Programming Language - Homework 3

Dicle Çoban - 220104004088

Part 1 - Flex:

```
%token KW_AND KW_OR KW_NOT KW_EQUAL KW_LESS KW_NIL
%token KW_LIST KW_APPEND KW_CONCAT KW_SET KW_DEFFUN
%token KW_FOR KW_IF KW_EXIT KW_LOAD KW_PRINT KW_TRUE KW_FALSE
%token OP_OP OP_CP OP_PLUS OP_MINUS OP_MULT OP_DIV
%token VALUEI VALUEF IDENTIFIER
```

I identified the necessary tokens according to the gpp_lexer.l file (the file that I used in the homework 2)

```
input: expression
     | expression_list
     | function_definition
     | control_statement
     | variable_definition
     | exit_statement
     | print_statement
     | load_statement;
```

Inputs can be this ones:

```
expression_list: OP_OP valid_expression_list OP_CP ;

valid_expression_list: expression_list expression
                     | /* empty */ ;

expression: OP_OP KW_AND expression expression OP_CP
          | OP_OP KW_OR expression expression OP_CP
          | OP_OP KW_NOT expression OP_CP
          | OP_OP KW_EQUAL expression expression OP_CP
          | OP_OP KW_LESS expression expression OP_CP
          | OP_OP KW_LIST expression_list OP_CP
          | OP_OP KW_APPEND expression expression OP_CP
          | OP_OP KW_CONCAT expression expression OP_CP
          |OP_OP OP_PLUS expression expression OP_CP
          | OP_OP OP_MINUS expression expression OP_CP
          | OP_OP OP_MULT expression expression OP_CP
          | OP_OP OP_DIV expression expression OP_CP
          | VALUEI
          | VALUEF
          | IDENTIFIER
          | KW_TRUE
          | KW_FALSE
          | KW_NIL;
```

These are the CFGs about expressions. Expression can be reduced or shifts into many things. To not having a shift/reduce conflict I have added valid_expression_list, if I dont use it and I use just

expression_list: OP expression_list expression OP_CP
| / empty */*

It makes conflicts, it just doesn't know when to shift or when to reduce.

```
function_definition: OP_OP KW_DEFFUN IDENTIFIER parameter_list expression_list OP_CP;

parameter_list: OP_OP parameter_list IDENTIFIER OP_CP
              | IDENTIFIER;

control_statement: OP_OP KW_IF expression expression_list OP_CP
                 | OP_OP KW_IF expression expression_list expression_list OP_CP
                 | OP_OP KW_FOR OP_OP IDENTIFIER expression expression OP_CP expression_list OP_CP;

variable_definition: OP_OP KW_SET IDENTIFIER expression OP_CP;

exit_statement: OP_OP KW_EXIT OP_CP;

print_statement: OP_OP KW_PRINT expression OP_CP;

load_statement: OP_OP KW_LOAD IDENTIFIER OP_CP;
```

These are the other CFGs that I think they're necessary.

PS: I also have used y.output to show the solutions of the CFG to check if there is any conflicts or not. You can find the file into the Flex Folder.

Part 2 - Lisp:

```lisp
(defun tokenize (input)
  (mapcar #'categorize-token (split-text input)))
```

Firstly I use tokenize function to categorize the tokens according to gpp_lexer.lisp (I did that in homework 2)

```lisp
(defun cfg-validate (tokens)
  (when (and (equal (first tokens) 'OP_OP)
             (equal (last tokens) 'OP_CP))
    (cond
      ;; KW_AND, KW_OR, KW_NOT
      ((and (member operator '(KW_AND KW_OR KW_NOT KW_EQUAL KW_LESS))
            (every #'cfg-validate args)) t)
      ;; KW_LIST, KW_APPEND, KW_CONCAT
      ((and (member operator '(KW_LIST KW_APPEND KW_CONCAT))
            (every #'cfg-validate args)) t)
      ;; Arithmetic Operations
      ((and (member operator '(OP_PLUS OP_MINUS OP_MUL OP_DIV))
            (every #'numberp args)) t)
      ;; VALUEI, VALUEF, IDENTIFIER, KW_TRUE, KW_FALSE, KW_NIL
      ((or (member operator '(VALUEI VALUEF IDENTIFIER KW_TRUE KW_FALSE KW_NIL))) t)
      ;; iç paranthesis
      ((and (equal (first args) 'OP_OP)
            (equal (last args) 'OP_CP)
            (cfg-validate (subseq args 1 (1- (length args))))) t)
      (t nil))))
```

I use cfg-validate function to analyze the grammars. I categorize them as much as possible to understand them.

```lisp
(defun gppinterpreter (&optional file)
  (if file
      (progn
        (format t "Loading file: ~a~%" file)
        (process-file file))
      (interactive-mode)))

(defun process-file (file)
  (with-open-file (stream file)
    (loop for line = (read-line stream nil)
          while line
          unless (starts-with ";;" line)
          do (handle-input line))))


(defun handle-input (input)
  (let* ((tokens (tokenize input)))
    (if (cfg-validate tokens)
        (format t "Input: ~a~%Tokens: ~{~a~^ ~}~%Valid Syntax!~%~%" input tokens)
        (format t "Input: ~a~%Tokens: ~{~a~^ ~}~%Error: Invalid Syntax!~%~%" input tokens))))
```

I use gppinterpreter function for the inputs it can be used with file or just simply with the user input.

PS: I also have addition function to check the result and stuff but it doesnt work so I commented them.

```
;;(defun parse (tokens)
;;  (let ((token (pop tokens)))
;;    (cond
;;      ((string= token "(")
;;       (let ((operator (pop tokens))
;;             (args '()))
;;         (loop while (not (string= (first tokens) ")"))
;;               do (push (parse tokens) args))
;;         (pop tokens)
;;         (cons operator (reverse args))))
;;      ((every #'digit-char-p token)
;;       (parse-integer token))
;;      (t token))))


;;(defun evaluate (parsed)
;;  (cond
;;    ((listp parsed)
;;     (let ((operator (first parsed))
;;           (args (rest parsed)))
;;       (case operator
;;         (+ (apply #'+ (mapcar #'evaluate args)))
;;         (- (apply #'- (mapcar #'evaluate args)))
;;         (* (apply #'* (mapcar #'evaluate args)))
;;         (/ (apply #'/ (mapcar #'evaluate args)))
;;         (if (if-evaluator (first args) (second args)))
;;         (set (set-evaluator (first args) (second args)))
;;         (t (error "Unknown işlem: ~a" operator)))))
;;   ((numberp parsed) parsed)
;;    (t parsed)))
```

Final PS: My codes doesn't work. Lisp code says "invalid syntax" even tho they're invalid. I couldn't understand the problem. I'm trying to not to use chatgpt for everything and I'm trying to be an active student so I'll leave them like this.

```
Loading file: test_input.lisp
Input: (+ 5 10)
Tokens: OP_OP OP_PLUS NUMBER NUMBER OP_CP
Error: Invalid Syntax!

Input: (- 15 3)
Tokens: OP_OP OP_MINUS NUMBER NUMBER OP_CP
Error: Invalid Syntax!

Input: (* 2 8)
Tokens: OP_OP OP_MULT NUMBER NUMBER OP_CP
Error: Invalid Syntax!

Input: (/ 20 4)
Tokens: OP_OP OP_DIV NUMBER NUMBER OP_CP
Error: Invalid Syntax!

Input: (if true (+ 1 2))
Tokens: OP_OP KW_IF KW_TRUE OP_OP OP_PLUS NUMBER NUMBER OP_CP OP_CP
Error: Invalid Syntax!
```