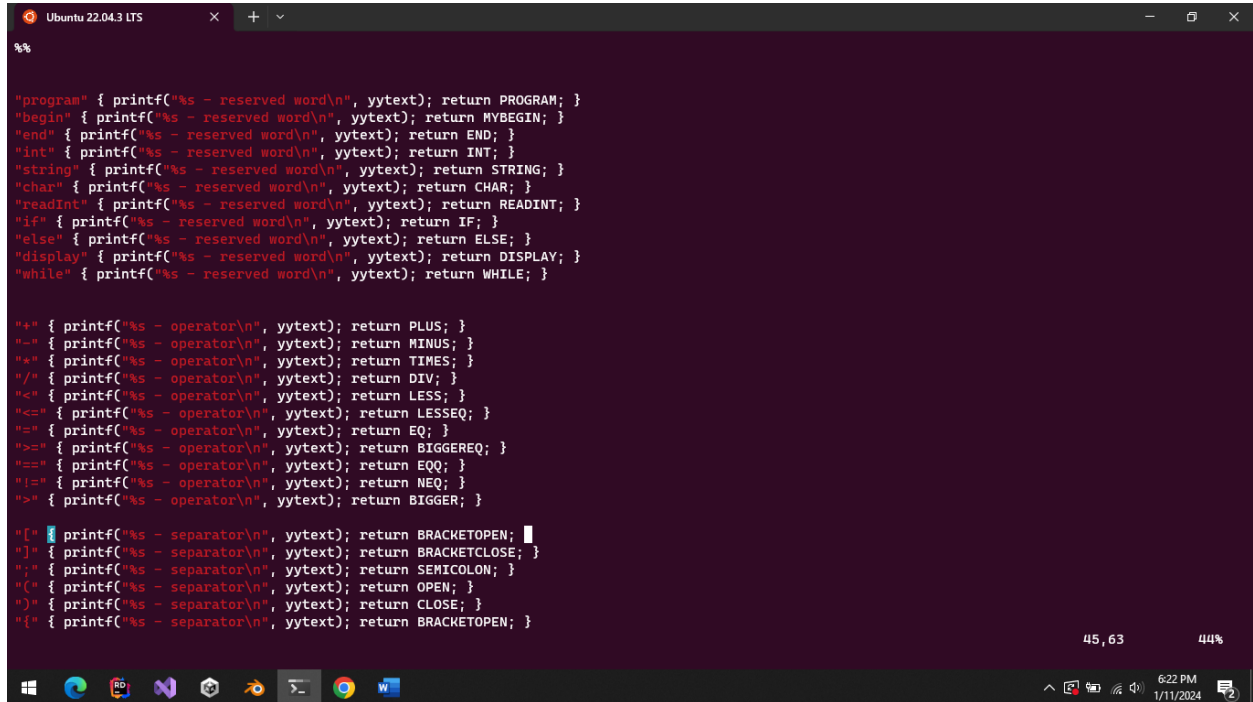


# Documentation lab10

Source code: <https://github.com/davidalexandru1370/LabLFTC/tree/lab9/lab10>

Changes from previous laboratory at scanner file:

-scanner returns the matched token



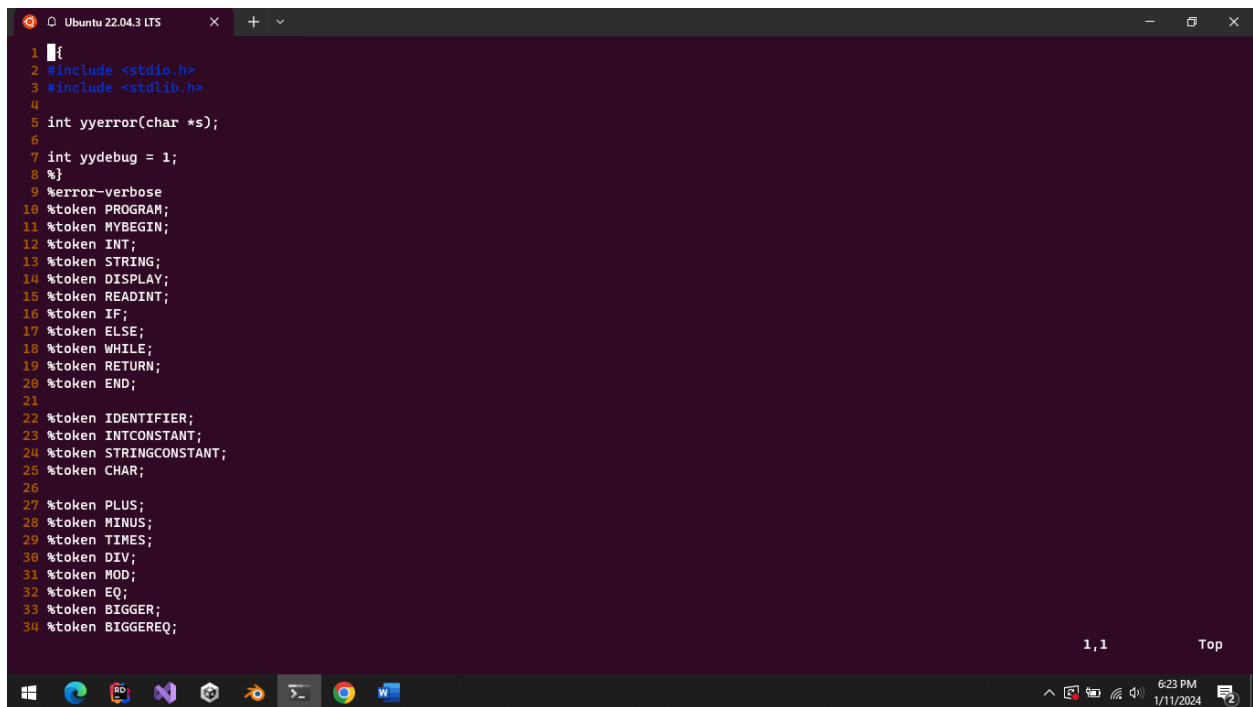
```
%%

"program" { printf("%s - reserved word\n", yytext); return PROGRAM; }
"begin" { printf("%s - reserved word\n", yytext); return MYBEGIN; }
"end" { printf("%s - reserved word\n", yytext); return END; }
"int" { printf("%s - reserved word\n", yytext); return INT; }
"string" { printf("%s - reserved word\n", yytext); return STRING; }
"char" { printf("%s - reserved word\n", yytext); return CHAR; }
"readInt" { printf("%s - reserved word\n", yytext); return READINT; }
"if" { printf("%s - reserved word\n", yytext); return IF; }
"else" { printf("%s - reserved word\n", yytext); return ELSE; }
"display" { printf("%s - reserved word\n", yytext); return DISPLAY; }
"while" { printf("%s - reserved word\n", yytext); return WHILE; }

"+" { printf("%s - operator\n", yytext); return PLUS; }
"-" { printf("%s - operator\n", yytext); return MINUS; }
"*" { printf("%s - operator\n", yytext); return TIMES; }
"/" { printf("%s - operator\n", yytext); return DIV; }
"<" { printf("%s - operator\n", yytext); return LESS; }
"<=" { printf("%s - operator\n", yytext); return LESSEQ; }
"==" { printf("%s - operator\n", yytext); return EQ; }
">=" { printf("%s - operator\n", yytext); return BIGGEREQ; }
"==" { printf("%s - operator\n", yytext); return EQQ; }
"!=" { printf("%s - operator\n", yytext); return NEQ; }
">" { printf("%s - operator\n", yytext); return BIGGER; }

"[" { printf("%s - separator\n", yytext); return BRACKETOPEN; }
"]" { printf("%s - separator\n", yytext); return BRACKETCLOSE; }
";" { printf("%s - separator\n", yytext); return SEMICOLON; }
"(" { printf("%s - separator\n", yytext); return OPEN; }
")" { printf("%s - separator\n", yytext); return CLOSE; }
"{" { printf("%s - separator\n", yytext); return BRACKETOPEN; }
```

Those tokens have been taken from yacc file

A screenshot of a terminal window titled 'Ubuntu 22.04.3 LTS'. The terminal displays a yacc grammar file with 34 lines of code. The code includes standard headers, function declarations, and a list of tokens. The tokens are: PROGRAM, MYBEGIN, INT, STRING, DISPLAY, READINT, IF, ELSE, WHILE, RETURN, END, IDENTIFIER, INTCONSTANT, STRINGCONSTANT, CHAR, PLUS, MINUS, TIMES, DIV, MOD, EQ, BIGGER, and BIGGEREQ. The terminal has a dark purple background. The bottom of the window shows a taskbar with various application icons and a system tray with the date and time '6:23 PM 1/11/2024'.

```
1 {}  
2 #include <stdio.h>  
3 #include <stdlib.h>  
4  
5 int yyerror(char *s);  
6  
7 int yydebug = 1;  
8 %}  
9 %error-verbose  
10 %token PROGRAM;  
11 %token MYBEGIN;  
12 %token INT;  
13 %token STRING;  
14 %token DISPLAY;  
15 %token READINT;  
16 %token IF;  
17 %token ELSE;  
18 %token WHILE;  
19 %token RETURN;  
20 %token END;  
21  
22 %token IDENTIFIER;  
23 %token INTCONSTANT;  
24 %token STRINGCONSTANT;  
25 %token CHAR;  
26  
27 %token PLUS;  
28 %token MINUS;  
29 %token TIMES;  
30 %token DIV;  
31 %token MOD;  
32 %token EQ;  
33 %token BIGGER;  
34 %token BIGGEREQ;
```

The grammar contains all the statements and expression to parse all the input files.

To run this yacc file we have to do the following steps:

Execute: `yacc -d parser.yacc` to get the `y.tab.c` and `y.tab.h` files

Execute: `flex -l scanner.lxi` to get the `lex.yy.c` file

Merge those together using: `gcc -o a.exe y.tab.c lex.yy.c`

This will output an executable file at `a.exe`

Running `a.exe` will print the following output:

```
Ubuntu 22.04.3 LTS
david@DESKTOP-CAJCL3P /mnt/c/Users/david/Desktop/folders/LabLFTC/lab10 (lab9)$ ./a.exe p1.txt
program - reserved word
{ - separator
begin - reserved word
int - reserved word
Type -> int
Identifier: a
; - separator
DeclarationStatement -> TYPE IDENTIFIER
Statement -> DeclarationStatement
int - reserved word
Type -> int
Identifier: b
; - separator
DeclarationStatement -> TYPE IDENTIFIER
Statement -> DeclarationStatement
int - reserved word
Type -> int
Identifier: c
; - separator
DeclarationStatement -> TYPE IDENTIFIER
Statement -> DeclarationStatement
readInt - reserved word
( - separator
Identifier: a
) - separator
ReadStatement -> readInt(IDENTIFIER)Statement -> ReadStatement
; - separator
readInt - reserved word
( - separator
Identifier: b
) - separator
ReadStatement -> readInt(IDENTIFIER)Statement -> ReadStatement
; - separator
readInt - reserved word
```