

# Teoria da Computação e Compiladores

Análise Sintática

- Yacc

# Sumário

- Introdução ao Yacc
- Estrutura do Yacc
- Lex & Yacc
- Livraria e opções do Yacc



# O que é YACC?

**YACC** do inglês **Y**et **A**nother **C**ompiler-**C**ompiler

‘mais um compilador de compiladores’

É um **gerador de analisadores sintáticos** que incorpora o algoritmo de análise sintática LALR(1).

É um programa que recebe como **entrada** uma dada especificação de Gramática Livre de Contexto (GLC) escrita em uma notação semelhante à Backus-Naur Form (BNF), e produz como **saída** um procedimento de análise sintático para aquela especificação em C.

Foi desenvolvido por Stephen c. Johnson da Bell Labs para Unix no início dos anos 70's.



O YACC foi reescrito para outras linguagens, incluindo Pascal, Java, Python, Ruby, Go, Erlang.

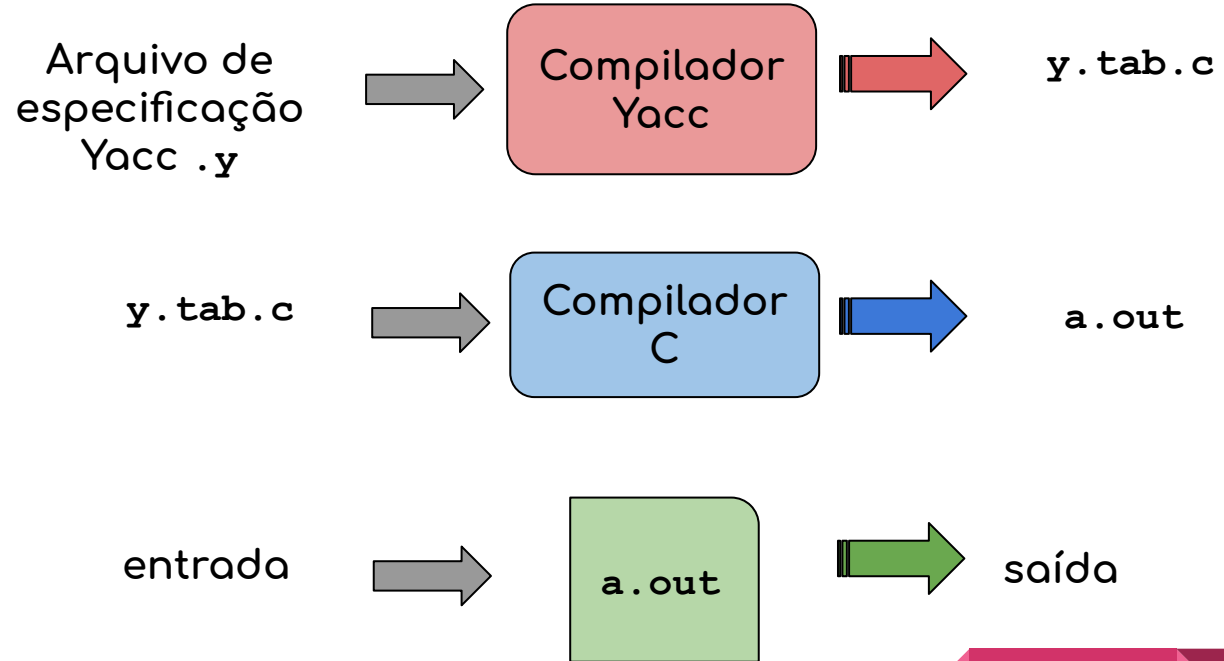
**Bison:** versão GNU do Yacc

**PLY** (Python Lex-Yacc): implementação alternativa de Lex e Yacc em python



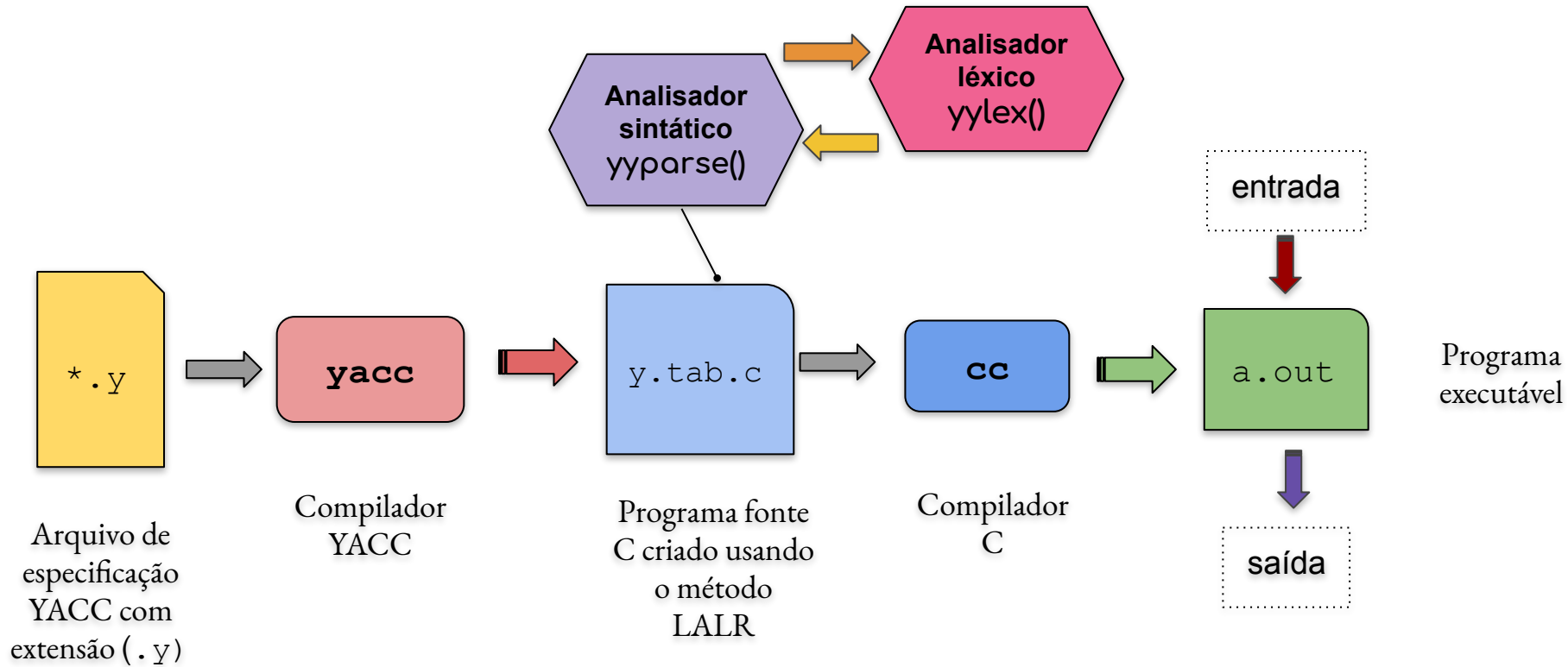
# Yacc

Para construir um tradutor, o Yacc opera da seguinte forma:



- O analisador gerado pelo Yacc é uma função C `yyparse()` – um analisador LALR(1)
- `yyparse()` chama `yylex()` repetidamente para obter o próximo token de entrada.
- A função `yylex()` pode ser codificada a mão em C ou gerada por Lex.
- `yyparse()` retorna um valor inteiro
  - 0 se a análise for bem sucedida e o final do arquivo for atingido
  - 1 se a análise falhar devido a um erro de sintaxe.





# Estrutura do Yacc

Declarações

1

%%

Regras

2

%%

Rotinas auxiliares

3





# Declarações

# Declarações

```
%{
```

## Declarações em C

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

```
%}
```

```
/* Declarações YACC */
```

## Declarações de tokens

```
%token DIGITO
```

```
%left '+' '-'
```

```
%%
```

- Declarações C delimitadas por %{ e %}

- Declarações de tokens da forma  
%token DIGITO NUMERO ...  
%"+ token PLUS

(expr : expr PLUS expr)

%left %right %nonassoc:  
Associatividade e precedência

- Pode usar caracteres com aspas simples como tokens sem declará-los.  
No caso  
expr : expr '+' expr

não precisamos declarar "=", "+" ou "-".

---

# Regras

# Regras

Contém regras gramaticais na forma BNF

%%

```
start: expr '\n'      {printf("%d\n", $1);}  
      ;
```

```
expr: expr '+' expr    {$$ = $1 + $3;}  
     | term            ;
```

```
term: expr '*' factor   {$$ = $1 * $3;}  
     | factor           ;
```

```
factor: '(' 'expr' ')'  { $$ = $2;}  
      | DIGITO          ;
```

%%

## Produção

```
expr: expr '+' expr  
     | term  
     ;
```

## Ação semântica \*

```
expr : expr '+' expr    { $$ = $1 + $3;}  
     | term             { $$ = $1; } (default)  
     ;
```

- Ação semântica é um fragmento de código delimitado por '{ }'.
- No final de cada regra colocamos ' ; '.

<cabeça> : <corpo> atributo - valor

\$\$ : valor do atributo associado ao não-terminal  
<cabeça> do lado esquerdo da produção

\$i : (\$1, \$2, \$3, ..) valor do i-ésimo símbolo da  
gramática (terminal ou não-terminal) de <corpo>

---

# Rotinas auxiliares

# Rotinas Auxiliares

O analisador léxico é fornecido como `yyllex()` geralmente é produzido por Lex.

`main()`

Rotinas de recuperação de erros  
`yyerror()`

Se `token - nome` e `atributo - valor` é produzido por `yyllex()` então

`token - nome` deve ser declarado na seção de definições do Yacc

`atributo - valor` é comunicado ao analisador através da variável `yyval()` definido por Yacc.

---

# Programa Yacc

t1.y

1

2

3

```
1  %{
2  #include <stdio.h>
3  #include <string.h>
4  %}
5
6  %token NUMERO CALOR ESTADO TEMPERATURA
7
8  %%
9
10 s: /* empty */
11   | s comando
12   ;
13 comando: interruptor
14         | tempo
15         ;
16 interruptor: CALOR ESTADO
17             {printf("\tCalor ligado ou desligado\n");}
18             ;
19 tempo: TEMPERATURA NUMERO
20       {printf("\tNova temperatura definida\n");}
21       ;
22
23 %%
24 void yyerror(const char *str)
25 {
26     fprintf(stderr, "error: %s\n", str);
27 }
28
29 int yywrap()
30 {
31     return 1;
32 }
33
34 main()
35 {
36     yyparse();
37 }
```

# Opções Yacc

```
yacc -v *.y
```

**-v opção verbose** produz um arquivo `y.output`, o arquivo contém uma descrição textual da tabela de análise sintática LALR(1) utilizada pelo analisador sintático.

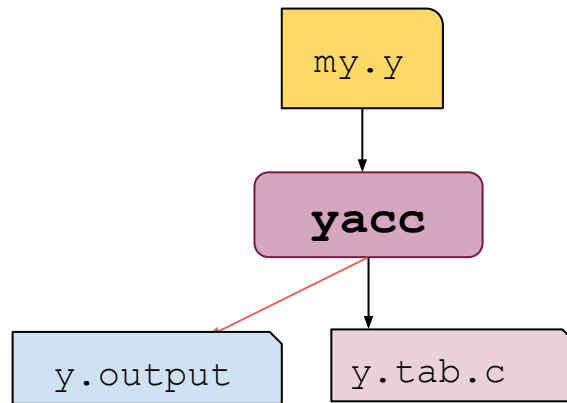
Uso importante:

Investigação de conflitos de análise sintática

Idéia:

Executar o Yacc com a opção verbose apenas na gramática para garantir que o analisador gerado pelo Yacc funcione como esperado.

Exemplo.





Arquivo gerado por

>> yacc -v t1.y

“y.output”

contém descrição textual  
da **tabela de análise**  
sintática LALR (1)

```
y.output x
Gramática
1
2
3 0 $accept: comandos $end
4
5 1 comandos: /* vazio */
6 2 | comandos comando
7
8 3 comando: interruptor
9 4 | tempo
10
11 5 interruptor: CALOR ESTADO
12
13 6 tempo: TEMPERATURA NUMERO
14
15
16 Terminais, com as regras onde eles aparecem
17
18 $end (0) 0
19 error (256)
20 NUMERO (258) 6
21 CALOR (259) 5
22 ESTADO (260) 5
23 TEMPERATURA (261) 6
24
25
26 Não-terminais com as regras onde eles aparecem
27
28 $accept (7)
29 | à esquerda: 0
30 comandos (8)
31 | à esquerda: 1 2, à direita: 0 2
32 comando (9)
33 | à esquerda: 3 4, à direita: 2
34 interruptor (10)
35 | à esquerda: 5, à direita: 3
36 tempo (11)
37 | à esquerda: 6, à direita: 4
38
39
40 estado 0
41
42 0 $accept: . comandos $end
43
44 $padrão reduzir usando a regra 1 (comandos)
45
46 comandos ir ao estado 1
47
```

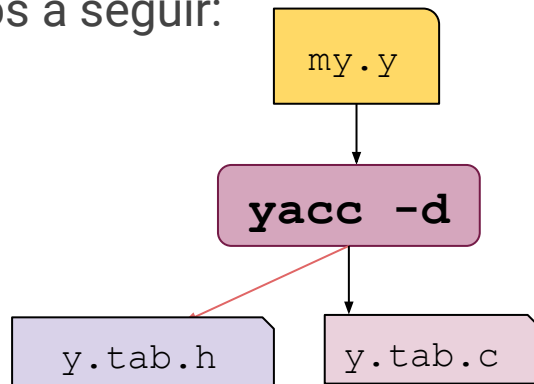
# Opções Yacc

`yacc -d *.y`

`-d` produz um arquivo de cabeçalho `y.tab.h` (além do arquivo `y.tab.c`) o conteúdo varia, mas normalmente inclui itens como os a seguir:

```
#ifnder YYSTYPE
#define YYSTYPE int
#endif
#define NUMBER      258

extern YYSTYPE yylval;
```



Esse arquivo pode ser utilizado na colocação do código para `yyllex` em um arquivo diferente, pela inserção da linha

```
#include y.tab.h
```


no arquivo.

Arquivo gerado por

>> yacc -d t1.y

“y.tab.h”

que será usado por  
Lex



```
y.tab.h
34
35
36 /* Tokens. */
37 #ifndef YYTOKENTYPE
38 # define YYTOKENTYPE
39     /* Put the tokens into the symbol table, so that GDB and other debuggers
40      know about them. */
41     enum yytokentype {
42         NUMERO = 258,
43         CALOR = 259,
44         ESTADO = 260,
45         TEMPERATURA = 261
46     };
47 #endif
48 /* Tokens. */
49 #define NUMERO 258
50 #define CALOR 259
51 #define ESTADO 260
52 #define TEMPERATURA 261
53
54
55
56
57 #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
58 typedef int YYSTYPE;
59 # define YYSTYPE_IS_TRIVIAL 1
60 # define YYSTYPE YYSYNTAX /* obsolescent; will be withdrawn */
61 # define YYSTYPE_IS_DECLARED 1
62 #endif
63
64 extern YYSTYPE yylval;
```

Arquivo gerado  
por Yacc -d

“y.tab.c”

onde se encontra a  
função `yyparse()`

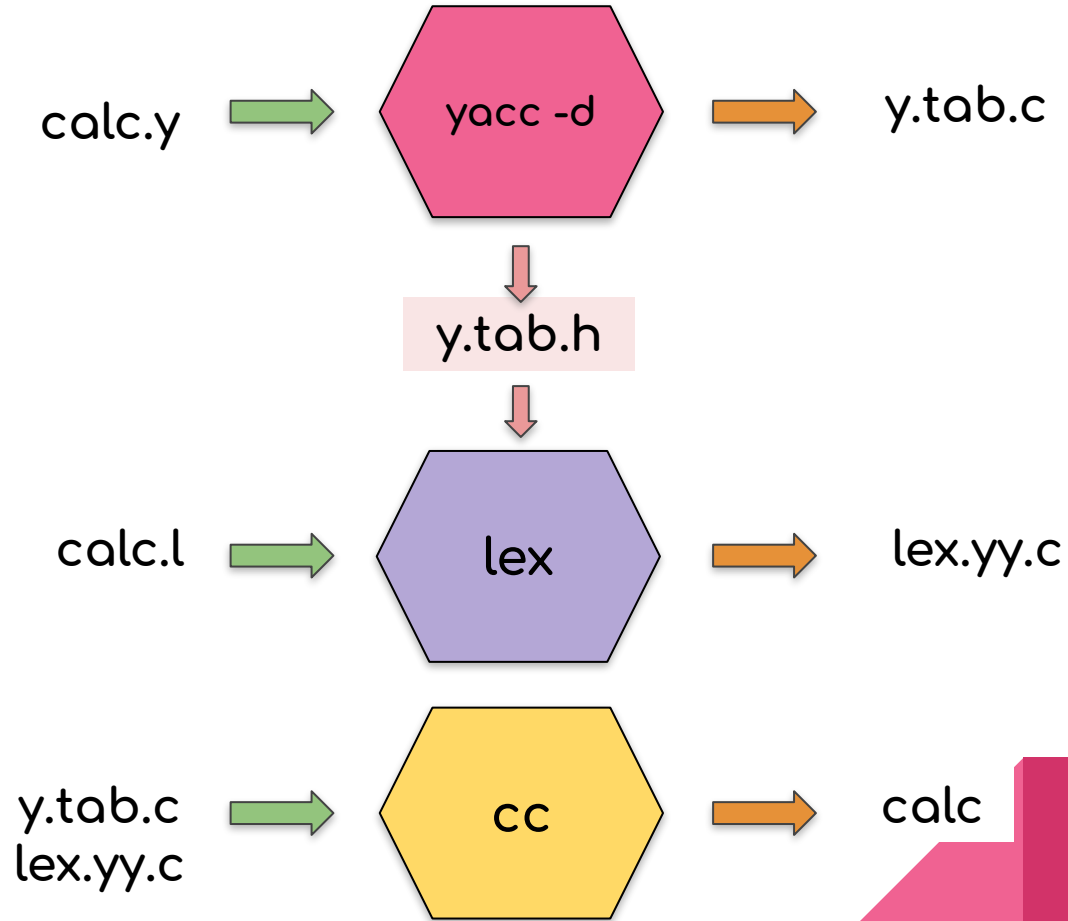


```
1047 /* Prevent warnings from -Wmissing-prototypes. */
1048 #ifdef YYPARSE_PARAM
1049 #if defined __STDC__ || defined __cplusplus
1050 int yyparse (void *YYPARSE_PARAM);
1051 #else
1052 int yyparse ();
1053 #endif
1054 #else /* ! YYPARSE_PARAM */
1055 #if defined __STDC__ || defined __cplusplus
1056 int yyparse (void);
1057 #else
1058 int yyparse ();
1059 #endif
1060 #endif /* ! YYPARSE_PARAM */
1061
1062 /* The lookahead symbol. */
1063 int yychar;
1064
1065 /* The semantic value of the lookahead symbol. */
1066 YYSTYPE yylval;
1067
1068 /* Number of syntax errors so far. */
1069 int yynerrs;
1070
1071
1072 /*-----
1073 | yyparse. |
1074 `-----*/
1075
1076 #ifdef YYPARSE_PARAM
1077 #if (defined __STDC__ || defined __C99_FUNC__ \
1078 || || defined __cplusplus || defined _MSC_VER)
1079 int
1080 yyparse (void *YYPARSE_PARAM)
1081 #else
1082 #endif
1083 int
1084 yyparse (YYPARSE_PARAM)
1085 | void *YYPARSE_PARAM;
1086 #endif
1087 #else /* ! YYPARSE_PARAM */
1088 #if (defined __STDC__ || defined __C99_FUNC__ \
1089 || || defined __cplusplus || defined _MSC_VER)
1090 int
1091 yyparse (void)
1092 #else
1093 int
1094 yyparse ()
1095 #endif
1096 #endif
1097 #endif
```

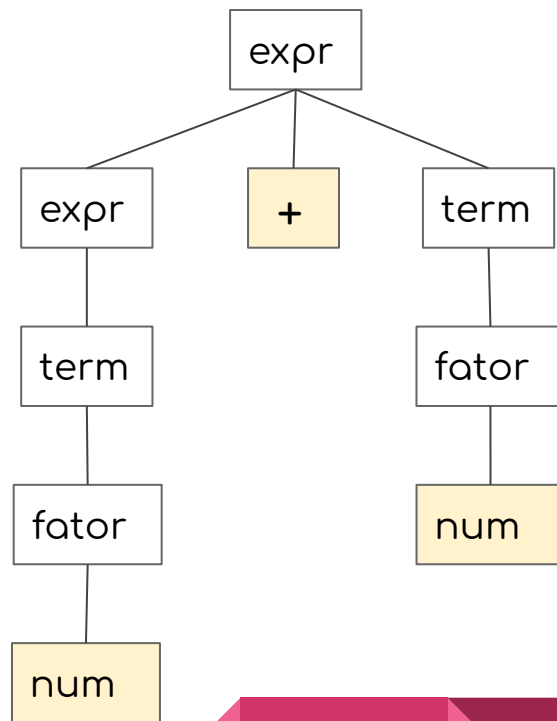
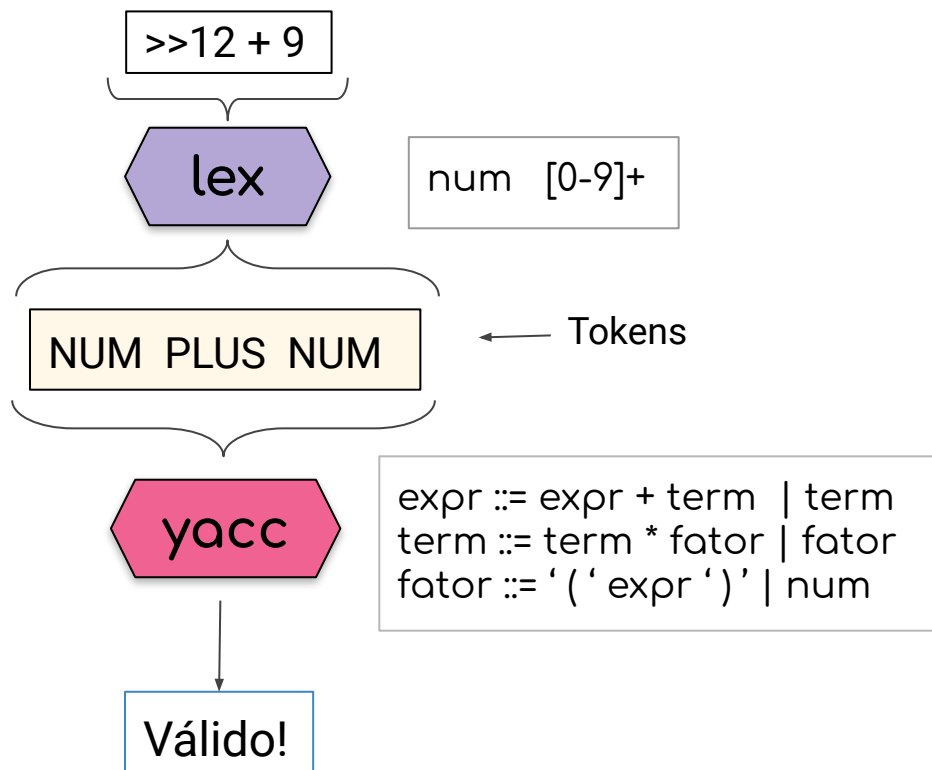
```
1589 /* Line 2067 of yacc.c */
1590 #line 30 "t1.y"
1591
1592
1593 void yyerror(const char *str)
1594 {
1595     fprintf(stderr,"error: %s\n",str);
1596 }
1597
1598 int yywrap()
1599 {
1600     return 1;
1601 }
1602
1603 main()
1604 {
1605     yyparse();
1606 }
1607
```

# Lex & Yacc

# Lex & Yacc



# Exemplo



# Exemplos



## Exemplo: Interação com um termostato

### Gramática

s	→ s comando   $\lambda$
comando	→ interruptor   tempo
interruptor	→ CALOR ESTADO
tempo	→ TEMPERATURA NUMERO

Terminais : CALOR , ESTADO, TEMPERATURA, NUMERO

Não-terminais : s , comando, interruptor, tempo

tokens

**Yacc**  
NUMERO  
CALOR  
ESTADO  
TEMPERATURA

**Lex**  
[0-9]+  
calor  
ligado | desligado  
temperatura

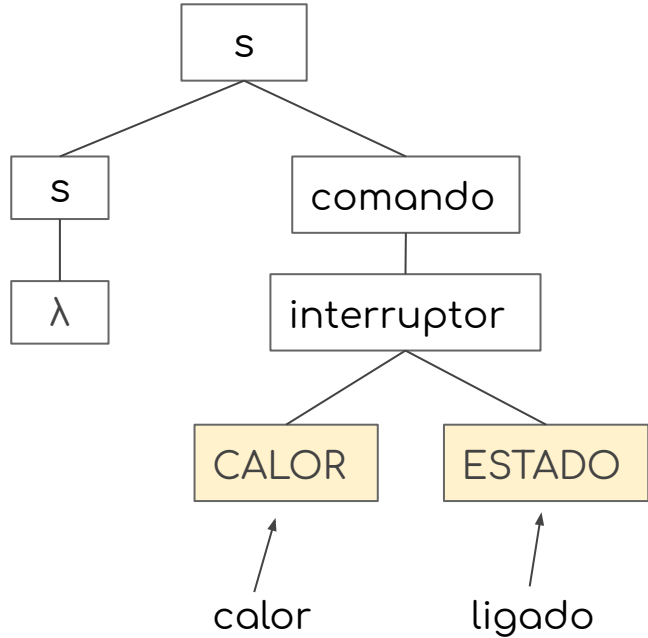
>> calor ligado

>> calor desligado

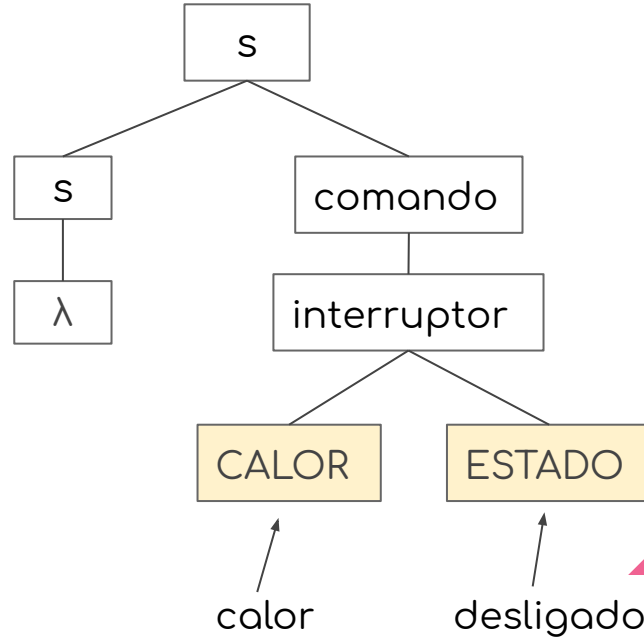
>> temperatura 35

## Exemplo: Interação com um termostato

calor ligado



calor desligado



**Yacc**

NUMERO

CALOR

ESTADO

TEMPERATURA

**Lex**

[0-9]+

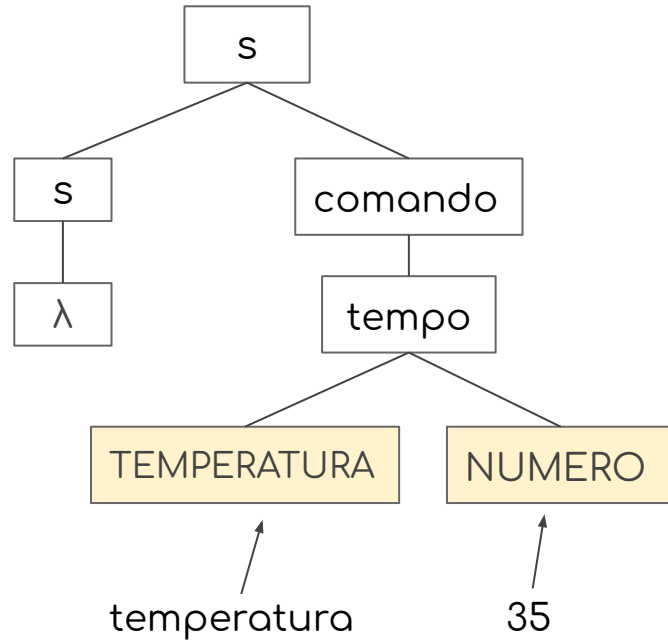
calor

ligado | desligado

temperatura

## Exemplo: Interação com um termostato

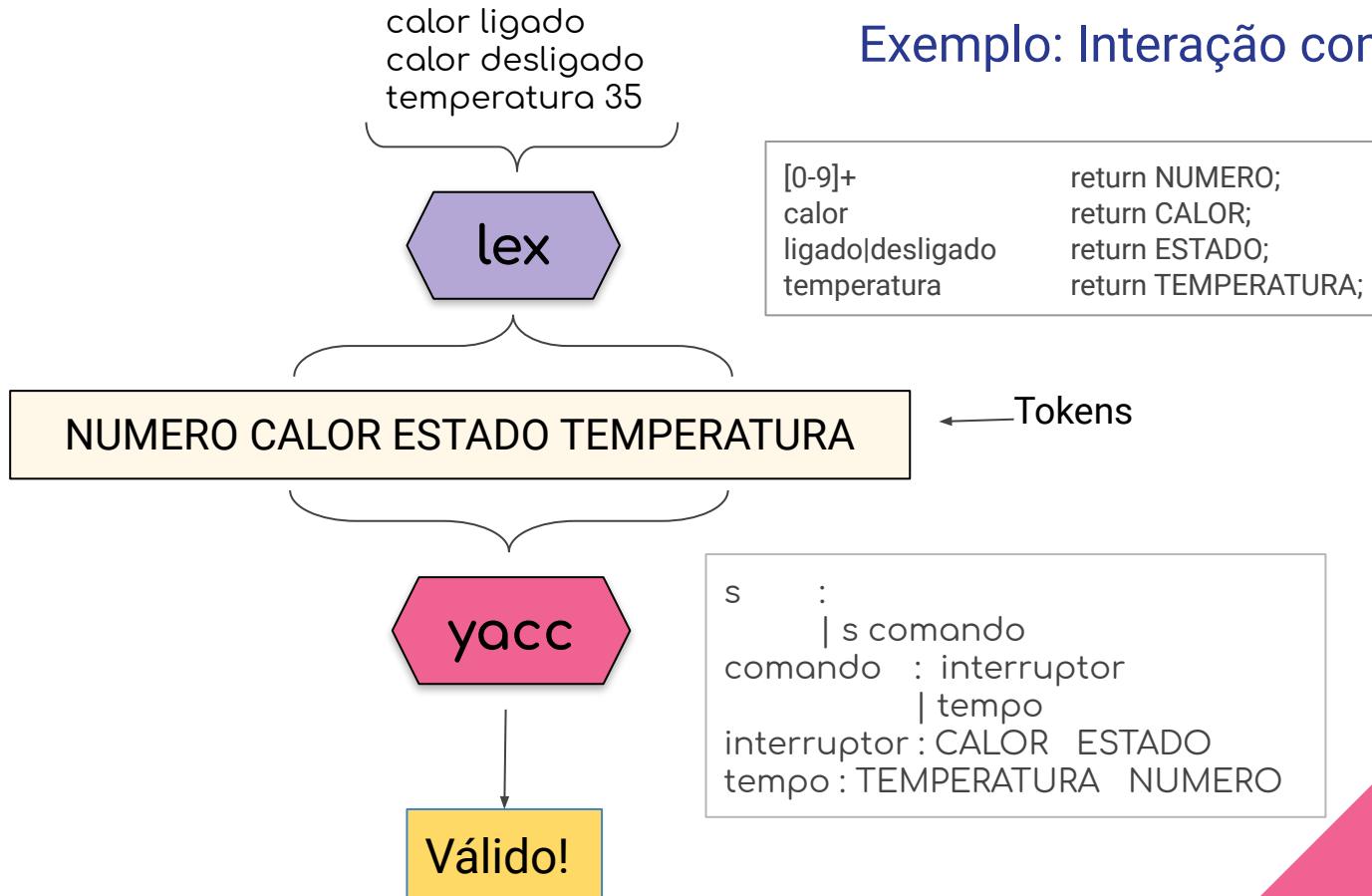
**temperatura 35**



**Yacc**  
NUMERO  
CALOR  
ESTADO  
TEMPERATURA

**Lex**  
[0-9]+  
calor  
ligado | desligado  
temperatura

## Exemplo: Interação com um termostato



## Exemplo: Interação com um termostato

```
1  %{
2  #include <stdio.h>
3  #include "y.tab.h"
4  %}
5
6  %%
7  [0-9]+  return NUMERO;
8  calor   return CALOR;
9  ligado|desligado  return ESTADO;
10 temperatura return TEMPERATURA;
11 \n ;
12 [ \t]+ ;
13
14 %%
```

```
% yacc -d t1.y
% lex t1.l
% cc lex.yy.c y.tab.c -o temperatura
% ./temperatura
```

```
t1.y x
4  %{
5
6  %token NUMERO CALOR ESTADO TEMPERATURA
7  %%
8
9  comandos: /* empty */
10           | comandos comando
11           ;
12  comando:
13           | interruptor
14           |
15           | tempo
16           ;
17  interruptor:
18           | CALOR ESTADO
19           | { printf("\tCalor ligado ou desligado\n"); }
20           |
21           ;
22
23  tempo:
24           | TEMPERATURA NUMERO
25           | { printf("\tNova temperatura definida\n"); }
26           ;
27
28
29
30 %%
31
32 void yyerror(const char *str)
33 {
34     fprintf(stderr,"error: %s\n",str);
35 }
36
37 int yywrap()
38 {
39     return 1;
40 }
41
42 main()
43 {
44     yyparse();
45 }
```

## Exemplo: Validação de uma expressão

```
1  %{
2  #include "y.tab.h"
3  %}
4  %%
5  [a-zA-Z] {return ID;}
6  [0-9]+ {return NUMBER;}
7  [ \t] {;}
8  \n {return 0;}
9  . {return yytext[0];}
10 %%
11
```

```
% yacc -d ex1.y
% lex ex1.l
% cc -o ex1 lex.yy.c y.tab.c
% ./ ex1
```

```
1  %{
2  #include<stdio.h>
3  %}
4  %token ID NUMBER
5  %left '+' '-'
6  %left '*' '/'
7
8  %%
9  stmt: expr
10 |
11 |
12 |
13 |
14 |
15 |
16 |
17 |
18 |
19 %%
20
21 void main()
22 {
23     printf("digite expressão : \n");
24     yyparse();
25     printf("expressão válida! \n");
26 }
27 yywrap(){ }
28 yyerror()
29 {
30     printf("Error\n");
31 }
32
```

## Exemplo: Duplo balanceamento

```
1  %{
2  #include "y.tab.h"
3  #include<math.h>
4  extern yyval;
5  %{
6  %%
7  [a] return A;
8  [b] return B;
9  [.|\\n] {return yytext[0];}
10 %%
```

```
% yacc -d ex2.y
% lex ex2.l
% cc -o ex2 lex.yy.c y.tab.c
% ./ ex2
```

```
1  %{
2  #include<stdio.h>
3  int valid=1;
4  %{
5
6  %token A B
7  %%
8  str: S'\\n' {return 0;}
9  S:A S B
10 |
11 ;
12 %%
13 main()
14 {
15     printf("Insira a string:\\n");
16     yyparse();
17     if(valid ==1)
18         printf("string válida\\n");
19 }
20 yywrap(){}
21 yyerror()
22 {
23     printf("Error\\n");
24 }
25
```

## Exemplo: String válida

```
1  %{
2  #include "y.tab.h"
3  %}
4
5  %%
6  [aA] {return A;}
7  [bB] {return B;}
8  \n {return NL;}
9  . {return yytext[0];}
10 %%
11
```

```
% yacc -d ex3.y
% lex ex3.l
% cc -o ex3 lex.yy.c y.tab.c
% ./ ex3
```

```
1  %{
2  #include<stdio.h>
3  #include<stdlib.h>
4  int valid=1;
5  %}
6
7  %token A B NL
8  %%
9  stmt: A A A A A S B NL {printf("string válida\n");
10 |      exit(0);}
11 ;
12 S: S A
13 |
14 ;
15 %%
16
17 main()
18 {
19     printf("Insira a string:\n");
20     yyparse();
21 }
22 yywrap(){}
23 yyerror()
24 {
25     printf("Error\n");
26 }
```



# Livraria Yacc

-**ly** ao final da linha de comando inclui uma livreria, cujo conteúdo varia entre implementações, mas sempre contém `main()` e `yyerror()`.

## `main()`

```
main(ac, av)
{
    yyparse();
    return 0;
}
```

Exemplo

## `yyerror()`

```
yyerror(char
*errmsg)
{
    fprintf(stderr
, "%s\n",
errmsg);
}
```

## Exemplo: Calculadora simples

```
ch3-01.l
1  %{
2  #include "y.tab.h"
3  extern int yylval;
4  %}
5
6  %%
7  [0-9]+ { yylval = atoi(yytext); return NUMBER; }
8  [ \t] ; /* ignore white space */
9  \n return 0; /* logical EOF */
10 . return yytext[0];
11 %%
12
```

```
% yacc -d ch3-01.y
% lex ch3-01.l
% cc -o ch3 lex.yy.c y.tab.c -ly
% ./ ch3
```

```
ch3-01.y
1  %{
2  #include<stdio.h>
3  %}
4  %token NAME NUMBER
5  %%
6  statement: NAME '=' expression
7  |         expression      { printf("= %d\n", $1); }
8  ;
9
10 expression: expression '+' NUMBER { $$ = $1 + $3; }
11 |          expression '-' NUMBER { $$ = $1 - $3; }
12 |          NUMBER                { $$ = $1; }
13 ;
14 %%
15 yyerror()
16 {
17     printf("Error\n");
18 }
19
```

# Nomes internos e mecanismos do Yacc

# Nomes internos e mecanismos de definições Yacc

Nome Interno Yacc	Significado/Usado
y.tab.c	Nome do arquivo de saída Yacc
y.tab.h	Arquivo de cabeçalho gerado pelo Yacc que contém as definições de marcas
yparse	Rotina de análise sintática Yacc
yylval	Valor da marca corrente na pilha
yyerror	Impressora de mensagens de erro definida pelo usuário e utilizada pelo Yacc
error	Pseudomarca de erro Yacc
yyerrork	Procedimento que reinicia o analisador depois de um erro
yychar	Contém a marca de verificação à frente que provocou um erro
YYSTYPE	Símbolo de pré-processador que define o tipo de valor da pilha de análise sintática
yydeug	Variável que, se ajustada pelo usuário para 1, leva à geração de informação em tempo de execução sobre as ações do analisador sintático

# Nomes internos e mecanismos de definições Yacc

Mecanismo de definição Yacc	Significado/Uso
<b>%token</b>	Define os símbolos pré-processadores de marcas
<b>%start</b>	Define o símbolo não-terminal inicial
<b>%union</b>	Define uma união <b>YYSTYPE</b> , com a permissão de valores de tipos distintos na pilha do analisador sintático
<b>%type</b>	Define o tipo diferenciado de união retornado por um símbolo
<b>%left %right %nonassoc</b>	Define a associatividade e precedência (por posição) dos operadores

## Outras ferramentas para análise sintática

Nome	Analizador léxico	Análise semântica	Linguagem de saída
Yacc	LALR(1)	Texto	Não
Bison	LALR(1), LR(1) E GR	Texto	Não
SableCC	LALR(1)	Texto	Não
COCO/R	LL(1)	Texto	Simples
AntLR 4	LL(*)	Texto	Sim

Nome	Analizador léxico	Análise semântica	Linguagem de saída
Yacc	Externo	Rotinas em linguagem própria	C
Bison	Externo	Rotinas em linguagem própria	C, C++ ou Java
SableCC	Junto à especificação sintática	Rotinas em linguagem Java	Java
COCO/R	Junto à especificação sintática	Rotinas em linguagem C/C++, Java, Pascal, C#, entre outras	C/C++, Java, PAscal, C#, entre outras
AntLR 4	Junto à especificação sintática	Rotinas em linguagem Java e C#	Java e C#