Projeto e Implementação de uma Ferramenta de Compilação para a Linguagem TPP

Alexandre Borges Baccarini Júnior
[1]Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

Abstract

This report presents the development of a Lexical Analyzer for the C programming language using a Moore machine. The objective is to recognize tokens from a source code and generate a list of tokens as output. The developed Lexical Analyzer is capable of identifying tokens such as reserved words, identifiers, numbers, and symbols.

Additionally, this report discusses the lexical analysis process, the implementation of the automaton, and the results obtained.

Resumo

Esse trabalho apresenta a elaboração de um Analisador Léxico para a linguagem C utilizando a máquina de moore. O seu objetivo é reconhecer tokens a partir de um código fonte e gerar uma lista de tokens como saída. O Analisador Léxico desenvolvido é capaz de identificar tokens como palavras reservadas, identificadores, números e símbolos.

Também, será apresentado o processo de análise léxica, a implementação do autômato e os resultados obtidos.

1 Instruções

Para executar o código desenvolvido é necessário ter instalado automata python e pytest. No projeto está disponibilizado alguns arquivos de exemplos para testes.

Para executar um teste em específico a implementação do analisador léxico ‘analex.py‘ pode ser chamada, o parâmetro ‘-k‘ pode ser utilizado para que somente tokens e chaves de erros sejam impressas, o comando é ”python3 analex.py -k tests/prog-000.cm”. Para executar todos os testes deve ser usado o comando ”pytest”.

A saída do Analisador Léxico pode ser apresentada de duas formas. De forma detalhada, exibe informações sobre o autômato utilizado, a entrada processada e a lista de tokens gerados. Já na forma simplificada, quando a opção -k é utilizada, apenas a lista de tokens é exibida. Caso ocorram erros léxicos, o analisador inclui uma mensagem de erro indicando o caractere inválido.

Entretanto, ele apresenta algumas limitações, não suporta comentários, não reconhece números em formato hexadecimal ou ponto flutuante, mas esses pontos não comprometem seu funcionamento.

Corresponding author(s):

2 Introdução

A base do projeto foi disponibilizada pelo professor Rogério Aparecido Gonçalves por meio do Github Classroom. O repositório continha um arquivo README que explicava o funcionamento do código, oferecendo uma visão geral sobre a estrutura e os componentes principais do projeto. Com isso, foi possível compreender o ponto de partida e dar continuidade ao desenvolvimento.

Segundo Louden, a análise léxica é a primeira fase do processo de compilação, responsável por transformar um fluxo de caracteres em uma sequência de tokens, que são as unidades básicas de uma linguagem de programação. Essa fase é fundamental para garantir que o código-fonte seja corretamente interpretado e validado antes de ser processado pelo compilador. Seguindo com esse conceito, este trabalho tem como objetivo implementar um Analisador Léxico para a linguagem C-, uma versão simplificada da linguagem C, utilizando uma Máquina de Moore. Esse tipo de autômato finito, cuja saída depende exclusivamente do estado atual, é particularmente adequado para a geração de tokens durante a análise léxica. O Analisador Léxico desenvolvido será capaz de reconhecer diversos tipos de tokens, como palavras reservadas, identificadores, números, operadores aritméticos e símbolos especiais, garantindo que o código seja analisado e validado.

A escolha da linguagem C- se da por sua simplicidade, sendo uma versão reduzida da linguagem C. A implementação deste analisador permitirá explorar de maneira prática conceitos fundamentais de compilação, como a construção de autômatos finitos e o reconhecimento de padrões em linguagens regulares. A linguagem C- mantém a estrutura básica da linguagem C, mas foi simplificada com um conjunto reduzido de palavras reservadas, operadores e regras sintáticas.

3 Metodologia

Para realizar a implementação do analisador léxico foi usado AFD's, que a partir de expressões regulares definem tokens. A ideia do código é ler caractere por caractere, assim fazendo simulando as transições de estados e identificando o token.

4 Análise Léxica

A primeira etapa da compilação é a varredura, ou análise léxica, cuja função é converter o código-fonte em uma sequência de tokens.

O analisador léxico processa o código caractere por caractere, identificando padrões com base em expressões regulares. Para cada padrão reconhecido, um token correspondente é gerado e enviado à análise sintática. Esse processo ocorre em duas fases principais: Leitura do código-fonte, onde o texto é percorrido sequencialmente para identificar possíveis padrões léxicos; E Reconhecimento de tokens, Expressões regulares são aplicadas para extrair tokens válidos, como palavras-chave e operadores. Também verifica a existência do arquivo de entrada e trata erros de uso.

O analisador foi implementado com base em uma máquina de Moore (analex.py), para representar a máquina foi usada uma classe definida como "Moore". A classe, possui estados, que vão de q0 a q69, que processam diferentes partes do código. "alfabeto", que possue caracteres reconhecidos. E por final, tokens, que são "palavras"e as "transições".

5 Análise Sintática

A análise sintática é a segunda fase do processo de compilação. Ela recebe como entrada a sequência de tokens gerada pelo Analisador Léxico e verifica se essa sequência está de acordo com as regras.

6 Análise Semântica

A análise semântica é a terceira fase. Ela utiliza as informçaões geradas pela análise sintática para verificar se o código-fonte possui significado válido dentro das regras da linguagem. Enquanto a análise sintática verifica a estrutura do código, a análise semântica verifica o significado das construções.

7 Geração de Código

O Analisador Léxico foi implementado com base em uma Máquina de Moore, utilizando a classe Moore definida no arquivo analex.py. Essa classe é composta por estados, alfabeto, tokens e transições. Os estados, que vão de q0 a q69, são responsáveis por processar diferentes partes do código-fonte. O alfabeto inclui um conjunto de caracteres reconhecidos, como letras, números, operadores e símbolos especiais. Os tokens representam as unidades léxicas da linguagem, como palavras reservadas, identificadores, números e símbolos. As transições definem como a máquina muda de estado com base no caractere lido, permitindo o reconhecimento de padrões no código-fonte.

Para validar o funcionamento do Analisador Léxico, foi desenvolvido o arquivo analex_test.py, que contém testes automatizados utilizando a biblioteca pytest. Esses testes verificam se a saída gerada pelo analisador corresponde à saída esperada, armazenada em arquivos .lex.out. O fluxo de execução dos testes ocorre da seguinte forma: primeiro, o script busca os arquivos de teste no diretório especificado. Em seguida, para cada arquivo de entrada, o Analisador Léxico é executado.

Os testes podem ser executados de três maneiras. A primeira é arquivo por arquivo detalhadamente, que exibe todos os estados, transições, alfabeto e saída de tokens, esse modo é executado com o comando python3 analex.py tests/prog-'num'.cm. A segunda maneira é arquivo por arquivo sem detalhes, que exibe apenas a lista de tokens, utilizando o comando python3 analex.py -k teste/prog-'num'.cm. A terceira forma é todos os arquivos sem detalhes, que executa todos os testes de uma vez, exibindo apenas o resultado, com o comando pytest.

Durante a fase de testes, observou-se que alguns arquivos de saída esperada (Testes 4, 5 e 7) continham inconsistências. Esses erros impediam a validação correta do Analisador Léxico. Para resolver o problema, foram feitas correções nos arquivos de saída, garantindo que eles refletissem o código e a verdadeira saída esperada. Assim, permitiu a execução bem-sucedida dos testes.

Referências

LOUDEN, Kenneth C. Compiladores: Princípios e Práticas. São Paulo, SP: Thomson, 2004. ISBN 8522104220.

MENEZES, Paulo Blauth. Linguagens Formais e Autômatos. Porto Alegre, RS: Bookman, 2011. ISBN 9788577808403.

RODGER, Susan H. JFLAP: An Interactive Formal Languages and Automata Package. Sudbury, MA: Jones & Bartlett Learning, 2006. ISBN 9780763738347.

SIPSER, Michael. Introdução à Teoria da Computação. São Paulo, SP: Cengage Learning,

2007. ISBN 9788522104997.