22/04/09

44315 - Alberto Noronha

analex

```
%{
#include "tokens.h"
%}
\[      return ABRE;
\]      return FECHA;
\,      return VIRG;
[0-9]+  return NUM;
<<EOF>> return fim;
[ \t\n] ;
        return ERRO;
{ yylval = atoi(yytext);
  return NUM;
}
```

anasin.c

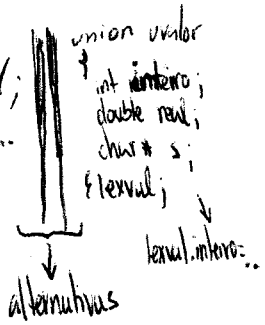
```
int main()
{
 prox_simb = yylex();
 rec_Z();
}
int rec_Z()
{
 rec_list();
 rec_term(fim);
}
int rec_list()
{
 rec_term(ABRE);
 rec_List2();
}
```

Gramática Independente do Contexto

```
Z → List '$'
List → '[' List2
List2 → ']'
      | Elemlist ']'
Elemlist → NUM Elemlist2
Elemlist2 → ',' Elemlist
          | ε
```

tokens.h

```
#ifndef _TOKENS
#define _TOKENS
#define ABRE 100
...
int lexval;
lexval=...
#endif
```



GRAMÁTICA ABSTRACTA

~~Z → List~~

~~List → List2~~

List → ε
| Elemlist

Elemlist → NUM Elemlist2

Elemlist2 → Elemlist
| ε

(passando os valores para as identidades)

f_1 List → NUM List
f_2 | ε

1. Para cada X ∈ N (não terminal):

2. Para cada p ∈ P
$p: X \to x_1, x_2, \ldots, x_n$

```
typedef struct slist
{
 int flag;
 union {
   struct {
     int s1;
     struct slist *s2;
   } p1;
   struct {} p2;
 } derivacoes;
} NodeList;
```

rl.h

```
typedef struct slist *List;
list s2;
(o c permite declarações "dummy")
```

```
List consp1(int s1, List s2)
{
 List aux;
 aux=(List) malloc(sizeof(Nodelist));
 aux->flag = p1;
 aux->derivacoes.p1.s1 = s1;
 aux->derivacoes.p1.s2 = s2;
 return aux;
}

List consp2()
{
 List aux;
 aux = (List) malloc(sizeof(Nodelist));
 aux->flag = p2;
 return aux;
}
```

rl.c