

Compiladores

Aula 04 – Análise Sintática (Implementação)

Edirlei Soares de Lima

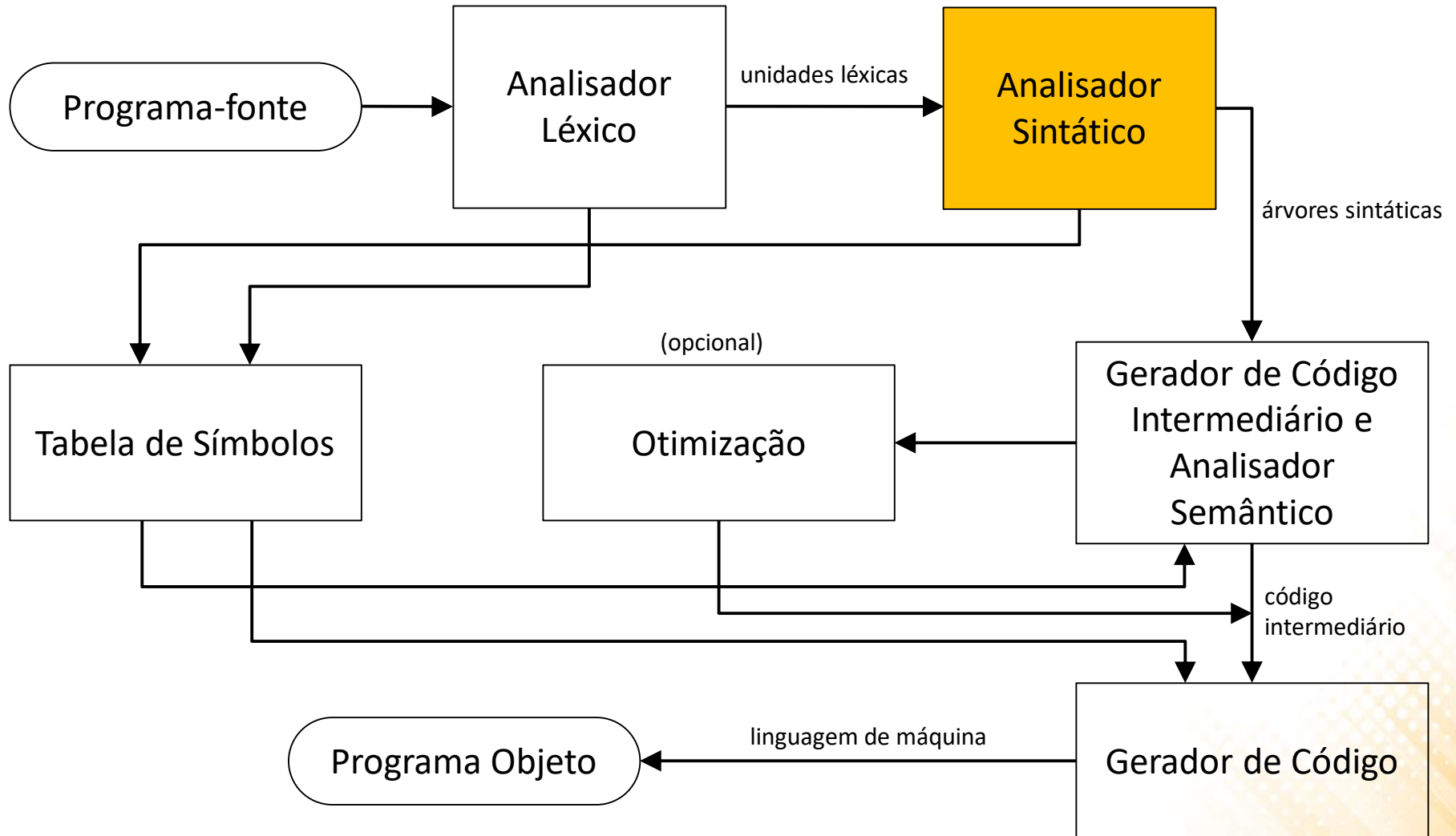
<edirlei.lima@universidadeeuropeia.pt>



Análise Sintática

- A maioria dos compiladores separam a tarefa da **análise sintática** em duas partes distintas:
 - Análise Léxica;
 - Análise Sintática;
- O **analisador léxico** trata as construções de pequena escala da linguagem (nomes, literais numérico, símbolos...);
- O **analisador sintático** trata as construções de larga escala da linguagem (instruções, expressões, unidades do programa...);

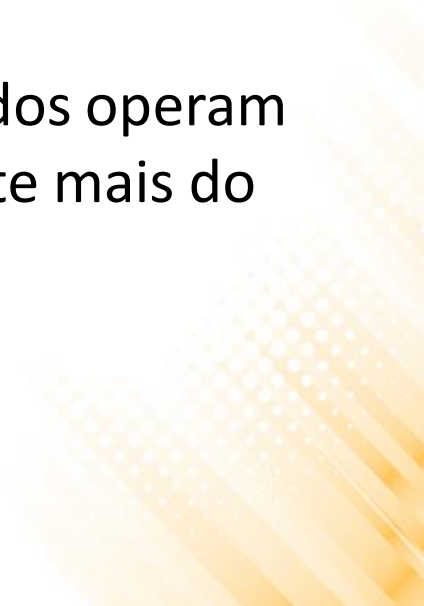
Processo de Compilação



Análise Sintática

- Os analisadores sintáticos constroem árvores sintáticas (**parse trees**) para os programas dados.
 - Em alguns casos, a árvore sintática é construída implicitamente (apenas o resultado de percorrer a árvore é gerado);
- Objetivos da Análise Sintática:
 - Verificar o programa de entrada para determinar se ele está sintaticamente correto;
 - Produzir árvores sintáticas (parse trees).

Análise Sintática

- Os analisadores sintáticos são classificados de acordo com a direção na qual eles constroem a árvore de análise:
 - **Cima-Baixo (top-down)**: a árvore é construída da raiz para as folhas;
 - **Baixo-cima (bottom-up)**: a árvore é construída das folhas para a raiz.
 - Todos os algoritmos de análise comumente utilizados operam sob a obrigação de que eles jamais olharam à frente mais do que um símbolo no programa de entrada.
- 


Analísadores Cima-Baixo

- Realizam uma derivação **mais à esquerda** (*leftmost derivations*).
- Dada uma forma sentencial, que é parte de uma derivação mais à esquerda, a tarefa do analisador é encontrar a próxima forma sentencial nessa derivação.
- Um **analisador descendente recursivo** é uma versão codificada de um analisador sintático baseado diretamente na descrição BNF da linguagem.
 - Alternativa para a recursão: tabela de análise para implementar as regras da BNF.

Analísadores Baixo-Cima

- Realizam uma derivação **mais à direita** (*rightmost derivations*).
- Dada uma forma sentencial à direita α , o analisador deve determinar qual sub-cadeia de α é o lado direito de alguma regra na gramática que deve ser reduzida para produzir a forma sentencial dada na derivação mais à direita.

Análise Descendente Recursiva

- Um **analisador descendente recursivo** consiste de uma coleção de funções (muitas das quais são recursivas).
 - A implementação destas funções descreve naturalmente as regras gramaticais da BNF.
 - O analisador descendente recursivo tem uma função para cada não-terminal da gramática.
- 

Análise Descendente Recursiva (Implementação)

- Gramática:

```
<expr> → <termo> + <termo>  
      | <termo> - <termo>  
      | <termo>
```

```
<termo> → <fator> * <fator>  
        | <fator> / <fator>  
        | <fator>
```

```
<fator> → ident  
        | numero  
        | ( <expr> )
```

Análise Descendente Recursiva (Implementação)

```
...

/*
<expr> -> <termo> + <termo>
        | <termo> - <termo>
*/

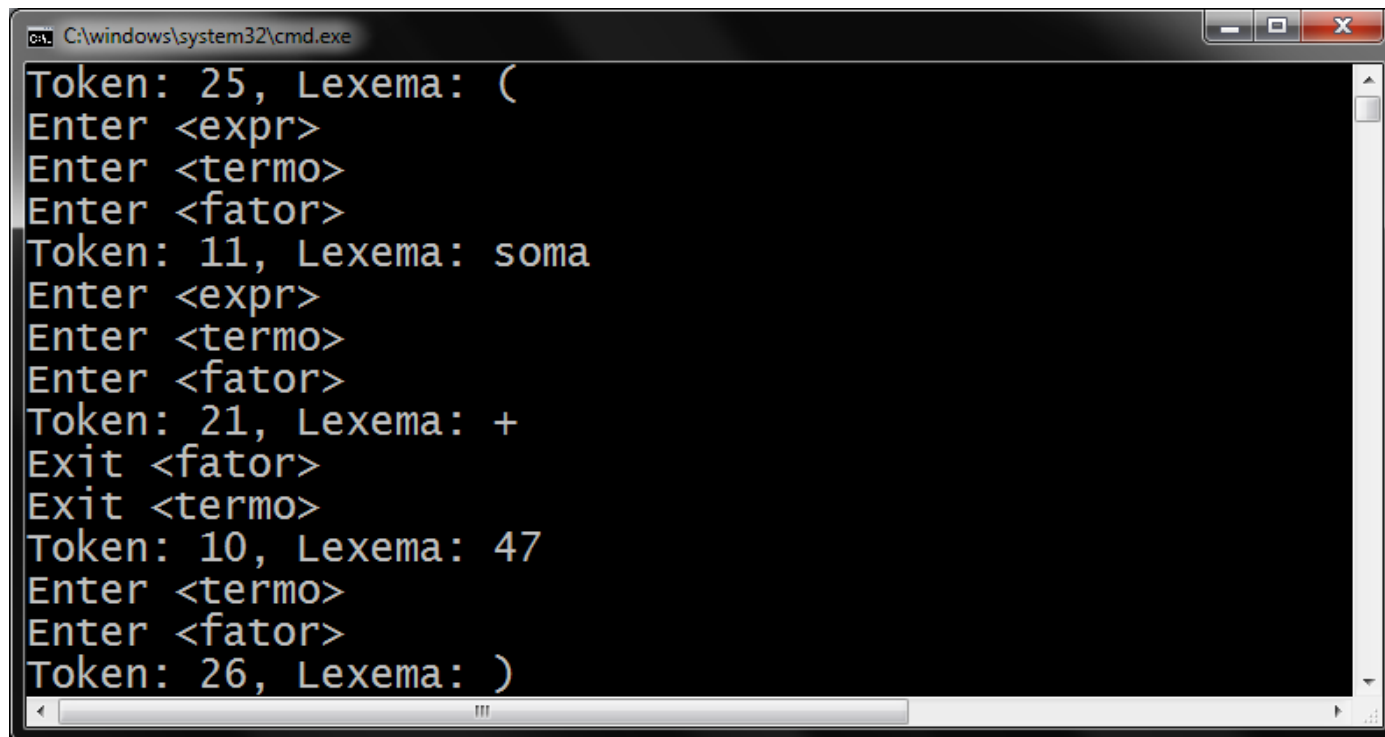
int expr(FILE *code_file, int next_token, NextChar *next_char)
{
    printf("Enter <expr>\n");
    next_token = termo(code_file, next_token, next_char);
    while (next_token == ADD_OP || next_token == SUB_OP)
    {
        next_token = lex(code_file, next_char);
        next_token = termo(code_file, next_token, next_char);
    }
    printf("Exit <expr>\n");
    return next_token;
}

...
```

Análise Descendente Recursiva (Implementação)

- Entrada:

```
(soma + 47) / total
```



```
C:\windows\system32\cmd.exe
Token: 25, Lexema: (
Enter <expr>
Enter <termo>
Enter <fator>
Token: 11, Lexema: soma
Enter <expr>
Enter <termo>
Enter <fator>
Token: 21, Lexema: +
Exit <fator>
Exit <termo>
Token: 10, Lexema: 47
Enter <termo>
Enter <fator>
Token: 26, Lexema: )
```

Análise Descendente Recursiva (Implementação)

- Saída:

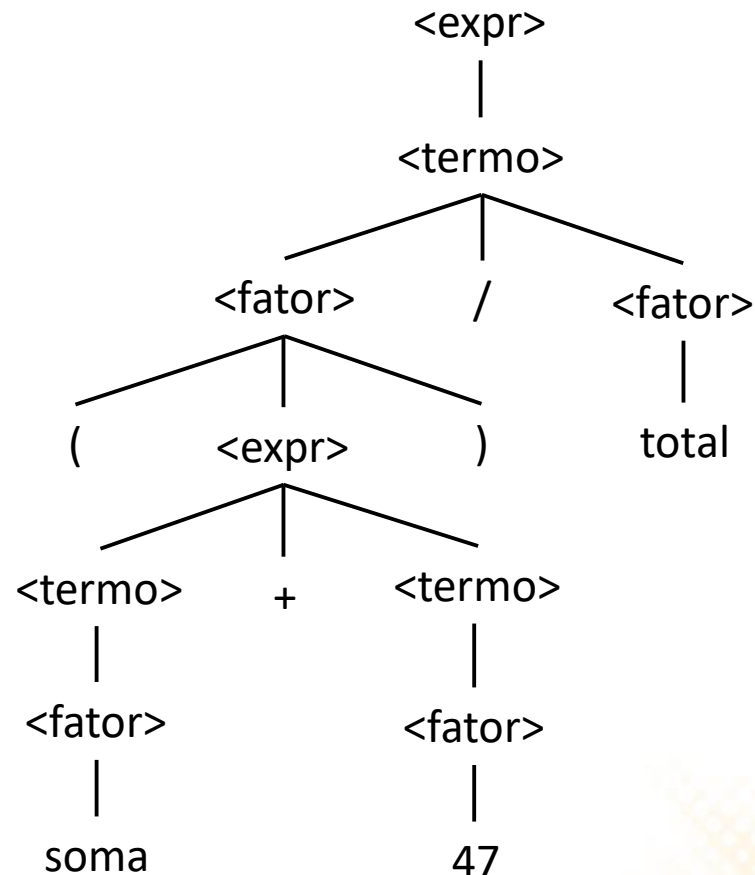
```
Token: 25, Lexema: (  
Enter <expr>  
Enter <termo>  
Enter <fator>  
Token: 11, Lexema: soma  
Enter <expr>  
Enter <termo>  
Enter <fator>  
Token: 21, Lexema: +  
Exit <fator>  
Exit <termo>  
Token: 10, Lexema: 47  
Enter <termo>  
Enter <fator>  
...
```

```
...  
Token: 26, Lexema: )  
Exit <fator>  
Exit <termo>  
Exit <expr>  
Token: 24, Lexema: /  
Exit <fator>  
Token: 11, Lexema: total  
Enter <fator>  
Token: -1, Lexema: EOF  
Exit <fator>  
Exit <termo>  
Exit <expr>
```

Análise Descendente Recursiva (Implementação)

- Saída:

```
Token: 25, Lexema: (  
Enter <expr>: 25  
Enter <termo>: 25  
Enter <fator>: 25  
Token: 11, Lexema: soma  
Enter <expr>: 11  
Enter <termo>: 11  
Enter <fator>: 11  
Token: 21, Lexema: +  
Exit <fator>: 11  
Exit <termo>: 21  
Token: 10, Lexema: 47  
Enter <termo>: 10  
Enter <fator>: 10  
Token: 26, Lexema: )  
Exit <fator>: 10  
Exit <termo>: 26  
Exit <expr>: 11  
Token: 24, Lexema: /  
Exit <fator>: 25  
Token: 11, Lexema: total  
Enter <fator>: 11  
Token: -1, Lexema: EOF  
Exit <fator>: 11  
Exit <termo>: -1  
Exit <expr>: 25
```



Exercício 01

1) Continue a implementação do analisador léxico, integrando-o com o analisador sintático implementado em aula. Em seguida, acrescente as seguintes características à gramática reconhecida pelo analisador sintático:

a) Dar suporte a expressões de atribuição. Exemplos:

```
resultado = (soma + 87) / total
```

b) Suportar programas compostos por mais de uma expressão de atribuição (separadas por ponto e vírgula). Exemplo:

```
a = 8 + 42;  
b = a * 8;
```

c) Exiba mensagens de erro adequadas quando o programa de entrada violar as regras da gramática.

d) Apresente a gramática suportada pelo analisador sintático desenvolvido em notação BNF.

Leitura Complementar

- Aho, A. V., Lam, M. S., Jeffrey, R. S.
Compiladores: Princípios, Técnicas e Ferramentas. 2ª edição, Pearson, 2007.
ISBN: 978-8588639249.
 - Capítulo 3: Lexical Analysis
 - Capítulo 4: Syntax Analysis

- Sebesta, R. W. **Conceitos de Linguagens de Programação.** 9ª edição Editora Bookman, 2011. ISBN: 978-8577807918.
 - Capítulo 3: Descrevendo a Sintaxe e a Semântica
 - Capítulo 4: Análise Léxica e Sintática

