https://github.com/trutadan/University-Work/tree/main/Semester%205/Formal%20Languages%20and%20Compiler%20Design/lab08

# Lex specification file

The provided Lex file defines a lexical analyzer for my simple programming language defined in the first lab. This analyzer recognizes various lexical elements in the source code and prints corresponding messages for each recognized token:

1. Reserved Words:
- Keywords of the programming language are recognized and printed with their line numbers.
- Reserved words include: array, bool, string, int, char, elif, if, return, else, do, end, gets, puts, cuts, to_i, while, start, stop, nil, and, break, or, for, in.

2. Operators:
- Arithmetic and comparison operators are recognized and printed with their line numbers.
- Operators include: +, -, *, /, %, <, <=, ==, !=, >=, >, =.

3. Separators:
- Various separators are recognized and printed with their line numbers.
- Separators include: [, ], {, }, (, ), ,, :, ;, ', ".

4. Constants:
- Boolean, integer, character, and string constants are recognized and printed with their line numbers.

5. Identifiers:
- Valid identifiers are recognized and printed with their line numbers.

6. Ignored Elements:
- Whitespaces (spaces and tabs) are ignored.
- Newline characters increase the current line number.

7. Illegal Elements:
- Illegal identifiers, integer constants, character constants, and string constants are recognized and appropriate error messages are printed.

8. Unrecognized Tokens:
- Any unrecognized token is identified(if it does not belong to any category from above) and an error message is printed.

spec.lxi:

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

%option noyywrap

BOOL_CONST       true|false
INT_CONST 0|[+|-]?[1-9][0-9]*
CHAR_CONST       [\'][a-zA-Z0-9][\']
STRING_CONST     [\"][a-zA-Z0-9 ]*[\"]
IDENTIFIER  [a-zA-Z_][a-zA-Z0-9_]*

%%

"array"|"bool"|"string"|"int"|"char"|"elif"|"if"|"return"        {printf("Line %d - reserved word:
%s\n", currentLine, yytext);}
"else"|"do"|"end"|"gets"|"puts"|"cuts"|"to_i"|"while"                {printf("Line %d - reserved
word: %s\n", currentLine, yytext);}
"start"|"stop"|"nil"|"and"|"break"|"or"|"for"|"in"               {printf("Line %d - reserved word:
%s\n", currentLine, yytext);}

"+"|"-"|"*"|"/"|"%"|"<"|"<="|"=="|"!="|">="|">"|"="    {printf("Line %d - operator: %s\n",
currentLine, yytext);}
"["|"]"|"{"|"}"|"("|")"|","|":"|";"|""|"\""             {printf("Line %d - separator: %s\n", currentLine,
yytext);}

{BOOL_CONST}     {printf("Line %d - boolean constant: %s\n", currentLine, yytext);}
{INT_CONST}      {printf("Line %d - integer constant: %s\n", currentLine, yytext);}
{CHAR_CONST}     {printf("Line %d - character constant: %s\n", currentLine, yytext);}
{STRING_CONST}  {printf("Line %d - string constant: %s\n", currentLine, yytext);}
{IDENTIFIER}     {printf("Line %d - identifier: %s\n", currentLine, yytext);}

[ \t]+    {/* Ignore whitespaces */}
[\n]+    {currentLine++;}

[0-9][a-zA-Z0-9_]*                                {printf("@ Line %d - illegal identifier: %s
@\n", currentLine, yytext);}
```

```
[+|-]0|[+|-]?0[0-9]+                                    {printf("@ Line %d - illegal integer
constant: %s @\n", currentLine, yytext);}
[\'][a-zA-Z0-9 ]{2}[\']|[\'][a-zA-Z0-9 ]{2,}[\'] {printf("@ Line %d - illegal character constant:
%s @\n", currentLine, yytext);}
[\"][a-zA-Z0-9_]+|[a-zA-Z0-9_]+[\"]                      {printf("@ Line %d - illegal string
constant: %s @\n", currentLine, yytext);}
.                                                       {printf("@ Line %d - unrecognized
token: %s @\n", currentLine, yytext);}

%%

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();
}
```

## **Usage:**

- Use the Lex compiler to generate the Lex file:

```
flex spec.lxi
```

- Compile the Lex file:

```
gcc lex.yy.c -o lex
```

- Run the lexer: lex.exe <source_code_file>

```
lex.exe p1.txt
```

You can see the results of running the lexer on the lab1 program files by checking the output text files found in the repository.