Compiladores
Análise Sintática
Analisador Sintático Descendente Recursivo *TINY*

Prof. Dr. Luiz Eduardo G. Martins
(adaptado por Profa Dra Ana Carolina Lorena)
UNIFESP

BNF *TINY*

programa → *decl-seqüência*
decl-seqüência → *decl-seqüência* **;** *declaração* | *declaração*
declaração → *cond-decl* | *repet-decl* | *atrib-decl* | *leit-decl* | *escr-decl*
cond-decl → **if** *exp* **then** *decl-seqüência* **end**
| **if** *exp* **then** *decl-seqüência* **else** *decl-seqüência* **end**
repet-decl → **repeat** *decl-seqüência* **until** *exp*
atrib-decl → **identificador** **:=** *exp*
leit-decl → **read identificador**
escr-decl → **write** *exp*
exp → *exp-simples comp-op exp-simples* | *exp-simples*
comp-op → **<** | **=**
exp-simples → *exp-simples soma termo* | *termo*
soma → **+** | **-**
termo → *termo mult fator* | *fator*
mult → * | /
fator → **(***exp***)** | **número** | **identificador**

Figura 3.6 Gramática da linguagem TINY em BNF.

EBNF *TINY*

programa → *decl-seqüência*
decl-seqüência → *declaração* { **;** , *declaração* }
declaração → *if-decl* | *repeat-decl* | *atribuição-decl* | *read-decl* | *write-decl*
if-decl → **if** *exp* **then** *decl-seqüência* [**else** *decl-seqüência*] **end**
repeat-decl → **repeat** *decl-seqüência* **until** *exp*
atribuição-decl → **identificador** **:=** *exp*
read-decl → **read** **identificador**
write-decl → **write** *exp*
exp → *simples-exp* [*comparação-op simples-exp*]
comparação-op → **<** | **=**
simples-exp → *termo* { *soma termo* }
soma → **+** | **-**
termo → *fator* { *mult fator* }
mult → ***** | **/**
fator → **(** *exp* **)** | **número** | **identificador**

Figura 4.9 Gramática da linguagem TINY em EBNF.

Analisador Sintático

- 11 funções, equivalentes aos não-terminais: decl-sequência, declaração, if-decl, repeat-decl, atribuição-decl, read-decl, write-decl, exp, simples-exp, termo e fator

Analisador Sintático

- Exemplo: declaração

declaração → if-decl | repeat-decl | atribuição-decl | read-decl | write-decl

```
NóÁrvore * declaração(void){
    NóÁrvore *t = NULL;
    switch (token){
        case IF: t = if_decl(); break;
        case REPEAT: t = repeat_decl(); break;
        case ID: t = atribuição_decl(); break;
        case READ: t = read_decl(); break;
        case WRITE: t = write_decl(); break;
        default: erroSintático("Token inesperado → ");
                 imprimeToken(token,tokenString);
                 token = getToken();
                 break;
    }
    return t;
}
```

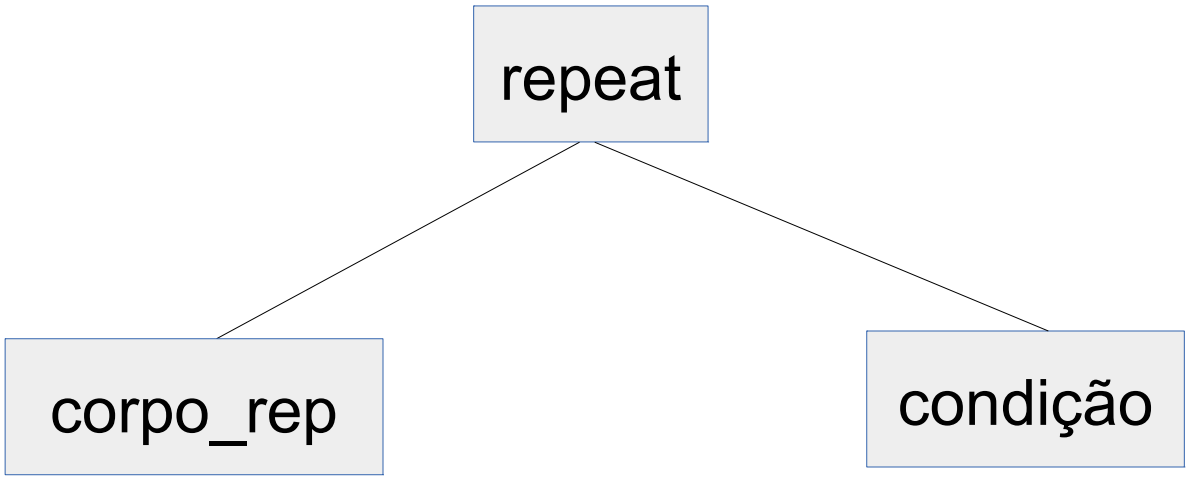
```
static void casa(TokenType esperado){
    if(token == esperado)
        token = getToken();
    else{
        erroSintático("Token
            inesperado → ");
imprimeToken(token,tokenString);
    }
}
```

Analisador Sintático

- Exemplo: repeat-decl

repeat-decl → repeat decl-sequência until exp

```
NóÁrvore * repeat-decl(void){
    NóÁrvore *t = novoNóDecl(Repeat);
    casa(REPEAT);
    if (t!=NULL) t->filho[0] = decl-sequência;
    casa(UNTIL);
    if (t!=NULL) t->filho[1] = exp();
    return t;
}
```

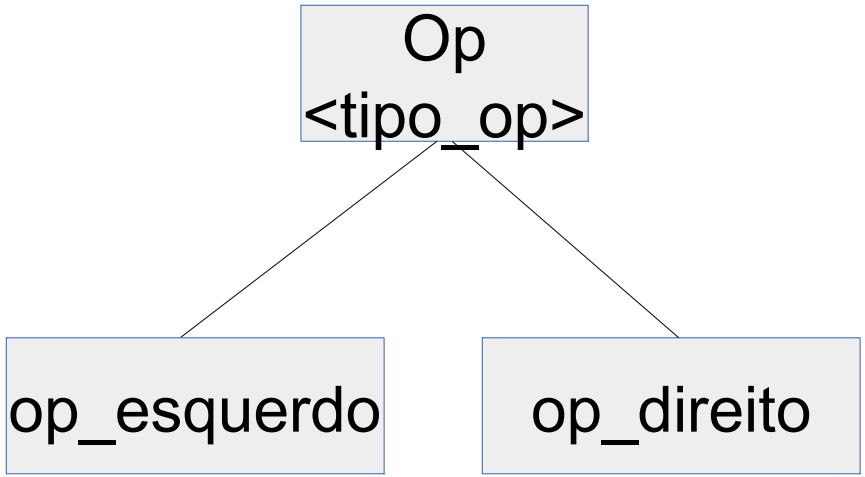


Analisador Sintático

- Exemplo: exp

exp → simples-exp [comparação-op simples-exp]

```
NóÁrvore * exp(void){
    NóÁrvore *t = simples-exp();
    if((token == LT) || (token == EQ)){
        NóÁrvore p = novoNóExp(Op);
        if(p!=NULL){
            p->filho[0] = t;
            p-> atrib.op = token;
            t = p;
        }
        casa(token);
        if(t!=NULL)
            t->filho[1] = simples-exp();
    }
    return t;
}
```



Analisador Sintático

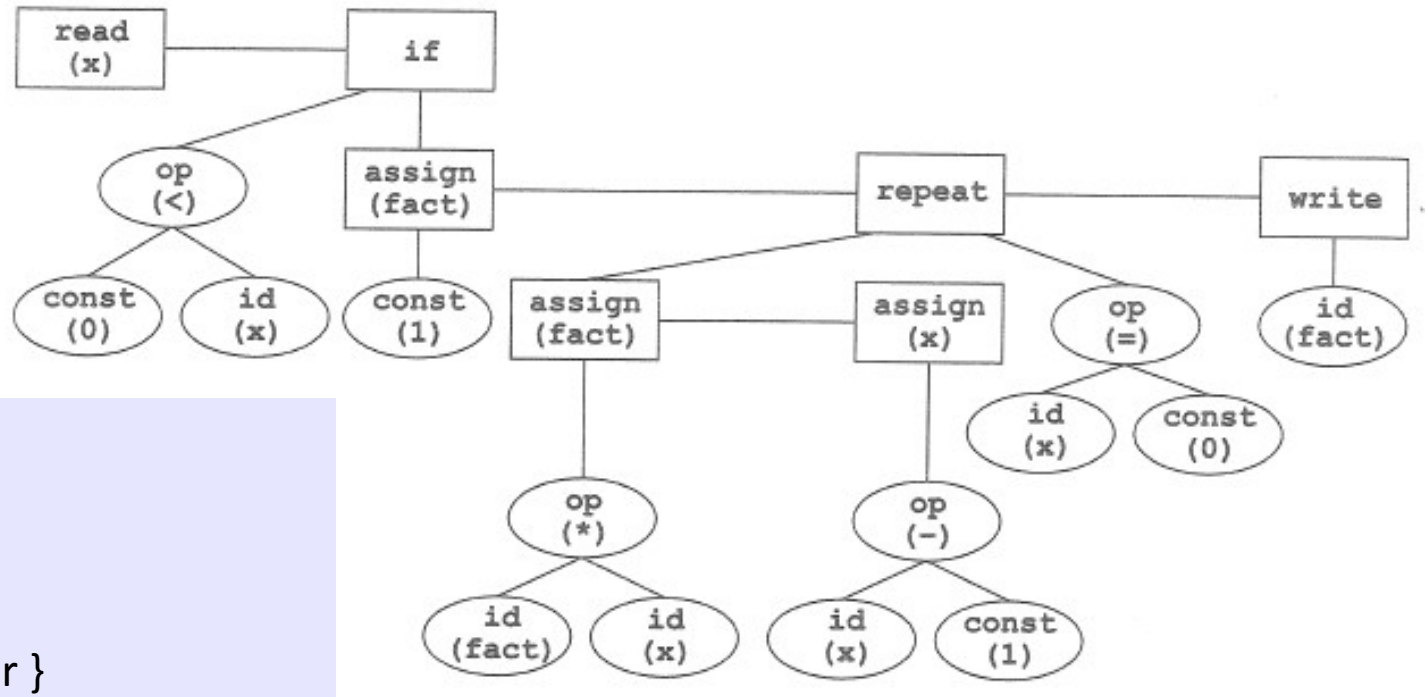
- Exemplo: simples-exp

simples-exp → termo {soma termo}

```
NóÁrvore * simples-exp(void){
    NóÁrvore *t = termo();
    while((token == PLUS) || (token == MINUS)){
        NóÁrvore p = novoNóExp(Op);
        if(p!=NULL){
            p->filho[0] = t;
            p-> atrib.op = token;
            t = p;
            casa(token);
            t->filho[1] = termo();
    }
    return t;
}
```

```
// função básica do Parser
NóÁrvore * parse(void){
    NóÁrvore *t;
    token = getToken();
    t = decl-sequência();
    return t;
}
```

A Linguagem *TINY*



ura 3.9 Árvore sintática para o programa TINY da Figura 3.8.

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact  { output factorial of x }
end
```

A Linguagem *TINY*

- Bibliografia consultada

Capítulo 4 e Apêndice B: LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004