INSTITUTO FEDERAL DE MINAS GERAIS
Campus Formiga
Curso Ciência da Computação

TRABALHO DE COMPILADORES
ANÁLISE LÉXICA E SINTÁTICA

Aluno: Rafael Augusto de Rezende Neto

Formiga - MG
2022

INTRODUÇÃO

Este trabalho tem como objetivo apresentar a implementação de um *frontend* do compilador, principalmente sua parte inicial, composta pelos analisadores léxico e sintático, além da tabela de símbolos e o tratamento de erros no modo pânico. Este trabalho se justifica como uma atividade prática da disciplina de Compiladores, ofertada pelo professor Mário Luiz Rodrigues Oliveira, no curso de Ciência da Computação. Por fim, utilizando as linguagens de programação Python, pode-se obter como resultado, um analisador léxico e um analisador sintático, além do modo pânico como tratamento de erros encontrados para a linguagem MiniPascal.

OBJETIVOS

Objetivo geral

O objetivo deste trabalho é o projeto e a implementação dos analisadores léxico, sintático, tabela de símbolos e tratamento de erros para a linguagem de programação MiniPascal, gerada pela gramática descrita na especificação do trabalho.

Objetivos específicos

1. Implementação de um analisador léxico, capaz de identificar tokens, e classificá-los baseado nos tipos de tokens disponíveis.
2. Implementação de um analisador sintático, responsável por validar as sentenças de entrada (tokens), retornadas pelo analisador léxico.
3. Implementação do tratamento de erro baseado no modo pânico.

MATERIAIS

1. Python
2. Sistema Operacional (Windows/Linux)
3. PyCharm IDEA
4. Github

Python

Atualmente com uma vasta gama de bibliotecas com suporte para todas as

áreas de ciência da computação, o Python se estabeleceu como uma linguagem de programação gratuita e universalmente disponível, sendo uma das mais populares da computação científica.

Algumas características que justificam a vanguarda da linguagem de programação Python, são a licença de código aberto, a execução de tarefas em multiplataformas, a sintaxe limpa mas com construções sofisticadas, uma poderosa interatividade, a capacidade de extensão do código compilado e incorporação em aplicativos, o grande número de bibliotecas instaladas, a ligação a todos os kits de ferramenta GUI, a forte comunidade de desenvolvedores e um repositório de módulos que simplificam o gerenciamento de software (OLIPHANT, 2007).

Sistema Operacional

Segundo Tanenbaum (2003), os sistemas operacionais realizam duas funções essencialmente não relacionadas: fornecer a programadores de aplicativos (e programas aplicativos, claro) um conjunto de recursos abstratos limpo em vez de recursos confusos de hardware, e gerenciar esses recursos de hardware. Microsoft Windows é uma família de sistemas operacionais desenvolvidos, comercializados e vendidos pela Microsoft, já o Linux é um sistema operacional, assim como o Windows e o Mac OS, que possibilita a execução de programas em um computador e outros dispositivos. Linux pode ser livremente modificado e distribuído (SIMIONI, 2021).

PyCharm

O PyCharm IDEA é desenvolvido pela JetBrains e está disponível como uma edição da comunidade licenciada do Apache 2 e em uma edição comercial proprietária. Além disso, foi projetado para maximizar a produtividade do desenvolvedor. Juntos, a assistência para codificação inteligente e o design ergonômico tornam o desenvolvimento, principalmente em Python, não apenas produtivo, mas também agradável.

Github

De acordo com o LONGEN (2022), o GitHub é um serviço baseado em nuvem que hospeda um sistema de controle de versão chamado Git. Ele permite que os

desenvolvedores colaborem e façam mudanças em projetos compartilhados enquanto mantêm um registro detalhado do seu progresso.

MÉTODOS

Para que o objetivo do trabalho fosse realizado, primeiramente, foi realizado um estudo aprofundado com base na principal referência da disciplina, o livro Compiladores do escritor Aho (também conhecido como livro do dragão). Após o estudo teórico, os exemplos práticos de implementação disponibilizados pelo professor Mário, serviram como principal referência para a implementação dos analisadores Léxico e Sintático.

Baseado nos exemplos “toy” disponibilizados pelo professor, foi reutilizado as mesmas estruturas de dados, sendo elas:

- Classe TipoToken: responsável pelas definições dos tipos de token existentes, organizados de forma enumerada, além dos tipos descritos na especificação, foram criados os tipos ERRO e FIMARQ, que representam um erro léxico e o fim de arquivo, respectivamente;
- Classe Token: TAD que representa um token, contendo atributos como o tipo, linha em que está localizado e lexema;
- Classe Lexico: responsável por toda execução feita na análise léxica, como a identificação e classificação dos tokens.
- Classe Sintatico: responsável por toda execução feita na análise sintática, como a verificação das sentenças de entradas (tokens), recebidos pela classe Lexico.

Na análise Léxica, o tratamento dos tokens foram separados em estados, sendo eles:

- estado 1: responsável pela primeira classficação;
- estado 2: responsável por tratar os identificadores e palavras reservadas;
- estado 3: responsável por tratar constantes numéricas;
- estado 4: responsável por tratar caracteres especiais;

- estado 5: responsável por tratar as cadeias de caracteres;
- estado 6: responsável por tratar comentários de linha;
- estado 7: responsável por tratar comentários de bloco.

Ainda na análise léxica, foi criado um método (*proxChar()*) que verifica qual o próximo caractere do código de entrada, pois assim, os operadores >=, <=, <> e := são verificados e retornados diretamente, sem a necessidade de serem vinculados a um estado.

Já na análise sintática, o método de consumo de um terminal é o *consome(token)* que valida se o token encontrado é o esperado. Para isso, foi criado um método (*proxToken()*) que recebe um novo token, verifica se este é do tipo ERRO e retorna o token. Caso os tokens sejam do mesmo tipo, a execução do programa continua de forma contínua.

O modo pânico foi implementado juntamente a análise sintática, e um novo erro é adicionado a um vetor de erros quando:

- um erro sintático é encontrado na validação de tokens;
- um token do tipo ERRO é recebido no método *proxToken(),* indicando um erro léxico;
- o token não é aceito na produção esperada, também havendo um erro sintático.

Por fim, foi implementado um buffer de token na análise sintática, que é preenchido com o próximo token do programa sempre que é encontrado um erro sintático, e o token é igual ao atual, isto é, sempre que é encontrado um token inválido, então ele é ignorado, resgatando o próximo para ser utilizado no buffer.

Dessa forma, o compilador sempre continua sua execução, ignorando os erros, mas preenchendo um vetor com as informações necessárias para identificação do erro no final da execução.

RESULTADOS E DISCUSSÕES

Foram obtidos bons resultados após a implementação das etapas de análise léxica e sintática. Os tokens retornados pela análise Léxica demonstraram estarem de acordo com o esperando, tendo assim, 100% exatidão na identificação e classificação dos tokens, utilizando os 45 exemplos de códigos de entradas como

testes. A verificação dos tokens recebidos na análise sintática também demonstraram um excelente resultado, não foram encontradas execuções fora do esperado durante a realização dos testes.

O armazenamento de erros no modo pânico também se mostrou eficiente, pois sempre que nos testes utilizando códigos de entrada com erros, todos eram descritos como esperado, exibindo a linha exata do erro, uma descrição objetiva do erro, e a origem do erro, seja Léxico e Sintático. Segue saída do exemplo22.txt que apresenta um erro léxico na linha 5, e erros sintáticos nas linhas 5, 8 e 9:

Figura 1: Exemplo de saída com erros

```
(venv) D:\Desktop\CC\2022.1\C\compiler>python main.py palavras\exemplo22.txt
ERRO LÉXICO [linha 5]: Caracter inválido: $
ERRO DE SINTAXE [linha 5]: Era esperado um tipo.
ERRO DE SINTAXE [linha 8]: Sem argumentos
ERRO DE SINTAXE [linha 9]: era esperado "id" mas veio ")"
```

Conclusão

Neste trabalho foi apresentado e discutido brevemente a implementação de analisadores léxico e sintático como etapas iniciais de construção de um *frontend* de um compilador, assim como, a implementação do modo pânico para tratamento de erros. A tecnologia que mais foram adequadas para realizar tal integração, foi a utilização da linguagem Python para a implementação das etapas iniciais do compilador.

Por fim, o trabalho se mostra útil na aplicação a qual se destinou, pois foi possível realizar a identificação e classificação de tokens, assim como a validação de acordo com a gramática especificada, e também o tratamento de erros de acordo com o modo pânico, porém, em alguns casos foram encontrada dificuldades no tratamento de erros, nas quais a saída não englobam todos os erros encontrados no programa. Mesmo assim, os resultados foram favoráveis, e contribuíram muito para o aprendizado na disciplina de compiladores, através da ação realizada em conjunto com a aquisição de conhecimentos e habilidades sobre compiladores, principalmente nas etapas léxica e sintátca.

MANUAL DE EXECUÇÃO:

Para executar (em linux) o compilador completo é necessário utilizar o comando:

python main.py <caminho-do-codigo-de-entrada>

A execução no windows necessita da instalação do python em sua versão apropriada (3 ou superior).

REFERÊNCIAS BIBLIOGRÁFICAS:

AHO, A. V. et al. **Compiladores**. 2 ed. São Paulo: Pearson Addison-Wesley, 2008.

SIMIONI, Dionatan. Entenda como funciona o Linux. **Hostinger**, 2021. Disponível em: <https://www.hostgator.com.br/blog/entenda-como-funciona-o-linux/>. Acesso em: 14 de maio de 2022.

LONGEN, Andrei. O que é Github e como usá-lo. **Hostinger**, 2022. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github>. Acesso em: 15 de maio de 2022.

TANENBAUM, A. S., **Sistemas Operacionais Modernos**. Segunda Edição, Prentice Hall, 2003. Bibliografias Complementares. GALVIN, S., Operating System Concepts.

OLIPHANT, T. E. Python for scientific computing. **Computing in Science & Engineering**, v. 9, n. 3, p. 10-20, 2007.