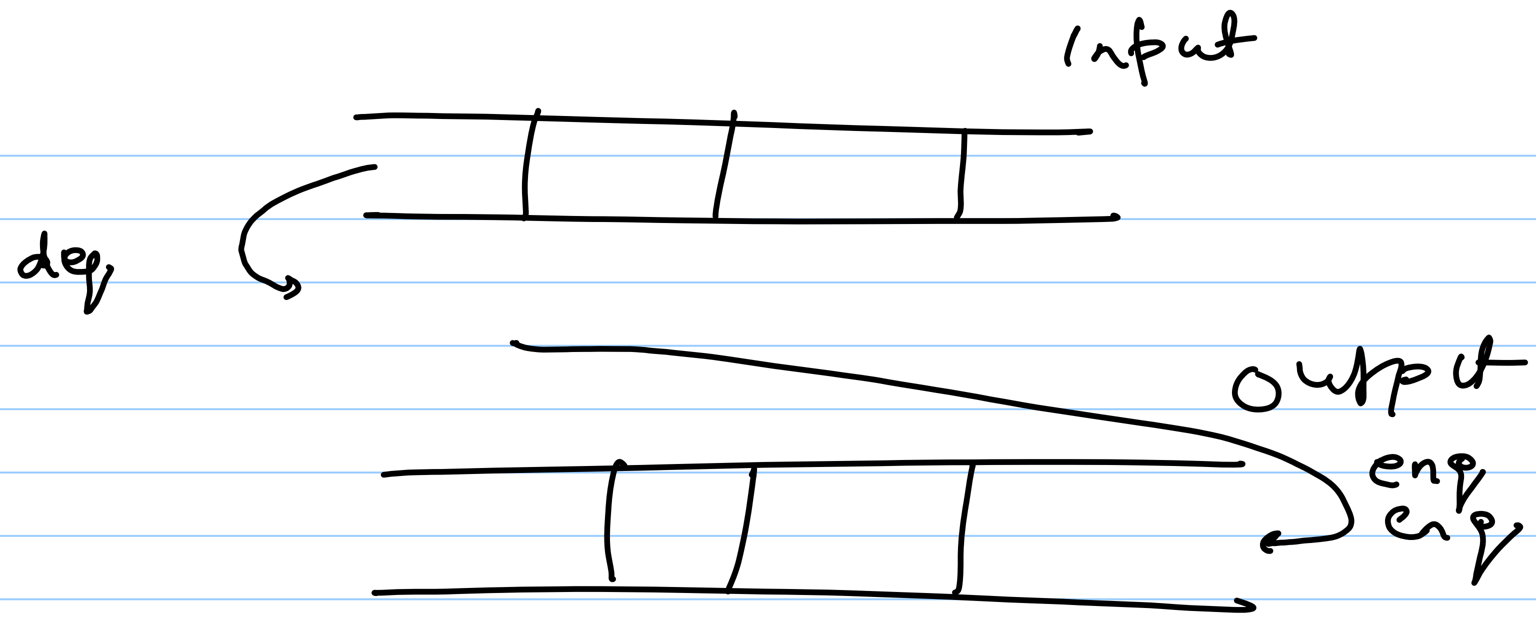
```
while (q.size() > 0)
{
        cout << q.dequeue() << " ";
}
```

Exercise:

Write a function "<u>duplicate</u>" that accepts a queue of integers and returns a new queue with every element with two copies of itself.

eg. Input : f { 1 , 2 , 3 } b

Output : f { 1, 1, 2, 2, 3, 3 } b

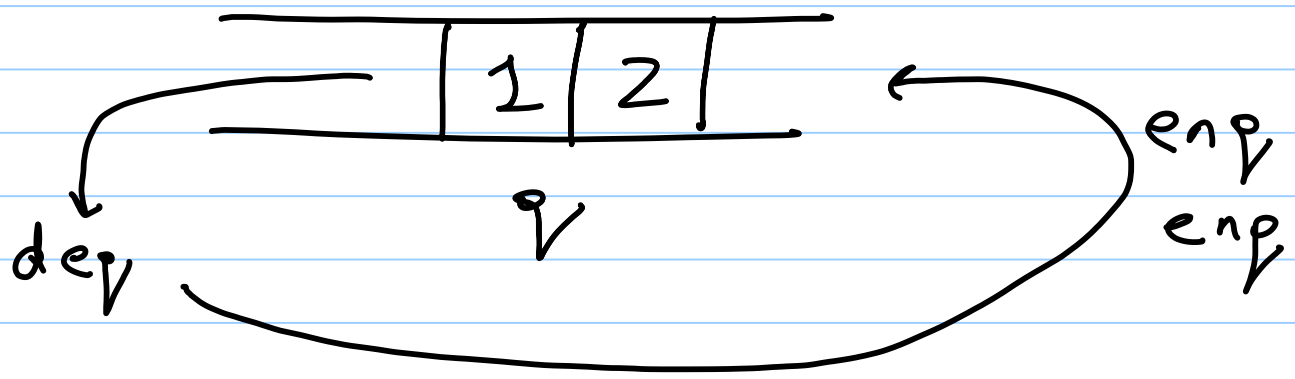

Input
deq
output
enq
enq

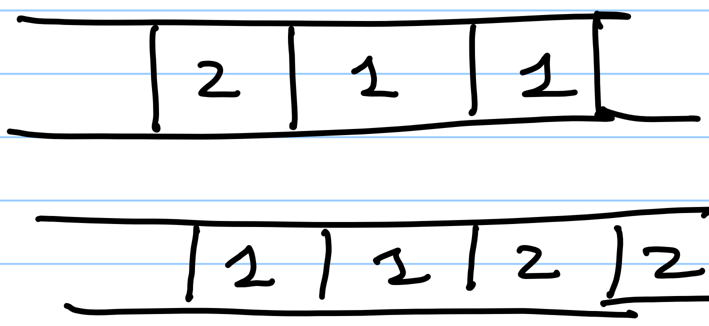
```
Queue<int>  duplicate( Queue<int>  iq )
{     int ipsize = iq.size();
      Queue<int>   oq;   // f{ }b
      for (int i = 0;   i < iq size  , i++)
      {
              int  x =  iq. deque(); ✓
               oq. enqueue (x);
               oq. enqueue (x);
      }

      return  oq;

}
```

```
Queue<int>    duplicate (Queue<int>   iq)
{          Queue<int> oq;
          while  (iq.size()  >0)
          { int x= iq. dequeue();
                oq.enqueue(x);
                 oq. enqueue(x);
          }

           return oq ;
 }
```

change the question: change the input queue instead

```
void   duplicate_in_place(Queue<int> & q)
{
        int   qsize = q.size();
        for (int  i=0;  i< qsize;   i++)
        {
            int x = q.dequeue();
                q.enqueue(x);
                q.enqueue(x);
        }
}
```


1
2
q
deq
enq
enq


2
1
1
1
1
2
2

~~Sow~~ Be careful about :

→ If you are counting elements in a queue, store them in a variable first

eg.

```
int qsize = q.size();
for (int i = 0; i < qsize; i++)
{
    // do something with q

}
```

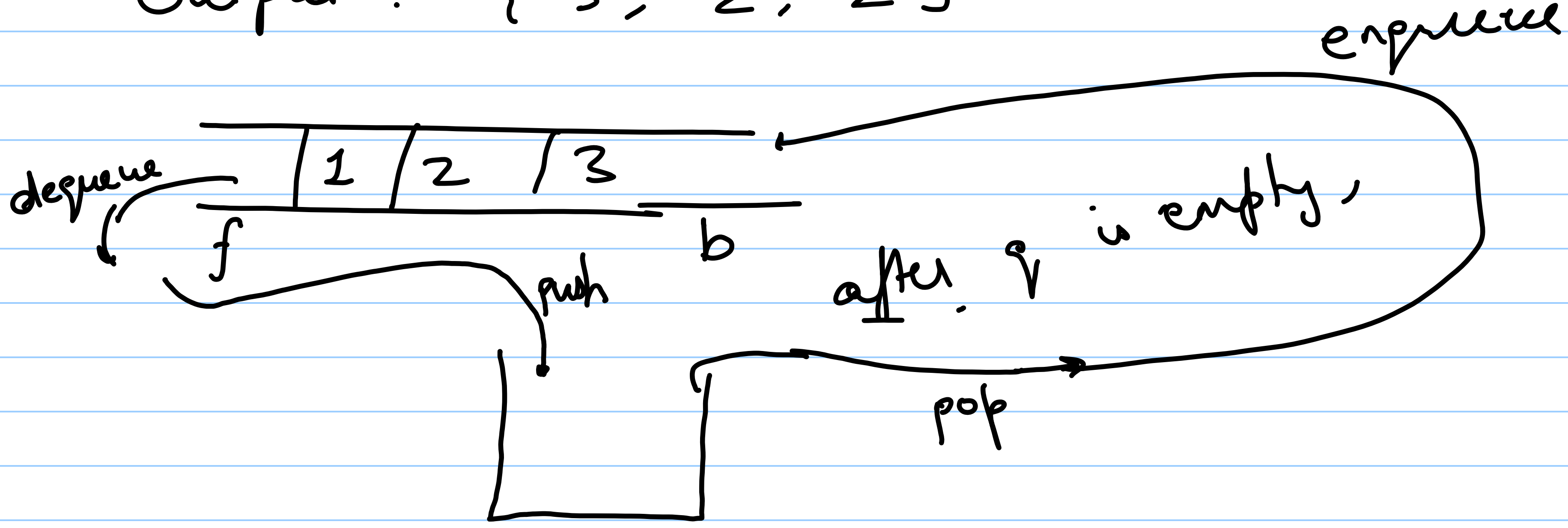
Another pattern:

```
while ( ! q. isEmpty() )
{
        // do something with
        //   q. dequeue() ;
 }
```

Exercise: Write a method to Reverse the elements in a q

Input: { 1, 2, 3 }

Output: { 3, 2, 1 }

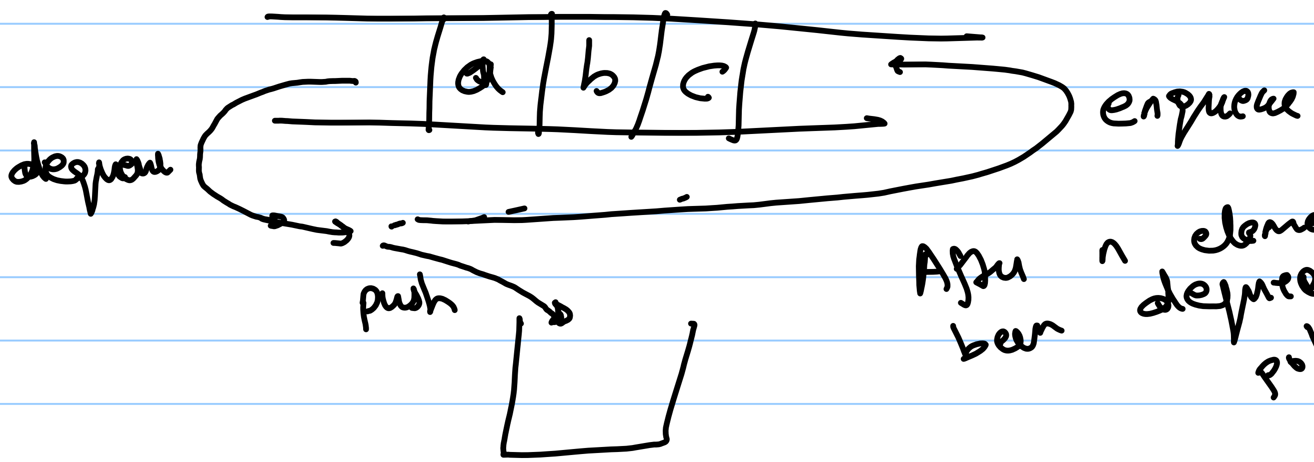


```
void    reverse( Queue<int >      & q)
{
        Stack<int> s ;
         while   (! q.isEmpty( ) )
         {
                int   x=  q.dequeue();
                 s.  push(x);
         }
         while ( ! s.is Empty() )
         {
               int    x=   s.pop();
               q. enqueue( x);
          }
 }
```

Exercise: write a method "mirror" that appends the elements of the q to itself in reverse order.

Input: { "a", "b", "c" }

Output: { "a", "b", "c", "c", "b", "a" }



After n elements have been dequeued, pop the stack and enqueue

```
void    mirror (Queue <int>   &q)
{
        Stack  <int>   s;
        int  qsize =  q.size();
        for (int  i=0;  i< qsize;  i++)
        {
             int  x =  q.dequeue(); // O(1)
             s.push(x);             // O(1)
             q.enqueue(x);          // O(1)
        }
O(n)    while (!s.isEmpty())
        {
             int  x =  s.pop();
             q.enqueue(x);
        }
}
```

Do

Debugging

Steps:

→ Determine that you have a "bug"

→ Isolate the bug's location

- Try and find the smallest input that reproduces the bug

Debugger

Breakpoint :

program will pause at that line of code

foo.c

```
        0   # include <iostream>
        1
        2   int   foo ( int  a )
      → 3   {
        4         a = a + 2;
        5         return a;
        6   }
start   7   int   main ()
        8   {
        9         cout << "hello" ;
       10         cout << foo (10);
       11         return 0;
       12   }
```

GDB
GNU Debugger

```
(gdb) breakpoint
        foo.c : 3

(gdb) run
hello
Breakpoint reached
(gdb) print a
10
```