lang.lxi

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

%option noyywrap

IDENTIFIER              [a-zA-Z_]([a-zA-Z0-9_]*)
NUMBER_CONST            0|[+|-][1-9]([0-9])*|[1-9]([0-9])*
STRING_CONST [\"][a-zA-Z0-9_?!#*,./%-+=<>;(){} ]*[\"]



%%

"if"          {printf("Reserved word: %s\n", yytext); return IF;}
"@"           {printf("Reserved word: %s\n", yytext); return AT;}
"else"        {printf("Reserved word: %s\n", yytext); return ELSE;}
"read"        {printf("Reserved word: %s\n", yytext); return READ;}
"write"       {printf("Reserved word: %s\n", yytext); return WRITE;}
"integer"     {printf("Reserved word: %s\n", yytext); return INTEGER;}
"string"      {printf("Reserved word: %s\n", yytext); return STRING;}
"for"         {printf("Reserved word: %s\n", yytext); return FOR;}
"in"          {printf("Reserved word: %s\n", yytext); return IN;}
"range"       {printf("Reserved word: %s\n", yytext); return RANGE;}
```

```
"while"        {printf("Reserved word: %s\n", yytext); return WHILE;}

"+"            {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 division;}
"%"            {printf("Operator %s\n", yytext); return mod;}
"<="           {printf("Operator %s\n", yytext); return lessOrEqual;}
">="           {printf("Operator %s\n", yytext); return moreOrEqual;}
"<"            {printf("Operator %s\n", yytext); return  less;}
">"            {printf("Operator %s\n", yytext); return more;}
"=="           {printf("Operator %s\n", yytext); return equal;}
"!="           {printf("Operator %s\n", yytext); return different;}
"="            {printf("Operator %s\n", yytext); return eq;}
"and"          {printf("Operator %s\n", yytext); return and;}
"or"           {printf("Operator %s\n", yytext); return or;}

"{"            {printf("Separator %s\n", yytext); return leftCurlyBracket;}
"}"            {printf("Separator %s\n", yytext); return rightCurlyBracket;}
"("            {printf("Separator %s\n", yytext); return leftRoundBracket;}
")"            {printf("Separator %s\n", yytext); return rightRoundBracket;}
"["            {printf("Separator %s\n", yytext); return leftBracket;}
"]"            {printf("Separator %s\n", yytext); return rightBracket;}
":"            {printf("Separator %s\n", yytext); return colon;}
";"            {printf("Separator %s\n", yytext); return semicolon;}
","            {printf("Separator %s\n", yytext); return comma;}
"'"            {printf("Separator %s\n", yytext); return apostrophe;}
"\""           {printf("Separator %s\n", yytext); return quote;}
" "            {printf("Separator %s\n", yytext); return space;}
```

```
"."           {printf("Separator %s\n", yytext); return dot;}

{IDENTIFIER}    {printf("Identifier: %s\n", yytext); return IDENTIFIER;}
{NUMBER_CONST}  {printf("Number: %s\n", yytext); return NUMBER_CONST;}
{STRING_CONST}  {printf("String: %s\n", yytext); return STRING_CONST;}

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

[0-9][a-zA-Z0-9_]*                  {printf("\nINVALID IDENTIFIER at line %d, %s\n", currentLine,yytext);}
[+|-]0              {printf("\nINVALID NUMBER CONSTANT at line %d\n", currentLine);}

%%
```

lang.y

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

#define YYDEBUG 1
%}


%token IF
%token AT
%token ELSE
%token READ
%token WRITE
%token INTEGER
```

```
%token STRING
%token FOR
%token IN
%token RANGE
%token WHILE

%token plus
%token minus
%token mul
%token division
%token mod
%token lessOrEqual
%token moreOrEqual
%token less
%token more
%token equal
%token different
%token eq
%token and
%token or

%token leftCurlyBracket
%token rightCurlyBracket
%token leftRoundBracket
%token rightRoundBracket
%token leftBracket
%token rightBracket
%token colon
%token semicolon
```

```
%token comma
%token apostrophe
%token quote
%token space
%token dot

%token IDENTIFIER
%token NUMBER_CONST
%token STRING_CONST

%start program


%%


program : AT compound_statement AT

compound_statement : declaration_list statement_list

declaration_list : declaration | declaration declaration_list

declaration : type identifier_list semicolon | type assignment_stmt semicolon

type : INTEGER | STRING

identifier_list : identifier | identifier leftBracket noconst rightBracket | identifier comma identifier_list

statement_list : statement | statement statement_list
```

statement : read_stmt | write_stmt | if_stmt | for_stmt | while_stmt | assignment_stmt

read_stmt : READ leftRoundBracket identifier_list rightRoundBracket semicolon

write_stmt : WRITE leftRoundBracket identifier_list | identifier_list stringconst | stringconst rightRoundBracket semicolon

if_stmt : IF condition colon leftCurlyBracket statement_list rightCurlyBracket ELSE colon leftCurlyBracket statement_list rightCurlyBracket | IF condition colon leftCurlyBracket statement_list rightCurlyBracket

for_stmt : FOR identifier IN RANGE range_expr colon leftCurlyBracket statement_list rightCurlyBracket

range_expr : leftRoundBracket expression comma expression rightRoundBracket | leftRoundBracket expression comma expression comma noconst rightRoundBracket

while_stmt : WHILE condition colon leftCurlyBracket statement_list rightCurlyBracket

assignment_stmt : identifier eq expression semicolon

condition : expression operator expression | condition operator condition

operator : less | lessOrEqual | more | moreOrEqual | eq | and | or | equal

expression : term | expression plus term | expression minus term

term : factor | term mul factor | term division factor | term mod factor

factor : identifier_list | noconst | stringconst | leftRoundBracket expression rightRoundBracket

```
identifier : IDENTIFIER

noconst : NUMBER_CONST

stringconst : STRING_CONST




%%

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

extern FILE *yyin;

void main(int argc, char** argv)
{
if (argc > 1)
 {
   yyin = fopen(argv[1], "r");
   if (!yyin)
   {
     printf("'%s': Could not open specified file\n", argv[1]);
     return 1;
   }
 }
 if (argc > 2 && strcmp(argv[2], "-d") == 0)
```

```
  {
    yydebug = 1;
  }

  printf("Starting parsing...\n");

  if (yyparse() == 0)
  {
    printf("\tProgram is syntactically correct.\n");
    return 0;
  }

  printf("\tProgram is NOT syntactically correct.\n");
  return 0;
}
```