Lab 8-9

Georgiana Loba

https://github.com/GeorgianaLoba/Formal-Languages-and-Compiler-Design/tree/master/lex-yacc

Statement: Use lex

You may use any version (LEX or FLEX)

1. Write a LEX specification containing the regular expressions corresponding to your language specification - see lab 1
2. Use Lex in order to obtain a scanner. Test for the same input as in lab 1 (p1, p2).

Deliverables: pdf file containing lang.lxi (lex specification file) + demo

Statement: Use yacc

You may use any version (yacc or bison)

1. Write a specification file containing the production rules corresponding to the language specification (use syntax rules from lab1).
2. Then, use the parser generator (no errors)

Deliverables: lang.y (yacc specification file)

Lang.lxi is:

```
%{
   #include<stdio.h>
   #include <string.h>
   #include "y.tab.h"
   int current = 1;
%}

%option noyywrap
```

```
%option caseless

LETTER [a-ZA-Z]
DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
INTEGER [+-]?{NON_ZERO_DIGIT}{DIGIT}*
CHAR \'{LETTER}\'
STRING ["][a-zA-Z]*["]
CONSTANT {INTEGER}|{CHAR}|{STRING}
IDENTIFIER [a-zA-Z][a-ZA-Z0-9_]*

%%

let {return LET;}
func {return FUNC;}
returns {return RETURNS;}
is {return IS;}
and {return AND;}
or {return OR;}
print {return PRINT;}
while {return WHILE;}
if {return IF;}
else {return ELSE;}
then {return THEN;}
print {return PRINT;}
scan {return SCAN;}
```

```
integer {return INTEGER;}

string {return STRING;}

char {return CHAR;}

boolean {return BOOLEAN;}

true {return TRUE;}

false {return FALSE;}

; {return SEMI_COLON;}

"," {return COMMA;}

\{ {return OPEN_CURLY_BRACKET;}

\} {return CLOSED_CURLY_BRACKET;}

\( {return OPEN_ROUND_BRACKET;}

\) {return CLOSED_ROUND_BRACKET;}

\[ {return OPEN_SQUARE_BRACKET;}

\] {return CLOSED_SQUARE_BRACKET;}

\+ {return PLUS;}

\- {return MINUS;}

\* {return MUL;}

\/ {return DIV;}

\% {return PERCENT;}

{IDENTIFIER} {printf("%s id \n", yytext); return IDENTIFIER;}

{CONSTANT} {printf("%s ct \n", yytext); return CONSTANT;}

[\n\r] {current++;}

[ \t] {}

[\"]{CHAR}* {printf("%s - wrong, close yo quotes my dude at line %d\n", yytext, current);}

\'[a-zA-Z]*\' {printf("%s - wrong, is a char, not a string my dude at line %d\n", yytext,
current);}

%%
```

Lang.y is:

```
%{
   #include<stdio.h>
   #include<stdlib.h>
   #define YYDEBUG 1
%}



%token LET
%token FUNC
%token RETURNS
%token IS
%token OR
%token PRINT
%token WHILE
%token RETURN
%token IF
%token ELSE
%token THEN
%token INTEGER
%token BOOLEAN
%token TRUE
%token STRING
%token CHAR
```

```
%token FALSE

%token SCAN

%token PRINT

%token IDENTIFIER

%token CONSTANT

%token RELATION

%token COMMA

%token SEMI_COLON

%token OPEN_SQUARE_BRACKET

%token CLOSED_SQUARE_BRACKET

%token OPEN_CURLY_BRACKET

%token CLOSED_CURLY_BRACKET

%token OPEN_ROUND_BRACKET

%token CLOSED_ROUND_BRACKET

%token PLUS

%token MINUS

%token DIV

%token MUL

%token PERCENT

%token EQ

%token NOT_EQ

%token AND

%start program


%%
```

program: LET FUNC IDENTIFIER function_arguments RETURNS type OPEN_CURLY_BRACKET statement_list CLOSED_CURLY_BRACKET SEMI_COLON

;

type: INTEGER

| BOOLEAN

| STRING

| CHAR

;

statement_list: statement

;

statement: simple_declaration

| assigned_declaration

| assignment

| if_statement

| while_statement

| output_statement

| input_statement

;

simple_declaration: LET type IDENTIFIER SEMI_COLON;

assigned_declaration: LET IDENTIFIER EQ expression SEMI_COLON;

assignment: IDENTIFIER EQ expression SEMI_COLON;

function_arguments: OPEN_ROUND_BRACKET type IDENTIFIER CLOSED_ROUND_BRACKET;

if_statement: IF condition THEN OPEN_CURLY_BRACKET statement_list CLOSED_CURLY_BRACKET ELSE

OPEN_CURLY_BRACKET statement_list CLOSED_CURLY_BRACKET;

while_statement: WHILE condition THEN OPEN_CURLY_BRACKET statement_list CLOSED_CURLY_BRACKET;

```
condition: OPEN_ROUND_BRACKET expression RELATION expression
CLOSED_ROUND_BRACKET;

expression: IDENTIFIER operand IDENTIFIER

   | IDENTIFIER operand CONSTANT

   | CONSTANT operand IDENTIFIER

   | CONSTANT operand CONSTANT

   ;

operand: PLUS

   | MINUS

   | DIV

   | MUL

   | PERCENT

   ;

output_statement: PRINT STRING SEMI_COLON

   | PRINT IDENTIFIER SEMI_COLON

   ;

input_statement: SCAN type IDENTIFIER SEMI_COLON;




%%


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, "no errors\n");
}
```

My program:

```
let func check (integer a) returns integer
{
   print a;
};
```