# Concurrent Algorithms and Data Structures – Theory Assignment 2-3

Parosh Aziz Abdulla
Sarbojit Das
Firjoff Peer Stoldt

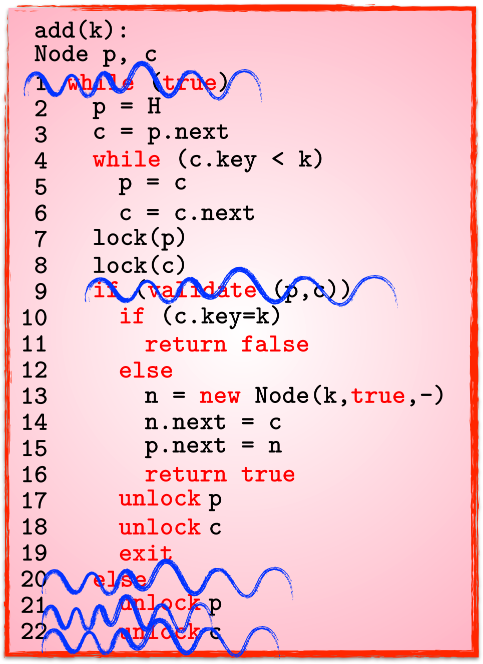
December 13, 2023

**Deadline: 2023–12–28**

**Please, submit your solutions in .pdf format.**

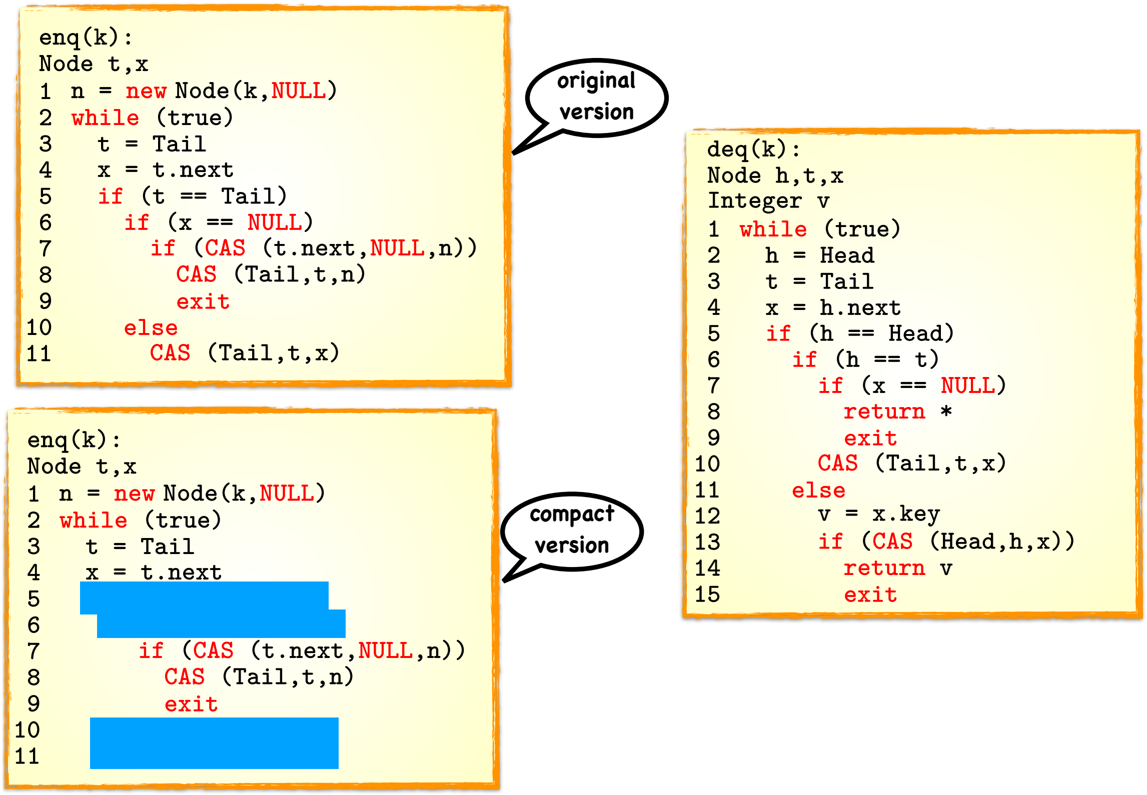
**Problem 1** Recall the `Optimistic List` algorithm. Suppose that the threads only perform `add` operations, i.e., no thread ever performs a `delete` or a `ctn` operation. However, we are still interested in guaranteeing two properties, namely that (i) we cannot add multiple copies of the same element to the list, and that (ii) we allow adding an element if we have not previously added the element to the list.
Assume that we remove the `validate` procedure form the code of the `add` operation, so we obtain the code depicted in Fig. 1. Is the resulting algorithm linearizable wrt. the two guarantees mentioned above? First, answer *yes* or *no*. If *yes* motivate your answer in no more than five lines. If *no* give a concurrent history that shows non-linearizability.

**Problem 2** Conisder the new version of the Michael-Scott algorithm depicted in Fig. 2. In the new algorithm we keep the dequeue part, but replace the enqueue part by a more compact code where we have removed some lines of code.
Is the resulting algorithm linearizable? First, answer *yes* or *no*. If *yes*, give the linearization policy, and motivate your answer in no more than five lines. If *no* give a concurrent history that shows non-linearizability.



```
add(k):
Node p, c
while (true)
  p = H
  c = p.next
  while (c.key < k)
    p = c
    c = c.next
  lock(p)
  lock(c)
  if (validate (p,c))
    if (c.key=k)
      return false
    else
      n = new Node(k,true,-)
      n.next = c
      p.next = n
      return true
    unlock p
    unlock c
    exit
  else
    unlock p
    unlock c
```

Figure 1: The `add` module optimistic list algorithm.



```
enq(k):
Node t,x
n = new Node(k,NULL)
while (true)
  t = Tail
  x = t.next
  if (t == Tail)
    if (x == NULL)
      if (CAS (t.next,NULL,n))
        CAS (Tail,t,n)
        exit
    else
      CAS (Tail,t,x)
```

```
enq(k):
Node t,x
n = new Node(k,NULL)
while (true)
  t = Tail
  x = t.next
[redacted]
      if (CAS (t.next,NULL,n))
        CAS (Tail,t,n)
        exit
[redacted]
```

```
deq(k):
Node h,t,x
Integer v
while (true)
  h = Head
  t = Tail
  x = h.next
  if (h == Head)
    if (h == t)
      if (x == NULL)
        return *
        exit
      CAS (Tail,t,x)
    else
      v = x.key
      if (CAS (Head,h,x))
        return v
        exit
```

Figure 2: The new version of the Michel-Scott algorithm.