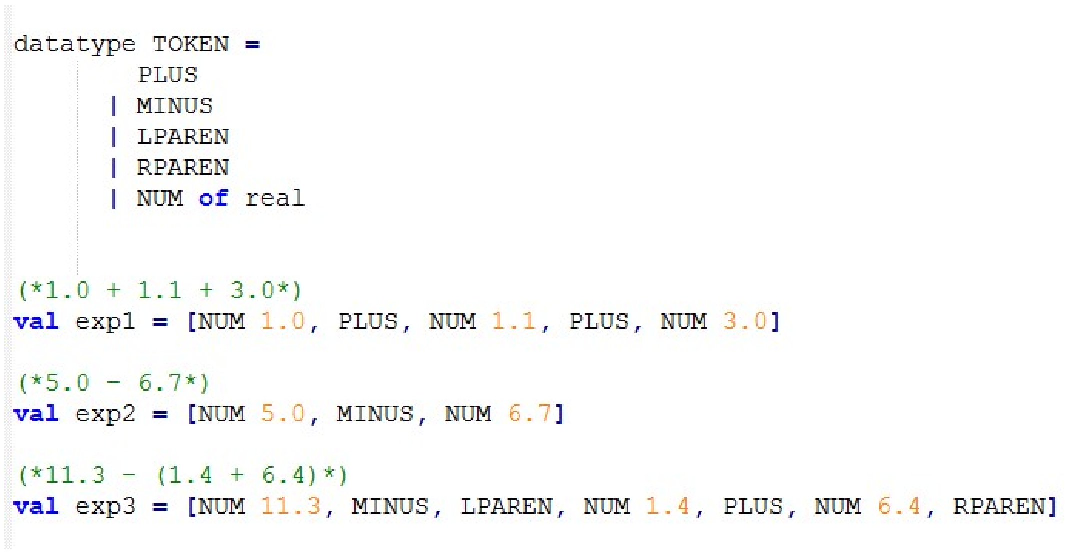
Name: Tara Walton, Brent Eaves

Programming Assignment #5 – A Parser for Calc

Part 1 – Please show you datatype for representing lexical tokens, and then show 3 expressions and how they can be represented as a list of tokens. (You can screenshot your code and paste it in the space below.)

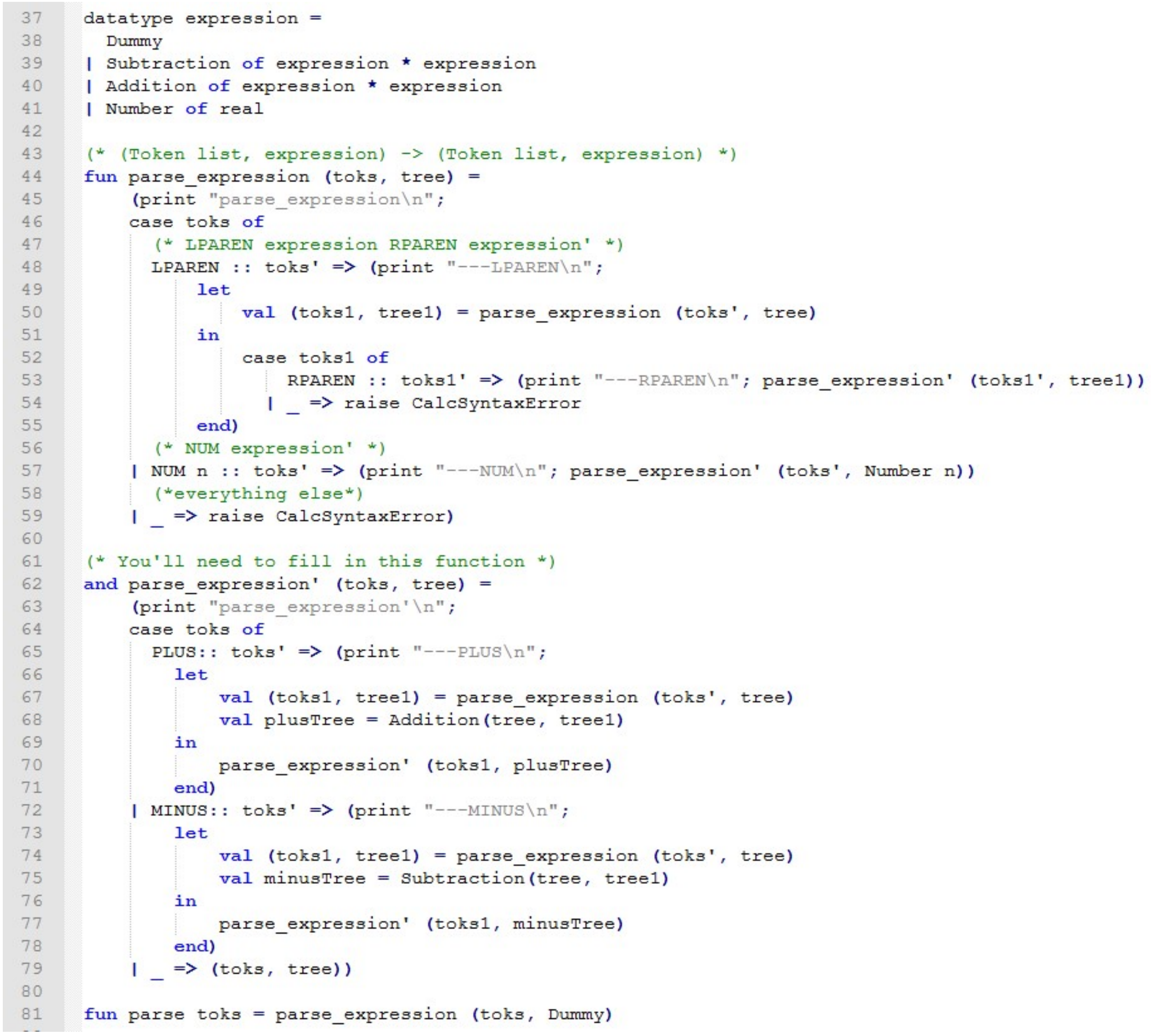


```
datatype TOKEN =
          PLUS
        | MINUS
        | LPAREN
        | RPAREN
        | NUM of real


(*1.0 + 1.1 + 3.0*)
val exp1 = [NUM 1.0, PLUS, NUM 1.1, PLUS, NUM 3.0]

(*5.0 - 6.7*)
val exp2 = [NUM 5.0, MINUS, NUM 6.7]

(*11.3 - (1.4 + 6.4)*)
val exp3 = [NUM 11.3, MINUS, LPAREN, NUM 1.4, PLUS, NUM 6.4, RPAREN]
```

Name: Tara Walton, Brent Eaves

Part 2 – Show you code for creating parse trees. This will include a datatype for representing trees and your functions for generating the trees themselves. (Please put a screenshot of your code in the space provided below.)

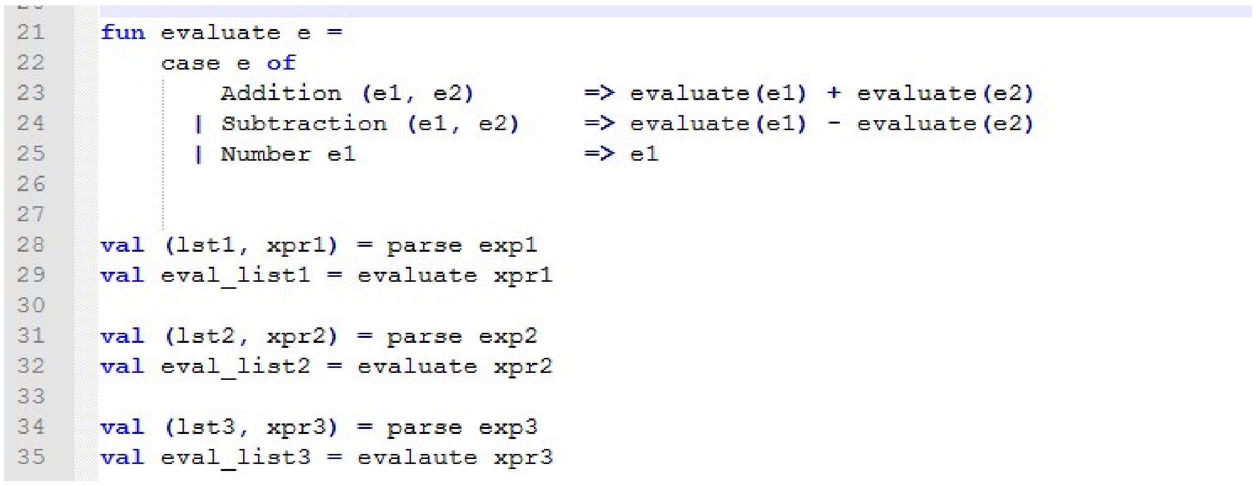
THIS MAKES MUCH MORE SENSE NOW. WE APPRECIATE ALL THE TIME AND HELP.
THANK YOU SO MUCH FOR THE DEBUGGING LINES! THEY MADE ALL THE DIFFERENCE ☺



```
datatype expression =
  Dummy
| Subtraction of expression * expression
| Addition of expression * expression
| Number of real

(* (Token list, expression) -> (Token list, expression) *)
fun parse_expression (toks, tree) =
    (print "parse_expression\n";
    case toks of
      (* LPAREN expression RPAREN expression' *)
      LPAREN :: toks' => (print "---LPAREN\n";
          let
              val (toks1, tree1) = parse_expression (toks', tree)
          in
              case toks1 of
                  RPAREN :: toks1' => (print "---RPAREN\n"; parse_expression' (toks1', tree1))
                | _ => raise CalcSyntaxError
          end)
      (* NUM expression' *)
    | NUM n :: toks' => (print "---NUM\n"; parse_expression' (toks', Number n))
      (*everything else*)
    | _ => raise CalcSyntaxError)

(* You'll need to fill in this function *)
and parse_expression' (toks, tree) =
    (print "parse_expression'\n";
    case toks of
      PLUS:: toks' => (print "---PLUS\n";
        let
            val (toks1, tree1) = parse_expression (toks', tree)
            val plusTree = Addition(tree, tree1)
        in
            parse_expression' (toks1, plusTree)
        end)
    | MINUS:: toks' => (print "---MINUS\n";
        let
            val (toks1, tree1) = parse_expression (toks', tree)
            val minusTree = Subtraction(tree, tree1)
        in
            parse_expression' (toks1, minusTree)
        end)
    | _ => (toks, tree))

fun parse toks = parse_expression (toks, Dummy)
```

Name: Tara Walton, Brent Eaves

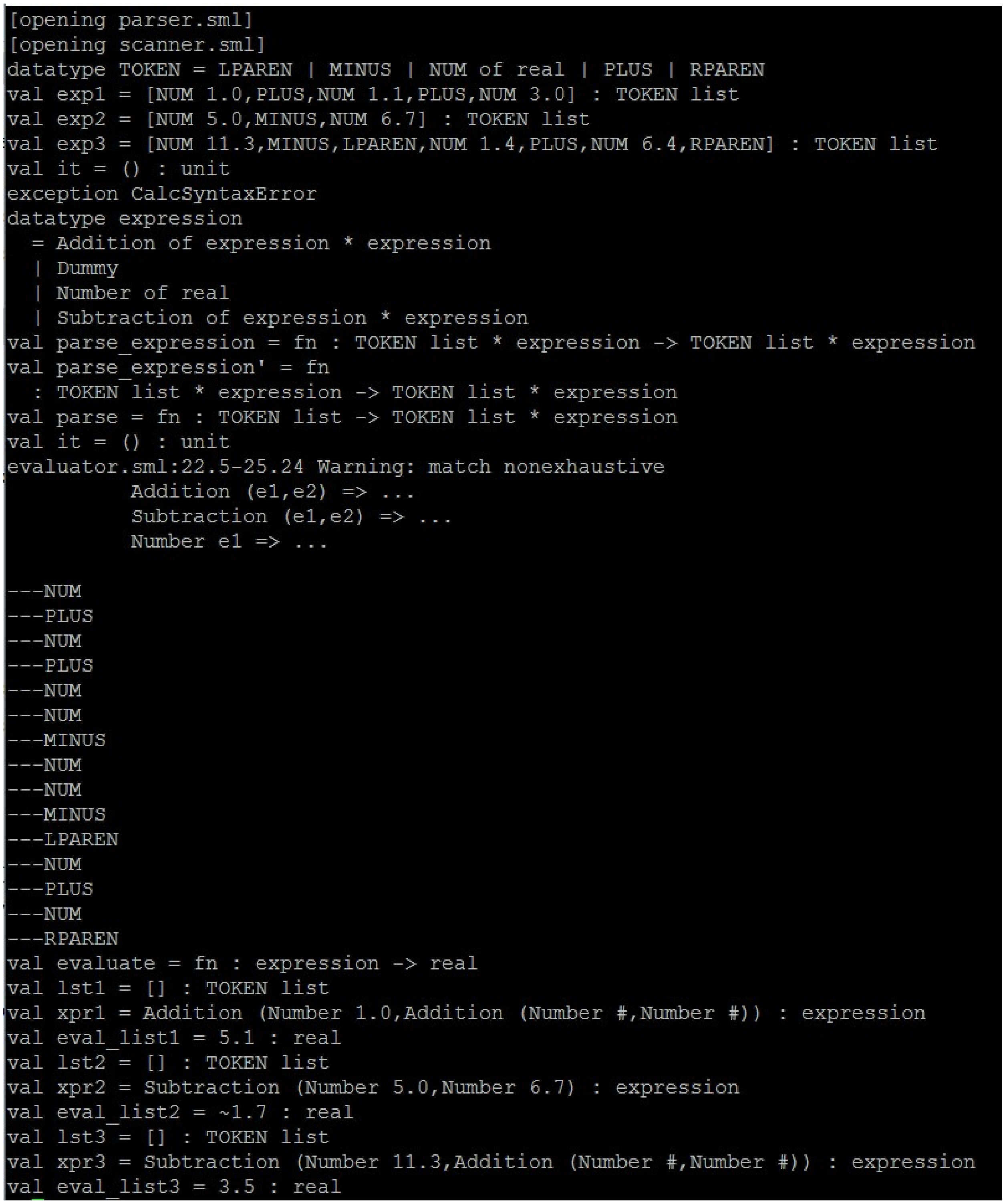
Part 3 – Code for evaluating parse trees (evaluating the expression to a single numerical value). (Please put a screenshot of your code in the space provided below.)



```
fun evaluate e =
    case e of
        Addition (e1, e2)       => evaluate(e1) + evaluate(e2)
      | Subtraction (e1, e2)    => evaluate(e1) - evaluate(e2)
      | Number e1               => e1


val (lst1, xpr1) = parse exp1
val eval_list1 = evaluate xpr1

val (lst2, xpr2) = parse exp2
val eval_list2 = evaluate xpr2

val (lst3, xpr3) = parse exp3
val eval_list3 = evalaute xpr3
```

Name: Tara Walton, Brent Eaves

Part 4 – Below, please show a screenshot of your interacting with your code in the SML/NJ interpreter. Show that it can turn a list of tokens into a parse tree, and that it can evaluate the given parse tree.



```
[opening parser.sml]
[opening scanner.sml]
datatype TOKEN = LPAREN | MINUS | NUM of real | PLUS | RPAREN
val exp1 = [NUM 1.0,PLUS,NUM 1.1,PLUS,NUM 3.0] : TOKEN list
val exp2 = [NUM 5.0,MINUS,NUM 6.7] : TOKEN list
val exp3 = [NUM 11.3,MINUS,LPAREN,NUM 1.4,PLUS,NUM 6.4,RPAREN] : TOKEN list
val it = () : unit
exception CalcSyntaxError
datatype expression
  = Addition of expression * expression
  | Dummy
  | Number of real
  | Subtraction of expression * expression
val parse_expression = fn : TOKEN list * expression -> TOKEN list * expression
val parse_expression' = fn
  : TOKEN list * expression -> TOKEN list * expression
val parse = fn : TOKEN list -> TOKEN list * expression
val it = () : unit
evaluator.sml:22.5-25.24 Warning: match nonexhaustive
          Addition (e1,e2) => ...
          Subtraction (e1,e2) => ...
          Number e1 => ...

---NUM
---PLUS
---NUM
---PLUS
---NUM
---NUM
---MINUS
---NUM
---NUM
---MINUS
---LPAREN
---NUM
---PLUS
---NUM
---RPAREN
val evaluate = fn : expression -> real
val lst1 = [] : TOKEN list
val xpr1 = Addition (Number 1.0,Addition (Number #,Number #)) : expression
val eval_list1 = 5.1 : real
val lst2 = [] : TOKEN list
val xpr2 = Subtraction (Number 5.0,Number 6.7) : expression
val eval_list2 = ~1.7 : real
val lst3 = [] : TOKEN list
val xpr3 = Subtraction (Number 11.3,Addition (Number #,Number #)) : expression
val eval_list3 = 3.5 : real
```