LEX

Problem Statement:

Use lex

You may use any version (LEX or FLEX)

1) Write a LEX specification containing the regular expressions corresponding to your language specification - see lab 1

2) Use Lex in order to obtain a scanner. Test for the same input as

in lab 1 (p1, p2).

Deliverables: pdf file containing lang.lxi (lex specification file) + demo

Lang.lxi:

```
%{
#include <stdio.h>
int line = 0;
%}

%option noyywrap
%option caseless

IDENTIFIER ([a-zA-Z][a-zA-Z0-9]{0,255})|([a-zA-Z][a-zA-Z0-9_]{0,254}[a-zA-Z0-9])
```

```
INTEGER_CONSTANT        ([+-]?[1-9][0-9]*)|0

CHARACTER_CONSTANT      \'[a-zA-Z0-9]\'

STRING_CONSTANT         \"[-a-zA-Z0-9_:;!?.+/%*&#<>= ^]*\"

BOOLEAN_CONSTANT        true|false


OPERATOR                [-+*/%<>]|">>"|"<<"|"<-"|"->"|"<="|">="|"=="|"!="


SEPARATOR               \[|\]|\{|\}|\(|\)|;|:|,


RESERVED
program|main|const|declarations|statements|integer|character|boolean|string|array|in|out|while|for|
if|else|and|or


%%


{OPERATOR}              printf("Operator: %s\n", yytext);

{SEPARATOR}             printf("Separator: %s\n", yytext);

{INTEGER_CONSTANT}      printf("Integer: %s\n", yytext);

{CHARACTER_CONSTANT}    printf("Character: %s\n", yytext);

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

{BOOLEAN_CONSTANT}      printf("Boolean: %s\n", yytext);

{RESERVED}              printf("Reserved: %s\n", yytext);

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

```
[ \t]                   /* eat up whitespace */
[\n]                     { line++; }
.                       { printf("Error: %s on line: %d\n", yytext, line); }

[0-9_]+{IDENTIFIER}|{IDENTIFIER}_+      printf("Error: %s on line: %d\n", yytext, line);
"+0"|"-0"|[-+]?"0"[0-9]+                printf("Error: %s on line: %d\n", yytext, line);
%%

void 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();
}
```

Tests:

P1.txt

```
PROGRAM
    MAIN -> {
        DECLARATIONS
            INTEGER: nr_1, nr_2, nr_3, min;
            STRING: output_message <- "The minimum of the 3 numbers is: ";
        STATEMENTS
        {
            in>>nr_1, nr_2, nr_3;
            if (nr_1 < nr_2)
                min <- nr_1;
            else
                min <- nr_2;
            if (nr_3 < min)
                min <- nr_3;
            out<<output_message, min;
        }
}
```

Reserved: PROGRAM

Reserved: MAIN

Operator: ->

Separator: {

Reserved: DECLARATIONS

Reserved: INTEGER

Separator: :

Identifier: nr_1

Separator: ,

Identifier: nr_2

Separator: ,

Identifier: nr_3

Separator: ,

Identifier: min

Separator: ;

Reserved: STRING

Separator: :

Identifier: output_message

Identifier: nr_3

Separator: ;

Reserved: out

Operator: <<

Operator: <-

String: "The minimum of the 3 numbers is: "

Separator: ;

Reserved: STATEMENTS

Separator: {

Reserved: in

Operator: >>

Identifier: nr_1

Separator: ,

Identifier: nr_2

Separator: ,

Identifier: nr_3

Separator: ;

Reserved: if

Separator: (

Identifier: nr_1

Operator: <

Identifier: nr_2

Identifier: output_message

Separator: ,

Identifier: min

Separator: ;

Separator:)

Identifier: min

Operator: <-

Identifier: nr_1

Separator: ;

Reserved: else

Identifier: min

Operator: <-

Identifier: nr_2

Separator: ;

Reserved: if

Separator: (

Identifier: nr_3

Operator: <

Identifier: min

Separator:)

Identifier: min

Operator: <-

Separator: }

Separator: }

P1err.txt

```
ROGRAM
    MAIN -> {
        DECLARATIONS
            INTEGER: 1_nr, nr_2, nr_3, min <- 01;
            STRING: output_message <- "min: ;
        STATEMENTS
        {
            in>>nr_1, nr_2, nr_3;
            if (nr_1 < nr_2)
                min <- nr_1;
            else
                min <- nr_2;
            if (nr_3 < min)
                min <- nr_3;
            out<<output_message, min;
        }
    }
```

Identifier: ROGRAM

Reserved: MAIN

Operator: ->

Separator: {

Reserved: DECLARATIONS

Reserved: INTEGER

Separator: :

Error: 1_nr on line: 3

Separator: ,

Identifier: nr_2

Separator: ,

Identifier: nr_3

Separator: ,

Identifier: min

Operator: <-

Error: 01 on line: 3

Separator: ;

Reserved: STRING

Separator: :

Identifier: output_message

Operator: <-

Error: " on line: 4

Identifier: min

Separator: :

Separator: ;

Reserved: STATEMENTS

Separator: {

Reserved: in

Operator: >>

Identifier: nr_1

Separator: ,

Identifier: nr_2

Separator: ,

Identifier: nr_3

Separator: ;

Reserved: if

Separator: (

Identifier: nr_1

Operator: <

Identifier: nr_2

Separator:)

Identifier: min

Operator: <-

Identifier: nr_1

Separator: ;

Reserved: else

Identifier: min

Operator: <-

Identifier: nr_2

Separator: ;

Reserved: if

Separator: (

Identifier: nr_3

Operator: <

Identifier: min

Separator:)

Identifier: min

Operator: <-

Identifier: nr_3

Separator: ;

Reserved: out

Operator: <<

Identifier: output_message

Separator: ,

Identifier: min

Separator: ;

Separator: }

Separator: }

P2.txt

```
PROGRAM
    MAIN -> {
    DECLARATIONS
        INTEGER: input_number, i;
        BOOLEAN: prime <- true;
    STATEMENTS
    {
        in("Give a number: ")>>input_number;
        for(i <- 2; i <= input_number / 2; i <- i+1)
        if (input_number % i == 0)
        prime <- false;
        if (prime == true)
            out<<"The number ", input_number, " is prime";
        else
            out<<"The number ", input_number, " is not prime";
    }
}
```

Separator: {
Reserved: in
Separator: (
String: "Give a number: "
Separator:)
Operator: >>
Identifier: input_number
Separator: ;
Reserved: for
Separator: (
Identifier: i
Operator: <-
Integer: 2
Separator: ;
Identifier: i
Operator: <=
Identifier: input_number
Operator: /
Integer: 2
Separator: ;
Identifier: i
Operator: <-
Identifier: i
Integer: +1
Separator:)
Reserved: if
Separator: (
Identifier: input_number
Operator: %
Identifier: i
Operator: ==
Integer: 0
Separator:)
Identifier: prime
Operator: <-
Boolean: false
Separator: ;
Reserved: if
Separator: (
Identifier: prime
Operator: ==
Boolean: true
Separator:)
Reserved: out
Operator: <<
String: "The number "
Separator: ,
Identifier: input_number
Separator: ,
String: " is prime"
Separator: ;
Reserved: else
Reserved: out
Operator: <<
String: "The number "
Separator: ,
Identifier: input_number
Separator: ,
String: " is not prime"
Separator: ;
Separator: }
Separator: }