FLCD/Lab9 at main · 913-Homescu-Monica/FLCD (github.com)

lang.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.tab.h"

int line = 1;
%}

%option noyywrap

IDENTIFIER              \#([A-Za-z]+)
INTEGER_CONSTANT  \-?[1-9][0-9]*|0
CHARACTER_CONSTANT              \'[a-zA-Z0-9]\'
STRING_CONSTANT             \"[a-zA-Z0-9]+\"

%%

"+" { printf("OPERATOR -> %s\n", yytext); return PLUS; }
"-" { printf("OPERATOR -> %s\n", yytext); return MINUS; }
"*" { printf("OPERATOR -> %s\n", yytext); return TIMES; }
"/" { printf("OPERATOR -> %s\n", yytext); return DIV; }
"%" { printf("OPERATOR -> %s\n", yytext); return MOD; }
"=" { printf("OPERATOR -> %s\n", yytext); return EQ; }
```

```
"<=" { printf("OPERATOR -> %s\n", yytext); return LESSEQ; }
"==" { printf("OPERATOR -> %s\n", yytext); return EQQ; }
"!=" { printf("OPERATOR -> %s\n", yytext); return NEQ; }
">=" { printf("OPERATOR -> %s\n", yytext); return BIGGEREQ; }
">" { printf("OPERATOR -> %s\n", yytext); return BIGGER; }
"<" { printf("OPERATOR -> %s\n", yytext); return LESS; }

"{" { printf("SEPARATOR -> %s\n", yytext); return BRACKETOPEN; }
"}" { printf("SEPARATOR -> %s\n", yytext); return BRACKETCLOSE; }
"(" { printf("SEPARATOR -> %s\n", yytext); return OPEN; }
")" { printf("SEPARATOR -> %s\n", yytext); return CLOSE; }
"[" { printf("SEPARATOR -> %s\n", yytext); return SQBRACKETOPEN; }
"]" { printf("SEPARATOR -> %s\n", yytext); return SQBRACKETCLOSE; }
":" { printf("SEPARATOR -> %s\n", yytext); return COLON; }
";" { printf("SEPARATOR -> %s\n", yytext); return SEMICOLON; }
"," { printf("SEPARATOR -> %s\n", yytext); return COMMA; }

"program" { printf("RESERVED WORD -> %s\n", yytext); return PROGRAM; }
"input" { printf("RESERVED WORD -> %s\n", yytext); return INPUT; }
"output" { printf("RESERVED WORD -> %s\n", yytext); return OUTPUT; }
"int" { printf("RESERVED WORD -> %s\n", yytext); return INT; }
"char" { printf("RESERVED WORD -> %s\n", yytext); return CHAR; }
"str" { printf("RESERVED WORD -> %s\n", yytext); return STR; }
"given" { printf("RESERVED WORD -> %s\n", yytext); return GIVEN; }
"then" { printf("RESERVED WORD -> %s\n", yytext); return THEN; }
"otherwise" { printf("RESERVED WORD -> %s\n", yytext); return OTHERWISE; }
"aslongas" { printf("RESERVED WORD -> %s\n", yytext); return ASLONGAS; }
"repeat" { printf("RESERVED WORD -> %s\n", yytext); return REPEAT; }
```

```
{IDENTIFIER}              { printf("IDENTIFIER -> %s\n", yytext); return IDENTIFIER; }

{INTEGER_CONSTANT}                  { printf("INTEGER CONSTANT -> %s\n", yytext); return
INTEGER_CONSTANT; }

{STRING_CONSTANT}           { printf("STRING CONSTANT -> %s\n", yytext); return
STRING_CONSTANT; }

{CHARACTER_CONSTANT}              { printf("CHARACTER CONSTANT -> %s\n", yytext);
return CHARACTER_CONSTANT; }


[ \t]+              {}

[\n]+    {line++;}


. {printf("Error for token %s on line %d!\n", yytext, line); exit(1);}


%%
```

parser.y

```
%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


int yylex();

void yyerror(char *s);


#define YYDEBUG 1

%}


%token PLUS
```

```
%token MINUS
%token TIMES
%token DIV
%token MOD
%token EQ
%token LESSEQ
%token EQQ
%token NEQ
%token BIGGEREQ
%token BIGGER
%token LESS

%token BRACKETOPEN
%token BRACKETCLOSE
%token OPEN
%token CLOSE
%token SQBRACKETOPEN
%token SQBRACKETCLOSE
%token COLON
%token SEMICOLON
%token COMMA

%token PROGRAM
%token INPUT
%token OUTPUT
%token INT
%token CHAR
%token STR
%token GIVEN
```

```
%token THEN
%token OTHERWISE
%token ASLONGAS
%token REPEAT

%token IDENTIFIER
%token INTEGER_CONSTANT
%token STRING_CONSTANT
%token CHARACTER_CONSTANT

%start program

%%

program : PROGRAM compound_statement

decllist : declaration | declaration COMMA decllist

statement : decllist | assignstmt| iostmt | ifstmt | whilestmt

statement_list : statement | statement statement_list

compound_statement : BRACKETOPEN statement_list BRACKETCLOSE

expression : expression PLUS term | expression MINUS term | term

term : term TIMES factor | term DIV factor | term MOD factor | factor

factor : OPEN expression CLOSE | IDENTIFIER | constant
```

constant : INTEGER_CONSTANT | CHARACTER_CONSTANT | STRING_CONSTANT

iostmt : INPUT OPEN IDENTIFIER CLOSE | OUTPUT OPEN IDENTIFIER CLOSE | OUTPUT OPEN constant CLOSE

type : simple_type | array_declaration

simple_type : INT | CHAR | STR

array_declaration : simple_type SQBRACKETOPEN INTEGER_CONSTANT SQBRACKETCLOSE

declaration : type COLON IDENTIFIER

assignstmt : IDENTIFIER EQ expression

ifstmt : GIVEN OPEN condition CLOSE THEN compound_statement | GIVEN OPEN condition CLOSE THEN compound_statement OTHERWISE compound_statement

whilestmt : ASLONGAS OPEN condition CLOSE REPEAT compound_statement

condition : expression relation expression

relation : LESS | LESSEQ | EQQ | NEQ | BIGGEREQ | BIGGER

%%

```
void yyerror(char *s) {
  printf("%s\n", s);
}
```

```
extern FILE *yyin;

int 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,"\nParsed successfully!\n");
}
```

demo

1. Save the files lang.y and parser.y in a folder
2. Open the folder and enter cmd
3. Run the command: *flex lang.y*
4. Run the command: *bison -d parser.y*
5. Run the command: *gcc lex.yy.c parser.tab.c -o parser -L -lfl*
6. Run the command: *parser.exe < p1.txt*