UniRuy & Área 1 | Wyden

PROGRAMA DE ENGENHARIA DA COMPUTAÇÃO

TEORIA DE COMPILADORES

JOÃO VICTOR DE DEUS MARTINS

Teoria de Compiladores: ANÁLISE LÉXICA (FASES DO COMPILADOR)

Salvador - Bahia - Brasil

2022

JOÃO VICTOR DE DEUS MARTINS

Teoria de Compiladores: ANÁLISE LÉXICA (FASES DO COMPILADOR)

Trabalho Acadêmico elaborado junto ao programa de Engenharia UniRuy & Área 1 | Wyden, como requisito para obtenção de nota parcial da AV1 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

TERMO DE APROVAÇÃO

JOÃO VICTOR DE DEUS MARTINS

TEORIA DE COMPILADORES: ANÁLISE LÉXICA (FASES DO COMPILADOR)

Trabalho Acadêmico aprovado como requisito para obtenção de nota parcial da AV1 na disciplina Teoria de Compiladores, UniRuy & Área 1 | Wyden, pela seguinte banca examinadora:

BANCA EXAMINADORA

Profº. MScº. Heleno Cardoso
Wyden

Salvador, 06 de Novembro de 2022

Dedico este trabalho acadêmico a todos que contribuíram direta ou indiretamente com minha formação acadêmica.

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.

E a todos aqueles que contribuíram direta ou indiretamente para a minha formação acadêmica.

"A educação tem raízes amargas, mas os seus frutos são doces".

Aristóteles.

Resumo

Ao definir o léxico de uma linguagem de programação, um alfabeto define o que constitui palavras válidas. As linguagens de programação normalmente usam Unicode ou ASCII como alfabeto; é por isso que alguns tokens consistem em caracteres consecutivos. A estrutura do léxico de um idioma também determina quantos caracteres consecutivos constituem um token. Todos os caracteres que seguem um caractere inválido encerram imediatamente o token.

Palavras-chaves: Símbolos léxicos, Gramática regulares, Expressões regulares, Autômatos finitos, Tabela de símbolos, Analisador léxico.

Abstract

When defining the lexicon of a programming language, an alphabet defines what constitutes valid words. Programming languages typically use Unicode or ASCII as the alphabet; that's why some tokens consist of consecutive characters. The lexicon structure of a language also determines how many consecutive characters make up a token. All characters following an invalid character immediately terminate the token.

Keywords: Lexical symbols, Regular grammar, Regular expressions, Finite automata, Symbol table, lexical Analyzer.

Lista de abreviaturas e siglas

Token	- Ficha ou Símbolo.
Swift	- Linguagem oficiais para desenvolvimento nas plataformas iOS.
String	- Cadeias de caracteres que armazenam dados textuais.

Sumário

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

1 ANÁLISE LÉXICA (FASES DO COMPILADOR)

1.1 Introdução

Esta resenha tem como objetivo apresentar os conceitos mais fundamentais em fases do compilador. Este texto examina os meandros das propriedades, abordando sobre a linguística. As linguagens de programação têm muitos conceitos integrados que as tornam mais fáceis de entender e usar. O uso desses conceitos permite que os programadores aprendam novas linguagens mais rapidamente e aproveitem ao máximo as construções de programação com cada nova linguagem.

1.2 Execução/Método

Resenha desenvolvida através de analises de artigos científicos apurados.

1.2.1 Repositório de Pesquisa

Google Acadêmico

1.2.2 String de Busca por Repositório

"Lexical Analysis"

1.2.3 Artigos Selecionados

LEXICAL ANALYSIS

Analyzing explanations of substitution reactions using lexical analysis and logistic regression techniques

Applications of Finite Automata in Lexical Analysis and as a Ticket Vending Machine – A Review

1.2.4 Resenha dos Artigos Selecionados

A estrutura lexical da linguagem descreve quais strings são palavras válidas por definição. Essas palavras funcionam como os blocos fundamentais que suportam o restante dos capítulos de linguagem. Um token consiste em qualquer um dos termos listados: palavra-chave, pontuação, literal, operador ou identificador. Os tokens são criados usando a substring mais longa possível de um arquivo de texto de entrada. Isso normalmente ocorre quando os caracteres estão presentes em um arquivo de origem do Swift. Muitas linguagens de programação têm uma gramática léxica que define as strings que elas suportam. Por exemplo, a gramática de muitas linguagens de programação diz que literais de string começam com um único caractere e continuam até que uma correspondência seja encontrada. Ele também afirma que as strings identificadoras devem ser alfanuméricas e não podem incluir zeros à esquerda. Os literais inteiros devem consistir apenas em dígitos, sem sublinhados ou zeros à esquerda. Além disso, certas palavras-chave podem ter o mesmo formato de strings identificadoras; geralmente são palavras alfabéticas. No entanto, essas palavras são categorizadas separadamente dos identificadores e marcadas de forma diferente de outros tokens na mesma categoria lexical. As gramáticas formais definem a estrutura de uma linguagem usando strings estruturadas. A maioria das discussões em torno de gramáticas formais relaciona-se a linguagens livres de contexto – chamadas de spanners no singular – mas elas também têm o mesmo conceito central: expressões regulares. Conforme aprendemos, as expressões regulares são gramáticas formais que operam em todas as linguagens. A sintaxe de uma linguagem de programação define as regras que determinam quais cadeias de caracteres são consideradas programas legais.

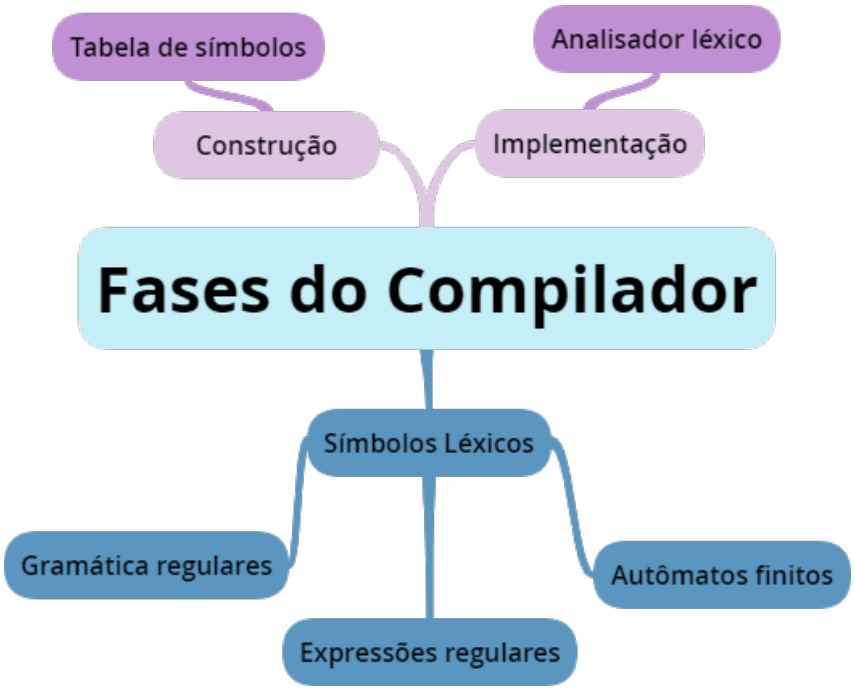
As linguagens de programação usam dois componentes em sua sintaxe. A estrutura lexical de uma linguagem determina como o comprimento do texto de um programa de computador é dividido em lexemas individuais. Lexemas são as "palavras"em um programa de computador que são separadas por espaços. A sintaxe de uma linguagem é sua estrutura gramatical. As palavras em um idioma são agrupadas em estruturas sintáticas menores chamadas sentenças por meio de um processo semelhante à gramática humana. O texto do programa que quebra a sintaxe do idioma ou as regras de uso de palavras não é aceitável como um programa legal. Programas que seguem as regras de uma linguagem específica não significam necessariamente que transmitem qualquer informação significativa. Por exemplo, falantes de inglês podem criar frases como esta. Para entender a estrutura

lexical de uma linguagem de programação, é útil estudar a linguagem regular de uma linguagem. Expressões regulares e autômatos finitos são formalismos importantes para entender ao estudar linguagens regulares. Compreender esses dois conceitos nos ajudará a criar definições precisas para linguagens regulares. Expressões regulares criam linguagens que seguem um padrão regular. Uma máquina que entende uma linguagem específica é chamada de autômato finito. Uma expressão regular pode gerar todas as strings possíveis em uma linguagem regular. A especificação de uma expressão regular é necessária ao criar um novo objeto Regex. Qualquer letra do alfabeto é uma expressão regular que gera, que pode ser representada por um símbolo. O caractere especial é uma expressão regular que representa uma lista vazia. Diretrizes para correspondência de expressões regulares são necessárias. Expressões regulares mais expansivas são criadas a partir de regras básicas. Colocar um parêntese em torno de um conjunto de palavras esclarece seu significado. Concatenação Uma expressão regular x e y sempre geram as mesmas expressões regulares Lx e Ly. Usando xy, você pode obter qualquer expressão que contenha todas as strings que xey concatenar para formar. alternância A relação x|y é um pai direto do idioma Lx|Ly. Também representa a união de Lx e Ly, que é o equivalente a Lx Ly. Se xey são expressões regulares que descrevem expressões idiomáticas, então aplicar x|y irá misturar Lx e Ly para produzir uma expressão idiomática inteira.

Repetição O operador Kleen+ é uma expressão regular que gera a linguagem regular Lx. Se x é uma expressão regular que gera Lx, então (x) + é uma expressão regular que gera todas as strings formadas pela concatenação de 1 ou mais membros de Lx. Qualquer string formada pela concatenação de 1 ou mais expressões regulares Lx é considerada um membro da linguagem regular Lx. Uma expressão regular que gera Lx é x, que também é conhecida como Kleene plus. Adicionar x a outra expressão Lx produz uma nova expressão que gera todas as strings em Lx. A tabela de símbolos é uma estrutura de dados chave usada pelos compiladores. Ele armazena informações sobre entidades como variáveis, funções, classes e interfaces, bem como outros objetos. A tabela de símbolos é criada e mantida pelas seções de síntese e análise sintática de um compilador. Diferentes linguagens requerem diferentes tabelas de símbolos para realizar tarefas específicas. Organize os nomes de todas as entidades em um único local. Para determinar se uma variável foi declarada. O código que verifica atribuições e expressões deve ser semanticamente correto para usar verificação de tipo, validação e mensagens de erro. Determinar o escopo de um nome envolve resolver seu comprimento. Uma tabela de símbolos armazena nomes em um dos

dois formatos: uma tabela linear ou uma tabela de hash. Cada nome é representado por uma entrada na tabela que contém as informações a seguir. O nome, o tipo e os atributos de um símbolo devem ser incluídos em sua definição. Programas que dividem o texto bruto em suas unidades lexicais individuais são lexers. Lexers contêm um scanner ou tokenizer, que invalida o texto se detectar um problema. O Lexical Analyzer ajuda os designers a compilar programas lendo fluxos de caracteres do código-fonte e verificando se cada token é legal. Quando um token exige dados, ele os passa para o analisador. Em seguida, ele cria tokens que o analisador usa para produzir resultados. Os analisadores léxicos confirmam todo o código do programa identificando cada token individual. Os scanners só podem produzir um token quando solicitados por um analisador léxico.

Mapa Mental



1.2.5 Perguntas e Respostas

1. Qual função da estrutura léxica?

A estrutura lexical da linguagem descreve quais strings são palavras válidas por definição.

2. Como são criados os tokens?

São criados usando a substring mais longa possível de um arquivo de texto de entrada.

3. Qual a função da gramática formal?

As gramáticas formais definem a estrutura de uma linguagem usando strings estruturadas

1.3 Conclusão

Compiladores que podem produzir linguagens de alto nível requerem uma sobrecarga significativa devido à sua compreensão pela maioria das pessoas. Este foi um avanço importante durante o estágio inicial de crescimento do computador. Entender como os criadores de compiladores projetam suas criações é essencial para entender como eles funcionam.

Referências[1]

L. DEREMER, F. Lexical analysis. 1 jan. 2005.

J. DOOD, A. et al. Analyzing explanations of substitution reactions using lexical analysis and logistic regression techniques. 19 out. 2019.

EZHILARASU, P.; KRISHNARAJ, N. Applications of Finite Automata in Lexical Analysis and as a Ticket Vending Machine – A Review. 6 maio 2015.

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