Tutorial 7

Exercise 1

pre: $s = \langle a_1, a_2, \ldots, a_n \rangle$ and $x \notin s$

```
    s.push(x)
```

post: $s' = \langle a_1, a_2, \ldots, a_n, x \rangle$ and $x' = x$

pre: $s = \langle a_1, a_2, \ldots, a_n \rangle$ and $n \geq 1$

```
    res:=s.top
```

post: res $= a_n$ and $s' = s$

pre: $s = \langle a_1, a_2, \ldots, a_n \rangle$ and $n \geq 1$

```
    res:=s.pop
```

post: res $= a_n$ and $s' = \langle a_1, a_2, \ldots, a_{n-1} \rangle$

Exercise 2

```
push(x:ENTRY_TYPE) =
x.next_entry:=last
last:=x
end
```

```
top:ENTRY_TYPE =
return last
```

```
pop:ENTRY_TYPE =
res:=last
last:=last.next_entry
return res
```

Worst-case time complexity of all operations is $O(1)$.

Exercise 3

Let s be a temporary stack. We define the procedure reverse as follows.

```
  reverse(l:LIST) =
  s.make
  while (NOT l.empty) do
    x:=l.first
    s.push(x)
    l.delete(x)
  end while
  while (NOT s.empty) do
    l.insert_last(s.pop)
  end while
```

Worst-case time complexity of reverse(l) is $O(|l|)$ where $|l|$ is the number of entries in the list l.

Remark: To be absolutely correct we should also realize that ENTRY_TYPE in lists and stacks are not the same. Hence instead of

s.push(x)

we should rather write:

s.push(s.new_entry(x.value)).

Similarly instead of

l.insert_last(s.pop)

we should write

l.insert_last(l.new_entry(s.pop.value)).

Exercise 4

pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ and $x \notin q$

```
    q.enqueue(x)
```

post: $q' = \langle a_1, a_2, \ldots, a_n, x \rangle$ and $x' = x$

pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ and $n \geq 1$

```
    res:=q.front
```

post: res $= a_1$ and $q' = q$

pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ and $n \geq 1$

```
    res:=q.dequeue
```

post: res $= a_1$ and $q' = \langle a_2, a_3 \ldots, a_n \rangle$

Exercise 5

```
enqueue(x:ENTRY_TYPE) =
if (last = void) then x.next_entry := x
else x.next_entry:=last.next_entry; last.next_entry:=x
endif
last := x
end
```

```
front:ENTRY_TYPE =
return last.next_entry
```

```
dequeue:ENTRY_TYPE =
res:=last.next_entry
if last.next_entry = last then last:= void
else last.next_entry:=res.next_entry
endif
return res
```

Worst-case time complexity of all operations is $O(1)$.

Exercise 6

```
member(x:ENTRY_TYPE; q:QUEUE):boolean =
res:= false
if (NOT q.empty) then
    first_entry:=q.front
    repeat
        y:=q.dequeue
        if y=x then
            res:=true
        end if
        q.enqueue(y)
    until first_entry = q.front
end if
return res
```

Worst-case time complexity of member(x,q) is $O(|q|)$ where $|q|$ is the number of entries in q.