$$sexpr ::= (\text{atom}\ e_1\ \cdots\ e_n)\ \text{for}\ n \geq 1$$
$$e ::= \text{atom} \mid sexpr$$

(a (b c) d (e f g))

(+ 3 (* 5 6))

(hello there)

```
(library
 (public_name lab4)
 (libraries stdlib320)
 (flags (:standard -nostdlib
     -nopervasives -open Stdlib320)))
```

((a z) (b c) d (e f g))

invalid

```
"(defun foo (args x y) (+ (/ x y) (/ y x)))"
```

`tokenize`

Provided for you

```
[Lparen; Atom "defun"; Atom "foo";  ...  ; Rparen; Rparen; Rparen]
```

`ntree_of_toks`

Should fail if s-expression is not well-formed

```
Node ("defun", [
  Node ("foo", []);
  Node ("args", [Node ("x",[]); Node ("y",[])]);
  Node ("+", [
    Node ("/", [Node ("x",[]); Node ("y",[])] );
    Node ("/", [Node ("y",[]); Node ("x",[])] );
  ]);
])
```

mutually recursive

Recommended structure for *ntree_of_toks*:

a. Helper function that pulls the first e off a sequence of e's

$(\text{atom}\ e_1\ e_2\ \cdots\ e_n\ ) \rightarrow (e_1\ ,\ e_2 \cdots e_n)$

Iterate over tokens, count parentheses:
+1 for (-1 for) Stop at count 0.

b. Helper function that parses one e into a tree

$(\text{atom}\ e_1\ \cdots\ e_n) \rightarrow \text{Node}(\text{atom}, [t_1; \ldots; t_n])$

$\text{atom} \rightarrow \text{Node}(\text{atom}, [\,])$

Invoke (c) on $e_1\ e_2\ \cdots\ e_n\ )$

c. Helper function that parses a sequence of e's into a list of trees

$(\text{atom}\ e_1\ \cdots\ e_n\ ) \rightarrow [t_1; \ldots; t_n]$

Repeatedly invoke (a) to grab each e_i.
Invoke (b) with each e_i.
Stop when) is reached.

d. The main function that parses one $sexpr$ into a tree

Invoke (b), but reject result if it's just one atom

a. Helper function that pulls the first e off a sequence of e's

$$(\text{atom}\ e_1\ e_2\ \cdots\ e_n\ )\ \rightarrow\ (e_1\ ,\ e_2 \cdots e_n)$$

```
let first_sexpr_of_toks (toks : token list) : (token list * token list) option =
  let rec f count acc toks =
    match toks with
    | [] → None
    | Atom s :: toks →
      if count = 0
      then Some ([Atom s],toks)
      else f count (Atom s::acc) toks
    | Lparen :: toks →
      f (count+1) (Lparen::acc) toks
    | Rparen :: toks →
      if count ≤ 0 then None else
      let acc = Rparen::acc in
      if count = 1 then Some (List.rev acc,toks) else
      f (count-1) acc toks
  in f 0 [] toks
```

b. Helper function that parses one e into a tree

$$(\text{atom } e_1 \cdots e_n) \rightarrow \text{Node}(\text{atom}, [t_1; \ldots; t_n])$$
$$\text{atom} \rightarrow \text{Node}(\text{atom}, [\,])$$

```
let rec ntree_of_toks_aux (toks : token list) : string ntree option =
  match toks with
  | [Atom s] → Some (Node (s, []))
  | Lparen :: Atom s :: toks →
    begin match ntrees_of_sexpr_sequence toks with
    | Some trees → Some (Node (s,trees))
    | _ → None
    end
  | _ → None
```

c. Helper function that parses a sequence of e's into a list of trees

$(\text{atom}\ e_1 \cdots e_n) \rightarrow [t_1; \ldots; t_n]$

mutually recursive with `ntree_of_toks_aux`

```
and ntrees_of_sexpr_sequence (toks : token list) : string ntree list option =
  let rec f acc toks =
    match toks with
    | [] → None
    | [Rparen] → Some (List.rev acc)
    | Rparen :: _ → None
    | _ →
    match first_sexpr_of_toks toks with None → None
    | Some (toks,toks') →
    match ntree_of_toks_aux toks with None → None
    | Some tree → f (tree::acc) toks'
  in f [] toks
```

d. The main function that parses one *sexpr* into a tree

```
let ntree_of_toks toks =
  match ntree_of_toks_aux toks with
  | Some (Node (_, [])) → None
  | res → res
```