

Quiz #11

Ilya Starikov

June 30, 2025

1 Source Code

```
1      /*      example.l
2
3          Illya Starikov
4      */
5
6      %{
7      /*
8      Definitions of constants, variables, & function prototypes
9      go here
10     */
11
12     #define T_IDENT      1
13     #define T_INTCONST   2
14     #define T_UNKNOWN    3
15     #define T_FOO        4
16     #define T_KEYWORD    5
17     #define T_OPERATOR   6
18     #define T_NUMBER     7
19     #define T_ALPHA      8
20
21     int numLines = 1;
22
23     void printTokenInfo(const char* tokenType, const char*
24     lexeme);
25
26     /* Defintions of regular expressions go here */
27
```

```

28 WSPACE      [ \t\v\r]+
29 NEWLINE     \n
30
31 KEYWORD      (var|if|then|else|while|true|false)
32 OPERATOR     (;|\{|\}|\;|\[|\]|=\|\<|\)|\+|\>|\<|!=|>=|<=|==)
33 NUMBER       -?{INTCONST}
34 ALPHA        {LETTER}
35
36 DIGIT        [0-9]
37 LETTER       [a-zA-Z]
38
39 IDENT        {LETTER}({LETTER}|{DIGIT})*
40 INTCONST     {DIGIT}+
41
42 %%
43
44 "foo"        {
45               printTokenInfo("FOO", yytext);
46               return T_FOO;
47             }
48 {KEYWORD}    {
49               printTokenInfo("KEYWORD", yytext);
50               return T_KEYWORD;
51             }
52 {OPERATOR}   {
53               printTokenInfo("OPERATOR", yytext);
54               return T_OPERATOR;
55             }
56 {ALPHA} {
57               printTokenInfo("ALPHA", yytext);
58               return T_ALPHA;
59             }
60 {INTCONST}   {
61               printTokenInfo("INTCONST", yytext);
62               return T_INTCONST;
63             }
64 {INTCONST}   {
65               printTokenInfo("INTCONST", yytext);
66               return T_INTCONST;
67             }
68 {IDENT}      {
69               printTokenInfo("IDENT", yytext);
70               return T_IDENT;
71             }
72 {NEWLINE}    {

```

```

73             numLines++;
74         }
75     {WSPACE}    { }
76     .          {
77                 printTokenInfo("UNKNOWN", yytext);
78                 return T_UNKNOWN;
79     }
80     %%
81
82     // User-written code goes here
83
84     void printTokenInfo(const char* tokenType, const char*
lexeme)
85     {
86         printf("TOKEN: %s LEXEME: %s\n", tokenType, lexeme);
87     }
88
89     // You should supply a yywrap function.
90     // Having it return 1 means only 1 input file will be
scanned.
91     int yywrap() { return(1); }
92
93     int main()
94     {
95         while ( yylex() ) ; // Process tokens until 0 returned
96
97         printf("Processed %d lines\n", numLines);
98         return(0);
99     }

```

2 Input

```
var
x; y; z;
A[4];
B[2][3];
\$\% \#
{
x = 12;
B[1][2] = x;
A[x] = 5;
while (x > 100) x = x + 1;
if (A[2] <= A[B[x][y]]) then x = y + 1 else z = 10;
\}
```

3 Output

```
TOKEN: KEYWORD  LEXEME: var
TOKEN: ALPHA    LEXEME: x
TOKEN: OPERATOR LEXEME: ;
TOKEN: ALPHA    LEXEME: y
TOKEN: OPERATOR LEXEME: ;
TOKEN: ALPHA    LEXEME: z
TOKEN: OPERATOR LEXEME: ;
TOKEN: ALPHA    LEXEME: A
TOKEN: OPERATOR LEXEME: [
TOKEN: INTCONST  LEXEME: 4
TOKEN: OPERATOR LEXEME: ]
TOKEN: OPERATOR LEXEME: ;
TOKEN: ALPHA    LEXEME: B
TOKEN: OPERATOR LEXEME: [
TOKEN: INTCONST  LEXEME: 2
TOKEN: OPERATOR LEXEME: ]
TOKEN: OPERATOR LEXEME: [
TOKEN: INTCONST  LEXEME: 3
TOKEN: OPERATOR LEXEME: ]
TOKEN: OPERATOR LEXEME: ;
TOKEN: UNKNOWN   LEXEME: \$
TOKEN: UNKNOWN   LEXEME: \%
TOKEN: UNKNOWN   LEXEME: \#
TOKEN: OPERATOR LEXEME: {
TOKEN: ALPHA    LEXEME: x
TOKEN: OPERATOR LEXEME: =
TOKEN: INTCONST  LEXEME: 12
TOKEN: OPERATOR LEXEME: ;
TOKEN: ALPHA    LEXEME: B
```

```
TOKEN: OPERATOR  LEXEME: [  
TOKEN: INTCONST  LEXEME: 1  
TOKEN: OPERATOR  LEXEME: ]  
TOKEN: OPERATOR  LEXEME: [  
TOKEN: INTCONST  LEXEME: 2  
TOKEN: OPERATOR  LEXEME: ]  
TOKEN: OPERATOR  LEXEME: =  
TOKEN: ALPHA     LEXEME: x  
TOKEN: OPERATOR  LEXEME: ;  
TOKEN: ALPHA     LEXEME: A  
TOKEN: OPERATOR  LEXEME: [  
TOKEN: ALPHA     LEXEME: x  
TOKEN: OPERATOR  LEXEME: ]  
TOKEN: OPERATOR  LEXEME: =  
TOKEN: INTCONST  LEXEME: 5  
TOKEN: OPERATOR  LEXEME: ;  
TOKEN: KEYWORD   LEXEME: while  
TOKEN: OPERATOR  LEXEME: (  
TOKEN: ALPHA     LEXEME: x  
TOKEN: OPERATOR  LEXEME: >  
TOKEN: INTCONST  LEXEME: 100  
TOKEN: OPERATOR  LEXEME: )  
TOKEN: ALPHA     LEXEME: x  
TOKEN: OPERATOR  LEXEME: =  
TOKEN: ALPHA     LEXEME: x  
TOKEN: OPERATOR  LEXEME: +  
TOKEN: INTCONST  LEXEME: 1  
TOKEN: OPERATOR  LEXEME: ;
```

```
TOKEN: KEYWORD  LEXEME: if
TOKEN: OPERATOR  LEXEME: (
TOKEN: ALPHA    LEXEME: A
TOKEN: OPERATOR  LEXEME: [
TOKEN: INTCONST  LEXEME: 2
TOKEN: OPERATOR  LEXEME: ]
TOKEN: OPERATOR  LEXEME: <=
TOKEN: ALPHA    LEXEME: A
TOKEN: OPERATOR  LEXEME: [
TOKEN: ALPHA    LEXEME: B
TOKEN: OPERATOR  LEXEME: [
TOKEN: ALPHA    LEXEME: x
TOKEN: OPERATOR  LEXEME: ]
TOKEN: OPERATOR  LEXEME: [
TOKEN: ALPHA    LEXEME: y
TOKEN: OPERATOR  LEXEME: ]
TOKEN: OPERATOR  LEXEME: ]
TOKEN: OPERATOR  LEXEME: )
TOKEN: KEYWORD  LEXEME: then
TOKEN: ALPHA    LEXEME: x
TOKEN: OPERATOR  LEXEME: =
TOKEN: ALPHA    LEXEME: y
TOKEN: OPERATOR  LEXEME: +
TOKEN: INTCONST  LEXEME: 1
TOKEN: KEYWORD  LEXEME: else
TOKEN: ALPHA    LEXEME: z
TOKEN: OPERATOR  LEXEME: =
TOKEN: INTCONST  LEXEME: 10
TOKEN: OPERATOR  LEXEME: ;
TOKEN: OPERATOR  LEXEME: }
```