Conceitos de Linguagens de Programação

Aula 06 – Análise Sintática (Implementação)

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

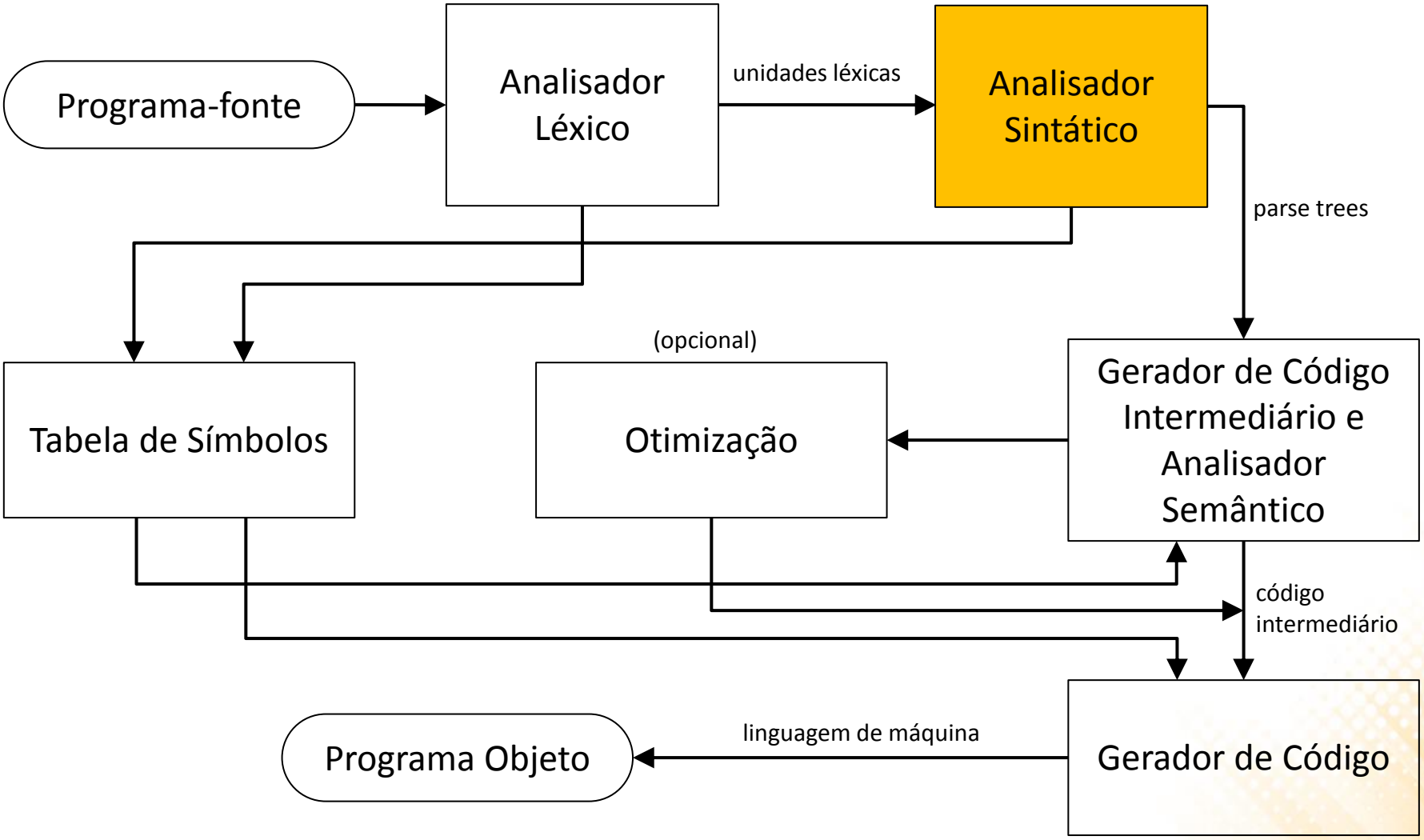
Análise Sintática

- 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 de análise (**parse trees**) para os programas dados.
  - Em alguns casos, a parse tree 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 de análise (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.

Analisadores 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.

Analisadores 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 gramatica 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 gramatica.

Análise Descendente Recursiva (Implementação)

- Gramatica:

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

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

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

Análise Descendente Recursiva (Implementação)

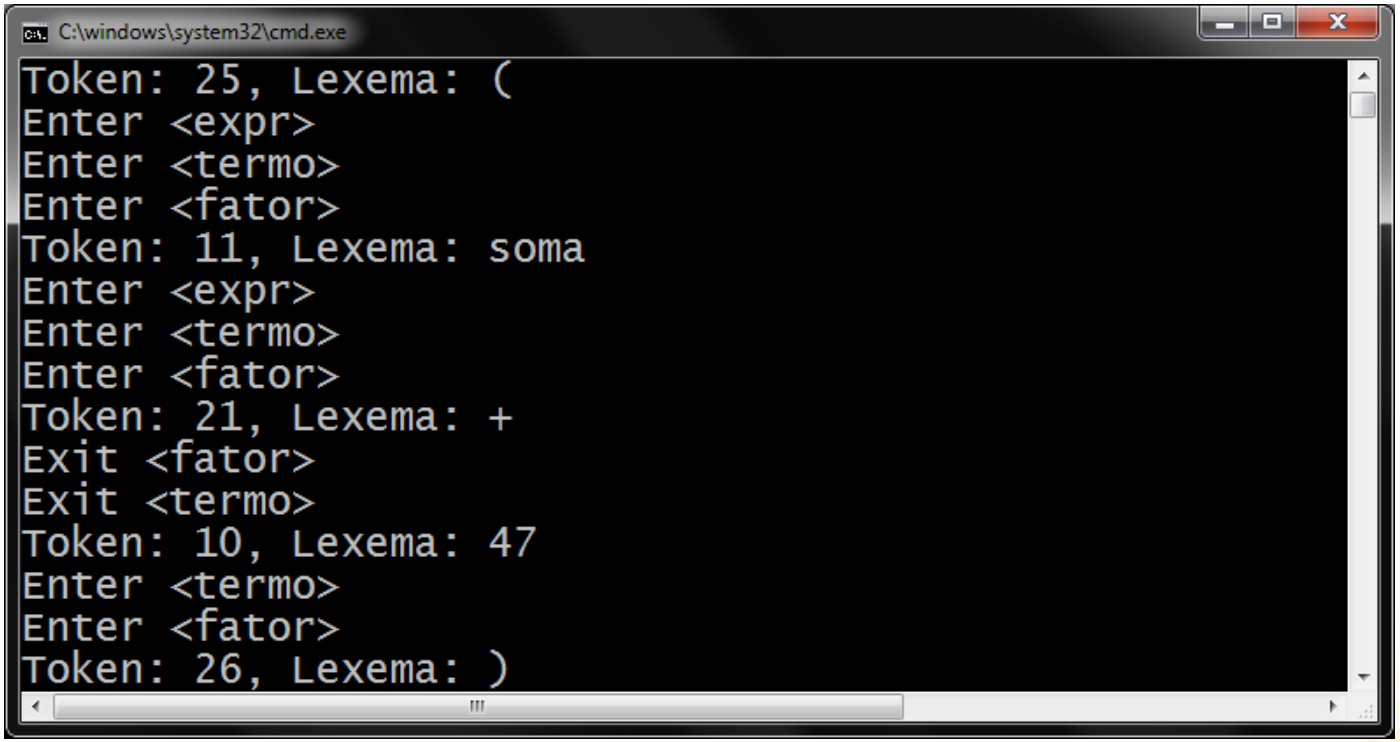
```
...

/*
<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;
}

...
```

http://www.inf.puc-rio.br/~elima/clp/projeto_compilador_v2.zip

Análise Descendente Recursiva (Implementação)

- Entrada:

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



Análise Descendente Recursiva (Implementação)

- Saída:

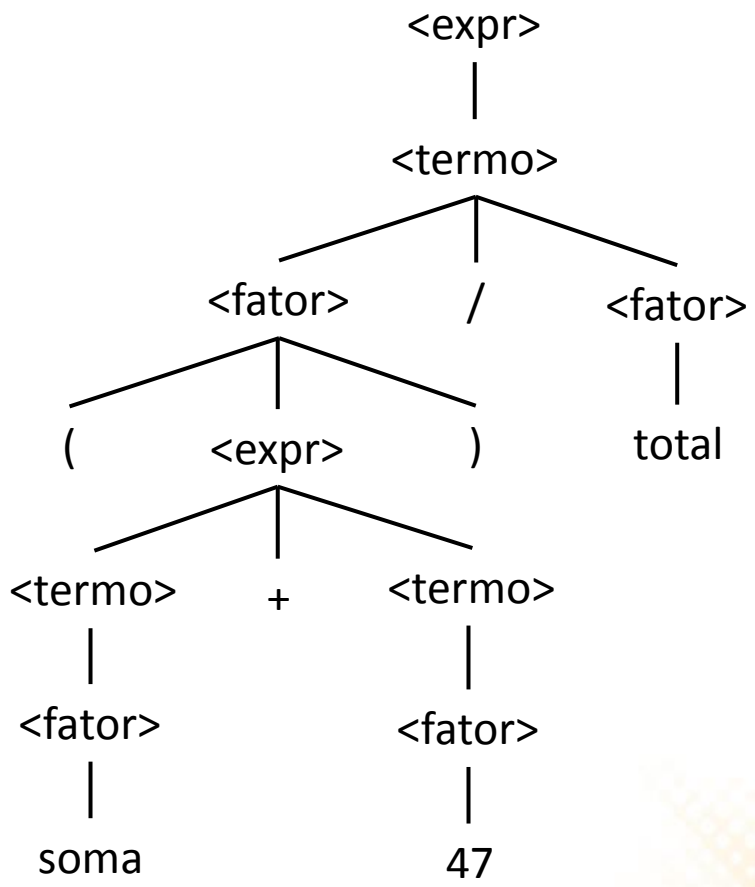
```
Token: 25, Lexema: (
Enter <expr>
Enter <termo>
Enter <fator>
Token: 11, Lexema: soma
Enter <expr>
Enter <termo>
Enter <fator>
Token: 21, Lexema: +
Exit <fator>
Exit <termo>
Token: 10, Lexema: 47
Enter <termo>
Enter <fator>
...
```

```
...
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
```



Leitura Complementar

- Sebesta, Robert W. **Conceitos de Linguagens de Programação**. Editora Bookman, 2011.

- **Capítulo 4: Análise Léxica e Sintática**