PROGRAMMING LANGUAGES – HW3 REPORT

ALPER YAŞAR – 151044072

Part 1:

HW3 G++ Language Syntax Analyzer

LEXER FOR FILE IMPLEMENTED CODE :

> make

> ./a.out base.lisp

LEXER FOR TERMINAL CODE :

> make

> ./a.out

when enter new line to terminal it will ending code and write to file. First read file or line from user and checking syntax errors. Check when needs to use open brackets before some special keywords. it checks it. if did not give error. If closed brackets not equal to open brackets it will give error.

Holding variable names in **id if it is variable, it is OK! But it is not declared as variable program give error. Checking each number variable and keywords it is ok or not ok and tokenize it. When program did not give any error create new temp.lisp and system(cmd) run it before delete temp.list.

Part 2:

In previous lecture, I looking error conditions. when enter new line to terminal it will ending code and write to file. First read file or line from user and checking syntax errors. I reading code and tokenize it. When tokenization I checking syntax is Ok!

“defun errorCheck (lexerList)” :

taking a count when it seen opened brackets it will increase

when closed brackets it will decrease

hold new element as last element when fuction done.

Looking Keyword, operator, integer and variables. Checking them are ok or nor ok.

“defun isKeyword(paramKeyword keywordList)”:

Looking is keyword mentioned. If keyword mentioned and open bracket is found previous keyword it will return syntax ok and add new list as Tokenize character in “defun tokenization(lexerList)”. If there is not open bracket before keyword it will give syntax error.

“defun isOperator(paramOperator operatorList)” :

Running as like as isKeyword function.

defun isInteger(numbers &optional (index 0) (negative 0)):

taking numbers and looking integer is positive or negative.

When number is found looking until end.

“defun isIdentifier(id)” :

“defun checkIdentifiersError (str)”:

Taking variables as local variables and global variables and control it. If there is variable is not declared program will give error. Or variable found in pointer it will return syntax ok!.

When there is no syntax error “defun tokenization(lexerList)” add all tokenize to new list and return it. After that parser section is starting. It is the new homework subject.

Parser for making parser tree. arser takes these lexemes and produces the parse trees as below.

```
"START
     INPUT
         EXPI
             (
             deffun
             ID
                 factorial
             IDLIST
                 (
                 IDLIST
                     ID
                         x
                 )
             EXPLISTI
                 EXPI
                     (
                     if
                     EXPB
                         (
                         ID
                             equal
                         EXPI
                             ID
                                 x
                         EXPI
                             VALUES
                                 IntegerValue
                                     0
                         )
                     EXPLISTI
                         EXPI
                             VALUES
                                 IntegerValue
```

```
                        1
                EXPLISTI
                    EXPI
                        (
                        *
                        EXPI
                            ID
                                x
                        EXPI
                            (
                            ID
                                factorial
                            EXPLISTI
                                EXPI
                                    (
                                    -
                                    EXPI
                                        ID
                                            x
                                    EXPI
                                        VALUES
                                            IntegerValue
                                                1
                                    )
                            )
                        )
                    )
            )"
```

“defun basic(token &optional (parserTree '("EXPI")))”:

This function is sent to the createList function to determine which structure the code in the desired coin has, and then extract the parserTree list of this code.

returns the list of the result parserTree.

;Expression list. This list makes sense of the commands in the list that come as input.

```
(defvar expi '("VALUES" ("while" "EXPB" "EXPLISTI") ("/" "EXPI" "EXPI")
                        ("*" "EXPI" "EXPI") ("+" "EXPI" "EXPI")
                        ("-" "EXPI" "EXPI") "ID" "integer"
                        ("if" "EXPB" "EXPLISTI" "EXPLISTI") ("if" "EXPB" "EXPLISTI")
                        ("deffun" "ID" "IDLIST" "EXPLISTI") ("and" "EXPB" "EXPB")
                        ("or" "EXPB" "EXPB") ("not" "EXPB") ("equal" "EXPI" "EXPI")
                        ("equal" "EXPB" "EXPB") "BinaryValue" ("concat" "EXPLISTI" "EXPLISTI")
                        ("append" "EXPI" "EXPLISTI") "null" "keyword" ("for" "ID" "EXPI" "EXPI" "EXPLISTI")
```

“defun createList(parserList parserResult &optional (rResult '()))” :

Create parseTree list

incoming codes (parser List) create a list according to the expression list.

parserList:sublist , parserResult;return of match-exp (+ expi expi)

“defun match-exp(match-exp-list expList)” :

There are codes from the user in the match-exp-list.

Tokens are determined by comparing with expList.

With the function named closeBrackets, the codes in the desired brackets are taken and

Which pattern they belong to is determined.

expi = list holding expressions checklist; list with (+ x 5)

“defun checkList(cnlliste cnlstring)” :

This method takes a list and a string.

true if string matches the innermost element of the list

If it doesn't match, return nil.

“defun print-tree (tree &optional (stringx ""))” :

hiding the necessary indentation, bottom line, keywords that should not be in the output.

“defun call-func (param)” :

Write parameter to file in lisp format.