Github repo: https://github.com/915-Petruta-Razvan/LFTC_Labs

Documentation

lang.lxi

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

    int currentLine = 1;
%}

%option noyywrap
%option case-insensitive

DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
UNSIGNED_INTEGER {NON_ZERO_DIGIT}{DIGIT}*
INT_CONSTANT [+-]?{UNSIGNED_INTEGER}|0
LETTER [a-zA-Z_]
OTHER_CHAR [ ?!,.]
STRING_CONSTANT (\"({LETTER}|{DIGIT}|{OTHER_CHAR})*\")
IDENTIFIER {LETTER}({LETTER}|{DIGIT})*
WRONG_IDENTIFIER {DIGIT}({LETTER}|{DIGIT})*{LETTER}

%%

"int"|"char"|"string"|"vector"|"if"|"else"|"while"|"read"|"w
rite" {printf("Keyword: %s\n", yytext);}

"+"|"-"|"*"|"/"|"%" {printf("Arithmetic operator: %s\n",
yytext);}

"=="|"!="|"<="|">="|"<"|">" {printf("Relational operator:
%s\n", yytext);}

"=" {printf("Assignment operator: %s\n", yytext);}

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

{IDENTIFIER} {printf("Identifier: %s\n", yytext);}

{WRONG_IDENTIFIER} {printf("Wrong identifier: %s\n",
yytext); exit(1);}
```

```
{INT_CONSTANT} {printf("Integer constant: %s\n", yytext);}

{STRING_CONSTANT} {printf("String constant: %s\n", yytext);}

[ \t]+ {}

[\n]+ {currentLine++;}

. {printf("Unknown token: %s at line %d\n", yytext,
currentLine); exit(1);}

%%

int main(int argc, char **argv)
{
    if (argc != 2)
    {
        printf("Usage: ./lang.exe <input file>\n");
        exit(1);
    }

    FILE *fp = fopen(argv[1], "r");
    if (fp == NULL)
    {
        printf("Cannot open file %s\n", argv[1]);
        exit(1);
    }

    yyin = fp;
    yylex();
    fclose(fp);

    return 0;
}
```

About

This Lex program serves as a lexical analyzer for a simple programming language. It reads an input file containing source code and tokenizes it, classifying various elements like keywords, identifiers, operators, and constants. The program also handles line number tracking and error reporting for unrecognized tokens.

Usage

- Generate the C code from the lex file:

```
flex lang.lxi
```
- Compile the generated C code to get an executable:
```
gcc lex.yy.c –o lang.exe
```
- Run the executable with an input file containing the source code:
```
./lang.exe p1.txt
```

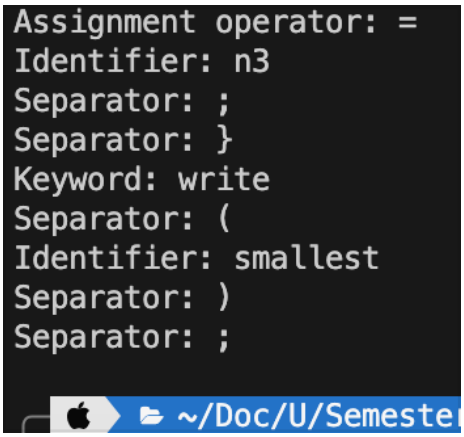
Header Section
- Added C libraries
- Initialised the line counter

Lex Options
- noyywrap: disables the default end-of-file behaviour of Lex
- case-insensitive: makes the token matching process case insensitive

Regular expressions
- DIGIT: from 0 to 9
- NON_ZERO_DIGIT: from 1 to 9
- UNSIGNED_INTEGER: starts with a non-zero digit and contains any number of digits afterwards
- INT_CONSTANT: can have a specified sign for a UNSIGNED_INTEGER or it can be 0
- LETTER: any letter from a to z, A to Z or _ character
- OTHER_CHARS: other possible chars in string constants
- STRING_CONSTANT: any combination of letters, digits and other chars, enclosed between “ and “
- IDENTIFIER: start with a letter, then can have any number of letters and digits afterwards
- WRONG_IDENTIFIER: cannot start with a digit

Running example for p1.txt:

```
~/Doc/U/Semester5/FormalLanguage
./lang.exe p1.txt
Keyword: int
Identifier: n1
Separator: ;
Keyword: int
Identifier: n2
Separator: ;
Keyword: int
Identifier: n3
Separator: ;
Keyword: int
Identifier: smallest
Separator: ;
Keyword: read
Separator: (
Identifier: n1
Separator: )
Separator: ;
Keyword: read
Separator: (
Identifier: n2
Separator: )
Separator: ;
Keyword: read
Separator: (
Identifier: n3
Separator: )
Separator: ;
Identifier: smallest
Assignment operator: =
Identifier: n1
Separator: ;
Keyword: if
Separator: (
Identifier: n2
Relational operator: <
Identifier: smallest
Separator: )
Separator: {
Identifier: smallest
Assignment operator: =
Identifier: n2
Separator: ;
Separator: }
Keyword: if
Separator: (
Identifier: n3
Relational operator: <
Identifier: smallest
```


Assignment operator: =
Identifier: n3
Separator: ;
Separator: }
Keyword: write
Separator: (
Identifier: smallest
Separator:)
Separator: ;
~/Doc/U/Semeste