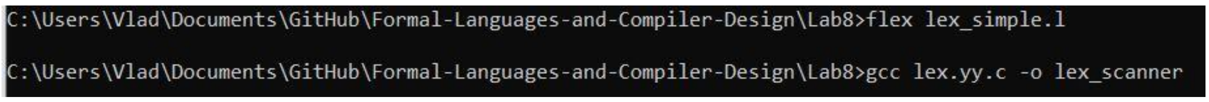
Lex Scanner

lex_simple.l file:

```
%{
	#include<stdio.h>
	#include <string.h>
	int currentLine = 1;
%}

%option noyywrap

LETTER [a-ZA-Z]
DIGIT [0-9]
NON_ZERO_DIGIT [1-9]
INTEGER 0|[+|-]?{NON_ZERO_DIGIT}{DIGIT}*
CHAR [\'][a-zA-Z0-9_][\']
STRING [\"][a-zA-Z0-9_]*[\"]
IDENTIFIER [a-zA-Z][a-zA-Z0-9_]*

%%

"if"		{printf("Reserved word: %s\n", yytext);}
"elif"		{printf("Reserved word: %s\n", yytext);}
"else"		{printf("Reserved word: %s\n", yytext);}
"while"		{printf("Reserved word: %s\n", yytext);}
"for"		{printf("Reserved word: %s\n", yytext);}
"and"		{printf("Reserved word: %s\n", yytext);}
"or"		{printf("Reserved word: %s\n", yytext);}
"read"		{printf("Reserved word: %s\n", yytext);}
"show"		{printf("Reserved word: %s\n", yytext);}


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

"+"		{printf("Operator: %s\n", yytext);}
"-"		{printf("Operator: %s\n", yytext);}
"*"		{printf("Operator: %s\n", yytext);}
"/"		{printf("Operator: %s\n", yytext);}
"//"		{printf("Operator: %s\n", yytext);}
"%"		{printf("Operator: %s\n", yytext);}
"="		{printf("Operator: %s\n", yytext);}
"<-"		{printf("Operator: %s\n", yytext);}
"<>"		{printf("Operator: %s\n", yytext);}
"<"		{printf("Operator: %s\n", yytext);}
```

```
">"             {printf("Operator: %s\n", yytext);}
"<="            {printf("Operator: %s\n", yytext);}
">="            {printf("Operator: %s\n", yytext);}

{INTEGER}       {printf("Number: %s\n", yytext);}
{STRING}        {printf("String: %s\n", yytext);}
{CHAR}                  {printf("Character: %s\n", yytext);}
{IDENTIFIER}            {printf("Identifier: %s\n", yytext);}

[ \t]+    {}
[\n]+ {currentLine++;}

[0-9_][a-zA-Z0-9_]*             {printf("Illegal identifier at line %d\n", currentLine); return -1;}
[+|-]0          {printf("Illegal numeric constant at line %d\n", currentLine); return -1;}
.               {printf("Illegal symbol at line %d\n", currentLine); return -1;}

%%
void main(argc, argv)
int argc;
char** argv;
{
if (argc > 1)
{
    FILE *file;
    file = fopen(argv[1], "r");
    if (!file)
    {
        fprintf(stderr, "Could not open %s\n", argv[1]);
        exit(1);
    }
    yyin = file;
}

yylex();
}
```

Commands used:



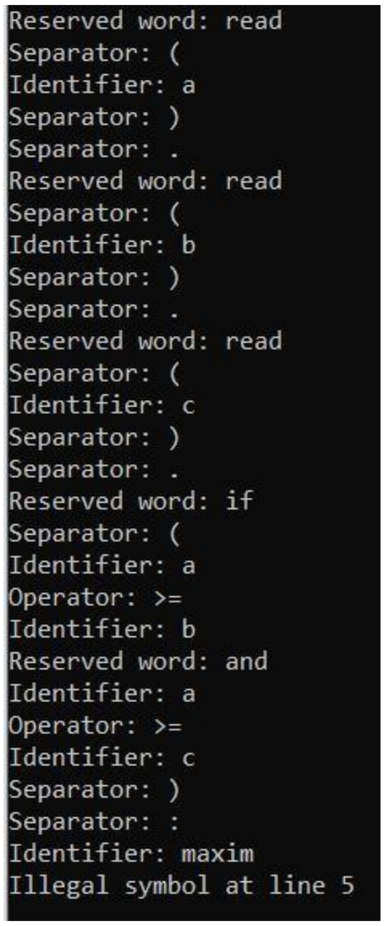
```
C:\Users\Vlad\Documents\GitHub\Formal-Languages-and-Compiler-Design\Lab8>flex lex_simple.l

C:\Users\Vlad\Documents\GitHub\Formal-Languages-and-Compiler-Design\Lab8>gcc lex.yy.c -o lex_scanner
```

A program with errors:

```
read(a).
read(b).
read(c).
if (a >= b and a >= c):
    maxim ^ a.
elif (b >= a and b >= c):
    maxim <- b.
```

```
else:
    maxim <- c.
show("The_" + "biggest_" + "number_" + "is_").
show(maxim).
```

The result:



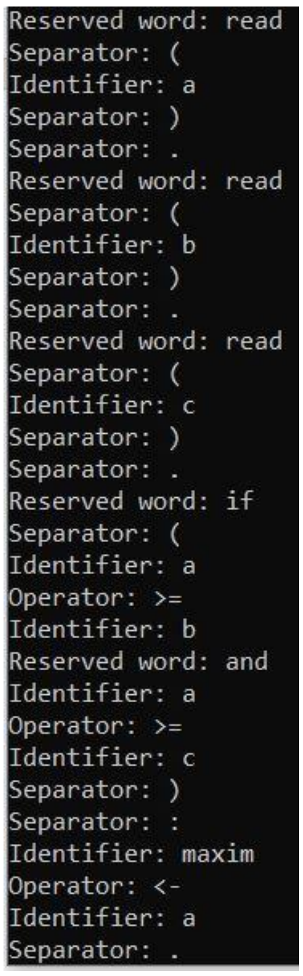
```
Reserved word: read
Separator: (
Identifier: a
Separator: )
Separator: .
Reserved word: read
Separator: (
Identifier: b
Separator: )
Separator: .
Reserved word: read
Separator: (
Identifier: c
Separator: )
Separator: .
Reserved word: if
Separator: (
Identifier: a
Operator: >=
Identifier: b
Reserved word: and
Identifier: a
Operator: >=
Identifier: c
Separator: )
Separator: :
Identifier: maxim
Illegal symbol at line 5
```

A correct program:

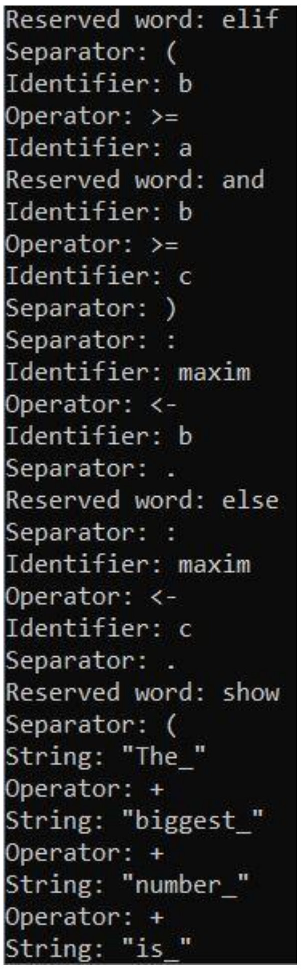
```
read(a).
read(b).
read(c).
if (a >= b and a >= c):
    maxim <- a.
elif (b >= a and b >= c):
    maxim <- b.
else:
    maxim <- c.
show("The_" + "biggest_" + "number_" + "is_").
```

show(maxim).

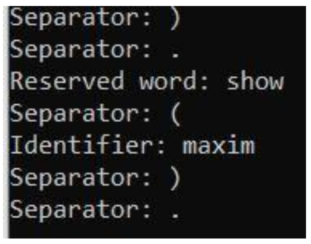
The result:



```
Reserved word: read
Separator: (
Identifier: a
Separator: )
Separator: .
Reserved word: read
Separator: (
Identifier: b
Separator: )
Separator: .
Reserved word: read
Separator: (
Identifier: c
Separator: )
Separator: .
Reserved word: if
Separator: (
Identifier: a
Operator: >=
Identifier: b
Reserved word: and
Identifier: a
Operator: >=
Identifier: c
Separator: )
Separator: :
Identifier: maxim
Operator: <-
Identifier: a
Separator: .
```



```
Reserved word: elif
Separator: (
Identifier: b
Operator: >=
Identifier: a
Reserved word: and
Identifier: b
Operator: >=
Identifier: c
Separator: )
Separator: :
Identifier: maxim
Operator: <-
Identifier: b
Separator: .
Reserved word: else
Separator: :
Identifier: maxim
Operator: <-
Identifier: c
Separator: .
Reserved word: show
Separator: (
String: "The_"
Operator: +
String: "biggest_"
Operator: +
String: "number_"
Operator: +
String: "is_"
```



```
Separator: )
Separator: .
Reserved word: show
Separator: (
Identifier: maxim
Separator: )
Separator: .
```