Formal Languages and Compiler Design - Lab9

Requirement

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)

BONUS: modify lex to return tokens and use yacc to return string of productions

Solution

lang.lxi

```
%{
#include <math.h>
%}

NONZERO_DIGIT     [1-9]
DIGIT          [0-9]
INTEGER_CT    0|(-?{NONZERO_DIGIT}{DIGIT}*)
CHAR_CT         \'[A-Z0-9]\'
STRING_CT    \"[A-Z0-9]*\"
BOOLEAN_CT    true|false
ID          [A-Z_][A-Z0-9_]*
ERROR         [+-]0|\".*|.*\"|'.|.'|0{DIGIT}+|{DIGIT}+[A-Z0-9_]+
%%

{INTEGER_CT}     { printf("Integer constant: %s\n", yytext); return ct;}

{CHAR_CT}    { printf("Char constant: %s\n", yytext); return ct; }

{STRING_CT}    { printf("String: %s\n", yytext); return ct; }

{BOOLEAN_CT}    { printf("Boolean constant: %s\n", yytext); return ct; }

"START"         { return START;}
"ENDPRG"    { return ENDPRG; }
"INT"         { return INT; }
"BOOLEAN"    { return BOOLEAN; }
"CHAR"         { return CHAR; }
"STRING"    { return STRING;}
"ARRAY"         { return ARRAY; }
"BEGIN"         { return BEGIN; }
"END"         { return END; }
```

```
"READ"        { return READ; }
"WRITE"        { return WRITE; }
"IF"        { return IF; }
"THEN"        { return THEN; }
"ELSE"        { return ELSE; }
"WHILE"        { return WHILE; }
"DO"        { return DO; }

{ID}         { return id;}

"+"|" - "|"*"|"/"|"%"|"<"|"<="|">"|">="|"="|"!="|":="|"AND"|"OR"        printf("Operat

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

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

"{"[^}\n]*"}"         /* eat up one-line comments */

[ \t\n]+         /* eat up whitespace */

. printf("Eroare\n");
%%
main( argc, argv )
int argc;
char **argv;
{
    ++argv, --argc; /* skip over program name */
    if ( argc > 0 )
    yyin = fopen( argv[0], "r" );
    else
     yyin = stdin;
    yylex();
}
```

lang.y

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

%token START
%token ENDPRG
%token BEGIN
%token END
%token READ
%token WRITE
%token IF
%token THEN
%token ELSE
%token WHILE
%token DO
```

```
%token id
%token ct

%token INT
%token BOOLEAN
%token CHAR
%token STRING
%token ARRAY

%token ADD
%token SUBTRACT
%token MULTIPLY
%token DIV
%token MOD
%token SMALLER
%token SMALLER_OR_EQUAL
%token GREATER
%token GREATER_OR_EQUAL
%token EQUAL
%token DIFFERENT
%token ASSIGNED
%token AND
%token OR

%token PARA_OPEN
%token PARA_CLOSE
%token SQUARE_BRACKET_OPEN
%token SQUARE_BRACKET_CLOSE
%token CURLY_BRACKET_OPEN
%token CURLY_BRACKET_CLOSE
%token SEMI_COLON
%token COLON

%%

program:    START decllist compstmt ENDPRG
        ;
decllist:
        | declaration SEMI_COLON decllist
        ;
declaration:     id COLON type
        ;
simple_type:    INT
        | BOOLEAN
        | CHAR
        | STRING
        ;
array_type:    ARRAY SQUARE_BRACKET_OPEN INT SQUARE_BRACKET_CLOSE simple_type
        ;
type:         simple_type
        | array_type
        ;
compstmt:    BEGIN stmtlist END
        ;
stmtlist:
```

```
        | stmt SEMI_COLON stmtlist
        ;
stmt:         simple_stmt
        | struct_stmt
        ;
simple_stmt:    assign_stmt
        | io_stmt
        ;
assign_stmt:    id SEMI_COLON expression
        ;
expression:    term signed_expression
        ;
signed_expresson:
            | operator expression
            ;
term:         id
        | ct
        ;
operator:    ADD
        | SUBTRACT
        | MULTIPLY
        | DIV
        | MOD
        ;
io_stmt:    READ PARA_OPEN id PARA_CLOSE
        | WRITE PARA_OPEN id PARA_CLOSE
        ;
struct_stmt:    compstmt
        | ifstmt
        | whilestmt
        ;
ifstmt:         IF condition THEN stmtlist elsestmt
        ;
elsestmt:
        | ELSE stmtlist
        ;
whilestmt:    WHILE condition DO stmtlist
        ;
condition:    expression RELATION expression
        ;
RELATION:    SMALLER
        | SMALLER_OR_EQUAL
        | GREATER
        | GREATER_OR_EQUAL
        | EQUAL
        | DIFFERENT
        | ASSIGNED
        | AND
        | OR
        ;
%%

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");
}
```