UniRuy & Área 1 | Wyden

PROGRAMA DE ENGENHARIA DA COMPUTAÇÃO

TEORIA DE COMPILADORES

DALVAN BATISTA DOS SANTOS

Teoria de Compiladores: Análise Léxica

Salvador - Bahia - Brasil

2022

DALVAN BATISTA DOS SANTOS

Teoria de Compiladores: Análise Léxica

Trabalho Acadêmico elaborado junto ao programa de Engenharia UniRuy & Área 1 | Wyden, como requisito para obtenção de nota parcial da AV2 na disciplina Teoria de Compiladores no curso de Graduação em Engenharia da Computação, que tem como objetivo consolidar os tópicos do plano de ensino da disciplina.

Orientador: Prof. MSc. Heleno Cardoso

Salvador - Bahia - Brasil
2022

da Tal, Aluno Fulano

Teoria de Compiladores: Resenha / Mapa Mental / Perguntas

– Aluno Fulano de Tal. Salvador, 2022.
18 f. : il.

Trabalho Acadêmico apresentado ao Curso de Ciência da Computação, UniRuy & Área 1 | Wyden, como requisito para obtenção de aprovação na disciplina Teoria de Compiladores.

Prof. MSc. Heleno Cardoso da S. Filho.

1. Resenha
2. Mapa Mental
3. Perguntas/Respostas (Mínimo de 03 – Máximo de 05)
4. Conclusão

I. da Silva Filho, Heleno Cardoso II. UniRuy & Área 1 | Wyden. III. Trabalho Acadêmico

CDD:XXX

TERMO DE APROVAÇÃO

DALVAN BATISTA DOS SANTOS

TEORIA DE COMPILADORES: ANÁLISE LÉXICA

Trabalho Acadêmico aprovado como requisito para obtenção de nota parcial da AV2 na disciplina Teoria de Compiladores, UniRuy & Área 1 | Wyden, pela seguinte banca examinadora:

BANCA EXAMINADORA

Prof^{o}. MSc^{o}. Heleno Cardoso
Wyden

Salvador, 08 de Outubro de 2022

Dedico este trabalho acadêmico a todos que contribuíram direta ou indiretamente com minha formação acadêmica. Aos meus pais que sempre acreditaram em meu potencial.

Agradecimentos

Primeiramente agradeço a Deus. Ele, sabe de todas as coisas, e através da sua infinita misericórdia, se fez presente em todos os momentos dessa trajetória, concedendo-me forças e saúde para continuar perseverante na minha caminhada. Aos meus amigos que sempre estiveram ao meu lado, aos meus pais, tios e tias de forma direta ou indireta contribuiram para meu amadurecimento, tanto pessoal, quanto profissional.

"A educação tem raízes amargas, mas os seus frutos são docesA ciência é mais que um corpo de conhecimento, é uma forma de pensar, uma forma cética de interrogar o universo, com pleno conhecimento da falibilidade humana".

Carl Sagan.

Resumo

A temática trabalhada na disciplina Teoria de Compiladores foi enriquecedora para minha formação acadêmica.

Palavras-chaves: Compiladores, Turing, Formalismo, Linguagem, arquitetura, grafos, algoritmos, autômato.

Abstract

The theme worked on in the Compiler Theory discipline was enriching for my academic training.

Keywords: Compilers, Turing, Formalism, Language, architecture, graphs, algorithms, automaton.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

Sumário

[1] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

1 Análise Léxica

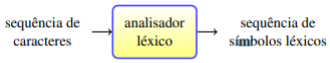
1.1 Introdução

A análise léxica é a primeira etapa do processo de compilação e seu objetivo é dividir o código fonte em símbolos léxicos, preparado-o para a análise sintática. Neste processo pode-se destacar três atividades como fundamentais:

1.2 Análise Léxica

(MALAQUIAS, 2014) Símbolos léxicos, ou tokens, são as palavras e sinais de pontuação utilizados para expressar a estrutura de um programa em um texto. O analisador léxico, ou scanner, é um módulo do compilador que tem como entrada uma sequência de caracteres (texto do programa), produzindo na saída uma sequência de símbolos léxicos. O analisador léxico atua como uma interface entre o texto de entrada e o analisador sintático. O analisador léxico, ou scanner, é um módulo do compilador que tem como entrada uma sequência de caracteres (texto do programa), produzindo na saída uma sequência de símbolos léxicos. O analisador léxico atua como uma interface entre o texto de entrada e o analisador sintático

Figura 1 – Analisador Lexico



Fonte: UFOP DECOM

(MALAQUIAS, 2014) As formas mais comuns de símbolos léxicos são: Identificadores: palavras utilizadas para nomear entidades do programa, como variáveis, funções, métodos, classes, módulos, etc. literais : sequência de caracteres que representa uma constante, como um número inteiro, um número em ponto flutuante, um caractere, uma string, um valor verdade (verdadeiro ou falso), etc. palavras chaves: palavras usadas para expressar estruturas da linguagem, como comandos condicionais, comandos de repetição, etc. Geralmente são reservadas, não podendo ser utilizadas como identificadores. sinais de pontuação: sequências de caracteres que auxiliam na construção das estruturas

do programa, como por exemplo servindo de separador de expressões em uma lista de expressões

2 Tabela de simbolos

(ERINALDO, 2011) São estruturas de dados usadas pelos compiladores para conter informações sobre as construções do programa fonte. As informações são coletadas de modo incremental pelas fases de análise e usadas pelas fases de síntese para gerar o código objeto. As entradas na tabela de símbolos são criadas e usadas durante a fase de análise pelo analisador léxico, analisador sintático e pelo analisador semântico. As entradas na tabela de símbolos contêm informações sobre um identificador (nome ou lexema, tipo, endereço na memória, etc.)

As tabelas de símbolos normalmente precisam dar suporte a múltiplas declarações do mesmo identificador dentro de um programa. O escopo de uma declaração é a parte de um programa à qual a declaração se aplica. Implementaremos os escopos definindo uma tabela de símbolos separada para cada um deles

Figura 2 – Tabela de Simbolos

```
Program  →                  {top = null;}
              Block
    Block    →   '{'        {saved = top;
                              Top = new Env(top);
                              Print("{");}
              Decls stmts'}'      {top = saved;
                                   Print("}");}
    Decls    →   decls decl
         |   ∈
    Decl →   type id;       {s = new Symbol;
                              s.type = type.lexeme
                              top.put(id.lexeme, s);}
    stmts    →   stmts stmt
         |   ∈
    Stmt →   block
         |   factor;        {print(";");}
    Factor   →   id         {s = top.get(id.lexeme);
                              Print(id.lexeme);
                              Print(":");}
                              Print(s.type);
```

Fonte: Erinaldo

3 implementação de um analisador léxico

De acordo com (JR, 2011) Existem diversas implementações para gerar analisadores léxicos para diferentes linguagens de programação. Dentre eles temos Flex, jflex, flex++ e csfelx.

Um analisador léxico pode ser implementado ou através do AEF que o representa ou através de sua gramática regular. Embora o resultado de ambas as técnicas deva ser o mesmo, estas formas tem diferenças fundamentais quanto à implementação. Se partirmos do AEF para a implementação do scanner, o trabalho de manipulação é maior do que se partirmos de sua gramática regular, que é um processo que pode ser feito de forma automática.

Implementação via AEF:

A implementação do scanner através de um AEF é bastante simples. Basicamente o que é feito é usar um par de comandos do tipo CASE (switch em C) para representar os estados e os caracteres sendo lidos pelo autômato. Assim, por exemplo, um primeiro CASE faz sua decisão a partir do estado atual do autômato (que está guardado numa variável atualizada a cada transição), enquanto que o segundo CASE tomará a sua decisão a partir do caracter que acaba de ser lido. A ordem em que estes dois cases são realizados pode ser alterada segundo a conveniência do processo de busca, isto é, caso seja mais conveniente primeiro selecionar pelo caracter de entrada e depois pelo estado, podemos fazê-lo assim sem maiores complicações. A decisão sobre qual ordem a ser adotada é deixada então a cargo do implementador. Como pode ser visto, uma vez definido o AEF temos que a sua implementação é trivial se o mesmo puder ser expresso por um conjunto reduzido de estados e de caracteres de entrada. Entretanto isso não é o que ocorre num compilador, fazendo com que a implementação do AEF se torne trabalhosa devido ao grande número de estados e do tamanho do alfabeto existentes numa gramática regular real. A solução para este problema é o uso de geradores automáticos de scanners, tais como o lex ou o flex presentes no ambiente UNIX.

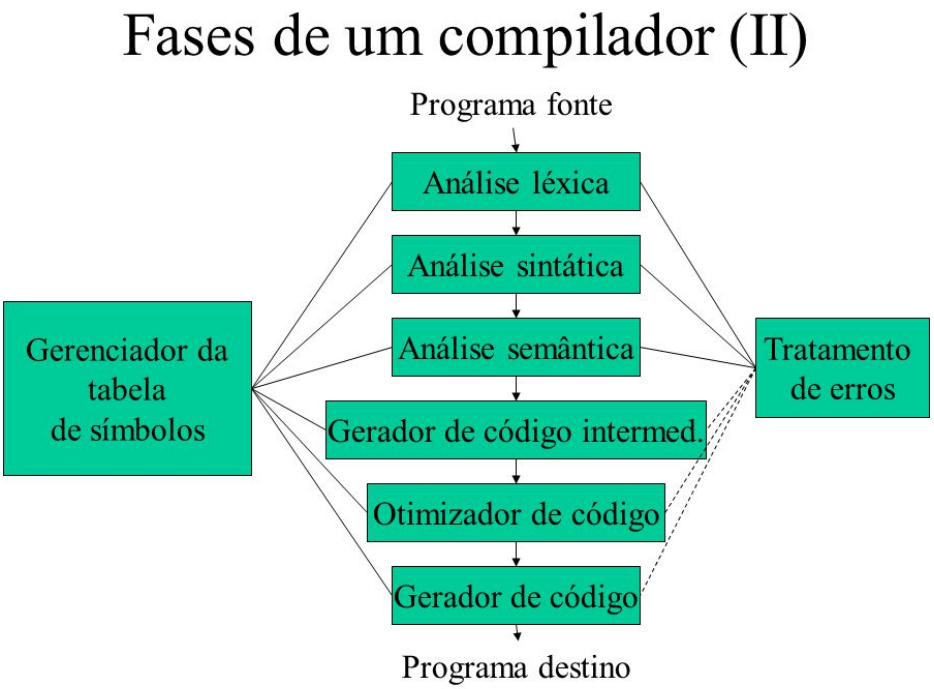
Implementação via gramática:

O scanner pode ser implementado usando-se um gerador automático, tal como o lex. Para tanto, usamos gramáticas reescritas na forma de expressões regulares, que devidamente arranjadas segundo a sintaxe do lex podem gerar de forma automática o código do scanner. Esta abordagem tem enormes vantagens com relação à implementação

direta a partir do autômato pois, como a gramática (e as expressões regulares que a representam) tem que ser definida de qualquer maneira, uma vez que é a partir dela que sabemos quais são os strings permitidos para a dada linguagem, torna-se desinteressante tranformarmos a gramática num autômato que teria que ser implementado manualmente, se sabemos que existem geradores automáticos de scanners a partir das produções da própria gramática. Entretanto, em ambos casos existem problemas que precisam ser resolvidos para que a implementação seja funcional, tanto quanto à sua correção como quanto à sua eficiência. A seção seguinte faz um tratamento destes problemas.

4 fases do compilador.

Figura 3 – Fases de um compildor



Fonte: https://slideplayer.com.br/slide/2262897/

5 Perguntas e Respostas - Mínimo de 2 e Máximo de 5

1) O que são simbolos Léxicos?

ou tokens, são as palavras e sinais de pontuação utilizados para expressar a estrutura de um programa em um texto.

2) o que faz um compilador?

Um compilador é basicamente um programa que traduz um texto de programa escrito em alguma linguagem denominada linguagem fonte (normalmente de alto nível) para uma outra linguagem denominada linguagem objeto (normalmente de baixo nível).

6 Conclusão

Conclui-se que analise lexica é uma parte essencial da disciplica de compiladores, pois atraves dela que vemos como funciona a implementação de um compilador, que faz a maquina interpretar a linguagem humana.

Referências[1]

ERINALDO, B. *Tabelas de Símbolos*. 2011. Citado na página 15.

JR, B. A. M. *Implementação do analisador léxico*. 2011. Citado na página 16.

MALAQUIAS, B. J. R. *Análise Léxica*. 2014. Citado na página 13.

[1] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.