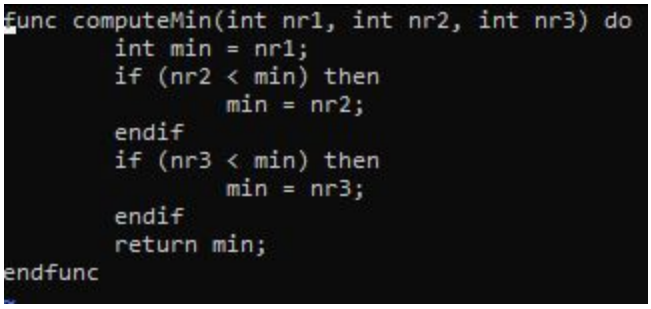
Lab 8 (lex)
Ternovan Darius-Daniel, 937/1
https://github.com/ternovandarius/FLCD

The lex is implemented as seen below:

```
%{
#include <stdio.h>
%}

%option noyywrap
%option caseless

letter [A-Za-z]
digit [0-9]
identifier {letter}[_a-zA-Z0-9]*

%%

func {printf("Reserved word: FUNCTION\n");}
int {printf("Reserved word: INTEGER\n");}
do {printf("Reserved word: DO\n");}
if {printf("Reserved word: IF\n");}
then {printf("Reserved word: THEN\n");}
endif {printf("Reserved word: ENDIF\n");}
return {printf("Reserved word: RETURN\n");}
endfunc {printf("Reserved word: END FUNCTION\n");}
for {printf("Reserved word: FOR\n");}
endfor {printf("Reserved word: END FOR\n");}
in {printf("Reserved word: IN\n");}


"("  {printf("Separator: (\n");}
")"  {printf("Separator: )\n");}
","  {printf("Separator: ,\n");}
";"  {printf("Separator: ;\n");}
"[" {printf("Separator: [\n");}
"]" {printf("Separator: ]\n");}

"="  {printf("Operator: =\n");}
"<"  {printf("Operator: <\n");}
">"  {printf("Operator: >\n");}
">=" {printf("Operator: >=\n");}
"<=" {printf("Operator: <=\n");}
"==" {printf("Operator: ==\n");}
"!=" {printf("Operator: !=\n");}
"+" {printf("Operator: +\n");}
"-"  {printf("Operator: -\n");}
"/"  {printf("Operator: /\n");}
"*" {printf("Operator: *\n");}

{identifier} {printf("IDENTIFIER\n");}

%%}
```

We input p1.in:



```
func computeMin(int nr1, int nr2, int nr3) do
        int min = nr1;
        if (nr2 < min) then
                min = nr2;
        endif
        if (nr3 < min) then
                min = nr3;
        endif
        return min;
endfunc
```

The result of inputting p1.in into the lex is:

```
Reserved word: FUNCTION
 IDENTIFIER
Separator: (
Reserved word: INTEGER
 IDENTIFIER
Separator: ,
 Reserved word: INTEGER
 IDENTIFIER
Separator: ,
 Reserved word: INTEGER
 IDENTIFIER
Separator: )
 Reserved word: DO

        Reserved word: INTEGER
 IDENTIFIER
 Operator: =
 IDENTIFIER
Separator: ;

        Reserved word: IF
 Separator: (
IDENTIFIER
 Operator: <
 IDENTIFIER
Separator: )
 Reserved word: THEN

                IDENTIFIER
 Operator: =
 IDENTIFIER
Separator: ;

        Reserved word: ENDIF

        Reserved word: IF
 Separator: (
IDENTIFIER
 Operator: <
 IDENTIFIER
Separator: )
 Reserved word: THEN

                IDENTIFIER
 Operator: =
 IDENTIFIER
Separator: ;

        Reserved word: ENDIF

        Reserved word: RETURN
 IDENTIFIER
Separator: ;

Reserved word: END FUNCTION
```