### 1)

**implement** Stack **of** T **where type** Stack **of** T = List **of** T

**fun** empty_stack() **ret** s: Stack s:= empty() **end fun**

**proc** push(**in** e: T, **in/out** s: Stack) addl(s, e) **end proc**

**fun** is_empty_stack(s: Stack ) **ret** b : bool b:= is_empty(s) **end fun**

**fun** top(s: Stack ) **ret** e : T e:= head(s) **end fun**

**proc** pop(**in/out** s: Stack) tail(s) **end proc**

### 2)

**implement** Stack **of** T **where type** Node **of** T = **tuple** elem: T next: **pointer to** (Node **of** T) **end tuple type** Stack **of** T = **pointer to** (Node **of** T )

**fun** empty_stack() **ret** s: Stack s:= null **end fun**

**proc** push(**in** e: T, **in/out** s: Stack) **var** p: **pointer to** Node alloc(p) p->elem:= e p->next:= s s:=p **end proc**

**fun** is_empty_stack(s: Stack ) **ret** b : bool b:= (s = null) **end fun**

**fun** top(s: Stack ) **ret** e : T e:= s->elem **end fun**

**proc** pop(**in/out** s: Stack) **var** p: **pointer to** Node p:= s s:= s->next free(s) **end proc**

### 3)

#### a)

**implement** Queue **of** T **where**

```
type Queue of T = tuple
  elems: array[0..N-1] of T
  size: nat
end tuple
fun empty_queue() ret q : Queue
  var a: array[1..N-1] of T
  q->elems:= a
  q->size:= 0
end fun
proc enqueue(in/out q: Queue of T, in e: T)
  q->size = q->size + 1
  for i := q->size to 2 do
    q->elems[i+1] = q->elems[i]
  od
  q->elems[1] := e
end proc
fun is_empty_queue(q: Queue of T) ret b : bool
  b:= (q->size = 0)
end fun
fun first(q: Queue of T) ret e : T
  e:= q->elems[q->size]
end fun
proc dequeue(in/out q: Queue of T)
  q->size = q->size - 1
end proc
```

**b)**

**implement** Queue **of** T **where**

```
type Queue of T = tuple
  elems: array[0..N-1] of T
  size: nat
  start: nat
end tuple
fun empty_queue() ret q : Queue
  var a: array[1..N-1] of T
  q->elems:= a
  q->size:= 0
  q->start:= 1
end fun
proc enqueue(in/out q: Queue of T, in e: T)
  q->size = q->size + 1
  q->elems[q->start + q->size] := e
end proc
fun is_empty_queue(q: Queue of T) ret b : bool
  b:= (q->size = 0)
end fun
fun first(q: Queue of T) ret e : T
  e:= q->elems[q->start]
end fun
proc dequeue(in/out q: Queue of T)
  q->start = q->start + 1
  q->size := q->size - 1
end proc
```