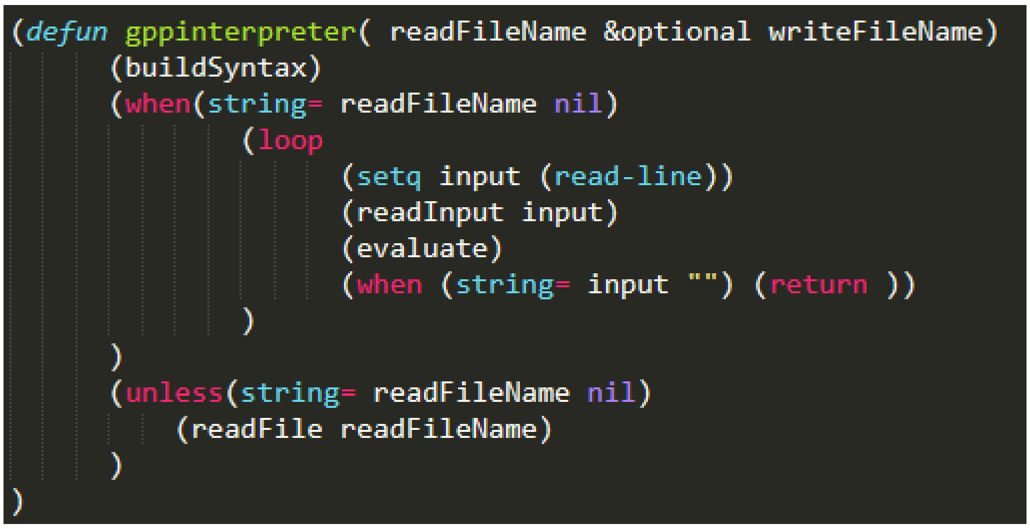
CSE 341 - PROGRAMMING LANGUAGES

HW3- REPORT

Elif Goral

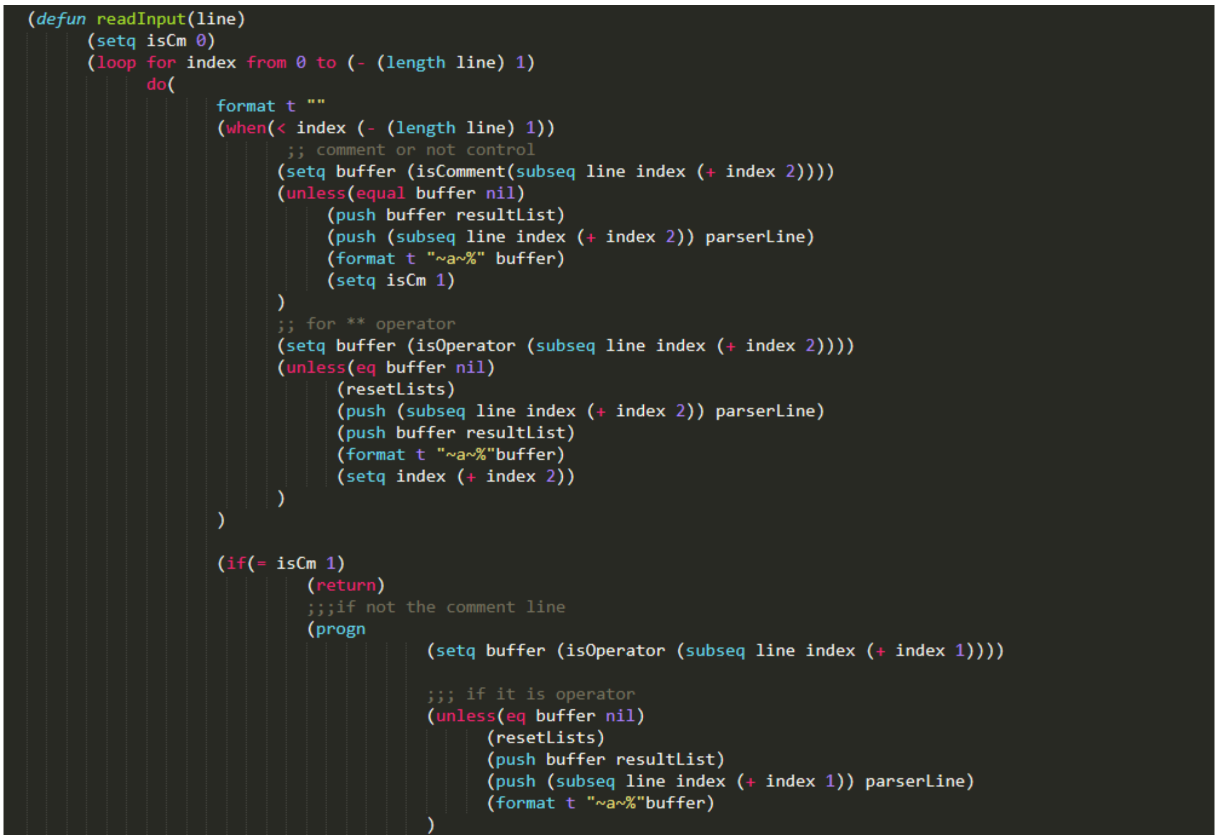
171044003

Gppinterpreter(): That function builds the keyword, operation and comment lists. Then controls the input.txt file comes with terminal or not. If input file entered, I read that file, otherwise I read from terminal. I read line by line and evalulate the results and write to file in evaluate function.

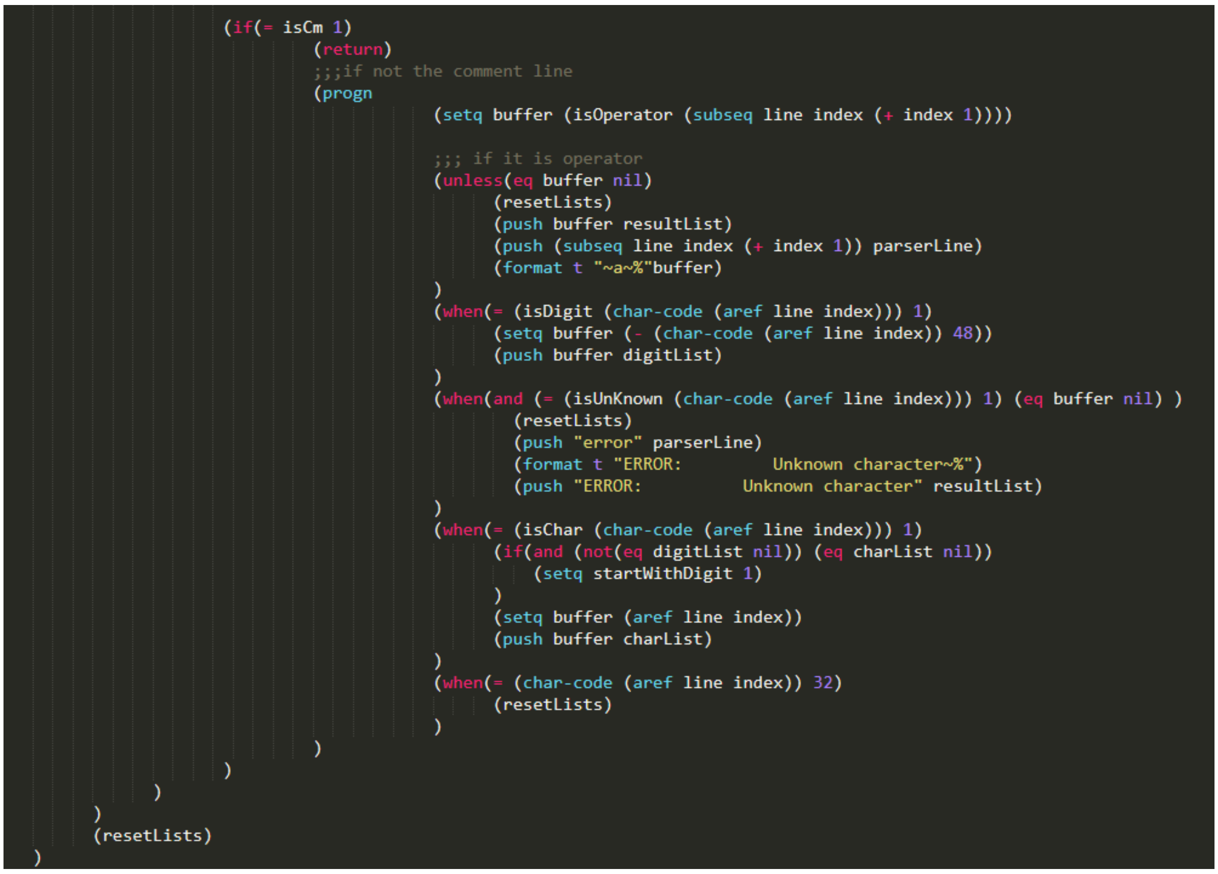


```lisp
(defun gppinterpreter( readFileName &optional writeFileName)
      (buildSyntax)
      (when(string= readFileName nil)
              (loop
                    (setq input (read-line))
                    (readInput input)
                    (evaluate)
                    (when (string= input "") (return ))
              )
      )
      (unless(string= readFileName nil)
          (readFile readFileName)
      )
)
```

ReadInput(): That function read the line and determine the types of words(keyword, operatör, comment vs). Then put them to the resultlist. And check them on resetList function.



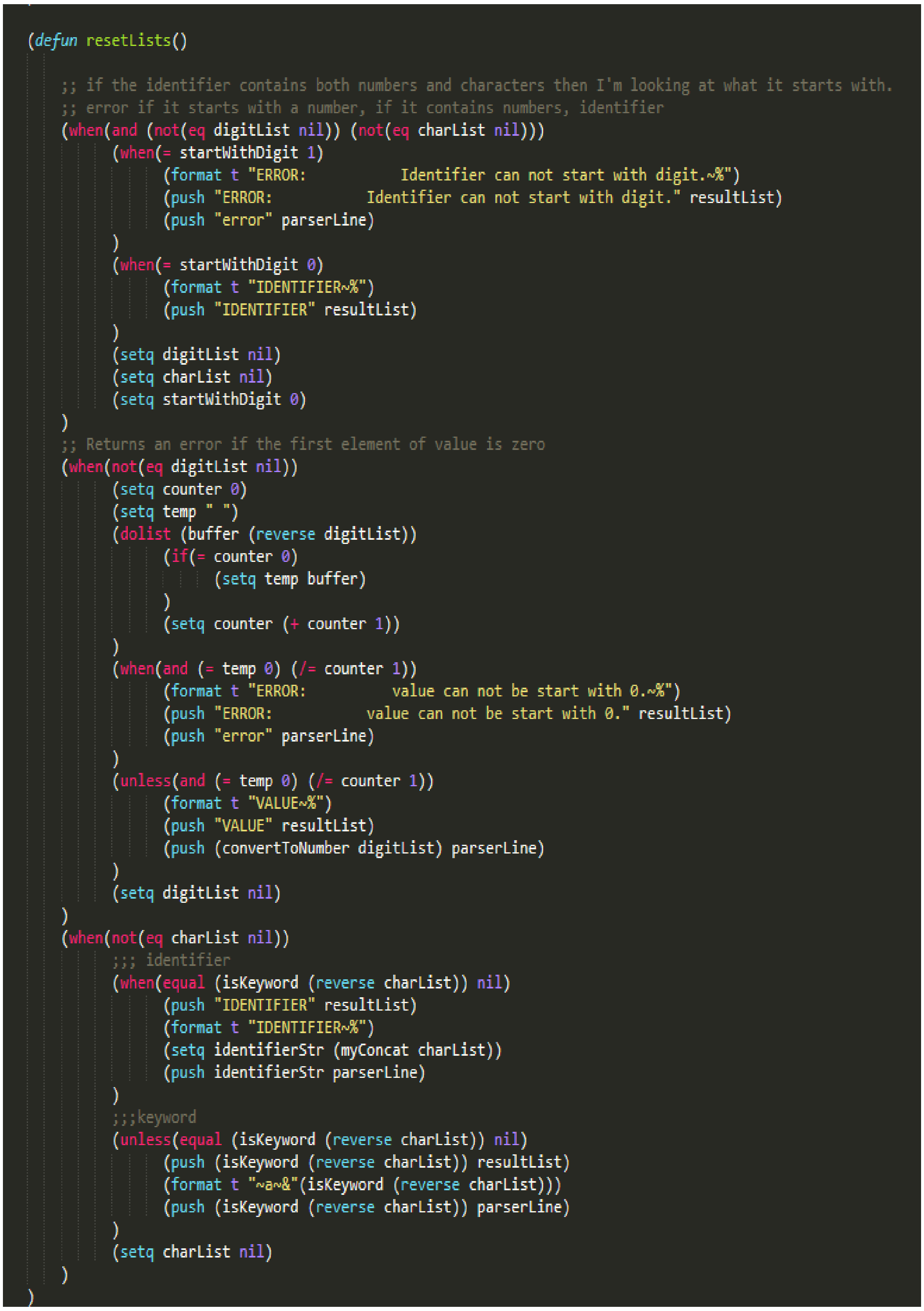
```lisp
 (defun readInput(line)
       (setq isCm 0)
       (loop for index from 0 to (- (length line) 1)
             do(
                   format t ""
                   (when(< index (- (length line) 1))
                          ;; comment or not control
                         (setq buffer (isComment(subseq line index (+ index 2))))
                         (unless(equal buffer nil)
                              (push buffer resultList)
                              (push (subseq line index (+ index 2)) parserLine)
                              (format t "~a~%" buffer)
                              (setq isCm 1)
                         )
                         ;; for ** operator
                         (setq buffer (isOperator (subseq line index (+ index 2))))
                         (unless(eq buffer nil)
                               (resetLists)
                               (push (subseq line index (+ index 2)) parserLine)
                               (push buffer resultList)
                               (format t "~a~%"buffer)
                               (setq index (+ index 2))
                         )
                   )

                   (if(= isCm 1)
                            (return)
                            ;;;if not the comment line
                            (progn
                                        (setq buffer (isOperator (subseq line index (+ index 1))))

                                        ;;; if it is operator
                                        (unless(eq buffer nil)
                                              (resetLists)
                                              (push buffer resultList)
                                              (push (subseq line index (+ index 1)) parserLine)
                                              (format t "~a~%"buffer)
                                        )
```



```
                   (if(= isCm 1)
                            (return)
                            ;;;if not the comment line
                            (progn
                                        (setq buffer (isOperator (subseq line index (+ index 1))))

                                        ;;; if it is operator
                                        (unless(eq buffer nil)
                                              (resetLists)
                                              (push buffer resultList)
                                              (push (subseq line index (+ index 1)) parserLine)
                                              (format t "~a~%"buffer)
                                        )
                                        (when(= (isDigit (char-code (aref line index))) 1)
                                              (setq buffer (- (char-code (aref line index)) 48))
                                              (push buffer digitList)
                                        )
                                        (when(and (= (isUnKnown (char-code (aref line index))) 1) (eq buffer nil) )
                                                (resetLists)
                                                (push "error" parserLine)
                                                (format t "ERROR:         Unknown character~%")
                                                (push "ERROR:          Unknown character" resultList)
                                        )
                                        (when(= (isChar (char-code (aref line index))) 1)
                                              (if(and (not(eq digitList nil)) (eq charList nil))
                                                  (setq startWithDigit 1)
                                              )
                                              (setq buffer (aref line index))
                                              (push buffer charList)
                                        )
                                        (when(= (char-code (aref line index)) 32)
                                              (resetLists)
                                        )
                            )
                   )
            )
      )
      (resetLists)
)
```

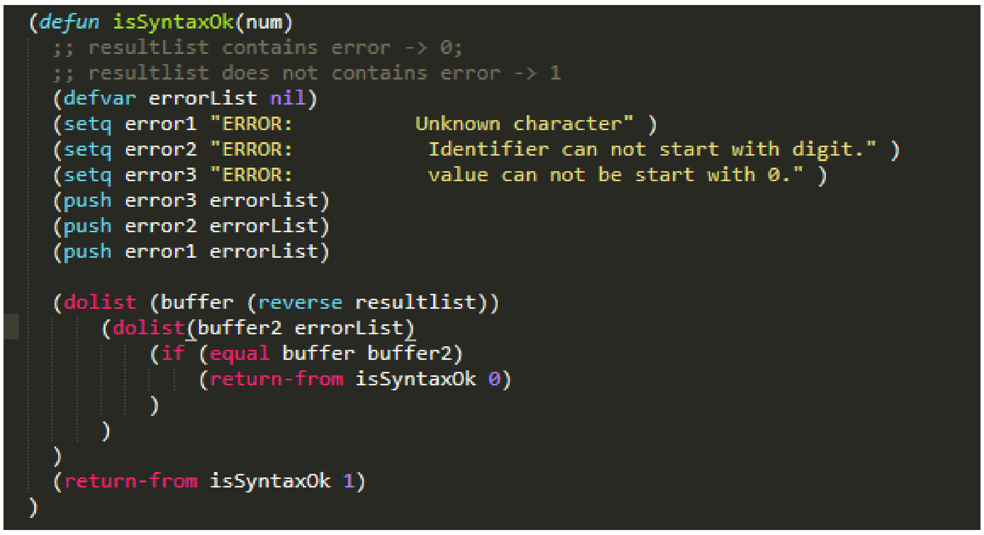
RestList():

If the identifier contains both numbers and characters then I'm looking at what it starts with. Error if it starts with a number, if it contains numbers, identifier. Returns an error if the first element of value is zero. Otherwise put the results to the resultList.



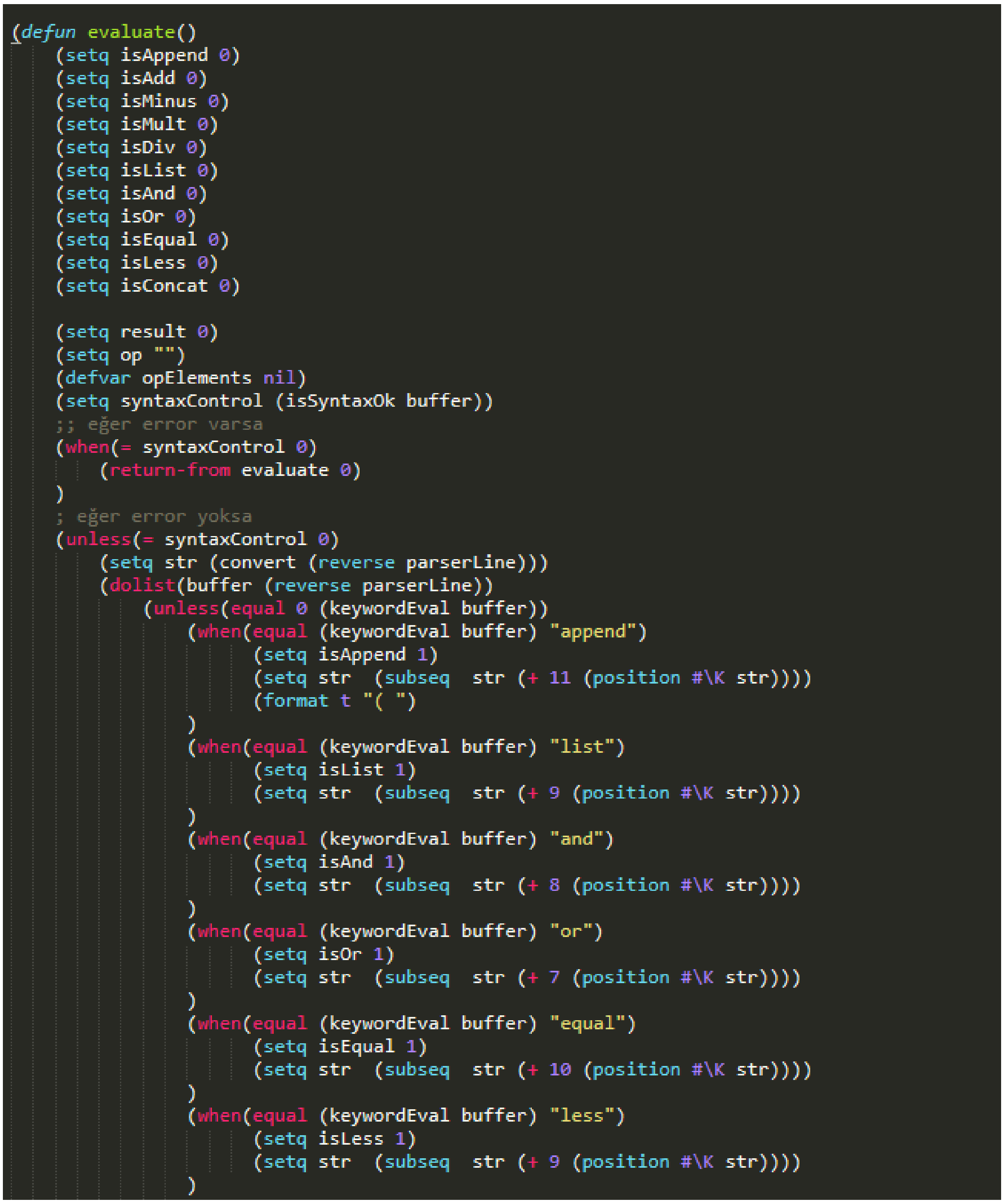
```lisp
(defun resetLists()

    ;; if the identifier contains both numbers and characters then I'm looking at what it starts with.
    ;; error if it starts with a number, if it contains numbers, identifier
    (when(and (not(eq digitList nil)) (not(eq charList nil)))
        (when(= startWithDigit 1)
            (format t "ERROR:           Identifier can not start with digit.~%")
            (push "ERROR:           Identifier can not start with digit." resultList)
            (push "error" parserLine)
        )
        (when(= startWithDigit 0)
            (format t "IDENTIFIER~%")
            (push "IDENTIFIER" resultList)
        )
        (setq digitList nil)
        (setq charList nil)
        (setq startWithDigit 0)
    )
    ;; Returns an error if the first element of value is zero
    (when(not(eq digitList nil))
        (setq counter 0)
        (setq temp " ")
        (dolist (buffer (reverse digitList))
            (if(= counter 0)
                (setq temp buffer)
            )
            (setq counter (+ counter 1))
        )
        (when(and (= temp 0) (/= counter 1))
            (format t "ERROR:          value can not be start with 0.~%")
            (push "ERROR:           value can not be start with 0." resultList)
            (push "error" parserLine)
        )
        (unless(and (= temp 0) (/= counter 1))
            (format t "VALUE~%")
            (push "VALUE" resultList)
            (push (convertToNumber digitList) parserLine)
        )
        (setq digitList nil)
    )
    (when(not(eq charList nil))
        ;;; identifier
        (when(equal (isKeyword (reverse charList)) nil)
            (push "IDENTIFIER" resultList)
            (format t "IDENTIFIER~%")
            (setq identifierStr (myConcat charList))
            (push identifierStr parserLine)
        )
        ;;;keyword
        (unless(equal (isKeyword (reverse charList)) nil)
            (push (isKeyword (reverse charList)) resultList)
            (format t "~a~&"(isKeyword (reverse charList)))
            (push (isKeyword (reverse charList)) parserLine)
        )
        (setq charList nil)
    )
)
```

IsSyntaxOk(): That function checks the line is syntaticly true.

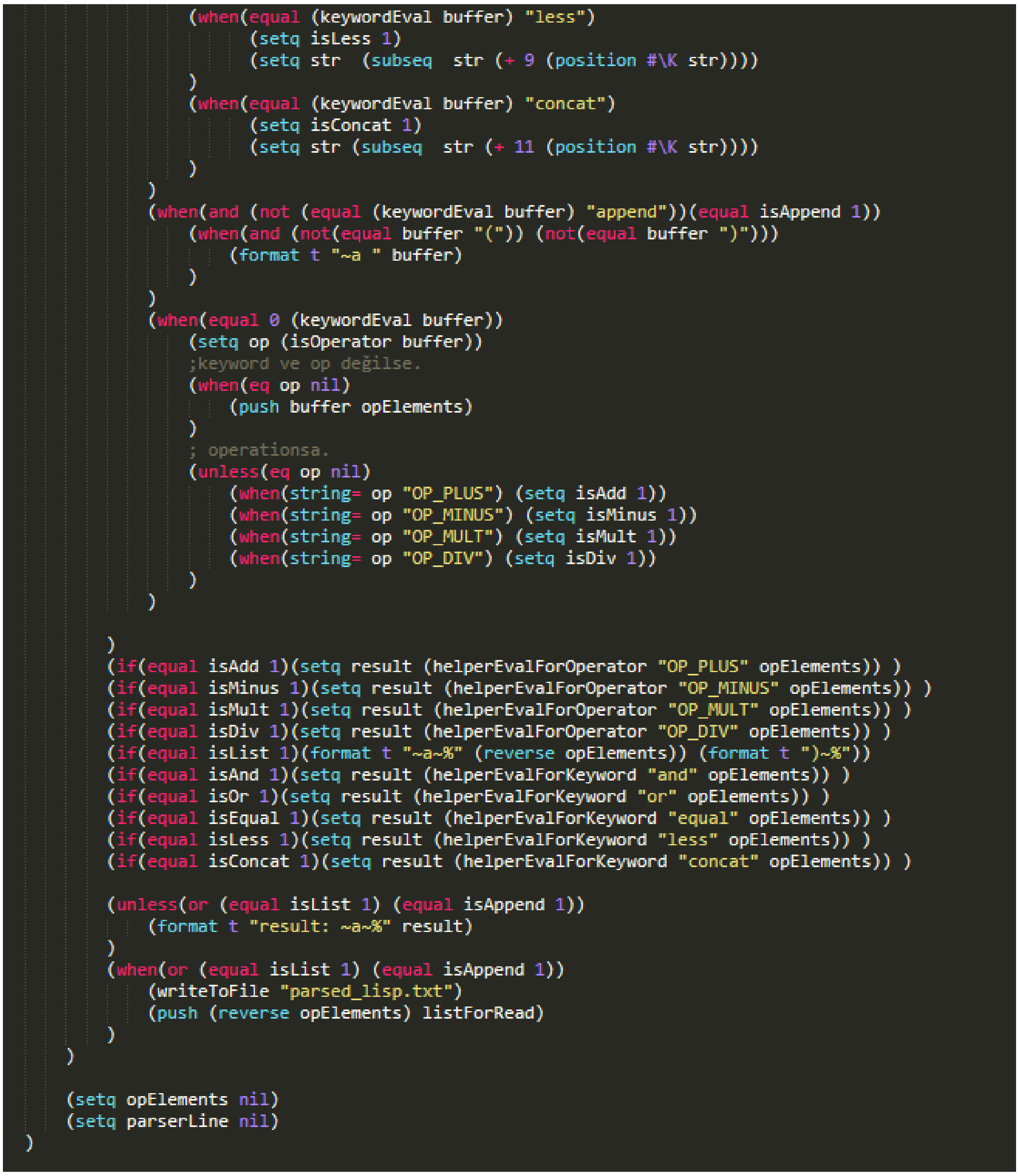


```
(defun isSyntaxOk(num)
  ;; resultList contains error -> 0;
  ;; resultlist does not contains error -> 1
  (defvar errorList nil)
  (setq error1 "ERROR:          Unknown character" )
  (setq error2 "ERROR:           Identifier can not start with digit." )
  (setq error3 "ERROR:           value can not be start with 0." )
  (push error3 errorList)
  (push error2 errorList)
  (push error1 errorList)

  (dolist (buffer (reverse resultlist))
      (dolist(buffer2 errorList)
          (if (equal buffer buffer2)
              (return-from isSyntaxOk 0)
          )
      )
  )
  (return-from isSyntaxOk 1)
)
```

Evaluate (): In this function, first I gave values (such as isAppend, isAdd) to understand what operation it will do. Then, after doing the syntax control of the line, if there is no problem, I combine the line with concatenate and put it to the str variable. And I'm starting to get around the elements on the line one by one. If I find the keyword, I throw the elements after the keyword to opElements. And I'm doing the calculation with the helperEvalKeyword function. But I'm only doing 2 elements calculations. If I found an operator, I also throw the elements after the operator to opElements in the same way. I'm sending it to the helperEvalOperator () function. Writing to the file is done in helper functions if there is no append or list keyword. But if there is an append or list, this operation is performed in the evaluate function. (list and append keywords work for more than two elements.)

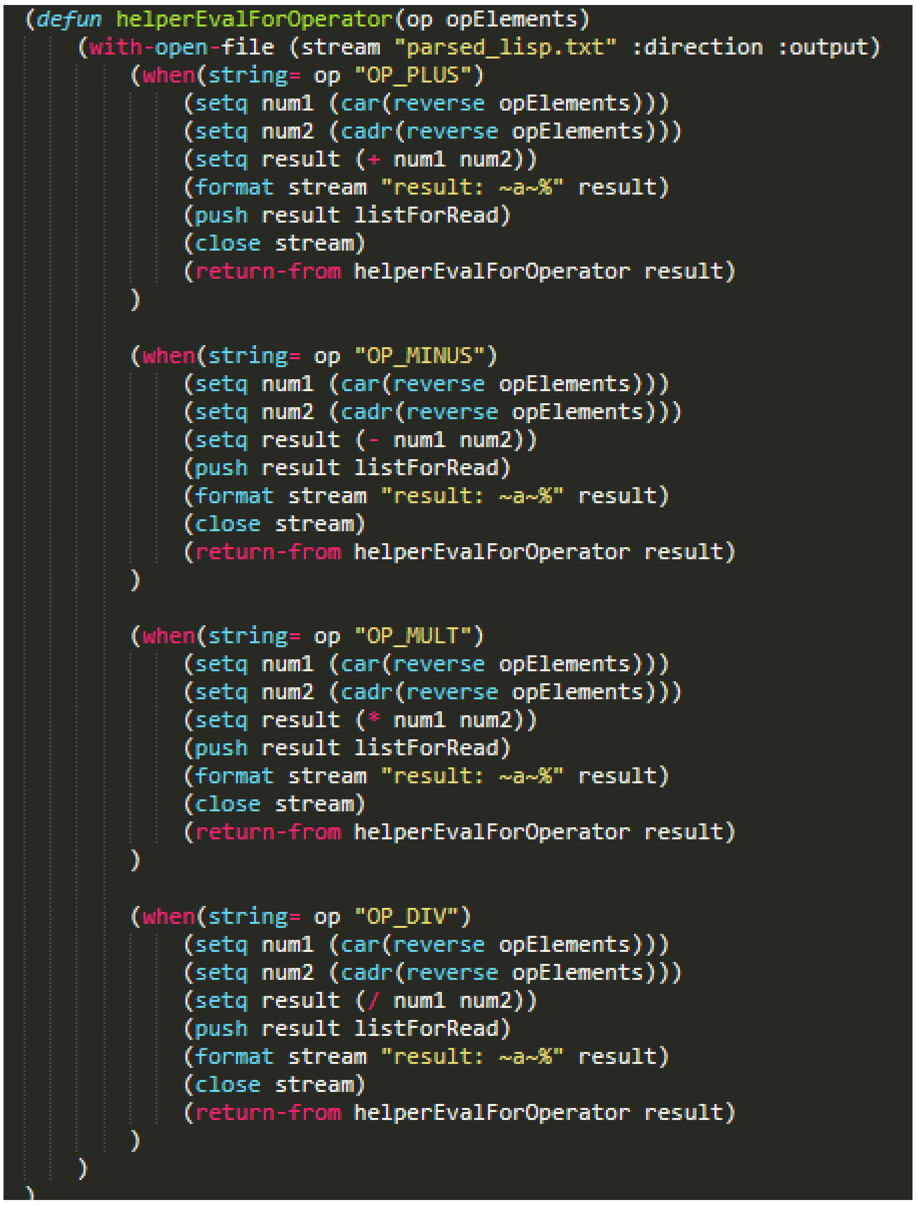


```lisp
(defun evaluate()
    (setq isAppend 0)
    (setq isAdd 0)
    (setq isMinus 0)
    (setq isMult 0)
    (setq isDiv 0)
    (setq isList 0)
    (setq isAnd 0)
    (setq isOr 0)
    (setq isEqual 0)
    (setq isLess 0)
    (setq isConcat 0)

    (setq result 0)
    (setq op "")
    (defvar opElements nil)
    (setq syntaxControl (isSyntaxOk buffer))
    ;; eğer error varsa
    (when(= syntaxControl 0)
        (return-from evaluate 0)
    )
    ; eğer error yoksa
    (unless(= syntaxControl 0)
        (setq str (convert (reverse parserLine)))
        (dolist(buffer (reverse parserLine))
            (unless(equal 0 (keywordEval buffer))
                (when(equal (keywordEval buffer) "append")
                      (setq isAppend 1)
                      (setq str  (subseq  str (+ 11 (position #\K str))))
                      (format t "( ")
                )
                (when(equal (keywordEval buffer) "list")
                      (setq isList 1)
                      (setq str  (subseq  str (+ 9 (position #\K str))))
                )
                (when(equal (keywordEval buffer) "and")
                      (setq isAnd 1)
                      (setq str  (subseq  str (+ 8 (position #\K str))))
                )
                (when(equal (keywordEval buffer) "or")
                      (setq isOr 1)
                      (setq str  (subseq  str (+ 7 (position #\K str))))
                )
                (when(equal (keywordEval buffer) "equal")
                      (setq isEqual 1)
                      (setq str  (subseq  str (+ 10 (position #\K str))))
                )
                (when(equal (keywordEval buffer) "less")
                      (setq isLess 1)
                      (setq str  (subseq  str (+ 9 (position #\K str))))
                )
```



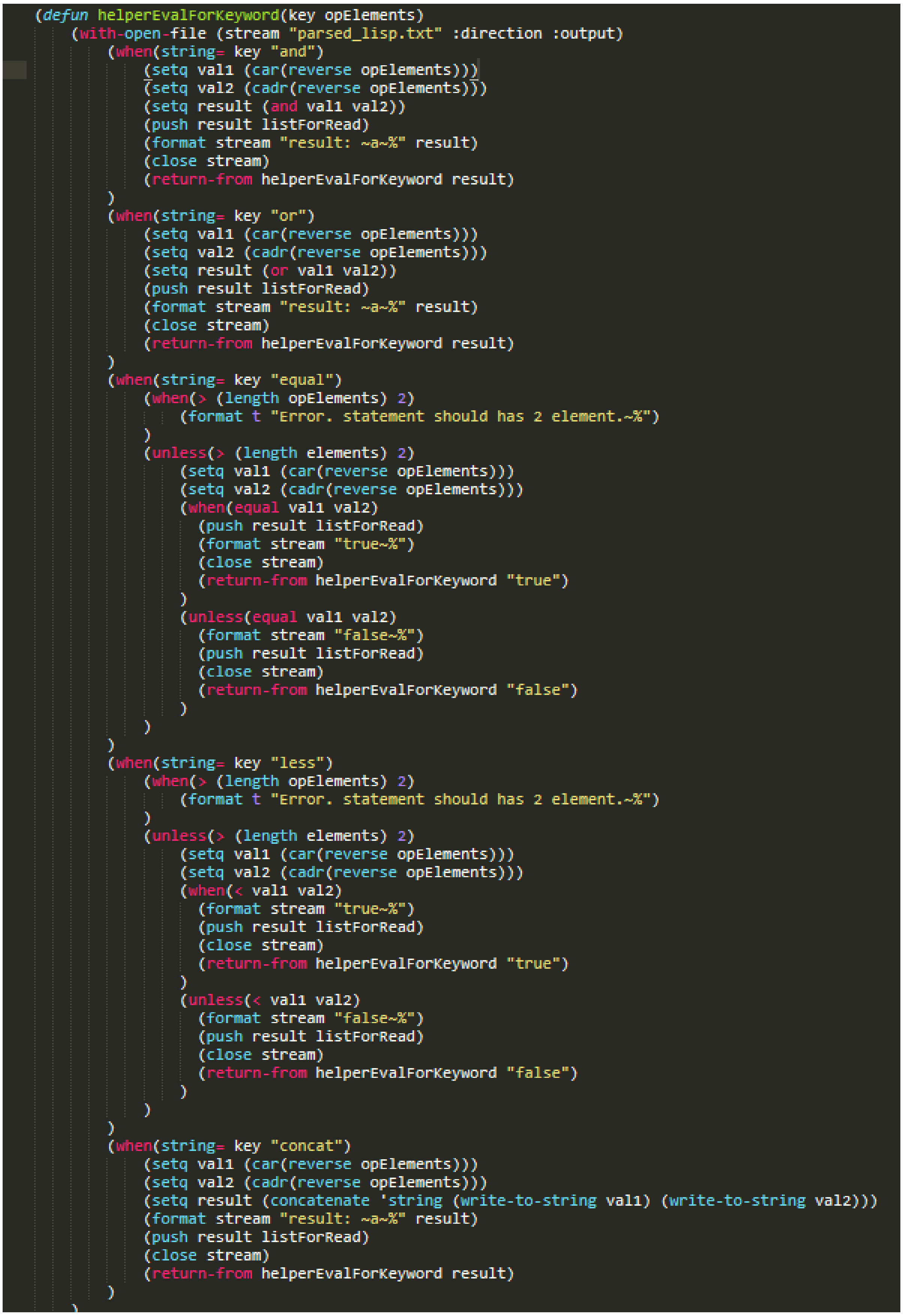
```
                (when(equal (keywordEval buffer) "less")
                      (setq isLess 1)
                      (setq str  (subseq  str (+ 9 (position #\K str))))
                )
                (when(equal (keywordEval buffer) "concat")
                      (setq isConcat 1)
                      (setq str (subseq  str (+ 11 (position #\K str))))
                )
            )
            (when(and (not (equal (keywordEval buffer) "append"))(equal isAppend 1))
                (when(and (not(equal buffer "(")) (not(equal buffer ")")))
                    (format t "~a " buffer)
                )
            )
            (when(equal 0 (keywordEval buffer))
                (setq op (isOperator buffer))
                ;keyword ve op değilse.
                (when(eq op nil)
                    (push buffer opElements)
                )
                ; operationsa.
                (unless(eq op nil)
                    (when(string= op "OP_PLUS") (setq isAdd 1))
                    (when(string= op "OP_MINUS") (setq isMinus 1))
                    (when(string= op "OP_MULT") (setq isMult 1))
                    (when(string= op "OP_DIV") (setq isDiv 1))
                )
            )

        )
        (if(equal isAdd 1)(setq result (helperEvalForOperator "OP_PLUS" opElements)) )
        (if(equal isMinus 1)(setq result (helperEvalForOperator "OP_MINUS" opElements)) )
        (if(equal isMult 1)(setq result (helperEvalForOperator "OP_MULT" opElements)) )
        (if(equal isDiv 1)(setq result (helperEvalForOperator "OP_DIV" opElements)) )
        (if(equal isList 1)(format t "~a~%" (reverse opElements)) (format t ")~%"))
        (if(equal isAnd 1)(setq result (helperEvalForKeyword "and" opElements)) )
        (if(equal isOr 1)(setq result (helperEvalForKeyword "or" opElements)) )
        (if(equal isEqual 1)(setq result (helperEvalForKeyword "equal" opElements)) )
        (if(equal isLess 1)(setq result (helperEvalForKeyword "less" opElements)) )
        (if(equal isConcat 1)(setq result (helperEvalForKeyword "concat" opElements)) )

        (unless(or (equal isList 1) (equal isAppend 1))
            (format t "result: ~a~%" result)
        )
        (when(or (equal isList 1) (equal isAppend 1))
            (writeToFile "parsed_lisp.txt")
            (push (reverse opElements) listForRead)
        )
    )

    (setq opElements nil)
    (setq parserLine nil)
)
```

HelperEvalForOperator():



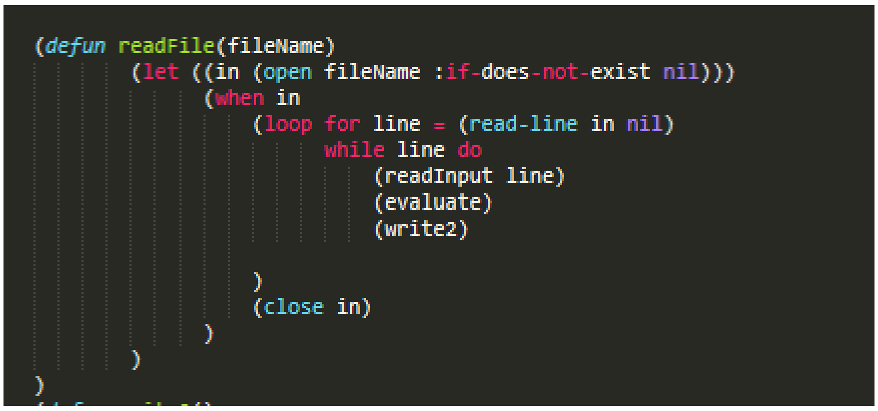
```
(defun helperEvalForOperator(op opElements)
    (with-open-file (stream "parsed_lisp.txt" :direction :output)
        (when(string= op "OP_PLUS")
            (setq num1 (car(reverse opElements)))
            (setq num2 (cadr(reverse opElements)))
            (setq result (+ num1 num2))
            (format stream "result: ~a~%" result)
            (push result listForRead)
            (close stream)
            (return-from helperEvalForOperator result)
        )

        (when(string= op "OP_MINUS")
            (setq num1 (car(reverse opElements)))
            (setq num2 (cadr(reverse opElements)))
            (setq result (- num1 num2))
            (push result listForRead)
            (format stream "result: ~a~%" result)
            (close stream)
            (return-from helperEvalForOperator result)
        )

        (when(string= op "OP_MULT")
            (setq num1 (car(reverse opElements)))
            (setq num2 (cadr(reverse opElements)))
            (setq result (* num1 num2))
            (push result listForRead)
            (format stream "result: ~a~%" result)
            (close stream)
            (return-from helperEvalForOperator result)
        )

        (when(string= op "OP_DIV")
            (setq num1 (car(reverse opElements)))
            (setq num2 (cadr(reverse opElements)))
            (setq result (/ num1 num2))
            (push result listForRead)
            (format stream "result: ~a~%" result)
            (close stream)
            (return-from helperEvalForOperator result)
        )
    )
)
```

HelperEvalForKeyword():



```
(defun helperEvalForKeyword(key opElements)
    (with-open-file (stream "parsed_lisp.txt" :direction :output)
        (when(string= key "and")
            (setq val1 (car(reverse opElements)))
            (setq val2 (cadr(reverse opElements)))
            (setq result (and val1 val2))
            (push result listForRead)
            (format stream "result: ~a~%" result)
            (close stream)
            (return-from helperEvalForKeyword result)
        )
        (when(string= key "or")
            (setq val1 (car(reverse opElements)))
            (setq val2 (cadr(reverse opElements)))
            (setq result (or val1 val2))
            (push result listForRead)
            (format stream "result: ~a~%" result)
            (close stream)
            (return-from helperEvalForKeyword result)
        )
        (when(string= key "equal")
            (when(> (length opElements) 2)
                (format t "Error. statement should has 2 element.~%")
            )
            (unless(> (length elements) 2)
                (setq val1 (car(reverse opElements)))
                (setq val2 (cadr(reverse opElements)))
                (when(equal val1 val2)
                  (push result listForRead)
                  (format stream "true~%")
                  (close stream)
                  (return-from helperEvalForKeyword "true")
                )
                (unless(equal val1 val2)
                  (format stream "false~%")
                  (push result listForRead)
                  (close stream)
                  (return-from helperEvalForKeyword "false")
                )
            )
        )
        (when(string= key "less")
            (when(> (length opElements) 2)
                (format t "Error. statement should has 2 element.~%")
            )
            (unless(> (length elements) 2)
                (setq val1 (car(reverse opElements)))
                (setq val2 (cadr(reverse opElements)))
                (when(< val1 val2)
                  (format stream "true~%")
                  (push result listForRead)
                  (close stream)
                  (return-from helperEvalForKeyword "true")
                )
                (unless(< val1 val2)
                  (format stream "false~%")
                  (push result listForRead)
                  (close stream)
                  (return-from helperEvalForKeyword "false")
                )
            )
        )
        (when(string= key "concat")
            (setq val1 (car(reverse opElements)))
            (setq val2 (cadr(reverse opElements)))
            (setq result (concatenate 'string (write-to-string val1) (write-to-string val2)))
            (format stream "result: ~a~%" result)
            (push result listForRead)
            (close stream)
            (return-from helperEvalForKeyword result)
        )
```

readFile(): If I read from file, I call the write2() function. And print listforread list.

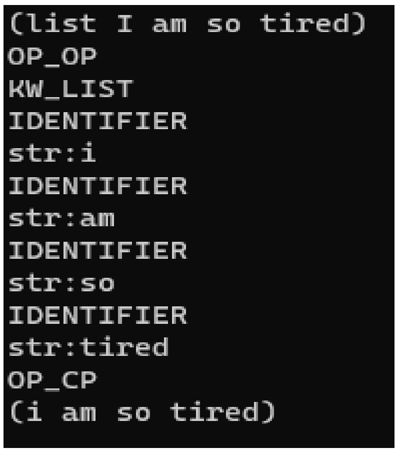


```
(defun readFile(fileName)
        (let ((in (open fileName :if-does-not-exist nil)))
              (when in
                  (loop for line = (read-line in nil)
                        while line do
                            (readInput line)
                            (evaluate)
                            (write2)

                  )
                  (close in)
              )
        )
)
```

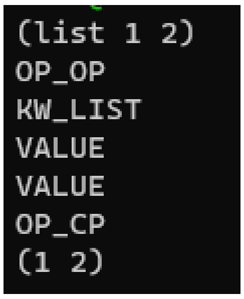
Test Results:

(LIST)

Terminal:

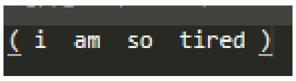


```
(list I am so tired)
OP_OP
KW_LIST
IDENTIFIER
str:i
IDENTIFIER
str:am
IDENTIFIER
str:so
IDENTIFIER
str:tired
OP_CP
(i am so tired)
```

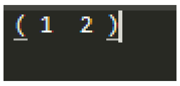


```
(list 1 2)
OP_OP
KW_LIST
VALUE
VALUE
OP_CP
(1 2)
```

Parsed_lisp.txt:



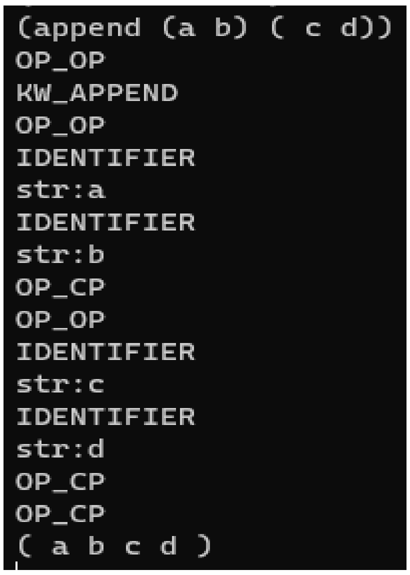
```
( i  am  so  tired )
```



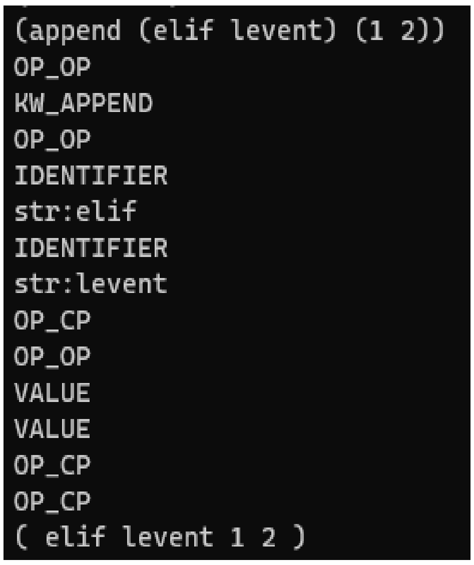
```
( 1  2 )
```

(APPEND)

Terminal:

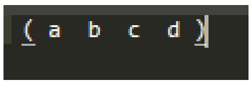


```
(append (a b) ( c d))
OP_OP
KW_APPEND
OP_OP
IDENTIFIER
str:a
IDENTIFIER
str:b
OP_CP
OP_OP
IDENTIFIER
str:c
IDENTIFIER
str:d
OP_CP
OP_CP
( a b c d )
```

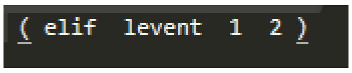


```
(append (elif levent) (1 2))
OP_OP
KW_APPEND
OP_OP
IDENTIFIER
str:elif
IDENTIFIER
str:levent
OP_CP
OP_OP
VALUE
VALUE
OP_CP
OP_CP
( elif levent 1 2 )
```

Parsed_lisp.txt:



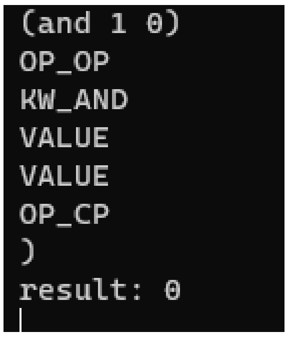
```
( a  b  c  d )
```



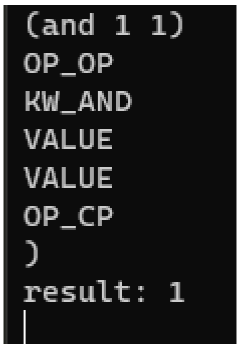
```
( elif  levent  1  2 )
```

(AND)

Terminal:

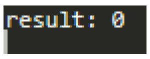


```
(and 1 0)
OP_OP
KW_AND
VALUE
VALUE
OP_CP
)
result: 0
```

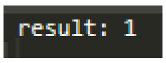


```
(and 1 1)
OP_OP
KW_AND
VALUE
VALUE
OP_CP
)
result: 1
```

Parsed_lisp.txt:



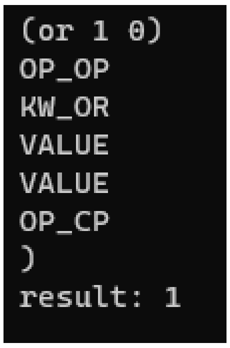
```
result: 0
```



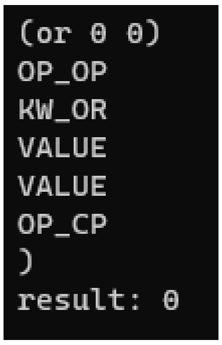
```
result: 1
```

(OR)

Terminal:

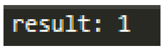


```
(or 1 0)
OP_OP
KW_OR
VALUE
VALUE
OP_CP
)
result: 1
```

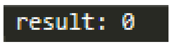


```
(or 0 0)
OP_OP
KW_OR
VALUE
VALUE
OP_CP
)
result: 0
```

Parsed_lisp.txt:

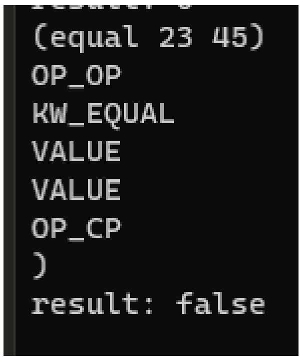


```
result: 1
```

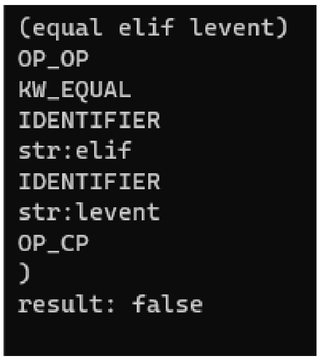


```
result: 0
```

Terminal:

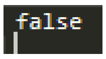


```
(equal 23 45)
OP_OP
KW_EQUAL
VALUE
VALUE
OP_CP
)
result: false
```

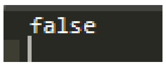


```
(equal elif levent)
OP_OP
KW_EQUAL
IDENTIFIER
str:elif
IDENTIFIER
str:levent
OP_CP
)
result: false
```

Parsed_lisp.txt:

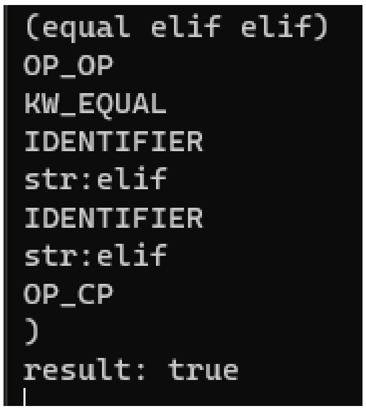


```
false
```

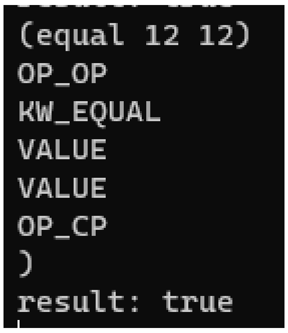


```
false
```

Terminal:

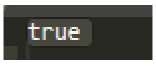


```
(equal elif elif)
OP_OP
KW_EQUAL
IDENTIFIER
str:elif
IDENTIFIER
str:elif
OP_CP
)
result: true
```

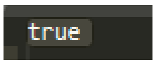


```
(equal 12 12)
OP_OP
KW_EQUAL
VALUE
VALUE
OP_CP
)
result: true
```

Parsed_lisp.txt:



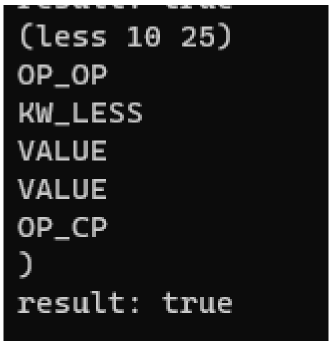
```
true
```



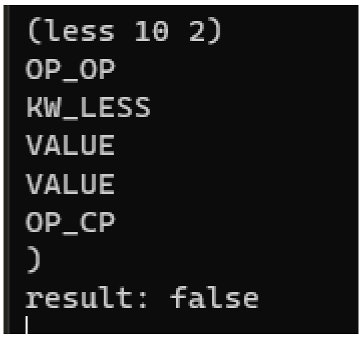
```
true
```

(LESS)

Terminal:

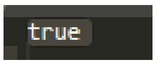


```
(less 10 25)
OP_OP
KW_LESS
VALUE
VALUE
OP_CP
)
result: true
```

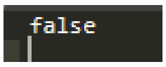


```
(less 10 2)
OP_OP
KW_LESS
VALUE
VALUE
OP_CP
)
result: false
```

Parsed_lisp.txt:



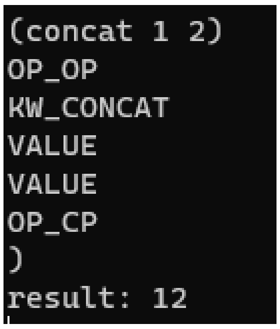
```
true
```



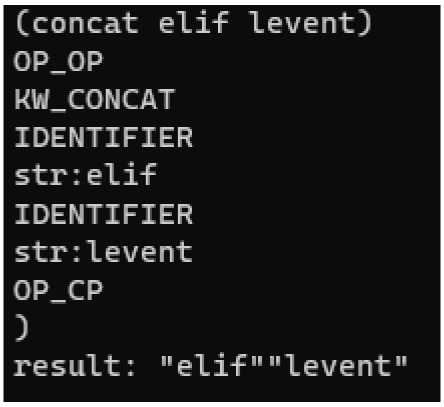
```
false
```

(CONCAT)

Terminal:

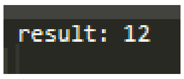


```
(concat 1 2)
OP_OP
KW_CONCAT
VALUE
VALUE
OP_CP
)
result: 12
```

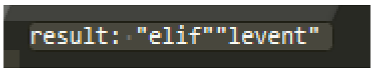


```
(concat elif levent)
OP_OP
KW_CONCAT
IDENTIFIER
str:elif
IDENTIFIER
str:levent
OP_CP
)
result: "elif""levent"
```

Parsed_lisp.txt:



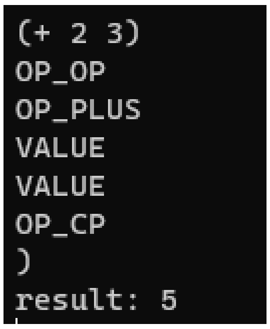
```
result: 12
```



```
result: "elif""levent"
```

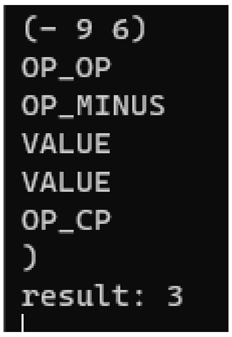
(+)

Terminal:



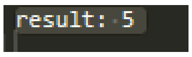
```
(+ 2 3)
OP_OP
OP_PLUS
VALUE
VALUE
OP_CP
)
result: 5
```

(-)

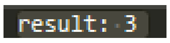


```
(- 9 6)
OP_OP
OP_MINUS
VALUE
VALUE
OP_CP
)
result: 3
```

Parsed_lisp.txt:



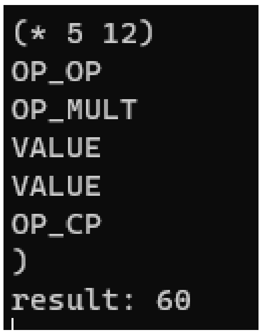
```
result: 5
```



```
result: 3
```

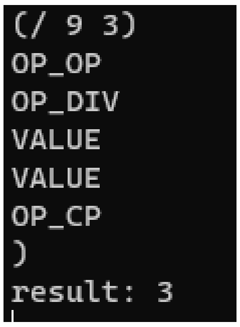
(*)

Terminal:



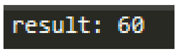
```
(* 5 12)
OP_OP
OP_MULT
VALUE
VALUE
OP_CP
)
result: 60
```

(/)

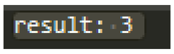


```
(/ 9 3)
OP_OP
OP_DIV
VALUE
VALUE
OP_CP
)
result: 3
```

Parsed_lisp.txt:

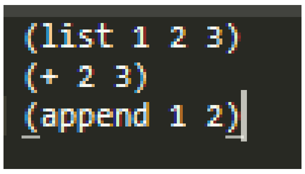


```
result: 60
```

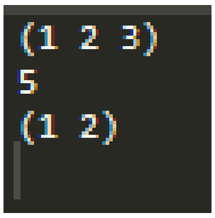


```
result: 3
```

Read from file:



```
(list 1 2 3)
(+ 2 3)
(append 1 2)
```



```
(1 2 3)
5
(1 2)
```