
# Teoria dos Compiladores - Geração de Código

UFJF

Matheus Gomes Luz Werneck (201835037)
Pedro Henrique Almeida Cardoso Reis (201835039)

Juiz de Fora, 09 de Julho de 2023

# Contents

# 1 Introdução

Este relatório do trabalho de Teoria dos Compiladores tem como objetivo explicar primeiramente como foi feita a Geração de Código para Java e Jasmin. Utilizamos o *String template*, uma técnica utilizada na geração de código, para facilitar a criação dinâmica de strings complexas.

# 2 Instruções para o uso do projeto

No projeto encontram-se dois diferentes Shell Scripts para a compilação do mesmo.

- **run-project.sh:** Em resumo, este primeiro Shell Script remove o arquivo de log do código desenvolvido e logo depois compila as gramáticas desenvolvidas com o auxílio do *ANTLR* e, usando o *Maven*, executa o projeto.

## 2.1 Como rodar o projeto

- Certifique de ter instalado em seu computador o Java, JDK e Maven.
- Para executar o programa, abra um terminal,
  - Execute o comando **mvn clean install** para instalar as dependências do projeto.

    ```
    mvn clean install
    ```

  - Execute o comando **./run-project.sh (opção)**. Passando -s é feito a geração de código em Java;
    Pode ser necessário utilizar o comando *chmod +x run-project.sh* para dar permissão de execução ao script caso esteja usando Linux.

    ```
    ./run-project.sh some_file.txt
    ```

  - Após a execução do script, teremos o código em Java gerado.

# 3 Tomada de decisões de implementação

Antes de nos aprofundarmos no Analisador Léxico deixaremos registrado nessa seção algumas das decisões tomadas durante a implementação do projeto.

A primeira das decisões tomadas foi a utilização do Maven para fazer o Build do projeto. Optamos pelo Maven pois segue um padrão de diretório e convenções predefinidas para estruturar projetos Java, uma vez que estávamos tendo problemas na hora de encontrar pacotes e classes para fazer a compilação. Graças

ao Maven ficou mais fácil de entender e estruturar o projeto.

Outra decisão importante foi a utilização da biblioteca *Log4j*. Com a utilização de logs ficou mais fácil para entender os erros, além de poder visualizar o passo-a-passo de como o nosso código estava se comportando ao longo de seu desenvolvimento.

Para a implementação do analisador semântico foi utilizado o padrão Visitor, foi criado uma classe que implementa o Visitor chamada TypeCheckVisitor e essa classe faz toda a avaliação semântica do código, e no final printa os erros na tela se houverem.

# 4 Geração de Código

A quarta parte do trabalho de Teoria dos Compiladores é a geração de código em uma linguagem de alto nível e em Jasmin.

## 4.1 Geração de código em Java

Escolhemos Java como nossa linguagem de alto nível para podermos fazer a geração de código. Escolhemos essa linguagem pois já estamos familiarizados com ela.

Particularmente não tivemos dificuldades em fazer a geração para a linguagem Java, mesmo o retorno das funções tendo sido bem desafiador. Todavia, em alguns momentos, foi preciso fazer algumas adaptações que explicaremos nas subseções abaixo:

### 4.1.1 Funções e retorno das funções

Para o retorno das variáveis criamos um array de objetos, logo, toda função ira retornar um array de objetos. Isso foi feito pois a linguagem *lang* permite múltiplos retornos.

```
    }

    public static ArrayList<Object> divMod(int n, int q) {

        ArrayList<Object> _returnList = new ArrayList<Object>();

        _returnList.add(n / q);
        _returnList.add(n % q);

        return _returnList;

    }
}
```

Figure 1: Type cast do retorno das variáveis

Ao chamar a função, criamos uma variável temporária para receber o retorno da mesma, como por exemplo *_r1*. Após isso, é feito o type cast para o tipo esperado nas variáveis que irão receber os retornos.

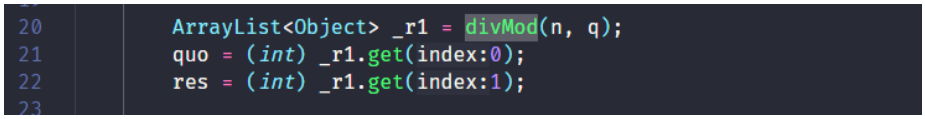


Figure 2: Variáveis no início da função

### 4.1.2 Iterate/For

Se temos um Iterate = 10 em lang, precisamos rodar uma instrução 10 vezes. Ao fazer a conversão do iterate para um for em Java, criamos algumas variáveis que sempre segue o seguinte padrão: *_a1 para o primeiro for, _a2 para o segundo for* e assim por diante para que a variável não seja repetida. Mas por que essas variáveis especiais começam com um _ ? Lang não pode começar variável com _, assim, podemos fazer uma diferenciação.


Figure 3: For em Java com as variáveis especiais

### 4.1.3 Declaração de variáveis

Na parte de declaração de variáveis, decidimos no começo da função já declarar todas as variáveis necessárias para aquela função. Como na etapa do analisador semântico já sabemos a tipagem de todas as variáveis, logo, já declaramos todas as variáveis no começo de cada função.


Figure 4: Exemplo de declaração de variáveis nas funções

## 4.2 Geração de código em Jasmin

A geração de código para a linguagem Jasmin foi muito desafiadora, pois Jasmin é uma linguagem de montagem para a Máquina Virtual Java (JVM). Além do mais, Jasmin requer uma sintaxe específica e uma compreensão profunda da JVM e suas instruções. Tentamos entender a sintaxe para conseguirmos implementar, porém, o tempo foi passando e não conseguimos avançar. Por conta dessas peculiaridades, e a falta de um tempo hábil, não conseguimos gerar o código para Jasmin.