Bison

Bison: Introdução

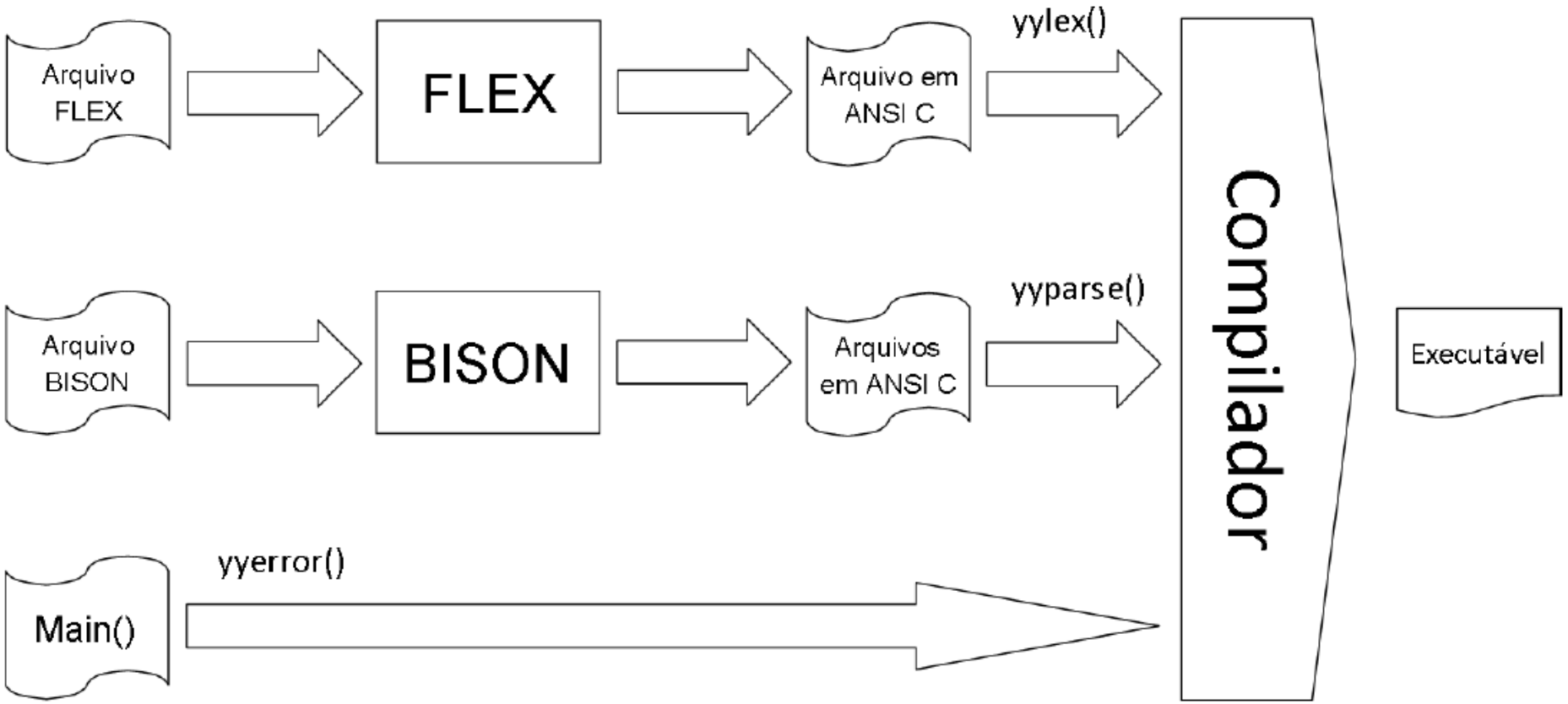
- O **Bison** é uma ferramenta que tem o objetivo de gerar o código de um programa para reconhecer a estrutura gramatical de uma entrada.

- O bison é a evolução do programa yacc**.**

- Foi originalmente desenvolvido para a construção de compiladores, sendo utilizando na geração de **analisadores sintáticos**.

- O Bison recebe como entrada, basicamente, uma sequência de produções de uma gramática livre de contexto; o que fazer quando um produção é reconhecida (ações); e produz como saída uma *parser* LALR.

Bison: Geração do Sintático em Conjunto com o Flex



Bison: Estrutura do Arquivo de Entrada

Os arquivos de entrada possuem, em geral, a extensão **`.y`** e são constituídos de 3 seções, delimitadas pelos caracteres **`%%`**.

Definições [opcional]

`%%`

Regras {ação} [padrão]

`%%`

Código Auxiliar em C/C++ [opcional]

Bison: Estrutura do Arquivo de Entrada

- A seção "***Definições***" é utilizada para a definição de macros e porções de código em C/C++. Toda porção de código C/C++ deve estar delimitado por ***%{*** e ***%}***.

- As "***Regras***" são destinadas a fornecer as regras gramaticais de todos os símbolos não-terminais e terminais, descrição da procedência de operadores além de tipos de dados dos valores semânticos dos diversos símbolos da gramática. As regras gramaticais definem a construção de cada símbolo não-terminal a partir dos *tokens* que o compõem.

- A seção "***Código Auxiliar em C/C++***" pode ser utilizada opcionalmente para descrever rotinas auxiliares, porém, no caso do programa escrito para o Bison ser independente, é necessário que essa seção contenha a função ***main()***.

Bison: Regras Gramaticais

Estrutura:

```
simbolo:definição { ação }
       |definição { ação }
       |          { ação }
;
```

- Dois pontos (:) separam o lado esquerdo do lado direito da regra, e o ponto-e-vírgula (;) finaliza a regra.
- A barra vertical “|” mostra as possibilidades de derivação para um mesmo não-terminal.
- Cada regra pode ter uma ação associada a ela. Essa ação está entre chaves ({}).

Exemplo 1:
Parser Simples

Exemplo 1: lexico.l

```
%option noyywrap
%{
#include <sintatico.tab.h>
%}

%%

"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
[0-9]+  { return NUMBER; }
\n      { return EOL; }
[ \t]   { /* espaço em branco/tabulação */ }
.       { printf("Caracter Misterioso: %c\n", *yytext); exit(0);}

%%
```

Exemplo 1: sintatico.y

```
%{
#include <stdio.h>
extern int yylex();
extern char* yytext;
void yyerror(void *s);
%}

%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist

%%

calclist: exp EOL { printf("Cadeia Aceita.\n"); return 0; }
;

exp: factor          { }
   | exp ADD factor  { }
   | exp SUB factor  { }
;

factor: term             { }
      | factor MUL term  { }
      | factor DIV term  { }
;

term: NUMBER   { }
;

%%

int main(int argc, char** argv)
{
    yyparse();
    return 0;
}
```

calclist → *exp* *EOL*
exp → *factor*
exp → *exp* *ADD* *factor*
exp → *exp* *SUB* *factor*
factor → *term*
factor → *factor* *MUL* *term*
factor → *factor* *DIV* *term*
term → *NUMBER*

Bison: Geração do Sintático

```
$ flex arquivo.l
$ bison -d arquivo.y
$ gcc *.c -I. -o programa
```

O programa gerado já pode ser executado:

```
$ ./programa < arquivo_de_entrada
```

A opção **-d** no bison é para a geração de um arquivo de cabeçalho a ser utilizado pelo flex.

Exemplo 1
na prática…

Exemplo 2:
Uma pequena calculadora

Exemplo 2: lexico.l

```
%option noyywrap
%{
#include <sintatico.tab.h>
extern YYSTYPE yylval; /* YYSTYPE possui tipo int por padrao */
%}

%%

"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
[0-9]+  { yylval = atoi(yytext); return NUMBER; }
\n      { return EOL; }
[ \t]   { /* espaço em branco/tabulação */ }
.       { printf("Caracter Misterioso: %c\n", *yytext); exit(0);}

%%
```

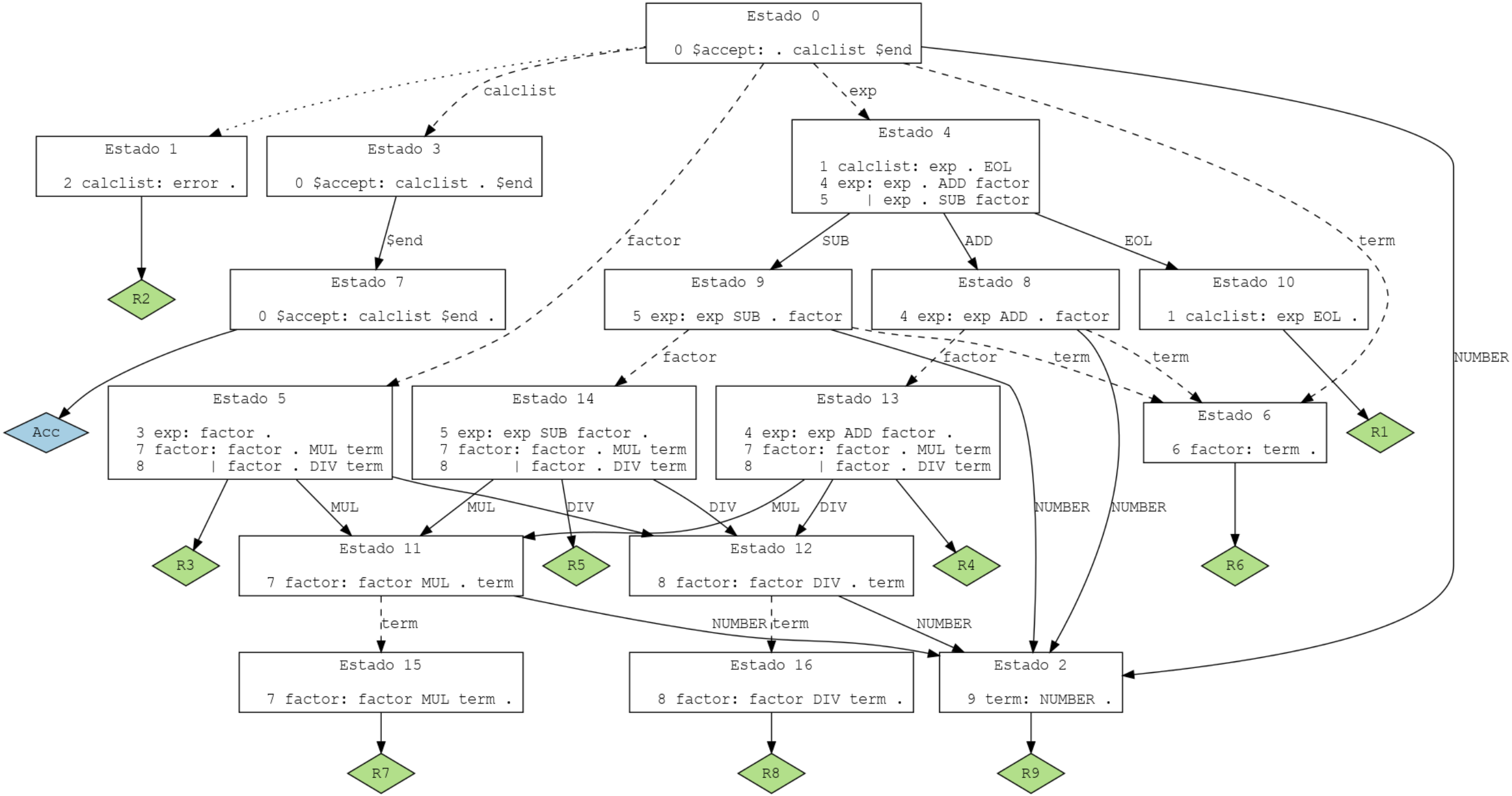
Exemplo 2: sintatico.y

```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

calclist → *exp* *EOL*
exp → *factor*
exp → *exp* *ADD* *factor*
exp → *exp* *SUB* *factor*
factor → *term*
factor → *factor* *MUL* *term*
factor → *factor* *DIV* *term*
term → *NUMBER*


Estado 0
0 $accept: . calclist $end
calclist
exp
factor
term
NUMBER
Estado 1
2 calclist: error .
R2
Estado 3
0 $accept: calclist . $end
$end
Estado 7
0 $accept: calclist $end .
Acc
Estado 4
1 calclist: exp . EOL
4 exp: exp . ADD factor
5 | exp . SUB factor
SUB
ADD
EOL
Estado 9
5 exp: exp SUB . factor
Estado 8
4 exp: exp ADD . factor
Estado 10
1 calclist: exp EOL .
R1
Estado 5
3 exp: factor .
7 factor: factor . MUL term
8 | factor . DIV term
Estado 14
5 exp: exp SUB factor .
7 factor: factor . MUL term
8 | factor . DIV term
Estado 13
4 exp: exp ADD factor .
7 factor: factor . MUL term
8 | factor . DIV term
Estado 6
6 factor: term .
R6
MUL
DIV
R3
R5
R4
Estado 11
7 factor: factor MUL . term
Estado 12
8 factor: factor DIV . term
Estado 15
7 factor: factor MUL term .
Estado 16
8 factor: factor DIV term .
Estado 2
9 term: NUMBER .
R7
R8
R9

Exemplo 2: sintatico.y

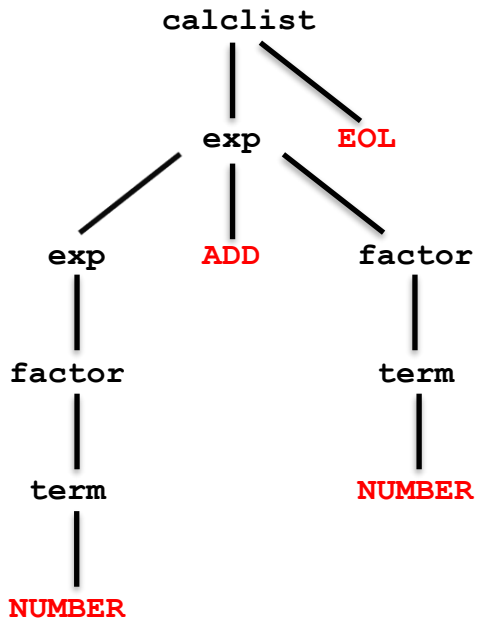
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term            { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

Exemplo 2: sintatico.y

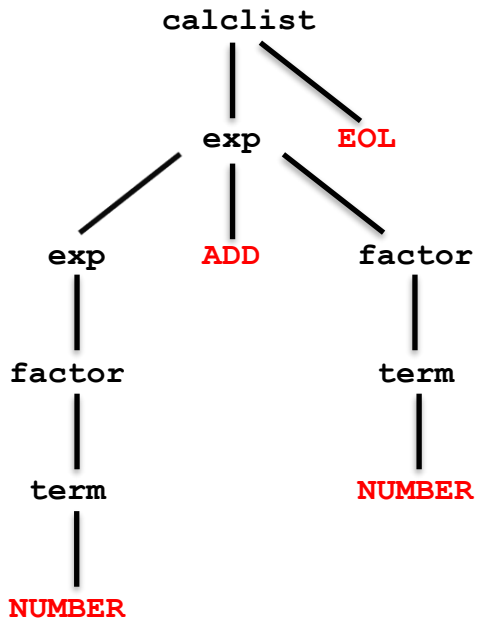
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

Stack: $_0$

NUMBER ADD NUMBER EOL $

Exemplo 2: sintatico.y

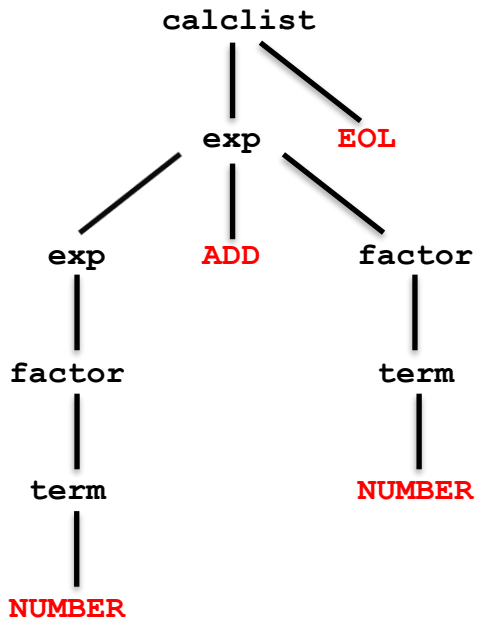
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term            { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

```
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
```

Stack: $_0$

NUMBER ADD NUMBER EOL $

Exemplo 2: sintatico.y

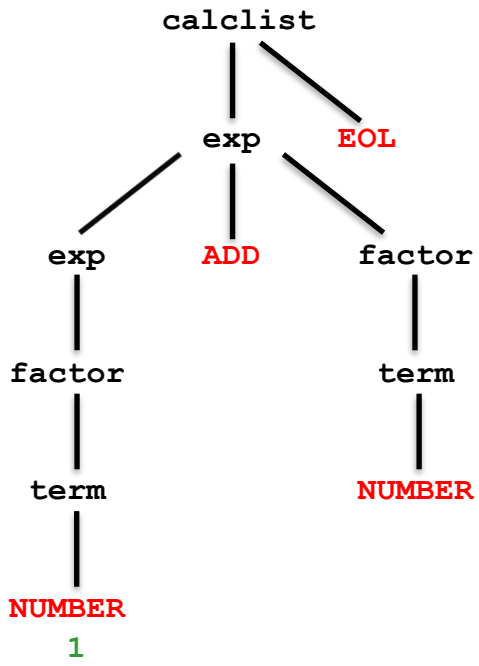
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1

Stack: $_0$**NUMBER**$_2$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

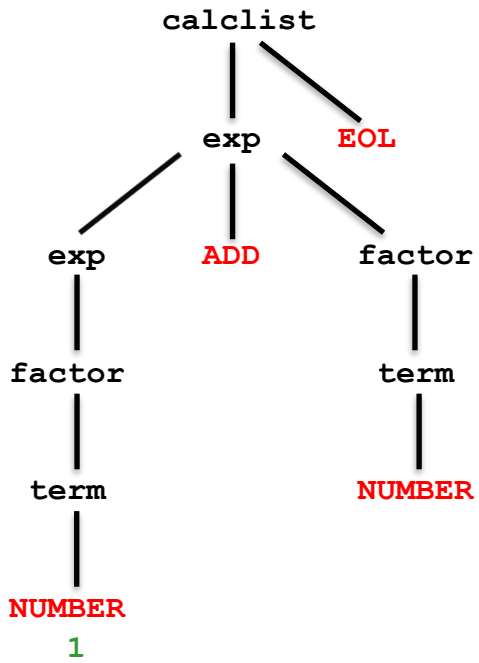
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1

Stack: $_0$NUMBER$_2$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

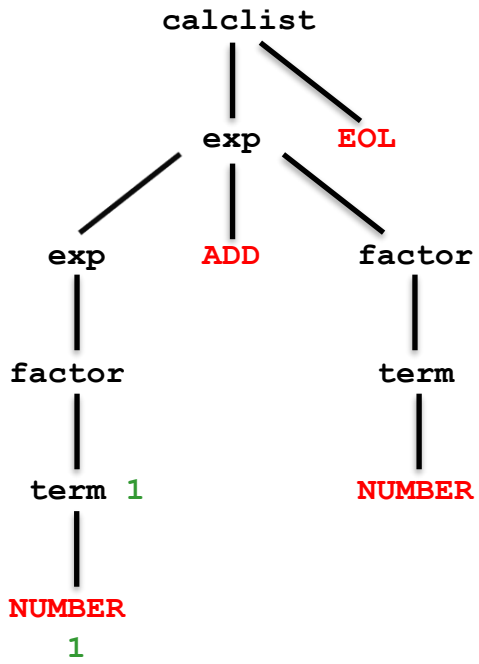
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1

Stack: $_0$term$_6$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

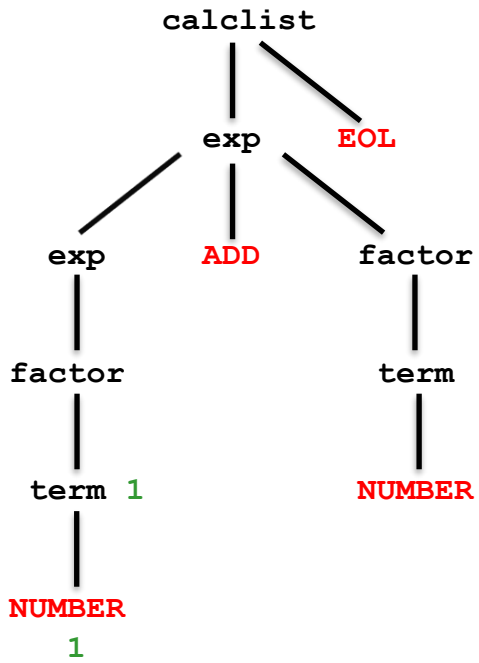
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1

Stack: $_0$term$_6$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

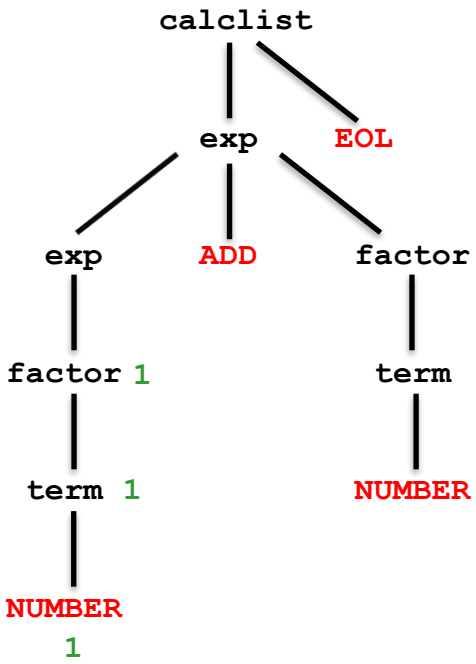
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
~~*factor → term*~~
~~*term → NUMBER*~~

1

Stack: $_0$factor$_6$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

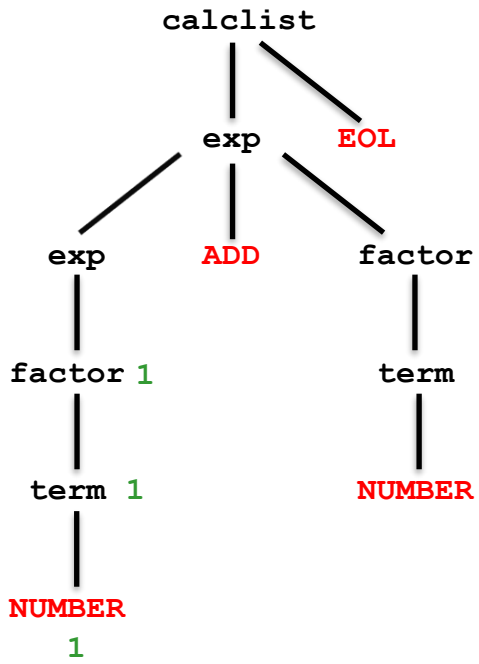
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
~~*factor → term*~~
~~*term → NUMBER*~~

1

Stack: $_0$factor$_6$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

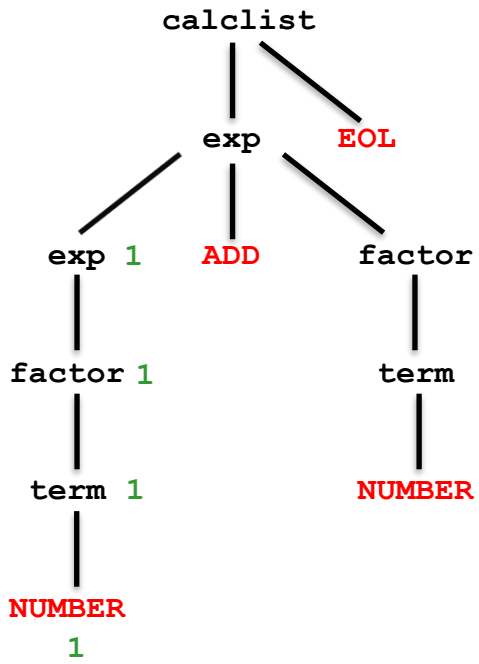
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1
Stack: $_0$exp$_6$

ADD NUMBER EOL $

Exemplo 2: sintatico.y

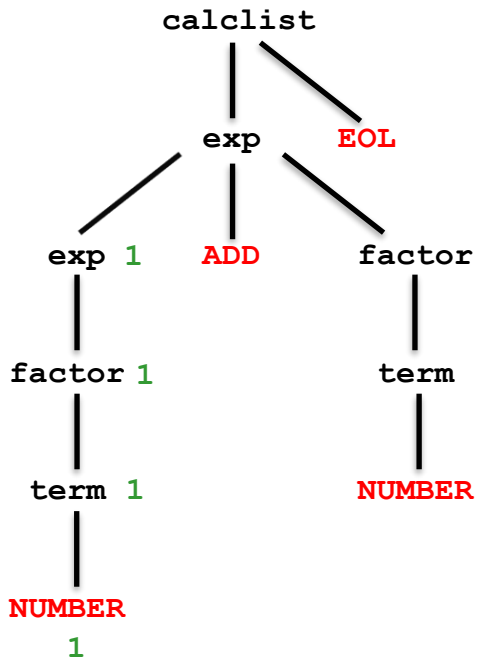
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

1
Stack: $_0$exp$_6$ADD$_8$

NUMBER EOL $

Exemplo 2: sintatico.y

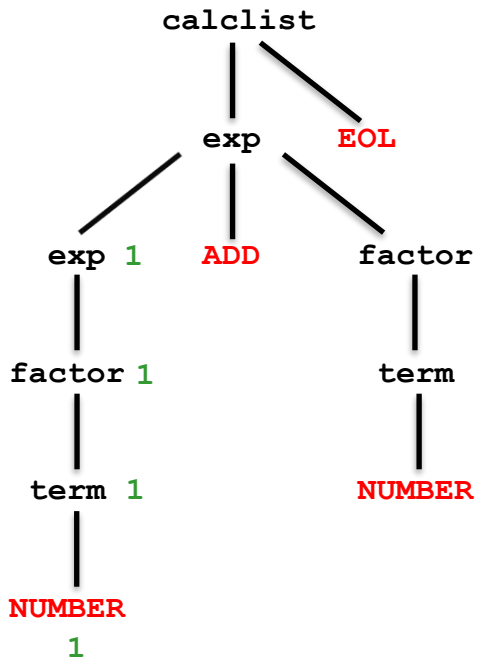
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

```
[0-9]+ { yylval = atoi(yytext); return NUMBER; }
```

Stack: $_0$exp$_6$ADD$_8$ (1)

NUMBER EOL $

Exemplo 2: sintatico.y

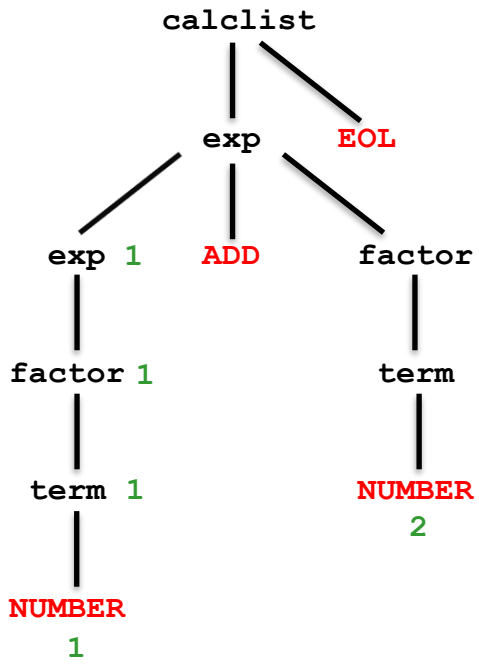
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
~~*exp → factor*~~
~~*factor → term*~~
~~*term → NUMBER*~~

Stack: $_0$exp $_6$ADD $_8$NUMBER $_2$ (1 2)

EOL $

Exemplo 2: sintatico.y

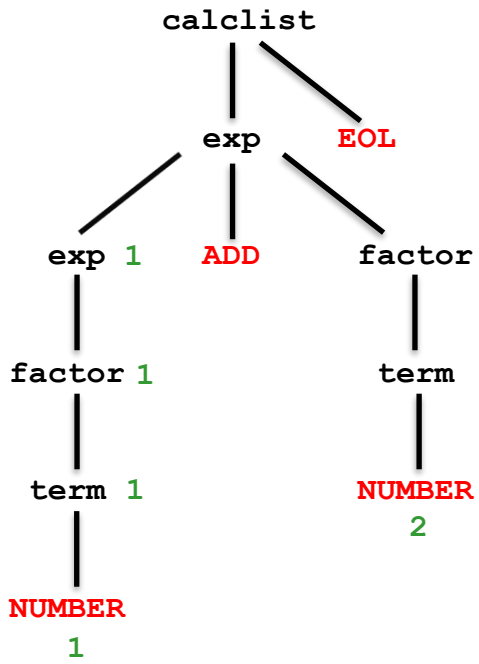
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

Stack: $_0$**exp** $_6$**ADD** $_8$**NUMBER** $_2$ **EOL $**

Exemplo 2: sintatico.y

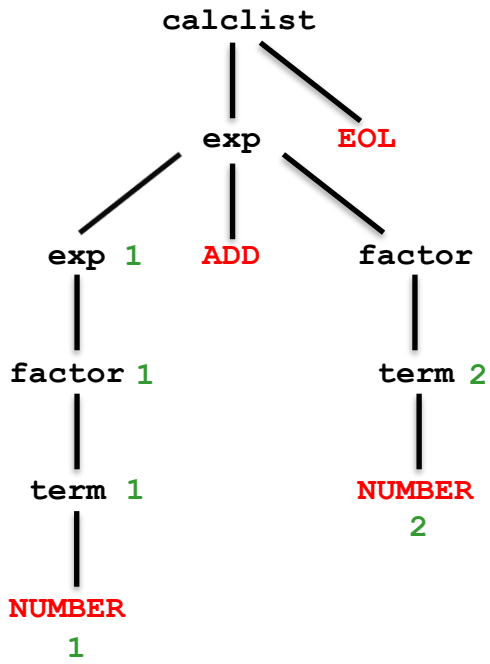
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

Stack: $_0$**exp**$_6$**ADD**$_8$**term**$_6$ (1, 2)

EOL $

Exemplo 2: sintatico.y

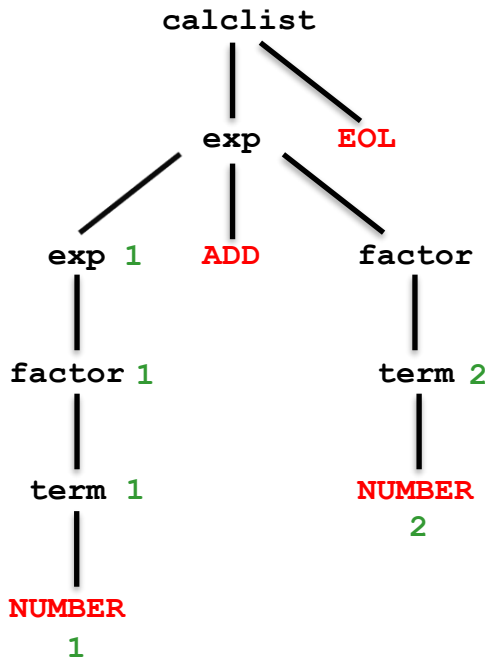
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

Stack: $_0$exp$_6$ ^{1}ADD$_8$ ^{2}term$_6$

EOL $

Exemplo 2: sintatico.y

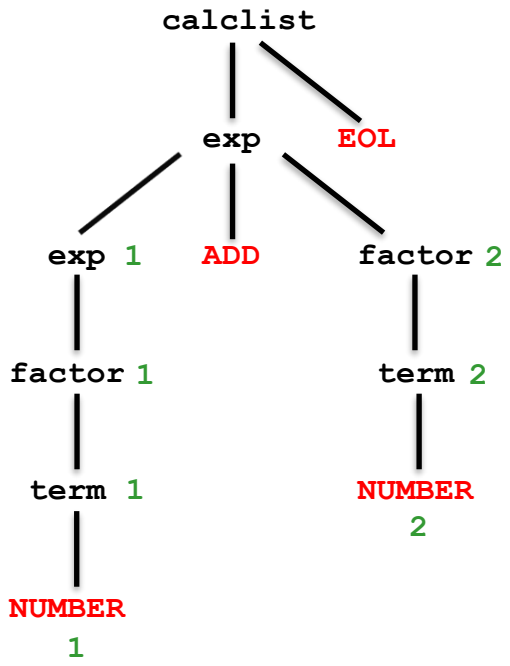
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
~~*factor → term*~~
~~*term → NUMBER*~~
~~*exp → factor*~~
~~*factor → term*~~
~~*term → NUMBER*~~

Stack: $_0$exp $_6$ADD $_8$factor $_{13}$ (1 2)

EOL $

Exemplo 2: sintatico.y

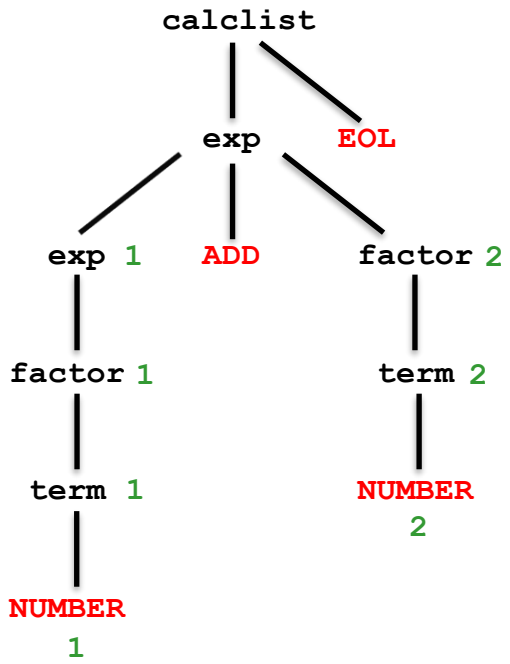
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
~~*factor → term*~~
~~*term → NUMBER*~~
~~*exp → factor*~~
~~*factor → term*~~
~~*term → NUMBER*~~

Stack: $_0$exp $_6$ADD $_8$factor $_{13}$ (1 2)

EOL $

Exemplo 2: sintatico.y

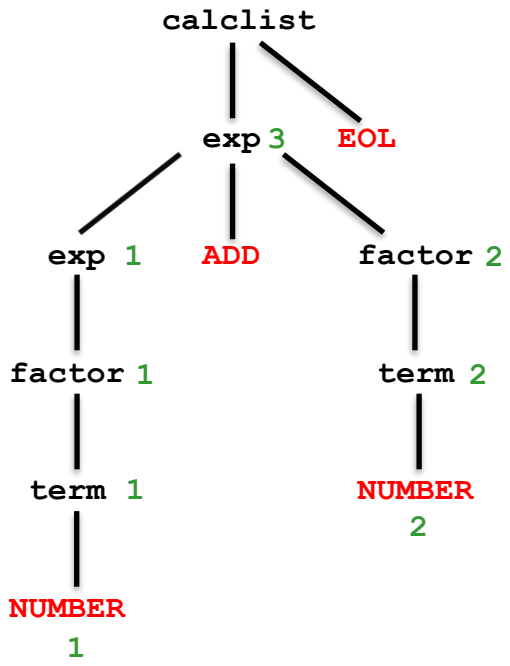
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

3

Stack: $_0$exp$_4$

EOL $

Exemplo 2: sintatico.y

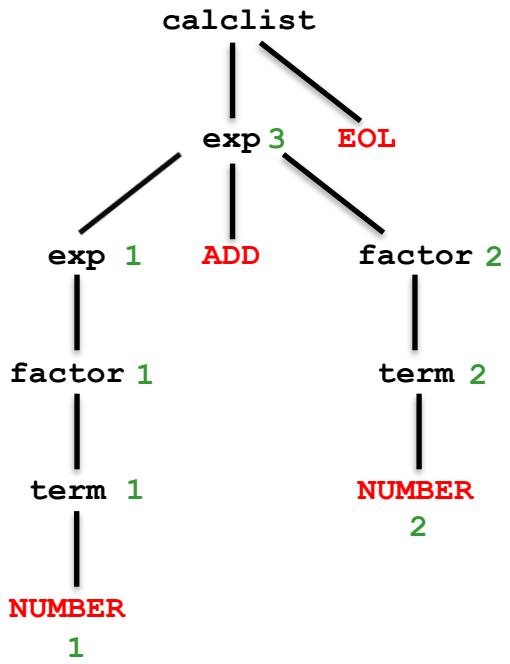
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

3
Stack: $_0$exp$_4$EOL$_{10}$ $

Exemplo 2: sintatico.y

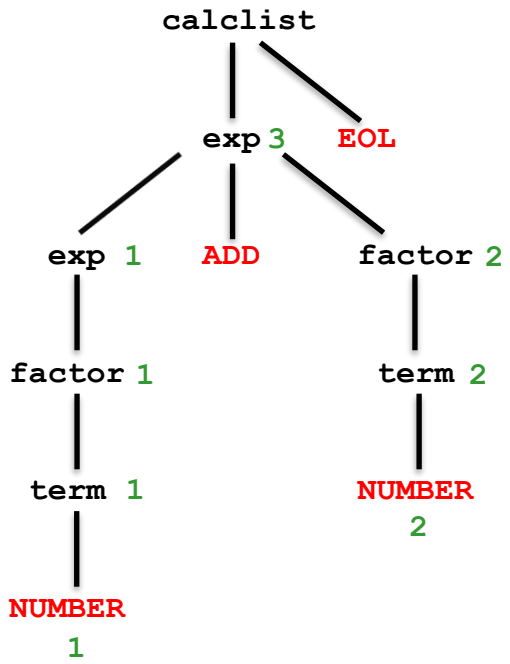
```
%{
#include <stdio.h>
extern int yylex();
extern char *yytext;
void yyerror(void *s);
%}
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%start calclist
%%
calclist: exp EOL { printf("= %d\n\n", $1); return $1;}
        | error   { return 0; }
;
exp: factor          { $$ = $1;}
   | exp ADD factor { $$ = $1 + $3; }
   | exp SUB factor { $$ = $1 - $3; }
;
factor: term             { $$ = $1;}
      | factor MUL term { $$ = $1 * $3; }
      | factor DIV term { $$ = $1 / $3; }
;
term: NUMBER   { $$ = $1;}
;

%%
int main(int argc, char** argv)
{
    while (yyparse());
    return 0;
}
```

1 + 2

NUMBER ADD NUMBER EOL $



calclist → exp EOL
exp → factor
exp → exp ADD factor
exp → exp SUB factor
factor → term
factor → factor MUL term
factor → factor DIV term
term → NUMBER

calclist → exp EOL
exp → exp ADD factor
factor → term
term → NUMBER
exp → factor
factor → term
term → NUMBER

3

Stack: $_0$exp$_4$EOL$_{10}$ $

Exemplo 2
na prática…

Exemplo 3:
Alocação de uma árvore binária e percurso em pós-ordem

Exemplo 3: ast.h

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node TreeNode;
struct node
{
        int node_type;
        int value;
        TreeNode* left;
        TreeNode* right;
};

void RPN_Walk(TreeNode* aux);
void Delete_Tree(TreeNode* aux);
```

Exemplo 3: ast.c

```
#include <ast.h>
#include <sintatico.tab.h>

void RPN_Walk(TreeNode* aux)
{
    if(aux)
    {
        RPN_Walk(aux->left);
        RPN_Walk(aux->right);
        switch(aux->node_type)
        {
            case ADD:{printf("+ ");};break;
            case SUB:{printf("- ");};break;
            case MUL:{printf("* ");};break;
            case DIV:{printf("/ ");};break;
            case NUMBER:{printf("%d ",aux->value);};break;
            default:{printf("ERROR: INVALID TYPE ");};break;
        }
    }
}

void Delete_Tree(TreeNode* aux)
{
    if(aux)
    {
        Delete_Tree(aux->left);
        Delete_Tree(aux->right);
        free(aux);
    }
}
```

Exemplo 3: lexico.l

```
%option noyywrap
%{
#include <ast.h>
#include <sintatico.tab.h>
%}

%%

"+"     { return ADD; }
"-"     { return SUB; }
"*"     { return MUL; }
"/"     { return DIV; }
[0-9]+  { yylval.integer = atoi(yytext); return NUMBER; }
\n      { return EOL; }
[ \t]   { /* espaço em branco/tabulação */ }
.       { printf("Caracter Misterioso: %c\n", *yytext); exit(0);}

%%
```

Exemplo 3: sintatico.y

```
%{

#include <stdio.h>
#include <ast.h>
extern int yylex();
extern char* yytext;
void yyerror(char *s);

TreeNode* AST = NULL;

%}

%union{
    TreeNode* ast;
    int integer;
}

/* declare tokens */
%token NUMBER
%token ADD
%token SUB
%token MUL
%token DIV
%token EOL

%type <ast> calc
%type <ast> exp
%type <ast> factor
%type <ast> term
%type <integer> NUMBER

%start calc

%%
```

Exemplo 3: sintatico.y

```
calc: exp EOL { AST = $1;
                if(AST)
                {
                    RPN_Walk(AST);
                    Delete_Tree(AST);
                }
                else
                {
                    printf("AST is NULL\n");
                }
                return 0;
              }
;

exp: factor           { $$ = $1;}
    | exp ADD factor {TreeNode* aux = (TreeNode*)malloc(sizeof(struct node));
                      aux->node_type = ADD;
                      aux->value = 0;
                      aux->left = $1;
                      aux->right = $3;
                      $$ = aux;
                     }
    | exp SUB factor {TreeNode* aux = (TreeNode*)malloc(sizeof(struct node));
                      aux->node_type = SUB;
                      aux->value = 0;
                      aux->left = $1;
                      aux->right = $3;
                      $$ = aux;
                     }
;
```

```
factor: term          { $$ = $1;}
    | factor MUL term {TreeNode* aux = (TreeNode*)malloc(sizeof(struct node));
                       aux->node_type = MUL;
                       aux->value = 0;
                       aux->left = $1;
                       aux->right = $3;
                       $$ = aux;
                      }
    | factor DIV term {TreeNode* aux = (TreeNode*)malloc(sizeof(struct node));
                       aux->node_type = DIV;
                       aux->value = 0;
                       aux->left = $1;
                       aux->right = $3;
                       $$ = aux;
                      }
;

term: NUMBER   { TreeNode* aux = (TreeNode*)malloc(sizeof(struct node));
                 aux->node_type = NUMBER;
                 aux->value = $1;
                 aux->left = NULL;
                 aux->right = NULL;
                 $$ = (TreeNode*) aux;
               }
;

%%

void yyerror(char *s)
{
    printf("Erro de Sintaxe: %s", yytext);
    exit(0);
}

int main(int argc, char **argv)
{
    return yyparse();
}
```

Exemplo 3
na prática…

Conteúdo Programático e Cronograma

1º Bimestre:

- ~~**Organização e estrutura de compiladores**~~
- ~~**Análise Léxica**~~
- ~~**Análise Sintática**~~
- ~~**Ferramentas de geração automática de compiladores**~~

2º Bimestre:

- **Análise Semântica**

Lista de Exercícios

Lista 14

- Exercício Prático.