Formal Languages and Compiler Design - Lab8

Requirement

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 <math.h>
%}

NONZERO_DIGIT     [1-9]
DIGIT         [0-9]
INTEGER_CT    0|(-?{NONZERO_DIGIT}{DIGIT}*)
CHAR_CT        \'[A-Z0-9]\'|\'\'
STRING_CT    \"[A-Z0-9]*\"
BOOLEAN_CT    true|false
ID         [A-Z_][A-Z0-9_]*
ERROR        [+-]0|\".*|.*\"|'.|.'|0{DIGIT}+|{DIGIT}+[A-Z0-9_]+
%%

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

{CHAR_CT}    printf("Char constant: %s\n", yytext);

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

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

"START"|"ENDPRG"|"id"|"ct"|"INT"|"BOOLEAN"|"CHAR"|"STRING"|"ARRAY"|"BEGIN"|"END"|"READ

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

"+"|" - "|"*"|"/"|"%"|"<"|"<="|">"|">="|"="|"!="|":="|"AND"|"OR"        printf("Operat

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

{ERROR}        printf("Error: %s\n", yytext);

"{"[^}\n]*"}"         /* eat up one-line comments */

[ \t\n]+         /* eat up whitespace */

. printf("Eroare\n");
```

```
%%
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();
}
```

demo

p1.txt

- **input**

```
START
 A:=-15;
 A := -7-10;
 A: INT; B: INT; C: INT; MX1: INT; MX: INT;
 BEGIN
  READ (A);
  READ (B);
  READ (C);

  IF A>B THEN
   MX1:=A;
  ELSE
   MX1 := B;
  IF C > MX1 THEN
   MX := C;
  ELSE
   MX := MX1;
  WRITE (MX);
 END
ENDPRG
```

- **output**

```
Keyword: START
Identifier: A
Operator: :=
Integer constant: -15
Separator: ;
Identifier: A
Operator: :=
Integer constant: -7
Operator:  -
Integer constant: 10
Separator: ;
```

```
Identifier: A
Separator: :
Keyword: INT
Separator: ;
Identifier: B
Separator: :
Keyword: INT
Separator: ;
Identifier: C
Separator: :
Keyword: INT
Separator: ;
Identifier: MX1
Separator: :
Keyword: INT
Separator: ;
Identifier: MX
Separator: :
Keyword: INT
Separator: ;
Keyword: BEGIN
Keyword: READ
Separator: (
Identifier: A
Separator: )
Separator: ;
Keyword: READ
Separator: (
Identifier: B
Separator: )
Separator: ;
Keyword: READ
Separator: (
Identifier: C
Separator: )
Separator: ;
Keyword: IF
Identifier: A
Operator: >
Identifier: B
Keyword: THEN
Identifier: MX1
Operator: :=
Identifier: A
Separator: ;
Keyword: ELSE
Identifier: MX1
Operator: :=
Identifier: B
Separator: ;
Keyword: IF
Identifier: C
Operator: >
Identifier: MX1
Keyword: THEN
```

```
Identifier: MX
Operator: :=
Identifier: C
Separator: ;
Keyword: ELSE
Identifier: MX
Operator: :=
Identifier: MX1
Separator: ;
Keyword: WRITE
Separator: (
Identifier: MX
Separator: )
Separator: ;
Keyword: END
Keyword: ENDPRG
```

p2.txt

- **input**

```
START
 A: INT; B: INT; AUX: INT; R: INT;

 BEGIN
  READ (A);
  READ (B);

  IF A > B THEN
   BEGIN
    AUX := A;
    A := B;
    B := AUX;
   END

  WHILE R != 0 DO
   BEGIN
    R := B % A;
    A := B;
    B := R;
   END

  WRITE (A);
 END
ENDPRG
```

- **output**

```
Keyword: START
Identifier: A
Separator: :
Keyword: INT
Separator: ;
```

```
Identifier: B
Separator: :
Keyword: INT
Separator: ;
Identifier: AUX
Separator: :
Keyword: INT
Separator: ;
Identifier: R
Separator: :
Keyword: INT
Separator: ;
Keyword: BEGIN
Keyword: READ
Separator: (
Identifier: A
Separator: )
Separator: ;
Keyword: READ
Separator: (
Identifier: B
Separator: )
Separator: ;
Keyword: IF
Identifier: A
Operator: >
Identifier: B
Keyword: THEN
Keyword: BEGIN
Identifier: AUX
Operator: :=
Identifier: A
Separator: ;
Identifier: A
Operator: :=
Identifier: B
Separator: ;
Identifier: B
Operator: :=
Identifier: AUX
Separator: ;
Keyword: END
Keyword: WHILE
Identifier: R
Operator: !=
Integer constant: 0
Keyword: DO
Keyword: BEGIN
Identifier: R
Operator: :=
Identifier: B
Operator: %
Identifier: A
Separator: ;
Identifier: A
```

```
Operator: :=
Identifier: B
Separator: ;
Identifier: B
Operator: :=
Identifier: R
Separator: ;
Keyword: END
Keyword: WRITE
Separator: (
Identifier: A
Separator: )
Separator: ;
Keyword: END
Keyword: ENDPRG
```

p3.txt

- **input**

```
START
 N: INT; SUM: INT; I: INT; X: INT;

 BEGIN
  READ (N);
  SUM := 0;
  I := 0;

  WHILE I < N DO
   BEGIN
    READ (X);
    SUM := SUM + X;
    I := I + 1;
   END

  WRITE (SUM);
 END
ENDPRG
```

- **output**

```
Keyword: START
Identifier: N
Separator: :
Keyword: INT
Separator: ;
Identifier: SUM
Separator: :
Keyword: INT
Separator: ;
Identifier: I
Separator: :
Keyword: INT
```

```
Separator: ;
Identifier: X
Separator: :
Keyword: INT
Separator: ;
Keyword: BEGIN
Keyword: READ
Separator: (
Identifier: N
Separator: )
Separator: ;
Identifier: SUM
Operator: :=
Integer constant: 0
Separator: ;
Identifier: I
Operator: :=
Integer constant: 0
Separator: ;
Keyword: WHILE
Identifier: I
Operator: <
Identifier: N
Keyword: DO
Keyword: BEGIN
Keyword: READ
Separator: (
Identifier: X
Separator: )
Separator: ;
Identifier: SUM
Operator: :=
Identifier: SUM
Operator: +
Identifier: X
Separator: ;
Identifier: I
Operator: :=
Identifier: I
Operator: +
Integer constant: 1
Separator: ;
Keyword: END
Keyword: WRITE
Separator: (
Identifier: SUM
Separator: )
Separator: ;
Keyword: END
Keyword: ENDPRG
```