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-z]{1}
STRING_CT    {CHAR_CT}{CHAR_CT}+
ID         [A-Z_][A-Z0-9_]*
%%

{INTEGER_CT}     printf( "An integer: %s (%d)\n", yytext, atoi( yytext ) );

{CHAR_CT}    printf( "A char: %s (%d)\n", yytext, atoi( yytext ) );

{STRING_CT}    printf( "A string: %s (%d)\n", yytext, atoi( yytext ) );

"true"|"false"    printf( "An boolean constant: %s (%d)\n", yytext, atoi( yytext ) );

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

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

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

"("|")"|"["|"]"|"{"|"}"|";"|":"        printf( "A separator: %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

```
A keyword: START
An identifier: A
An operator: :=
An integer: -15 (-15)
A separator: ;
An identifier: A
An operator: :=
An integer: -7 (-7)
An integer: -10 (-10)
A separator: ;
An identifier: A
A separator: :
A keyword: INT
A separator: ;
```

```
An identifier: B
A separator: :
A keyword: INT
A separator: ;
An identifier: C
A separator: :
A keyword: INT
A separator: ;
An identifier: MX1
A separator: :
A keyword: INT
A separator: ;
An identifier: MX
A separator: :
A keyword: INT
A separator: ;
A keyword: BEGIN
A keyword: READ
A separator: (
An identifier: A
A separator: )
A separator: ;
A keyword: READ
A separator: (
An identifier: B
A separator: )
A separator: ;
A keyword: READ
A separator: (
An identifier: C
A separator: )
A separator: ;
A keyword: IF
An identifier: A
An operator: >
An identifier: B
A keyword: THEN
An identifier: MX1
An operator: :=
An identifier: A
A separator: ;
A keyword: ELSE
An identifier: MX1
An operator: :=
An identifier: B
A separator: ;
A keyword: IF
An identifier: C
An operator: >
An identifier: MX1
A keyword: THEN
An identifier: MX
An operator: :=
An identifier: C
A separator: ;
```

```
A keyword: ELSE
An identifier: MX
An operator: :=
An identifier: MX1
A separator: ;
A keyword: WRITE
A separator: (
An identifier: MX
A separator: )
A separator: ;
A keyword: END
A 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**

```
A keyword: START
An identifier: A
A separator: :
A keyword: INT
A separator: ;
An identifier: B
A separator: :
A keyword: INT
A separator: ;
```

```
An identifier: AUX
A separator: :
A keyword: INT
A separator: ;
An identifier: R
A separator: :
A keyword: INT
A separator: ;
A keyword: BEGIN
A keyword: READ
A separator: (
An identifier: A
A separator: )
A separator: ;
A keyword: READ
A separator: (
An identifier: B
A separator: )
A separator: ;
A keyword: IF
An identifier: A
An operator: >
An identifier: B
A keyword: THEN
A keyword: BEGIN
An identifier: AUX
An operator: :=
An identifier: A
A separator: ;
An identifier: A
An operator: :=
An identifier: B
A separator: ;
An identifier: B
An operator: :=
An identifier: AUX
A separator: ;
A keyword: END
A keyword: WHILE
An identifier: R
An operator: !=
An integer: 0 (0)
A keyword: DO
A keyword: BEGIN
An identifier: R
An operator: :=
An identifier: B
An operator: %
An identifier: A
A separator: ;
An identifier: A
An operator: :=
An identifier: B
A separator: ;
An identifier: B
```

```
An operator: :=
An identifier: R
A separator: ;
A keyword: END
A keyword: WRITE
A separator: (
An identifier: A
A separator: )
A separator: ;
A keyword: END
A 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**

```
A keyword: START
An identifier: N
A separator: :
A keyword: INT
A separator: ;
An identifier: SUM
A separator: :
A keyword: INT
A separator: ;
An identifier: I
A separator: :
A keyword: INT
A separator: ;
An identifier: X
A separator: :
A keyword: INT
```

```
A separator: ;
A keyword: BEGIN
A keyword: READ
A separator: (
An identifier: N
A separator: )
A separator: ;
An identifier: SUM
An operator: :=
An integer: 0 (0)
A separator: ;
An identifier: I
An operator: :=
An integer: 0 (0)
A separator: ;
A keyword: WHILE
An identifier: I
An operator: <
An identifier: N
A keyword: DO
A keyword: BEGIN
A keyword: READ
A separator: (
An identifier: X
A separator: )
A separator: ;
An identifier: SUM
An operator: :=
An identifier: SUM
An operator: +
An identifier: X
A separator: ;
An identifier: I
An operator: :=
An identifier: I
An operator: +
An integer: 1 (1)
A separator: ;
A keyword: END
A keyword: WRITE
A separator: (
An identifier: SUM
A separator: )
A separator: ;
A keyword: END
A keyword: ENDPRG
```