CENTRO UNIVERSITÁRIO DAS FACULDADES ASSOCIADAS DE ENSINO - UNIFAE

**GUILHERME HENRIQUE TONETTI**

**Kavarador:** compilador para a linguagem Kavar

Documento de Requisitos

**SÃO JOÃO DA BOA VISTA - SP**

**AGOSTO DE 2022**

**Sumário**

[1 Introdução](#_ttwam2buqqkv) 2

[2 Compiladores](#_17dhvbxdsq4) 2

[2.1 Fases](#_yffa8bbf36c1) 3

[3 Estratégias e/ou ferramentas](#_5vhrffmrctxg) 4

[4 Características iniciais da linguagem Kavar](#_gt69xjs52o5r) 5

[5 Cronograma](#_cbk143wvf6xx) 6

[Referências](#_ox2b8tuayrye) 6

# 

# 1 Introdução

Uma vez que a Teoria dos Compiladores é uma área fundamental para a Ciência da Computação, a motivação deste projeto envolve entender como os compiladores funcionam, seus principais conceitos/teorias e motivações, compreendendo o funcionamento de uma linguagem de programação por dentro. Além de unir os conhecimentos de diversas disciplinas apreendidas durante a graduação em Engenharia de Software (algoritmos, paradigmas de programação, estrutura de dados, teoria da linguagem, arquitetura de computadores, padrões de projeto e a própria engenharia de software), construir um compilador é a chave para criar ferramentas que auxiliam na construção de novas tecnologias.

Inicialmente, o objetivo deste projeto é introduzir a construção de um compilador *tree walk* funcional, iniciando na análise léxica e finalizando na análise semântica, com a linguagem de programação Java para uma linguagem de programação imperativa de alto nível recentemente criada. Estudando compiladores não aprendemos apenas a construí-los, mas também a metodologia geral para solucionar problemas complexos.

Obviamente, o foco primário não é buscar por um algoritmo extremamente otimizado em termos de desempenho, pois o tempo disponível para o autor é escasso e outras disciplinas estão sendo apreendidas em paralelo. Todavia, caso este fosse o foco neste projeto, a melhor escolha seria construir o compilador na linguagem de programação C, devido ao ganho de controle sobre a memória e as estruturas de dados necessárias (vetores dinâmicos, grafos, árvores e tabelas *hash*). Entretanto, busca-se implementar o segundo objetivo, definido por AHO, 2007, que é um algoritmo correto que produza resultados coerentes com mensagens de erros amigáveis e esclarecedoras.

Segundo Nystrom (2021), construir um compilador é uma tarefa desafiadora, mas que o torna um programador mais robusto e capaz de utilizar eficientemente as estruturas de dados e algoritmos nos trabalhos do dia a dia. Para Yegge (2007), “Se você não sabe como os compiladores funcionam, então você não sabe como os computadores funcionam. Se você não tem 100% de certeza se sabe como os compiladores funcionam, então você não sabe como eles funcionam.”.

# 2 Compiladores

Para que seja possível executar um programa de *software* em determinado computador, ele precisa ser traduzido para uma linguagem que o próprio computador entenda. Quem faz essa tradução é denominado compilador. De maneira simplista, um compilador recebe um programa em uma linguagem (*fonte*) e o traduz para um programa equivalente em outra linguagem (*objeto*), detectando e relatando quaisquer erros que aconteçam durante a tradução (AHO et al., 2007).

Há duas partes que se dividem e cooperam para o processo de compilação, análise e síntese. A primeira parte (comumente chamada de *front-end)* é responsável por estabelecer uma estrutura gramatical sobre o programa, detectando se está sintaticamente malformado e fornecendo mensagens esclarecedoras para que possam ser corrigidas. Ademais, armazena as informações coletadas em uma estrutura de dados chamada *tabela de símbolos*.

A segunda parte (comumente chamada de *back-end*) constrói o programa objeto com base na tabela de símbolos.

## 2.1 Fases

A Figura 2.1 ilustra as fases de um compilador, que serão explicadas em seguida.

**Analisador Léxico**: é a primeira fase do compilador. Seu objetivo é ler o código-fonte, caractere por caractere, separar e identificar os elementos da linguagem, ou seja, seus símbolos léxicos (*tokens*). Os comentários, espaços em branco e tabulações são ignoradas.

**Analisador Sintático**: é responsável pelo agrupamento dos *tokens* para verificar se fazem sentido na linguagem e se podem ser gerados por uma gramática. Esta fase verifica se a formação estruturada dos *tokens* forma um programa válido.

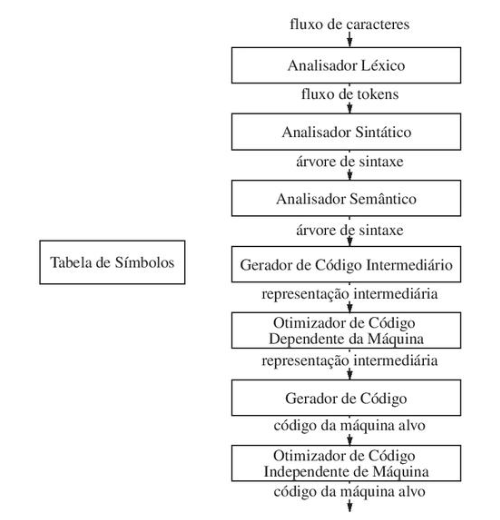
**Analisador Semântico**: utiliza a árvore de sintaxe e as informações da tabela de símbolos com o objetivo de identificar a consistência semântica do programa fonte com a definição da linguagem.

**Gerador de Código Intermediário**: é responsável por gerar uma representação intermediária explícita para uma linguagem de baixo nível, como um programa para uma máquina abstrata.

**Otimizador de Código Dependente da Máquina**: com o código intermediário como entrada, utiliza diversas técnicas para realizar transformações com o objetivo de produzir um código objeto melhor, seja em termos de eficiência, redução no consumo de energia ou um código menor (redução de código).

**Gerador de Código**: é a última fase da compilação, mapeando o código intermediário otimizado do programa fonte para a linguagem objeto. Os detalhes particulares da máquina devem ser levados em consideração, atribuindo cuidadosamente os registradores às variáveis do programa.

**Figura 2.1**: visão geral das fases de um compilador



**Fonte**: AHO et al, 2007, p. 4.

# 

# 3 Estratégias e/ou ferramentas

Diversas ferramentas que geram código automaticamente estão disponíveis para auxiliar no processo de construção de um compilador, como o Lex e o Yacc. Entretanto, me absterei de usar essas ferramentas e construirei tudo sem o usufruto de ferramentas automatizadas.

A estratégia adotada é dividir para conquistar, criando diversas etapas que complementam uma à outra. Eis algumas estratégias de documentação e um esboço dos passos a serem completados (a sequência ainda não foi definida):

* Criar os detalhes da linguagem e especificar as respectivas regras semânticas (quais são as palavras-chave disponíveis, estrutura de uma função, etc…).
* Criar o(s) programa(s) que será utilizado como entrada de teste no fim da análise.
* Documentar a linguagem
  + Criar um diagrama de classes;
  + Criar um autômato finito que a compreenda.
* Análise léxica
  + Reconhecer os *tokens*;
  + Validar os *tokens* e apresentar erros caso algum não esteja de acordo com as *tokens* definidos pela linguagem;
  + Armazenar o conhecimento obtido por esta análise na tabela de símbolos, representada por uma estrutura de dados compatível (que será utilizada ao decorrer do desenvolvimento).
* Análise sintática
  + Agrupar os *tokens*;
  + Reconhecimento de cadeias;
  + Entender e, possivelmente, utilizar gramáticas livres de contexto.
* Análise semântica
  + Utilizar a árvore semântica;
  + Detectar se as variáveis estão no escopo correto;
  + Verificar se os tipos das variáveis estão condizentes com o seu uso.
* Criar um manual do usuário para instruir a como executar o compilador com um programa previamente criado como entrada.

# 4 Características iniciais da linguagem Kavar

A linguagem:

* É estruturada e fortemente tipada;
* É *case-sensitive*, ou seja, as palavras-chave da linguagem somente são válidas se digitadas exclusivamente em maiúsculas ou minúsculas;
* Trabalha com números inteiros, reais e *strings*;
* Possui condições e controles de fluxo (*if*, *else,* *for* e *while*);
* Utiliza expressões aritméticas (soma, subtração, divisão e multiplicação); e
* Possui expressões relacionais de comparação (=, <>, >, <, >= e <=).

Características adicionais serão acrescentadas no próximo documento, bem como as regras semânticas que compõem a linguagem.

# 5 Cronograma

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Atividade | Ago | Set | Out | Nov |
| Definir os *tokens* e as regras avançadas da linguagem |  |  |  |  |
| Criar o autômato finito determinístico |  |  |  |  |
| Criar o trecho de código que será executado |  |  |  |  |
| Análise léxica |  |  |  |  |
| Identificar ao que cada variável se refere |  |  |  |  |
| Análise sintática |  |  |  |  |
| Análise semântica |  |  |  |  |
| Resultados e considerações finais |  |  |  |  |

# Referências

AHO, A. et al**. Compiladores**: princípios, técnicas e ferramentas. 2 ed. – São Paulo: Pearson Addison-Wesley, 2007.

AIKEN, A. **edX** - Stanford. Compilers. Disponível em: <https://www.edx.org/course/compilers>.

NYSTROM, R**. Crafting Interpreters**. Genever Beginning, 2021.

Wikipédia. **Compilador**. Disponível em: <https://pt.wikipedia.org/wiki/Compilador>. Acesso em: 15 ago. 2022.

YEGGE, S. Rich Programmer Food. **Stevey’s Blog Rants**. Washington, 21 jun. 2007. Disponível em: <https://steve-yegge.blogspot.com/2007/06/rich-programmer-food.html>. Acesso em: 17 ago. 2022.