**Lab 8**

Circu Mihai

Git: https://github.com/CMihai998/FLCD/tree/master/Lab8

%{

#include <stdio.h>

#include <string.h>

int lines = 0;

%}

%option noyywrap

%option caseless

DIGIT [0-9]

WORD \"[a-zA-Z0-9]\*\"

NUMBER [+-]?[1-9][0-9]\*|0

CHARACTER \'[a-zA-Z0-9]\'

CONST {WORD}|{NUMBER}|{CHARACTER}

ID [a-zA-Z][a-zA-Z0-9\_]

%%

daca {printf("Reserved word: %s\n", yytext);}

uite {printf("Reserved word: %s\n", yytext);}

tipami {printf("Reserved word: %s\n", yytext);}

nr {printf("Reserved word: %s\n", yytext);}

castron {printf("Reserved word: %s\n", yytext);}

DADADA {printf("Reserved word: %s\n", yytext);}

NAH {printf("Reserved word: %s\n", yytext);}

loop {printf("Reserved word: %s\n", yytext);}

si {printf("Reserved word: %s\n", yytext);}

sau {printf("Reserved word: %s\n", yytext);}

tineminte {printf("Reserved word: %s\n", yytext);}

{ID} {printf( "Identifier: %s\n", yytext );}

{CONST} {printf( "Constant: %s\n", yytext );}

"," {printf( "Separator: %s\n", yytext );}

"." {printf( "Separator: %s\n", yytext );}

"{" {printf( "Separator: %s\n", yytext );}

"}" {printf( "Separator: %s\n", yytext );}

"(" {printf( "Separator: %s\n", yytext );}

")" {printf( "Separator: %s\n", yytext );}

"[" {printf( "Separator: %s\n", yytext );}

"]" {printf( "Separator: %s\n", yytext );}

"$" {printf( "Separator: %s\n", yytext );}

"+" {printf( "Operator: %s\n", yytext );}

"-" {printf( "Operator: %s\n", yytext );}

"\*" {printf( "Operator: %s\n", yytext );}

"/" {printf( "Operator: %s\n", yytext );}

"<" {printf( "Operator: %s\n", yytext );}

">" {printf( "Operator: %s\n", yytext );}

"//" {printf( "Operator: %s\n", yytext );}

"bagasubradical" {printf( "Operator: %s\n", yytext );}

"bagatsubradical" {printf( "Operator: %s\n", yytext );}

"=!=" {printf( "Operator: %s\n", yytext );}

"@" {printf( "Operator: %s\n", yytext );}

"&" {printf( "Operator: %s\n", yytext );}

"===" {printf( "Operator: %s\n", yytext );}

"toarna" {printf( "Operator: %s\n", yytext );}

[ \t]+ {}

[\n]+ {lines++;}

[+-]?0[0-9]\* {printf("Illegal constant at line %d\n", lines);}

[0-9][a-zA-Z0-9] {printf("Illegal identifier at line %d\n", lines);}

\'[a-zA-Z0-9] {printf("Expected end of string on line %d\n", lines); }

%%

void main(int argc,char\*\* argv)

{

if (argc > 1)

{

FILE \*file;

file = fopen(argv[1], "r");

if (!file)

{

fprintf(stderr, "Could not open %s\n", argv[1]);

exit(1);

}

yyin = file;

}

yylex();

}

Output:

Reserved word: tineminte

Identifier: ma

Identifier: in

Separator: (

Separator: )

Separator: {

Reserved word: nr

xSeparator: $

Reserved word: castron

Identifier: pr

Identifier: im

Separator: $

Operator: toarna

xOperator: <

Constant: -1

Separator: $

Operator: toarna

Identifier: pr

Identifier: im

Operator: <

Reserved word: DADADA

Separator: $

Reserved word: uite

Operator: @

Expected end of string on line 5

Separator: $

Reserved word: tipami

Operator: &

xSeparator: $

Reserved word: daca

Separator: (

xOperator: bagatsubradical

Constant: 1

Separator: )

Separator: {

Operator: toarna

Identifier: pr

Identifier: im

Operator: <

Identifier: NU

Separator: $

Separator: }

Reserved word: loop

Separator: (

Reserved word: nr

iOperator: <

Constant: 2

Separator: $

iOperator: bagatsubradical

xOperator: //

Constant: 2

Reserved word: si

Identifier: pr

Identifier: im

Operator: ===

Reserved word: DADADA

Separator: $

iIdentifier: cr

Identifier: es

Identifier: te

Separator: )

Separator: {

Reserved word: daca

Separator: (

x%iOperator: ===

Constant: 0

Separator: )

Separator: {

Operator: toarna

Identifier: pr

Identifier: im

Operator: <

Identifier: NU

Separator: $

Separator: }

Separator: }

Reserved word: uite

Operator: @

Identifier: pr

Identifier: im

Separator: $

Separator: }

**PARSER**

%{

#include <stdio.h>

#include <stdlib.h>

#define YYDEBUG 1

%define parse.error verbose

%}

%token IDENTIFIER

%token CONSTANT

%token MAIN

%token IN

%token OUT

%token IF

%token FOR

%token BREAK

%token NUMBER

%token CHAR

%token BOOL

%token TRUE

%token FALSE

%token COLON

%token DOLLAR

%token COMA

%token DOT

%token PLUS

%token MINUS

%token MULTIPLY

%token DIVISION

%token DIVISION\_2

%token MOD

%token LEFT\_ROUND\_PARENTHESIS

%token RIGHT\_ROUND\_PARENTHESIS

%token LEFT\_SQUARE\_PARENTHESIS

%token RIGHT\_SQUARE\_PARENTHESIS

%token LEFT\_CURLY\_PARENTHESIS

%token RIGHT\_CURLY\_PARENTHESIS

%token LESS\_THAN

%token GREATER\_THAN

%token DIFFERENT

%token EQUAL

%token OR

%token AND

%token TOARNA\_STANGA

%token TOARNA\_DREAPTA

%token AFTER\_UITE

%token AFTER\_TIPAMI

%token INCREASE

%token DECREASE

%token ASSIGNMENT

%token SMALLER\_THAN

%token LARGER\_THAN

%start program

%%

program : MAIN stmtlist

cmpstmt : LEFT\_CURLY\_PARENTHESIS stmtlist RIGHT\_CURLY\_PARENTHESIS ;

stmtlist : stmt | stmt stmtlist;

stmt : decl DOLLAR | assignment DOLLAR | toarna DOLLAR | iostmt DOLLAR | ifstmt DOLLAR | forstmt DOLLAR | cmpstmt DOLLAR ;

decl : type IDENTIFIER ;

toarna : ASSIGNMENT term TOARNA\_STANGA CONSTANT | ASSIGNMENT term TOARNA\_DREAPTA CONSTANT | inc\_dec;

assignment : ASSIGNMENT term TOARNA\_STANGA expression | ASSIGNMENT term TOARNA\_DREAPTA expression ;

for\_assignment : type term TOARNA\_STANGA expression | type term TOARNA\_STANGA expression ;

inc\_dec : term INCREASE | term DECREASE ;

iostmt : OUT term | IN term ;

ifstmt : IF LEFT\_ROUND\_PARENTHESIS condition RIGHT\_ROUND\_PARENTHESIS cmpstmt ;

forstmt : FOR LEFT\_ROUND\_PARENTHESIS for\_assignment DOLLAR condition DOLLAR assignment RIGHT\_ROUND\_PARENTHESIS cmpstmt ;

relation : LESS\_THAN | GREATER\_THAN | DIFFERENT | EQUAL ;

expression : term | term PLUS expression | term MINUS expression | term MULTIPLY expression | term DIVISION expression | term MOD expression | LEFT\_ROUND\_PARENTHESIS expression RIGHT\_SQUARE\_PARENTHESIS ;

term : IDENTIFIER | CONSTANT | IDENTIFIER LEFT\_SQUARE\_PARENTHESIS term RIGHT\_SQUARE\_PARENTHESIS ;

type : primitiveType | arrayDeclaration ;

primitiveType : NUMBER | BOOL ;

arrayDeclaration : primitiveType LEFT\_SQUARE\_PARENTHESIS CONSTANT RIGHT\_SQUARE\_PARENTHESIS ;

condition : expression relation expression ;

%%

yyerror(char \*s)

{

printf("%s\n",s);

}

extern FILE \*yyin;

main(int argc, char \*\*argv)

{

if(argc>1) yyin : fopen(argv[1],"r");

if(argc>2 && !strcmp(argv[2],"-d")) yydebug: 1;

if(!yyparse()) fprintf(stderr, "\tO.K.\n");

}

**OUTPUT**

Reserved word: tineminte

Separator: {

Reserved word: nr

Identifier: aux

Separator: $

Reserved word: castron

Identifier: prm

Separator: $

Operator: toarna

Identifier: aux

Operator: <

Constant: 20

Separator: $

Operator: toarna

Identifier: prm

Operator: <

Reserved word: DADADA

syntax error