

TopScript

Vinicios Henrique Wentz¹, Rafael Henrique Tibola¹, Felipe Divensi¹

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Brasil, 4232 - Independência, Medianeira - PR, 85884-000

{vinicioswentz, tibola, felipee}@alunos.utfpr.edu.br

Abstract. *aaaa*

Resumo. *aaa*

1. Compilador

Linguagens de programação são notações para a descrição de computações para pessoas e máquinas. Hoje, no mundo em que vivemos as pessoas são totalmente dependentes das linguagens de programação, pois, todos os softwares os quais os computadores rodam são escritos em alguma linguagem de programação. Mas para que isso seja possível, precisa-se de alguma forma que os computadores entendam essas linguagens de programação, software que faz esta tradução é chamado de compilador [Alfred V. Aho 2007].

Pode-se definir compiladores como programas que fazem a tradução de outros programas os quais foram escritos em alguma linguagem e os preparam para serem executados. Para fazer esta transformação o compilador deve entender [K. Cooper 2012].

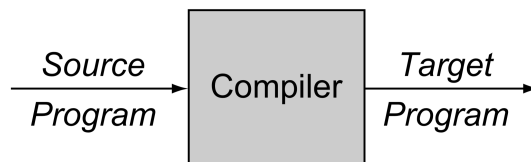


Figure 1. Compilador.

A estrutura de um compilador é composta por duas partes a parte de análise e a de síntese. A fase de análise quebra o programa fonte em peças constituintes e impõe uma estrutura gramatical. Essa estrutura é utilizada para a criação da representação intermediária do programa fonte. Caso a fase de análise detecte que o programa está sintaticamente mal formado ou não está semanticamente claro, essa fase deve fornecer informações informativas para que o usuário tome as devidas providências. A análise também coleta algumas informações importantes e as armazena em uma estrutura chamada tabela de símbolos, a qual é passada com a representação intermediária para a análise sintática. A análise sintática é responsável pela construção do programa alvo desejado. Ela usa a representação intermediária e a tabela de símbolos para tal processo [Alfred V. Aho 2007]. Essas fases são chamadas de *front end* e *back end* respectivamente. A fase será abordada com maior detalhes nas Seções 2, 4 e 3 e a síntese na Seção.

2. Analise Léxica

A tarefa principal do analisador léxico na primeira fase da compilação é de ler o programa fonte, agrupa-los em lexemes e produzir uma sequência de *tokens* para cada lexeme no programa fonte. Muitas vezes o analisador léxico interage com a tabela de símbolos. Quando o analisador léxico processa um lexeme que pertence a um identificador ele precisa adicioná-lo na tabela de símbolos. Em alguns casos informações sobre algum identificador deve ser lido da tabela de símbolos pelo analisador léxico para determinar o *token* apropriado que deverá ser passado para o *parser*. [Alfred V. Aho 2007]. Essa interação é demonstrada na Figura 2.

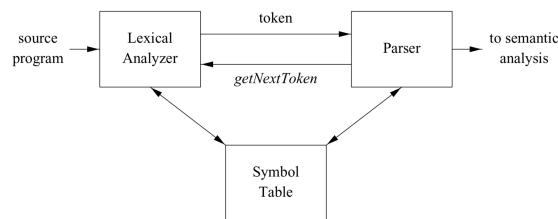


Figure 2. Iterações entre o analisador léxico e o *parser*.

3. Analise Sintática

O analisador sintático conhecido também como *parser* obtém *strings* dos tokens que do analisador léxico, ele verifica se os nomes dos tokens podem ser gerados pela gramática da linguagem fonte. É esperado que o *parser* informe qualquer erro de sintaxe. Para programas bem formados, o *parser* faz a construção da árvore sintática e passa para as próximas fases do compilador [Alfred V. Aho 2007].

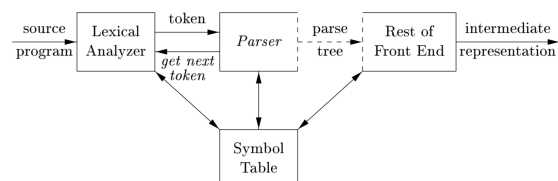


Figure 3. Iterações entre o analisador léxico e o *parser*.

Existem três tipos de analisadores de gramática, são eles, universal, *top-down* e *bottom-up*. O analisador universal pode fazer a análise de qualquer gramática, porém esse método é ineficiente em produção.

Os métodos mais comuns para fazer a análise sintática são os analisadores *top-down* e *bottom-up*. Os seus nomes são bem sugestivos, o analisador *top-down* faz a construção da árvore sintática do topo para baixo, ou seja, começa a construção a partir da raiz e termina nas folhas, o analisador *bottom-up* começa essa construção nas folhas indo em direção até a raiz. Em ambos os casos as análises são feitas da esquerda para direita um símbolo por vez.

Os analisadores *top-down* e *bottom-up* trabalham somente com uma sub-classe de linguagens, porém as sub-classes LL e LR são expressivas o bastante para a descrição da grande parte das construções sintáticas em linguagens de programação modernas.

4. Analise Semântica

5. Documentação

References

Alfred V. Aho, Monica S. Lam, R. S. J. D. U. (2007). *Compilers: principles, techniques, and tools*. Pearson/Addison Wesley, 2 edition.

K. Cooper, L. T. (2012). *Engineering a Compiler*. Morgan Kaufman, 2 edition.