Lab 8

Jeler Andrei-Iulian

934

Github: https://github.com/SummerRolls99/FLCD/tree/main/lab%20-%20lex%26yacc

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

lang.lxi

```
%{
#include <stdio.h>
#include <string.h>
int currentLine = 1;
%}

%option noyywrap
%option caseless

DIGIT               [0-9]
NZ_DIGIT    [1-9]
ZERO        [0]
INTEGER                     {ZERO}|-*{NZ_DIGIT}+{DIGIT}*
CHARACTER       '"'[^\n]'"'
```

```
STRING          [\"][^\n]*[\"]
CONSTANT                {STRING}|{INTEGER}|{CHARACTER}
IDENTIFIER              [a-zA-Z_][a-zA-Z0-9_]*

%%

and {printf("%s - reserved word\n", yytext);}
or {printf("%s - reserved word\n", yytext);}
not {printf("%s - reserved word\n", yytext);}
if {printf("%s - reserved word\n", yytext);}
else {printf("%s - reserved word\n", yytext);}
elif {printf("%s - reserved word\n", yytext);}
while {printf("%s - reserved word\n", yytext);}
for {printf("%s - reserved word\n", yytext);}
read {printf("%s - reserved word\n", yytext);}
write {printf("%s - reserved word\n", yytext);}
integer {printf("%s - reserved word\n", yytext);}
string {printf("%s - reserved word\n", yytext);}
char {printf("%s - reserved word\n", yytext);}
program {printf("%s - reserved word\n", yytext);}

{CONSTANT} {printf("%s - constant\n", yytext);}
{IDENTIFIER} {printf("%s - identifier\n", yytext);}

; {printf("%s - separator\n", yytext);}
\, {printf("%s - separator\n", yytext);}
\t {printf("%s - separator\n", yytext);}
\{ {printf("%s - separator\n", yytext);}
\} {printf("%s - separator\n", yytext);}
```

```
\[ {printf("%s - separator\n", yytext);}
\] {printf("%s - separator\n", yytext);}
\( {printf("%s - separator\n", yytext);}
\) {printf("%s - separator\n", yytext);}

\+ {printf("%s - operator\n", yytext);}
\- {printf("%s - operator\n", yytext);}
\* {printf("%s - operator\n", yytext);}
\/ {printf("%s - operator\n", yytext);}
\% {printf("%s - operator\n", yytext);}
\< {printf("%s - operator\n", yytext);}
\> {printf("%s - operator\n", yytext);}
\<= {printf("%s - operator\n", yytext);}
\>= {printf("%s - operator\n", yytext);}
"=" {printf("%s - operator\n", yytext);}
\== {printf("%s - operator\n", yytext);}
\!= {printf("%s - operator\n", yytext);}

[\n]+ {currentLine++;}
[ \t\n]+ {}

[a-zA-Z_0-9][a-zA-Z0-9_]* {printf("%s - illegal identifier found at line %d\n", yytext, currentLine);}
\'[a-zA-Z0-9]*\' {printf("%s - illegal char at line %d, did you mean string?\n", yytext, currentLine);}

%%

void main(argc, argv)
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();
}
```

p1.in (file for testing)

```
program
{
integer a, b, c;
string printMessage = "is the biggest number";
read(a);
read(b);
read(c);
a = -2;
if (a > b and a > c)
{
write("a", printMessage);
}
```

```
elif (b > a and b > c)
{
write("b", printMessage);
}
else
{
write("c", printMessage);
}
return 0;
}
```

How to run:

```
lex lang.lxi
gcc lex.yy.c -o lex.exe -ll
./lex.exe p1.in
```

Or

```
./lex.exe < p1.in
```