

# Compiladores

Análise sintática Bottom-Up  
YACC

# YACC

- Yet Another Compiler Compiler
- Produz um *parser* bottom-up para uma dada gramática
- Usado para produzir compiladores para Pascal, C, C++ entre outras
- Além disso, foi usado no desenvolvimento de:
  - bc - calculadora
  - eqn & pic
  - verificador de sintaxe SQL
  - Lex
- bison: Versão GNU

# YACC

Seqüência básica operacional

gram.y

Arquivo contendo gramática desejada no formato yacc

yacc

programa yacc

y.tab.c

programa fonte C criado pelo yacc

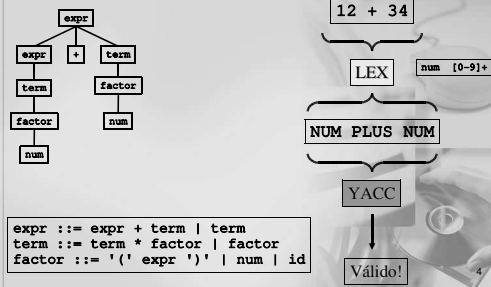
gcc / cc

Compilador C

a.out

Programa executável que faz a análise sintática da gramática descrita em parse.y

# Exemplo



# Formato do arquivo YACC

Definições

%%

Regras

%%

Código Suplementar

# Seções de Regras

- Normalmente escritas como segue:

```
expr : expr '+' term
      | term
      ;
term : term '*' factor
      | factor
      ;
factor : '(' expr ')'
        | ID
        | NUM
        ;
```

## Seção de Definições

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token ID NUM
%start expr
```

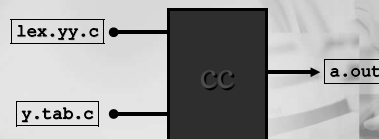
## Obs:

- lex/flex produz uma função yylex()
- yacc produz uma função yyparse()
- yyparse espera chamar uma yylex
- Como conseguir yylex?
  - Escrever sua própria!
  - Usar lex/flex

## Construindo yylex

```
int yylex()
{
    if(it's a num)
        return NUM;
    else if(it's an id)
        return ID;
    else if(parsing is done)
        return 0;
    else if(it's an error)
        return -1;
}
```

## lex & yacc



## Exemplo

- Suponha um arquivo lex scanner.l e um arquivo yacc chamado decl.y.
- Passo a serem feitos ...

```
yacc -d decl.y
lex scanner.l
gcc -c lex.yy.c y.tab.c
gcc -o parser lex.yy.o y.tab.o
-ll
```

Nota: scanner deve incluir na seção de definições:

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

## YACC

- As regras podem ser recursivas
- As regras não podem ser ambíguas\*
- Usa um parser bottom up Shift/Reduce -LALR(1)
  - Solicita um token
  - Empilha
  - Redução ?
    - Sim: reduz usando a regras correspondente
    - Não: pega outro token
- Yacc não pode olhar mais que um token de lookahead
- yacc -v gram.y gera a tabela de estados, em y.output

### Exemplo

scanner.l

```

scanner.l
%{
#include <stdio.h>
#include "parser.tab.h"
%}
id      [_a-zA-Z][_a-zA-Z0-9]*
wspc   [ \t\n]+
semi   [;]
comma  [,]
%%
int     { return INT; }
char   { return CHAR; }
float  { return FLOAT; }
{comma} { return COMMA; }
{semi}  { return SEMI; }
{id}    { return ID; }
{wspc}  {;}
  
```

### Exemplo

parser.y

```

parser.y
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token CHAR COMMA FLOAT
%token ID INT SEMI
%%
decl : type ID list
      { printf("Sucesso!\n"); };
list : COMMA ID list
      | SEMI
      ;
type : INT | CHAR | FLOAT
      ;
%%

parser.y (cont.)
extern FILE *yyin;

main() {
do {
  yyparse();
} while(!feof(yyin));
}

yyerror(char *s) {
/* Nada definido */
}
  
```

### Exemplo

Execução

```

$> bison -d parser.y
$> cat parser.tab.h
...
# define CHAR 257
# define COMMA 258
# define FLOAT 259
# define ID 260
# define INT 261
# define SEMI 262
...
$> flex parser.y
$> gcc parser.tab.c lex.yy.c -fl
  
```

**Gera:**

- parser.tab.c
- parser.tab.h

**Gera:**

- lex.yy.c

- ### A volta do atributo
- Cada símbolo tem um valor associado (atributo)
    - Poder uma quantidade numérica no caso de um número (42)
    - Pode ser um ponteiro para um string ("Hello, World!")
    - Pode ser um ponteiro para uma tabelas de símbolos no caso de uma variável
  - Quando usando lex junto com o Yacc , coloca-se o valor em yyval
    - Em situações complexas, yyval é uma union
  - Típico código lex:
 

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

### Definindo Valores

```

expr : expr '+' termo { $$ = $1 + $3; }
      | termo          { $$ = $1; }
      ;
termo : termo '*' fator { $$ = $1 * $3; }
        | fator        { $$ = $1; }
        ;
fator : '(' expr ')'   { $$ = $2; }
        | ID
        | NUM
        ;
  
```

- ### Resolução de Conflitos
1. Default:
    - Shift-Reduce: Escolhe o shift
    - Reduce-Reduce: Escolhe a regra que aparece primeiro
  2. Especificação de regras de precedências:
    - definida pelo usuário
- ```

%token INTEGER
%right '='
%left '+'
%left '*' '/'
%%
  
```
- precedência

```

E : E '+' E |
    E '-' E |
    E '*' E |
    E '/' E |
    E '=' E |
    '(' E ')' |
    INTEGER;
  
```

## Exemplos

```
%token INTEGER
%right '='
%left '+' '.'
%left '*' '/'
%%
```

5+3+2 ↔ (5+3)+2

5\*3\*2 ↔ (5\*3)\*2

5+3\*2 ↔ 5+(3\*2)

a=5+3 ↔ a=(5+3)

a=b=c=3+5 ↔ a=(b=(c=(3+5)))

19

## Resolução de Conflitos

Especificação de regras de precedências:

- mudança explícita de precedência (%prec)

```
%token INTEGER
%right '='
%left '+' '.'
%left '*' '/'
%nonassoc UMINUS
%%
```

### Precedência

```
E : E '+' E |
    E '-' E |
    E '*' E |
    E '/' E |
    E '=' E |
    '(' E ')' |
    -E %prec UMINUS |
    INTEGER;
```

## Conceitos adicionais

- Yacc permite que símbolos tenham múltiplos tipos de valores

```
%union {
    double dval;
    int vblno;
}
```

- Tabelas de símbolos são usadas para guardar informações sobre tipos:

- nomes
- tipos
- valores,
- etc

- TS podem ser implementadas de diversas formas.

21

## Resumo

- Escrever um compilador é difícil e requer tempo e esforço
- Scanners* e *parsers* podem ser construídos por métodos automáticos

Regras Léxicas ✓

Gramática ✓

Semântica

Compilador

Scanner

Parser

Gerador

Código

22

## Bibliografia

- Livro do dragão: Seção 4.9
- Bison Project Homepage [www.gnu.org/software/bison](http://www.gnu.org/software/bison)
- Thomas Niemann: A Compact Guide to Lex & Yacc
- C. Donnelly, R. Stallman: Bison - the YACC-compatible Parser Generator

23