Programming Languages and Techniques (CIS120)

Lecture 15
Feb 13, 2013

Linked Queues II
Announcements

• Homework 5 (queues) on the web
  – due Thursday, Feb 21st at midnight

• Midterm 1 will be in class on Friday, February 15th
  – ROOMS:
    • Towne 100 (here) last names: A – K
    • Cohen G17 last names: L – Z
  – TIME: 11:00a.m. sharp, 50 mins
  – Covers up to Feb 6th

• Labs today and tomorrow are review

• Review session 6-8PM tonight Wu & Chen
Queues

First-in first-out mutable data structures
module type QUEUE =
sig
  (* type of the data structure *)
  type 'a queue

  (* Make a new, empty queue *)
  val create : unit -> 'a queue

  (* Determine if the queue is empty *)
  val is_empty : 'a queue -> bool

  (* Add a value to the tail of the queue *)
  val enq : 'a -> 'a queue -> unit

  (* Remove the head value and return it (if any) *)
  val deq : 'a queue -> 'a

end
Data Structure for Mutable Queues

```ocaml
type 'a qnode = {
  v: 'a;
  mutable next : 'a qnode option
}

type 'a queue = { mutable head : 'a qnode option;
  mutable tail : 'a qnode option }
```

There are two parts to a mutable queue:

- the “internal nodes” of the queue with links from one to the next
- the head and tail references themselves

All of the references are options so that the queue can be empty (and so that the links can terminate).
Queues in the Heap

An empty queue

A queue with one element

A queue with two elements
Visual Shorthand: Abbreviating Options

An empty queue

A queue with one element

A queue with three elements

Val means
Some means None
Linked Queue Invariants

• Just as we imposed some restrictions on which trees are legitimate Binary Search Trees, Linked Queues must also satisfy some invariants:

Either:
(1) head and tail are both None (i.e. the queue is empty)

or
(2) head is Some n1, tail is Some n2 and
   - n2 is reachable from n1 by following ‘next’ pointers
   - n2.next is None

• We can check that these properties rule out all of the “bogus” examples.

• A queue operation may assume that these invariants hold of its inputs, and must ensure that the invariants hold when it’s done.
“Bogus” values of type int queue

head is None, tail is Some n

head is Some, tail is None

tail is not reachable from the head
More bogus int queues

The links contain a cycle!

tail doesn’t point to the last element of the queue
Implementing Linked Queues

LinkedQ.ml
create and is_empty

(* create an empty queue *)
let create () : 'a queue =
{ head = None;
  tail = None }

(* determine whether a queue is empty *)
let is_empty (q:'a queue) : bool =
  q.head = None

• **create establishes** the queue invariants  
  – both head and tail are None

• **is_empty assumes** the queue invariants  
  – it doesn’t have to check that q.tail is None
The code for `enq` is informed by the queue invariant:

- either the queue is empty, and we just update head and tail, or
- the queue is non-empty, in which case we have to “patch up” the “next” link of the old tail node to maintain the queue invariant.

```ocaml
let enq (x: 'a) (q: 'a queue) : unit =
  let newnode = {v=x; next=None} in
  begin match q.tail with
    | None ->
      (* Note that the invariant tells us that q.head is also None *)
      q.head <- Some newnode;
      q.tail <- Some newnode
    | Some n ->
      n.next <- Some newnode;
      q.tail <- Some newnode
    end
```
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> ...
    | Some n -> ...
  end

head

tail

v

next
Calling Enq on a non-empty queue

```ocaml
let newnode = {v=x; next=None}

match q.tail with
| None -> ...
| Some n -> ...
end
```
Calling Enq on a non-empty queue

```haskell
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> …
    | Some n -> …
  end
```

- Workspace: `2 q`
- Stack:
  - `enq`
  - `q`
- Heap:
  - `head`
  - `tail`
  - `v 1`
  - `next`
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ...
    | Some n -> ...
  end

head
tail

v
next
Calling Enq on a non-empty queue

Workspace

2

Stack

enq

q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None} in
  begin
    match q.tail with
    | None -> ...
    | Some n -> ...
  end

head

tail

v

l

next
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
let newnode = {v=x; next=none}
in begin match q.tail with
| None -> ...
| Some n -> ...
end
Calling Enq on a non-empty queue

Workspace

```
let newnode = {v=x; next=None} in
begin match q.tail with
  | None ->
    q.head <- Some newnode;
    q.tail <- Some newnode
  | Some n ->
    n.next <- Some newnode;
    q.tail <- Some newnode
end
```

Stack

- `enq`
- `q`

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None} in
  begin match q.tail with
    | None -> ...
    | Some n -> ...
  end
```

```
workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None} in
  begin match q.tail with
    | None -> ...
    | Some n -> ...
  end
```
Calling Enq on a non-empty queue

Workspace

```plaintext
let newnode = {v=x; next=None} in
begin match q.tail with
  | None ->
    q.head <- Some newnode;
    q.tail <- Some newnode
  | Some n ->
    n.next <- Some newnode;
    q.tail <- Some newnode
end
```

Stack

```
enq
q
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None} in
  begin match q.tail with
    | None -> ...
    | Some n -> ...
  end
```

```
head
x 2
```

```
tail
q
```

```
v 1
next
```
Calling Enq on a non-empty queue

Workspace

```haskell
let newnode = \{v=2; next=None\} in
begin match q.tail with
  | None ->
    q.head <- Some newnode;
    q.tail <- Some newnode
  | Some n ->
    n.next <- Some newnode;
    q.tail <- Some newnode
end
```

Stack

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>enq</td>
</tr>
<tr>
<td>q</td>
</tr>
</tbody>
</table>
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = \{v=x; next=None\} in
  begin match q.tail with
    | None -> ...
    | Some n -> ...
  end
```

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>head</td>
</tr>
<tr>
<td>tail</td>
</tr>
<tr>
<td>q</td>
</tr>
</tbody>
</table>
```

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
</tr>
<tr>
<td>next</td>
</tr>
</tbody>
</table>
```

(CIS120 / Spring 2013)
Calling Enq on a non-empty queue

Workspace

```
let newnode = {v=2; next=None} in
begin match q.tail with
| None ->
  q.head <- Some newnode;
  q.tail <- Some newnode
| Some n ->
  n.next <- Some newnode;
  q.tail <- Some newnode
end
```

Stack

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None} in
  begin match q.tail with
  | None -> ...
  | Some n -> ...
  end
```

Heap

```
head

x 2

q

v 1

next
```
Calling Enq on a non-empty queue

Workspace:

```
let newnode = in
  begin match q.tail with
  | None ->
    q.head <- Some newnode;
    q.tail <- Some newnode
  | Some n ->
    n.next <- Some newnode;
    q.tail <- Some newnode
  end
```

Stack:
- `enq`:
- `q`

Heap:
- `fun (x: 'a) (q: 'a queue) ->`
  - `let newnode = {v=x; next=None}`
  - `in begin match q.tail with`
    - `| None -> ...`
    - `| Some n -> ...`
  - `end`

- `head`:
- `tail`:
- `v`:
  - `1`
  - `next`:
- `q`:
- `x`:
  - `2`
- `q`:

Note: there is no “Some bubble” this is a qnode not a qnode option.
Calling Enq on a non-empty queue

Workspace

\[
\text{let newnode = \_ \_ in} \\
\text{begin match } q.\text{tail with} \\
\text{| None } \rightarrow \\
\text{q.head } \leftarrow \text{ Some newnode}; \\
\text{q.tail } \leftarrow \text{ Some newnode} \\
\text{| Some n } \rightarrow \\
\text{n.next } \leftarrow \text{ Some newnode}; \\
\text{q.tail } \leftarrow \text{ Some newnode} \\
\text{end}
\]

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
\text{let newnode = } \{v=x; next=None\}
\text{in begin match } q.\text{tail with} \\
\text{| None } \rightarrow \text{ ...} \\
\text{| Some n } \rightarrow \text{ ...} \\
\text{end}

head

\[
\begin{array}{c}
\text{v} \\
\text{next}
\end{array}
\]

tail

\[
\begin{array}{c}
v \\
\text{next}
\end{array}
\]

\[
\begin{array}{c}
x \\
2
\end{array}
\]

\[
\begin{array}{c}
q
\end{array}
\]
Calling Enq on a non-empty queue

Workspace

begin match q.tail with
| None -> q.head <- Some newnode;
| Some n -> n.next <- Some newnode;
end

Stack

enq

q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
in begin match q.tail with
  | None -> ...
  | Some n -> ...
end

head

tail

q

newnode

v

next

v

next

v

next

1

2

2

1
Calling Enq on a non-empty queue

```
begin match q.tail with
| None ->
  q.head <- Some newnode;
  q.tail <- Some newnode
| Some n ->
  n.next <- Some newnode;
  q.tail <- Some newnode
end
```

Workspace

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> ...
    | Some n -> ...
  end
```

Stack

Heap
Calling Enq on a non-empty queue

Workspace

```
begin match q.tail with
    | None ->
      q.head <- Some newnode;
      q.tail <- Some newnode
    | Some n ->
      n.next <- Some newnode;
      q.tail <- Some newnode
end
```

Stack

```
| enq |
| q  |
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> ... 
    | Some n -> ...
end
```

```
| head |
| tail |
| q    |
```

```
| newnode |
| v       |
| next    |
```

```
| v       |
| next    |
```
Calling Enq on a non-empty queue

Workspace

begin match q.tail with
| None ->
  q.head <- Some newnode;
  q.tail <- Some newnode
| Some n ->
  n.next <- Some newnode;
  q.tail <- Some newnode
end

Stack

<table>
<thead>
<tr>
<th>enq</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
  | None -> ...
  | Some n -> ...
end

head

<table>
<thead>
<tr>
<th>x</th>
<th>2</th>
</tr>
</thead>
</table>

tail

<table>
<thead>
<tr>
<th>q</th>
</tr>
</thead>
</table>

newnode

<table>
<thead>
<tr>
<th>v</th>
<th>1</th>
</tr>
</thead>
</table>

next

<table>
<thead>
<tr>
<th>v</th>
<th>2</th>
</tr>
</thead>
</table>

next
Calling Enq on a non-empty queue

Workspace

begin match   with
   | None ->
   | Some n ->
      n.next <- Some newnode;
   | Some n ->
      q.tail <- Some newnode
end

Stack

enq

q

Heap

fun (x: 'a) (q: 'a queue) ->
    let newnode = {v=x; next=None}
    in begin match q.tail with
       | None -> ...
       | Some n -> ...
    end

head

tail

x 2

q

newnode

v 1

next

v 2

next
Calling Enq on a non-empty queue

Workspace

```
begin match with
    | None ->
    | q.head <- Some newnode;
    | q.tail <- Some newnode
    | Some n ->
    | n.next <- Some newnode;
    | q.tail <- Some newnode
end
```

Stack

```
begin
    enq
    q
(____)
head
x 2
tail
q
newnode
eq
2
next
v
1
next
v
2
next
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
    let newnode = {v=x; next=None}
    in begin match q.tail with
        | None -> ...
        | Some n -> ...
    end
```

CIS120 / Spring 2013
Calling Enq on a non-empty queue

Workspace

```
begin match -- with
| None ->
| | q.head <- Some newnode;
| | q.tail <- Some newnode
| Some n ->
| | n.next <- Some newnode;
| | q.tail <- Some newnode
| end
```

Stack

```
enq
q

(____)
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
let newnode = {v=x; next=None}
in begin match q.tail with
| None -> ...
| Some n -> ...
end
```

```
head
|   |
| v 1
| next

tail
|   |
| v 2
| next

newnode
|   |
| v 1
| next

x 2

v 1

x 2

v 2
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> …
    | Some n -> …
  end

begin match q with
  | None ->
  q.head <- Some newnode;
  q.tail <- Some newnode
  Some n ->
  n.next <- Some newnode;
  q.tail <- Some newnode
end
Calling Enq on a non-empty queue

Workspace

n.next <- Some newnode;
q.tail <- Some newnode

Stack

enq
q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=Some None}
in begin match q.tail with
  None -> ...
  Some n -> ...
end

Note: n points to a qnode, not a qnode option.
Calling Enq on a non-empty queue

Workspace

\[
\begin{align*}
n.\text{next} & \leftarrow \text{Some newnode}; \\
q.\text{tail} & \leftarrow \text{Some newnode}
\end{align*}
\]

Stack

\[
\begin{array}{c}
enq \\
q
\end{array}
\]

Heap

\[
\text{fun } (x: \text{'a}) (q: \text{'a queue}) \rightarrow \\
\text{let newnode } = \{v=x; \text{next=None}\} \\
\text{in begin match q.tail with} \\
| \text{None } \rightarrow \ldots \\
| \text{Some n } \rightarrow \ldots \\
\text{end}
\]

Workspace: Stack: Heap:

- $n$.next <- Some newnode;
- q.tail <- Some newnode

```
fun (x: 'a) (q: 'a queue) ->
let newnode = {v=x; next=None}
in begin match q.tail with
| None -> ... 
| Some n -> ...
end
```
Calling Enq on a non-empty queue

Workspace

.enxxt <- Some newnode;
q.tail <- Some newnode

Stack

.enq
q

Heap

fun (x: 'a) (q: 'a queue) ->
let newnode = {v=x; next=None}
in begin match q.tail with
  | None -> …
  | Some n -> …
end

head
x 2
tail
q
newnode
v 1
next
n
v 2
next
Calling Enq on a non-empty queue

Workspace

```
   .next <- Some newnode;
   q.tail <- Some newnode
```

Stack

```
  enq
  q

(____)
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
  | None -> …
  | Some n -> …
  end

head
  v 1
  next

newnode
  v 2
  next
```

CIS120 / Spring 2013
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ...
    | Some n -> ...
  end

Workspace:

Stack:

Heap:

```haskell
next <- Some ;
q.tail <- Some newnode

let newnode = {v=x; next=None}
in begin
  match q.tail with
  | None -> …
  | Some n -> …
end
```
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    None -> ...
    Some n -> ...
  end

head

tail

q

newnode

v 1

next

v 2

next

v 2

next

Workspace

next <- Some ⬤;
q.tail <- Some newnode

c

Stack

enq

q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    None -> ...
    Some n -> ...
  end

head

tail

q

newnode

v 1

next

v 2

next

Workspace

next <- Some ⬤;
q.tail <- Some newnode

CIS120 / Spring 2013
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ...
    | Some n -> ...
  end

head

x 2

q

tail

newnode

v 1

next

n

v 2

next
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
  match q.tail with
  | None -> …
  | Some n -> …
  end

head

tail

x 2

q

c.tail <- Some newnode

enq

q
Calling Enq on a non-empty queue

Workspace

\[
() \;;
q.\text{tail} \leftarrow \text{Some newnode}
\]

Stack

fun (x: 'a) (q: 'a queue) \to
\begin{aligned}
&\text{let newnode} = \{v=x; \text{next=}\text{None}\} \\
&\text{in begin match q.tail with} \\
&\quad | \text{None} \to \ldots \\
&\quad | \text{Some n} \to \ldots \\
&\end{aligned}
\]

Heap

\[
\begin{array}{c}
\text{head} \\
\text{tail} \\
q \\
\text{newnode} \\
n \\
\end{array}
\]

\[
\begin{array}{c}
v \\
\text{next} \\
n \\
\end{array}
\]
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
let newnode = {v=x; next=None}
in begin match q.tail with
  | None -> …
  | Some n -> …
end

():
q.tail <- Some newnode

begin
  match q.tail with
  | None -> …
  | Some n -> …
end

head

x 2

q

newnode

v 1

next

v 2

next
Calling Enq on a non-empty queue

Workspace

q.tail <- Some newnode

Stack

enq
q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> …
    | Some n -> …
  end

head

x 2

q

newnode

v 1

next

n

v 2

next
Calling Enq on a non-empty queue

Workspace

g.tail <- Some newnode

Stack

<table>
<thead>
<tr>
<th>enq</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
</tr>
</tbody>
</table>

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
      None -> ...
      Some n -> ...
  end

head

tail

next

code: (___)

v 1
	node

v 2

Workspace

Stack

Heap

Workspace

Stack

Heap

Workspace

Stack

Heap
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> …
    | Some n -> …
  end

CIS120 / Spring 2013
Calling Enq on a non-empty queue

Workspace

```
.tail <- Some newnode
```

Stack

```
enq
q
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> …
    | Some n -> …
  end
```

```
head

\[ x \ 2 \]

\[ tail \]

\[ q \]

\[ newnode \]

\[ v \ 1 \]

\[ next \]

\[ n \]

\[ v \ 2 \]

\[ next \]
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> …
    | Some n -> …
  end
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> ...
    | Some n -> ...
  end

head

 tail

q

c newnode

v 1

next

v 2

next
Calling Enq on a non-empty queue

Workspace: .tail <-

Stack:
- enq
- q

(____)

Heap:
- fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ...
    | Some n -> ...
  end

- head
- tail
- x 2
- q
- newnode
- v 1
- next
- n
- v 2
- next
Calling Enq on a non-empty queue

Workspace

Stack
- enq
- q

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> …
    | Some n -> …
  end

head
- v 1
- next

tail
- x 2
- q
- newnode

.next
- v 2
- next
Calling Enq on a non-empty queue

Workspace

Stack

Heap

fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> …
    | Some n -> …
  end

CIS120 / Spring 2013
Calling Enq on a non-empty queue

Workspace

Stack
- enq
- q
- ()

Heap
- fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin match q.tail with
    | None -> …
    | Some n -> …
  end

- head
- tail
- x 2
- q
- v 1
- next
- n
- v 2
- next

POP!
Calling Enq on a non-empty queue

<table>
<thead>
<tr>
<th>Workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>enq</td>
</tr>
<tr>
<td>q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun (x: 'a) (q: 'a queue) -&gt;</td>
</tr>
<tr>
<td>let newnode = {v=x; next=None}</td>
</tr>
<tr>
<td>in begin match q.tail with</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>head</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Calling Enq on a non-empty queue

Workspace

```
()
```

Stack

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ... 
    | Some n -> ...
  end
```

Heap

```
fun (x: 'a) (q: 'a queue) ->
  let newnode = {v=x; next=None}
  in begin
    match q.tail with
    | None -> ... 
    | Some n -> ...
  end
```

Notes:
- the enq function imperatively updated the structure of q
- the new structure still satisfies the queue invariants
The code for `deq` must also “patch pointers” to maintain the queue invariant:
- The head pointer is always updated to the next element in the queue.
- If the removed node was the last one in the queue, the tail pointer must be updated to None.