Formal Languages and Compiler Design

Lab 8 – Documentation

Ariadna Vilasó Escarp

Link to the repository:

https://github.com/UO270119/Formal-Languages-and-Compiler-Design

Problem Statement:

Use lex. You may use any version (LEX or FLEX)

1) Write a LEX specification containing the regular expressions corresponding to your language specification - see lab 1

2) Use Lex in order to obtain a scanner. Test for the same input as in lab 1 (p1, p2).

Deliverables: pdf file containing lang.lxi (lex specification file) + demo

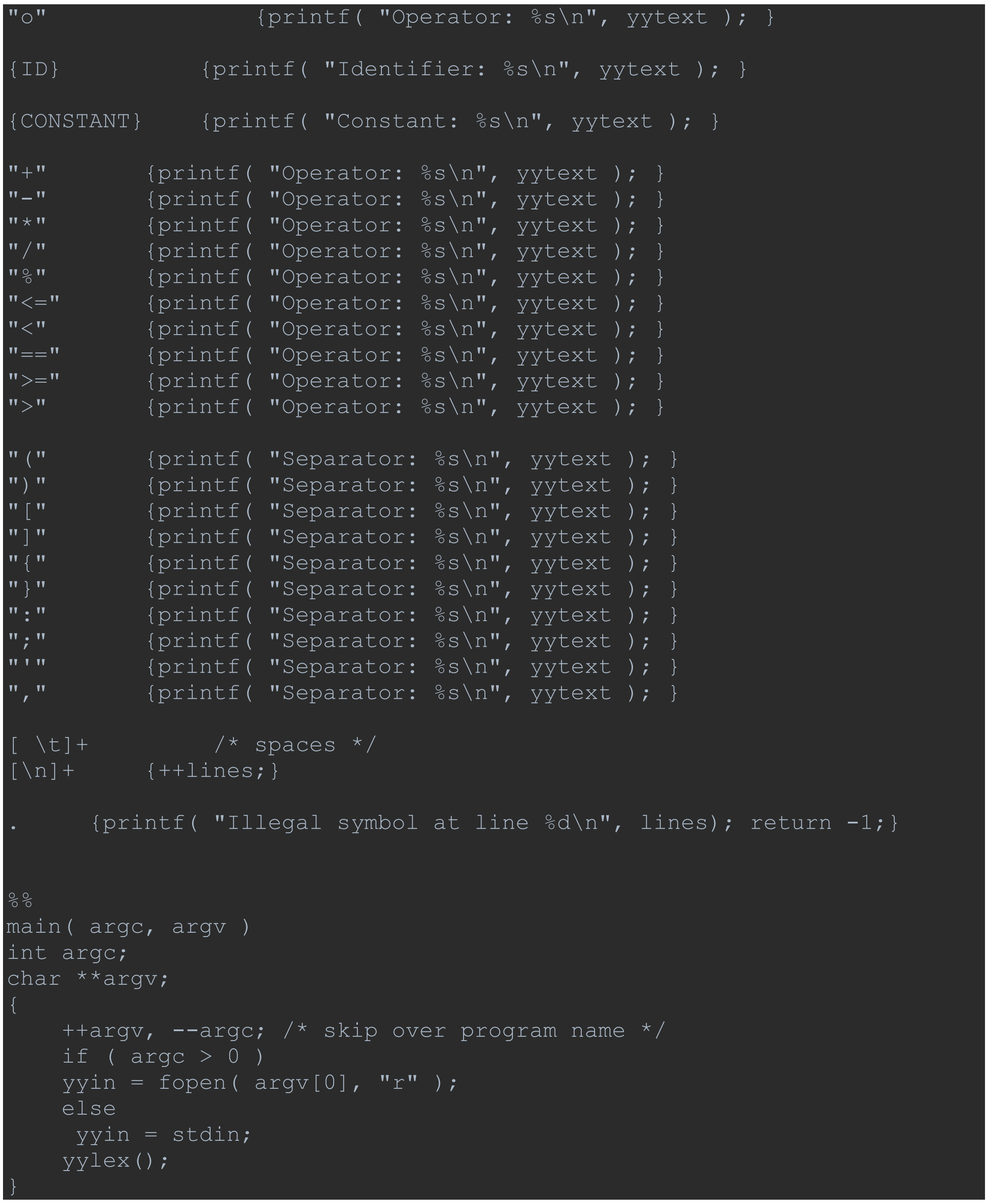
To solve this, I created the *lang.lxi* file following the example given in Moodle. I added my own operators, separators and reserved words according to what I had in my *token.in* file developed in previous sessions.

```
%{
#include <math.h>
int lines = 0;
%}
%option noyywrap

DIGIT     [0-9]
NUMBER     [1-9][O-9]*
STRING         \"[a-zA-Z]*\"
CONSTANT   {STRING}|{DIGIT}
ID        [a-zA-Z][a-zA-Z0-9]*

%%

"regular"     {printf( "Reserved word: %s\n", yytext ); }
"read(n)"     {printf( "Reserved word: %s\n", yytext ); }
"check"           {printf( "Reserved word: %s\n", yytext ); }
"entonces"    {printf( "Reserved word: %s\n", yytext ); }
"checkif"     {printf( "Reserved word: %s\n", yytext ); }
"not"         {printf( "Reserved word: %s\n", yytext ); }
"loop"        {printf( "Reserved word: %s\n", yytext ); }
"var"         {printf( "Reserved word: %s\n", yytext ); }
"rango(a,b)"   {printf( "Reserved word: %s\n", yytext ); }
"show"        {printf( "Reserved word: %s\n", yytext ); }
"end"         {printf( "Reserved word: %s\n", yytext );}
"matriz"      {printf( "Reserved word: %s\n", yytext ); }
"haz"         {printf( "Reserved word: %s\n", yytext );}
"mientras"    {printf( "Reserved word: %s\n", yytext ); }
"caracter"    {printf( "Reserved word: %s\n", yytext ); }
"constante"       {printf( "Reserved word: %s\n", yytext ); }
"programa"    {printf( "Reserved word: %s\n", yytext ); }
"empieza"     {printf( "Reserved word: %s\n", yytext ); }
"acaba"           {printf( "Reserved word: %s\n", yytext ); }
"de"          {printf( "Reserved word: %s\n", yytext ); }
"y"               {printf( "Operator: %s\n", yytext ); }
```



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

{ID}            {printf( "Identifier: %s\n", yytext ); }

{CONSTANT}      {printf( "Constant: %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( "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( "Separator: %s\n", yytext ); }
","         {printf( "Separator: %s\n", yytext ); }

[ \t]+          /* spaces */
[\n]+       {++lines;}

.     {printf( "Illegal symbol at line %d\n", lines); return -1;}


%%
main( argc, argv )
int argc;
char **argv;
{
    ++argv, --argc; /* skip over program name */
    if ( argc > 0 )
    yyin = fopen( argv[0], "r" );
    else
     yyin = stdin;
    yylex();
}
```

Then, I compiled the program and executed it using p1.txt with the following commands:

flex lang.lxi

gcc lex.yy.c -o result

result.exe p1.txt

And I obtained the following result:

```
Reserved word: empieza
Identifier: a
Identifier: equals
Reserved word: regular
Identifier: read
Separator: (
Identifier: a
Separator: )
Identifier: b
Identifier: equals
Reserved word: regular
Identifier: read
Separator: (
Identifier: b
Separator: )
Identifier: c
Identifier: equals
Reserved word: regular
Identifier: read
Separator: (
Identifier: c
Separator: )
Reserved word: check
Identifier: a
Identifier: greater
Identifier: b
Operator: y
Identifier: a
Identifier: greater
Identifier: c
```

```
Identifier: greatest

Identifier: equals

Identifier: a

Reserved word: checkif

Identifier: b

Identifier: greater

Identifier: a

Operator: y

Identifier: b

Identifier: greater

Identifier: c

Identifier: greatest

Identifier: equals

Identifier: b

Reserved word: not

Identifier: greatest

Identifier: equals

Identifier: c

Reserved word: show

Separator: (

Separator: '

Identifier: The

Identifier: largest

Identifier: number

Identifier: is

Separator: :

Separator: '

Separator: ,

Identifier: greatest

Separator: )

Reserved word: acaba
```