Documentation YACC

LEX file - specif.lxi:

```
%{
   #include <stdio.h>
   #include <string.h>
   #include "y.tab.h"
   int lines = 0;
%}

%option noyywrap
%option caseless

DIGIT               [0-9]
NON_ZERO_DIGIT [1-9]
INT_CONSTANT [-]?{NON_ZERO_DIGIT}{DIGIT}*|0
LETTER [a-zA-Z]
STRING_CONSTANT ["|'](({LETTER}|{DIGIT}|" ")*["|']
IDENTIFIER {LETTER}({LETTER}|{DIGIT})*

%%

func {printf("Reserved word: %s\n", yytext); return FUNC;}
Int    {printf( "Reserved word: %s\n", yytext); return INT;}
String    {printf( "Reserved word: %s\n", yytext); return STRING;}
Bool    {printf( "Reserved word: %s\n", yytext); return BOOL;}
Char    {printf( "Reserved word: %s\n", yytext); return CHAR;}
if    {printf( "Reserved word: %s\n", yytext); return IF;}
elif    {printf( "Reserved word: %s\n", yytext); return ELIF;}
else    {printf( "Reserved word: %s\n", yytext); return ELSE;}
let    {printf( "Reserved word: %s\n", yytext); return LET;}
var    {printf( "Reserved word: %s\n", yytext); return VAR;}
ret    {printf( "Reserved word: %s\n", yytext); return RET;}
True    {printf( "Reserved word: %s\n", yytext); return TRUE;}
False    {printf( "Reserved word: %s\n", yytext); return FALSE;}
read    {printf( "Reserved word: %s\n", yytext); return READ;}
print    {printf( "Reserved word: %s\n", yytext); return PRINT;}
loop    {printf( "Reserved word: %s\n", yytext); return LOOP;}
GO      {printf( "Reserved word: %s\n", yytext); return GO;}
STOP  {printf( "Reserved word: %s\n", yytext); return STOP;}

{IDENTIFIER}    {printf( "Identifier: %s\n", yytext ); return ID;}
{INT_CONSTANT}      {printf( "Constant: %s\n", yytext ); return INT_CONST;}
```

```
{STRING_CONSTANT}    {printf( "Constant: %s\n", yytext ); return STRING_CONST;}

"@"    {printf( "Separator: %s\n", yytext ); return AT_SIGN;}
"#"    {printf( "Separator: %s\n", yytext ); return HASHTAG;}
";"    {printf( "Separator: %s\n", yytext ); return SEMI_COLON;}
"'"    {printf( "Separator: %s\n", yytext ); return APOSTROPHE;}
"{"    {printf( "Separator: %s\n", yytext ); return OPEN_CURLY_BRACKET;}
"}"    {printf( "Separator: %s\n", yytext ); return CLOSED_CURLY_BRACKET;}
"("    {printf( "Separator: %s\n", yytext ); return OPEN_ROUND_BRACKET;}
")"    {printf( "Separator: %s\n", yytext ); return CLOSED_ROUND_BRACKET;}
"["    {printf( "Separator: %s\n", yytext ); return OPEN_STRAIGHT_BRACKET;}
"]"    {printf( "Separator: %s\n", yytext ); return CLOSED_STRAIGHT_BRACKET;}
"+"    {printf( "Operator: %s\n", yytext ); return PLUS;}
"-"    {printf( "Operator: %s\n", yytext ); return MINUS;}
"*"    {printf( "Operator: %s\n", yytext ); return MUL;}
"/"    {printf( "Operator: %s\n", yytext ); return DIV;}
"<"    {printf( "Operator: %s\n", yytext ); return LESS;}
">"    {printf( "Operator: %s\n", yytext ); return GREATER;}
"<="    {printf( "Operator: %s\n", yytext ); return LESS_EQ;}
">="    {printf( "Operator: %s\n", yytext ); return GREATER_EQ;}
"!="    {printf( "Operator: %s\n", yytext ); return DIFF;}
"=="    {printf( "Operator: %s\n", yytext ); return EQUAL;}
"="    {printf( "Separator: %s\n", yytext ); return ATRIBUTION;}
"!"    {printf( "Operator: %s\n", yytext ); return NEGATION;}
"?:"    {printf( "Operator: %s\n", yytext ); return TERNARY_OP;}
"%"    {printf( "Operator: %s\n", yytext ); return MOD;}


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

.           {printf("Error at token %s at line %d\n", yytext, lines); return -1;}

%%
```

YACC file - lang.y:

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

int yylex();
```

```
int yyerror(char *s);

#define YYDEBUG 1
%}

%token FUNC
%token INT
%token STRING
%token BOOL
%token CHAR
%token IF
%token ELIF
%token ELSE
%token LET
%token VAR
%token RET
%token TRUE
%token FALSE
%token READ
%token PRINT
%token LOOP
%token GO
%token STOP

%token ID
%token INT_CONST
%token STRING_CONST

%token AT_SIGN
%token HASHTAG
%token SEMI_COLON
%token APOSTROPHE
%token OPEN_CURLY_BRACKET
%token CLOSED_CURLY_BRACKET
%token OPEN_ROUND_BRACKET
%token CLOSED_ROUND_BRACKET
%token OPEN_STRAIGHT_BRACKET
%token CLOSED_STRAIGHT_BRACKET
%token PLUS
%token MINUS
%token MUL
%token DIV
%token LESS
%token GREATER
```

```
%token LESS_EQ
%token GREATER_EQ
%token DIFF
%token EQUAL
%token ATRIBUTION
%token NEGATION
%token TERNARY_OP
%token MOD

%left '+' '-' '*' '/'

%start program

%%

program : GO tempDecl STOP ;
tempDecl : /*Empty*/ | tempDecl declList | tempDecl stmtList ;
declList : declaration | declaration declList ;
declaration : variableDeclaration | constDeclaration ;
variableDeclaration : VAR ID AT_SIGN type ATRIBUTION expression SEMI_COLON | VAR ID
AT_SIGN type SEMI_COLON ;
constDeclaration : LET ID AT_SIGN type ATRIBUTION expression SEMI_COLON ;
type1 : BOOL | INT | CHAR | STRING ;
arrayDecl : OPEN_STRAIGHT_BRACKET type1 CLOSED_STRAIGHT_BRACKET ;
type : type1 | arrayDecl ;
stmtList : stmt | stmt stmtList ;
stmt : simplStmt | structStmt ;
simplStmt : assignStmt | ioStmt ;
assignStmt : ID ATRIBUTION expression SEMI_COLON;
expression : expression PLUS term | expression MINUS term | term | BOOL ;
term : term MUL factor | term DIV factor | term MOD factor | factor ;
factor : OPEN_ROUND_BRACKET expression CLOSED_ROUND_BRACKET | ID |
INT_CONST ;
ioStmt : READ OPEN_ROUND_BRACKET ID CLOSED_ROUND_BRACKET SEMI_COLON |
PRINT OPEN_ROUND_BRACKET stringExp CLOSED_ROUND_BRACKET SEMI_COLON ;
stringExp : STRING_CONST | ID ;
structStmt : ifStmt | whileStmt ;
ifStmt : IF condition OPEN_CURLY_BRACKET stmtList CLOSED_CURLY_BRACKET
tempElifStmt | IF condition OPEN_CURLY_BRACKET stmtList CLOSED_CURLY_BRACKET
tempElifStmt elseStmt ;
tempElifStmt : /*Empty*/ | tempElifStmt elifStmt ;
elseStmt : ELSE OPEN_CURLY_BRACKET stmtList CLOSED_CURLY_BRACKET ;
elifStmt : ELIF condition OPEN_CURLY_BRACKET stmtList CLOSED_CURLY_BRACKET ;
whileStmt : LOOP condition OPEN_CURLY_BRACKET stmtList CLOSED_CURLY_BRACKET ;
```

```
condition : expression relation expression ;
relation : LESS | LESS_EQ | EQUAL | GREATER_EQ | GREATER | DIFF ;

%%
int yyerror(char *s) {
	printf("Error: %s", s);
}

extern FILE *yyin;

int main(int argc, char** argv) {
	if (argc > 1)
	yyin = fopen(argv[1], "r");
	if (!yyparse())
	fprintf(stderr, "\tOK\n");
}
```

p1.txt:

```
GO
let count@Int = 0;
let n@Int = 432567;

loop n != 0 {
	n = n / 10;
	count = count + 1;
}
n = -3;
print(count);
if n < 2 {
	print("yes");
}
STOP
```

res.exe:

```
Reserved word: GO
Reserved word: let
Identifier: count
Separator: @
Reserved word: Int
Separator: =
Constant: 0
Separator: ;
Reserved word: let
```

Identifier: n
Separator: @
Reserved word: Int
Separator: =
Constant: 432567
Separator: ;
Reserved word: loop
Identifier: n
Operator: !=
Constant: 0
Separator: {
Identifier: n
Separator: =
Identifier: n
Operator: /
Constant: 10
Separator: ;
Identifier: count
Separator: =
Identifier: count
Operator: +
Constant: 1
Separator: ;
Separator: }
Identifier: n
Separator: =
Constant: -3
Separator: ;
Reserved word: print
Separator: (
Identifier: count
Separator:)
Separator: ;
Reserved word: if
Identifier: n
Operator: <
Constant: 2
Separator: {
Reserved word: print
Separator: (
Constant: "yes"
Separator:)
Separator: ;
Separator: }

Reserved word: STOP
OK