Projeto final

Transpilador de C para Python

Alunos:
- Vinicius Medeiros
- Saulo Mélo
- Pietro Gama
- Pedro Lucas
- Italo Santos
- Pedro Henrique

01. Introdução

Neste trabalho, desenvolvemos um transpilador capaz de converter código escrito em C para Python. Esse projeto foi escolhido com o objetivo de explorar as diferentes etapas envolvidas na criação de um compilador/transpilador, desde a análise léxica até a geração de código, passando pela análise sintática e semântica. A motivação principal foi proporcionar uma ferramenta útil para desenvolvedores que trabalham com ambas as linguagens, facilitando a migração de código C para Python, que é uma linguagem de alto nível e amplamente utilizada devido à sua simplicidade e versatilidade.

Para a implementação deste transpilador, seguimos rigorosamente as fases clássicas da compilação: análise léxica, análise sintática, análise semântica e geração de código. Cada uma dessas fases foi implementada de maneira a garantir a correta interpretação e conversão do código C para uma sintaxe equivalente em Python. Durante o desenvolvimento, enfrentamos desafios como a tradução de construções específicas do C, como ponteiros e gerenciamento manual de memória, para paradigmas mais idiomáticos em Python, que lida com a memória de maneira automática.

02. Análise de Complexidade Assintótica

A função parse na classe Parser se destaca pela complexidade, pois ela processa toda a lista de tokens e pode chamar outros métodos de parsing múltiplas vezes. Portanto, focaremos na parse e nas funções associadas como parse_if_statement e parse_function.

A função parse_if_statement é responsável por analisar estruturas de controle condicionais (if, else if, else). Assim, analisando a complexidade dessa função:

>Inicialização e Avanço dos Tokens

- A função avança os tokens para posicionar-se nas condições e blocos de código.
- **Complexidade:** Cada avanço é **O(1)**, mas o número total de avanços é proporcional ao número de tokens processados, o que é **O(n)** no pior caso.

>Parsing da Condição

- A condição é analisada usando parse_expression, que é **O(n)** no pior caso.

>Parsing do Corpo

- Cada bloco de if, else if ou else é analisado usando parse_statement.
- Cada parse_statement pode chamar outras funções de parsing, resultando em um comportamento recursivo linear.
- **Complexidade Total:** Se m é o número de condições e k o número de tokens por bloco, a complexidade é **O(m * k)**, que se simplifica para **O(n)** quando considerado o número total de tokens.

Pior Cenário

O pior cenário ocorre quando há uma longa cadeia de if-else if-else, cada um com seu próprio conjunto de tokens, resultando em uma complexidade linear em relação ao número total de tokens.

Possíveis Otimizações

>Cache de Resultados

- Utilizar um cache para armazenar resultados intermediários durante a análise de expressões que são repetitivas ou podem ser reutilizadas.

>Reducir Chamada a Métodos Internos

- Reduzir chamadas redundantes a métodos internos, especialmente aqueles que avançam o token ou analisam partes que poderiam ser combinadas em uma única chamada.

>Análise Preguiçosa

- Implementar a análise preguiçosa (lazy parsing) onde a análise detalhada de um bloco só ocorre quando necessário.

>Pré-processamento de Tokens

- Executar uma etapa de pré-processamento para identificar padrões complexos ou possíveis estruturas condicionais antes da análise detalhada.

03. Link do GitHub

https://github.com/Saulomelo/ProjetoFinal_Aspec-Teoricos-Comp