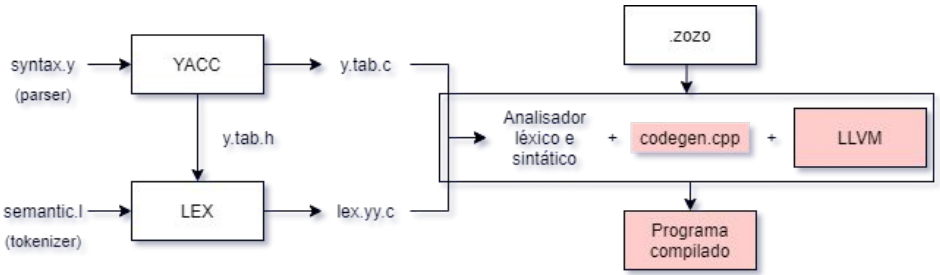
Linguagem Zozoba (.zozo)

Linguagem baseada nas palavras zozo e oba, em homenagem ao meu amigo Enzo e o resto do clã.

A linguagem Zozoba é adequada a uma gramática livre de contexto e apresenta estruturas básicas de uma linguagem de programação (variáveis, condicionais, loops e funções)

Processo de criação:

Com YACC, LEX e LLVM:



Com o compilador de Lógica da Computação:

- Tokenizador em Python;
- Parser em Python;
- Baseado em AST (gerada e executada em Python)

Ou seja

Tokenizador

Token	Valor
EOF	Não tem
INT_VAL	Número inteiro
FLOAT_VAL	Número decimal
PLUS	+
MINUS	-
MULT	*
POW	**
RESTO	%
DIV	/
DIV_INT	//

Token	Valor
ABRE_PAR	(
FECHA_PAR	)
P_VIRGULA	;
DOIS_P	:
VIRGULA	,
IGUAL	=
D_IGUAL	==
MAIOR	>
MENOR	<
NOT	!

Tokenizador (cont)

Token	Valor
PRINT	zrint
OR	obou
AND	obe
WHILE	ible
WHILE_E	zible
IF	ib
IF_E	zib
ELIF	ebib
ELSE	eble

Token	Valor
TRUE	true
FALSE	false
DEF	eb
DEF_E	zeb
RETURN	zeturn
INPUT	zinput
IDEN	Qualquer palavra não reservada
STRING_VAL	Qualquer coisa entre aspas duplas

Diagrama sintático

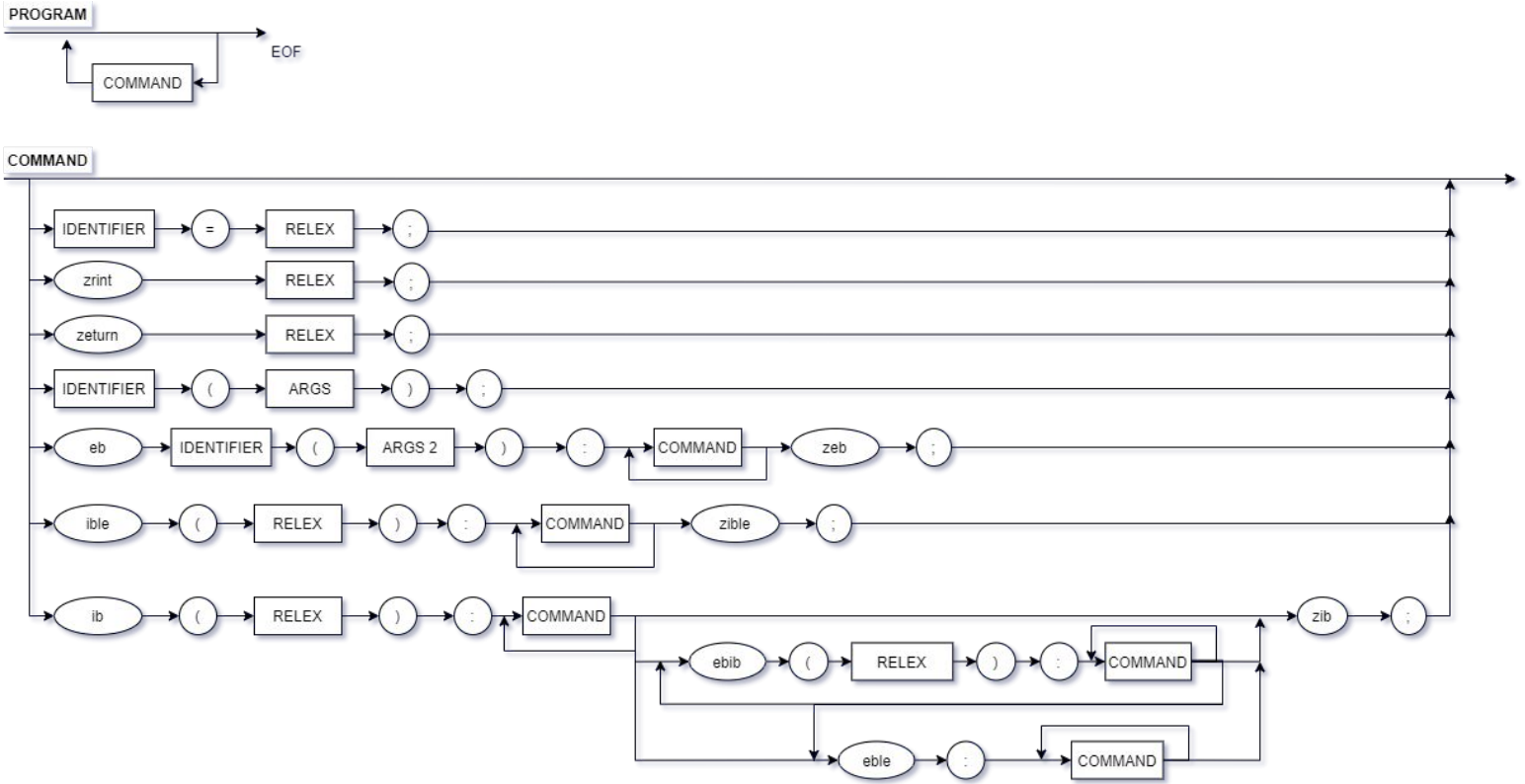
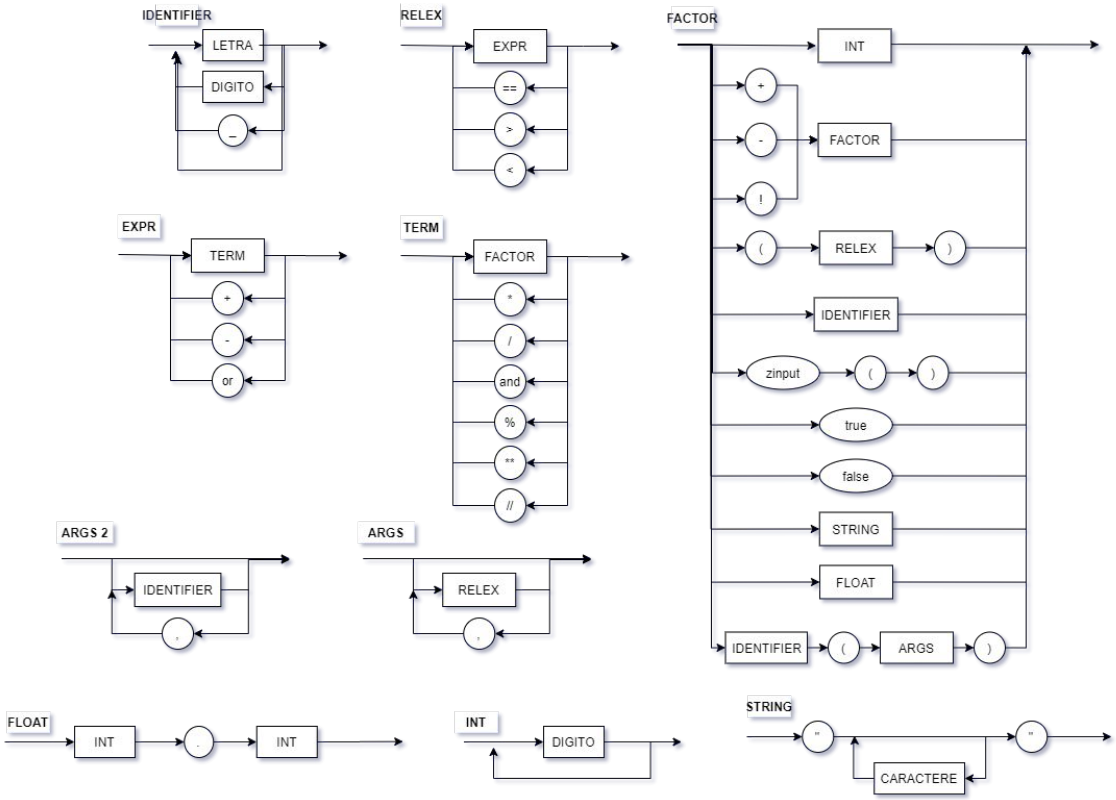


Diagrama sintático (cont)



EBNF

```
PROGRAM = {COMMAND};

COMMAND = ( IDENTIFIER, "=", RELEX, ";" |
          IDENTIFIER, "(", ARGS, ")", ";" |
          "zrint", RELEX, ";" |
          "zeturn", RELEX, ";" |
          "eb", IDENTIFIER, "(", ARGS2, ")", ":" COMMAND, {COMMAND}, "zeb", ";" |
          "ible", "(", RELEX, ")", ":" COMMAND, {COMMAND}, "zible", ";" |
          "ib", "(", RELEX, ")", ":" COMMAND, {COMMAND}, ( ELIF | ("zib", ";") ) );

ELIF = "ebib", "(", RELEX, ")", ":", COMMAND, {COMMAND}, ( ELIF | ("zib", ";") |
             "eble", ":", COMMAND, {COMMAND}, "zib", ";");

RELEX = EXPR, { ( "==" | ">" | "<" ), EXPR };

EXPR = TERM, { ( "+" | "-" | 'obou" ), TERM };

TERM = FACTOR, { ( "*" | "/" | "%" | "//" | "**" | "obe" ), FACTOR };

FACTOR = ( INT |
         FLOAT |
         ( "+" | "-" | "!"), FACTOR |
         "(", RELEX ")" |
         IDENTIFIER |
         "zinput", "(", ")" |
         "true" | "false" |
         STRING |
         IDENTIFIER, "(", ARGS, ")" );

ARGS = {RELEX, {",", RELEX}};

ARGS2 = {IDENTIFIER, {",", IDENTIFIER}};

INT = DIGITO, {DIGITO};

STRING = "", CARACTERE, "";

FLOAT = INT, ".", INT;

IDENTIFIER = LETRA, { (LETRA | DIGITO | "_") };
```

AST - Tratamento de operadores

Para operadores aritméticos e relacionais, os valores booleanos são convertidos para integers (true = 1; false = 0)

Para operadores booleanos, os valores integers são convertidos para booleans (0 = false; qualquer outro = true)

Especificidades:

- “+” contendo string gera a concatenação.
- “*” contendo string e integer gera a string repetida *n* vezes.

Problemas:

Operadores não tratam floats (floats são tratados como integer)

AST - Tratamento de Identifiers

Única symbol table para variáveis e funções.
Possibilidade de redefinir qualquer variável

Symbol Table = {“identifier”: {“value”: 5, “val_type”: “int”; “id_type”: “iden”},
{“value”: True, “val_type”: “bool”; “id_type”: “iden”},
{“value”: FuncDef.Object, “id_type”: “func”}}

Definição na Symbol Table:

- Variáveis
  Node Assign
- Funções:
  Node FuncDef

Chamada na Symbol Table:

- Variáveis
  Node Iden
- Funções:
  Node FuncCall

Problemas:
Não necessidade de “id_type” na Symbol Table

Exemplos

```
/* Operadores aritmeticos */
zrint 1+1;          /*2*/
zrint -1;           /*-1*/
zrint 20/10;        /*2*/
zrint 20/5+5;       /*9*/
zrint +----++--2;   /*2*/
zrint (3+6)/(4-1);  /*3*/
zrint 2**3;         /*8*/
zrint 3%2;          /*1*/
zrint 5//2;         /*2*/
```

```
$ python compilador.py testes/test0.zozo
2
-1
2
9
2
3
8
1
2
```

Exemplos

```
/* Identifiers, booleanos e relacionais*/
a = 1;
b = True;
c = falSe;
d = 0;
e = 6;

zrint ">aritimeticos";

f = a + b;                              /*1 + true*/
zrint f;                                /*2*/

j = b + e + c;                          /*True + 6 + 0*/
zrint j;                                /*7*/

zrint ">booleanos";

g = d obe b;                            /*0 and true*/
zrint g;                                /*false*/

h = (e obe b) obou (a obe c);           /*(6 and true) or (1 and false)*/
i = ((d obe a) obe b) obou (a obe c);   /*((0 and 1) and true) or (1 and false)*/
zrint h;                                /*true*/
zrint i;                                /*false*/

zrint ">relacionais";

k = e < c;                              /*6 < false*/
zrint k;                                /*false*/

l = e > false;                          /*6 > false*/
zrint l;                                /*true*/

m = b > false;                          /*true > false*/
zrint !m;                               /*false*/
```

```
$ python compilador.py testes/test1.zozo
>aritimeticos
2
7
>booleanos
False
True
False
>relacionais
False
True
False
```

Exemplos

```
/* Leitura do terminal, if e while */

zrint "enter um numero";
a = zinput();
b = 5;

ib ((a == b) obou (a == b + 1)):
    zrint "normal";
    zrint "pp";
ebib (a > b):
    zrint "big";
    zrint "pp";
ebib (a < 0):
    zrint "no";
    zrint "pp";
eble:
    zrint "smol";
    zrint "pp";
zib;


ible (a < 20):
    zrint a;
    ib (a == 13):
        zrint "aaaaaaaaaaa";
    zib;
    a = a + 1;
zible;
```

```
$ python compilador.py testes/test2_zozo
enter um numero
8
big
pp
8
9
10
11
12
13
aaaaaaaaaaa
14
15
16
17
18
19
```

Exemplos

```
/* Strings e floats */

a = "OBa";
zrint a;            /*OBa*/

zrint a + 3;        /*OBa3*/

zrint a * 2;        /*OBaOBa*/

d = true;
zrint a + d;        /*OBaTrue*/

b = 3.5;
zrint b;            /*3.5*/

zrint b + 5.2;      /*8*/

zrint a == "OBa"; /*true*/
```

```
$ python compilador.py testes/test3.zozo
OBa
OBa3
OBaOBa
OBaTrue
3.5
8
True
```

Exemplos

```
/* Funcoes */

zrint "enter um numero";
a = zinput();

eb Oba(a,z):
    l = a + z;

    eb Noba():
        zrint "noba";  /*noba*/
    zeb;
    Noba();

    zeturn l;
zeb;

c = Oba(a, 4);

zrint "c: " + c;    /*c: a+4*/

Oba = 2;

c = Oba;

zrint "c: " + c;    /*c: 2*/
```

```
$ python compilador.py testes/test4.zozo
enter um numero
7
noba
c: 11
c: 2
```