Documentation

Link to GitHub : https://github.com/diana-dr/Formal-Languages-and-Compiler-Design/tree/master/Lab%208

spec.lxi

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

int currentLine = 1;
%}

%option noyywrap
%option caseless

DIGIT           [0-9]
NZ_DIGIT    [1-9]
ZERO        [0]
NUMBER      {NZ_DIGIT}{DIGIT}*
SIGN        [+]|[-]
INTEGER         {ZERO}|{NUMBER}|{SIGN}{NUMBER}
SIGNER_INTEGER {SIGN}{NUMBER}
SPECIAL_CHAR
"_"|"."|","|";"|":"|"?"|"!"|"@"|"/"|"("|")"|"-"|"+"|"="|"{"|"}"|"*"|"["|"]"|"$"|
"%"|"^"|" "
CHAR        {DIGIT}|{SPECIAL_CHAR}|[a-zA-Z]
CHARACTER       "'"{CHAR}"'"
STRING          [\"]{CHAR}*[\"]
CONSTANT                {STRING}|{INTEGER}|{CHARACTER}
IDENTIFIER              [a-zA-Z_][a-zA-Z0-9_]*

%%

and {printf("Reserved word - %s\n", yytext);}
or {printf("Reserved word - %s\n", yytext);}
not {printf("Reserved word - %s\n", yytext);}
if {printf("Reserved word - %s\n", yytext);}
do {printf("Reserved word - %s\n", yytext);}
else {printf("Reserved word - %s\n", yytext);}
elif {printf("Reserved word - %s\n", yytext);}
while {printf("Reserved word - %s\n", yytext);}
for {printf("Reserved word - %s\n", yytext);}
read {printf("Reserved word - %s\n", yytext);}
write {printf("Reserved word - %s\n", yytext);}
int {printf("Reserved word - %s\n", yytext);}
string {printf("Reserved word - %s\n", yytext);}
char {printf("Reserved word - %s\n", yytext);}
function {printf("Reserved word - %s\n", yytext);}
bool {printf("Reserved word - %s\n", yytext);}
return {printf("Reserved word - %s\n", yytext);}


{CONSTANT} {printf("Constant - %s\n", yytext);}
{IDENTIFIER} {printf("Identifier - %s\n", yytext);}

; {printf("Separator - %s\n", yytext);}
\, {printf("Separator - %s\n", yytext);}
\t {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);}

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

(\+0)|(\-0) printf("! Lexical error: %s\n", yytext);
{INTEGER}{IDENTIFIER} printf("! Lexical error: %s\n", yytext);
0{INTEGER} printf("! Lexical error: %s\n", yytext);

%%

int 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]);
        }
        yyin = file;
}
yylex();
}
```

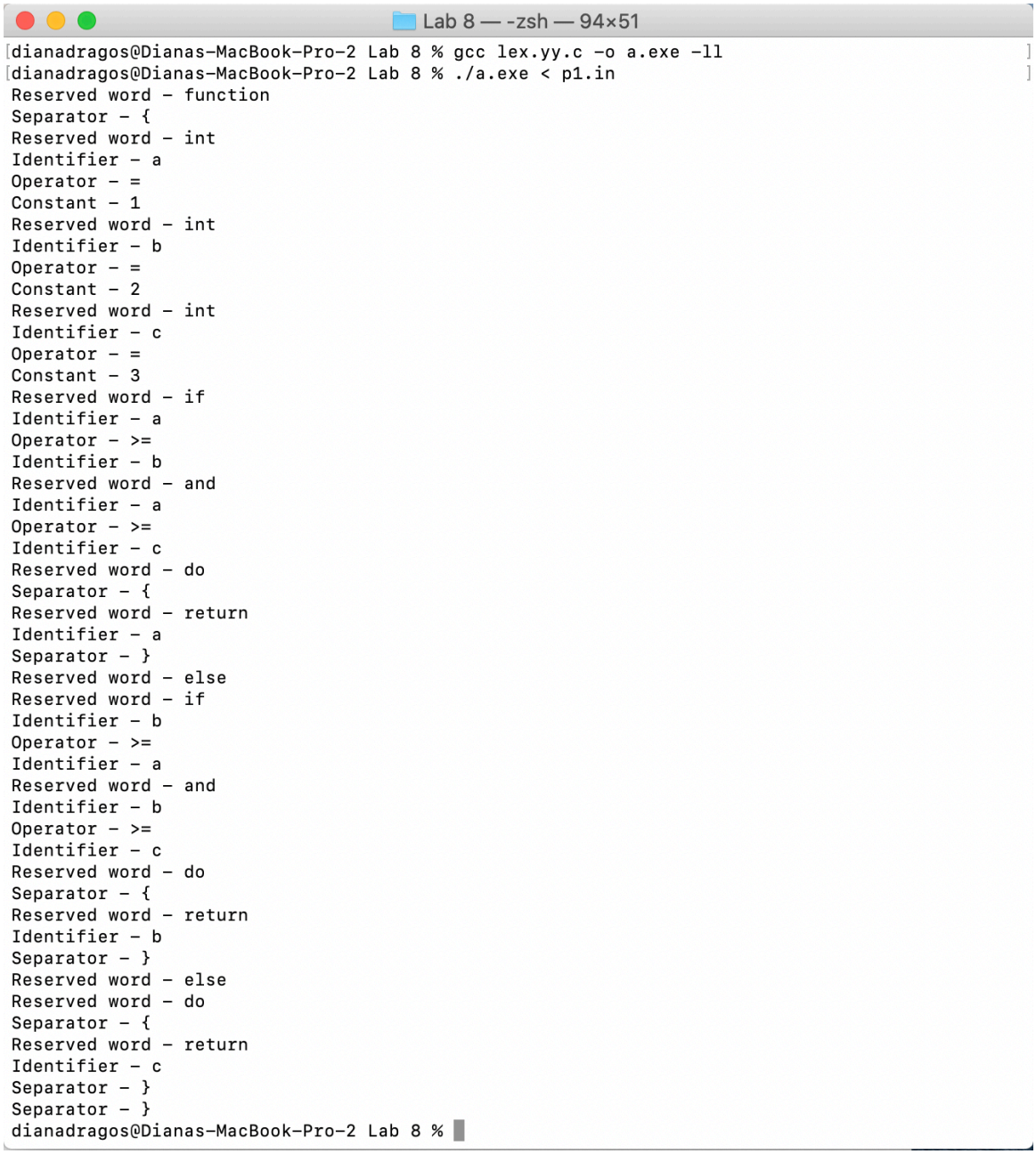
p1.in

```
function
{
    int a = 1
    int b = 2
    int c = 3

    if a >= b and a >= c do {
        return a
    }

    else if b >= a and b >= c do {
        return b
    }

    else do {
        return c
    }
}
```

p1err.in

```
function
{
    int a = +0
    return a
}
```

Example - p1.in



Example - p1err.in