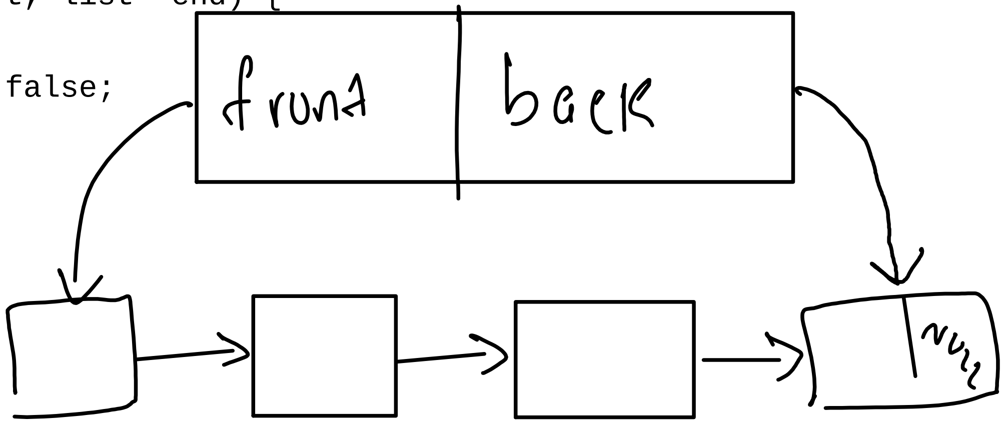
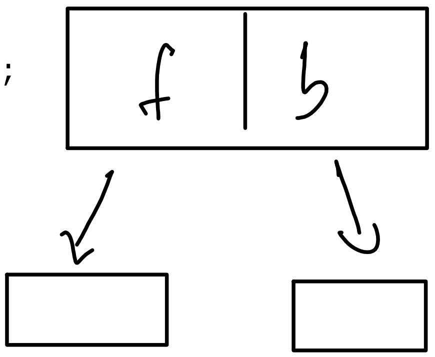
```
/* Queues, implemented with linked lists
 *
 * 15-122 Principles of Imperative Computation, Spring 2012
 * Frank Pfenning
 */

/* Interface to queues of strings */

typedef struct queue_header* queue;

bool queue_empty(queue Q);       /* O(1) */
queue queue_new();               /* O(1) */
void enq(queue Q, string s);     /* O(1) */
string deq(queue Q)              /* O(1) */
//@requires !queue_empty(Q);
  ;

/* Implementation of queues */

/* Aux structure of linked lists */
struct list_node {
  string data;
  struct list_node* next;
};
typedef struct list_node list;

/* is_segment(start, end) will diverge if list is circular! */
bool is_segment(list* start, list* end) {
  list* p = start;
  while (p != end) {
    if (p == NULL) return false;
    p = p->next;
  }
  return true;
}

/* Queues */

struct queue_header {
  list* front;
  list* back;
};

bool is_queue(queue Q) {
  if (Q == NULL) return false;
  if (Q->front == NULL || Q->back == NULL) return false;
  return is_segment(Q->front, Q->back);
}

bool queue_empty(queue Q)
//@requires is_queue(Q);
{
  return Q->front == Q->back;
}
```





Nothing is stopping us from creating this, which is why always checking is_queue() is so important.

```
queue queue_new()
//@ensures is_queue(\result);
//@ensures queue_empty(\result);
{
  queue Q = alloc(struct queue_header);
  list* p = alloc(struct list_node);
  Q->front = p;
  Q->back = p;
  return Q;
}

void enq(queue Q, string s)
//@requires is_queue(Q);
//@ensures is_queue(Q);
{
  list* p = alloc(struct list_node);
  Q->back->data = s;
  Q->back->next = p;
  Q->back = p;
  return;
}

string deq(queue Q)
//@requires is_queue(Q);
//@ensures is_queue(Q);
{
  string s = Q->front->data;
  Q->front = Q->front->next;
  return s;
}
```