**Departamento de Ciência da Computação - UFJF**
**DCC045 - Teoria dos compiladores**

# Interpretador para a Linguagem *lang*


Alunos:

Daniel Souza Ferreira - 201665519B

Matheus de Oliveira Carvalho - 201665568C


**Julho / 2021**

# 1 Introdução

Neste trabalho desenvolvemos o interpretador para a criação de um compilador para a linguagem *'lang'*.

Interpretadores são programas que interpretam algoritmos, geralmente a partir de uma AST ou a partir de uma interpretação intermediária permitindo a execução de instruções sem transformá-las em linguagem de máquina ou baixo nível. Podendo ser implementados através de estrategias diferentes eles apresentam algumas vantagens como portabilidade, facilidade de manutenção e maleabilidade, ao custo de eficiência de tempo de execução, uso maior de memória e falta de acesso a recursos de baixo nível.

# 2 Características do Projeto

Para a construção do interpretador foi utilizada a estratégia de implementação através de uma AST (Árvore Sintática Abstrata), sendo essa uma árvore em que seus nós são compostas por símbolos que podem ser terminais ou não. Essa AST foi criada tendo como base a Parse Tree gerada no analisador sintático. Tendo isso em mente, foram implementadas novas classes utilizando como base o projeto da etapa anterior. Outros arquivos gerados pela ANTLR foram utilizados e partes de alguns códigos de arquivos já existentes, fornecidos pelo professor, foram adaptados ao projeto.

Para a criação da AST foi necessário a implementação de algumas classes sendo que cada uma dela representa os nós existentes em sua geração. Além disso foi criado arquivos que utilizam a lógica de implementação de interpretação do tipo *visitor* para poderem realizar a interpretação da AST.

## 2.1 Novos Arquivos Presentes no Trabalho

- Arquivos presentes na pasta *ast*

  Nessa pasta estão as classes referentes aos nós da AST, como são muitos e na sua maior parte seguem um princípio comum. não iremos citá-los um por um.

- Main.java

  Ao invés de criar a ParseTree como fazia anteriormente a Main agora é responsável por instanciar a classe respónsavel pela construção da AST.

- lang.g4

- Novos arquivos gerados pelo ANTLR

  - langBaseVisitor.java
  - langVisitor.java
- Visitor.java
- VisitorInterpretator.java
- TestVisitorInterpretator.java

## 3 Implementação do Trabalho

Como citado anteriormente a árvore AST foi gerada com base na ParseTree gerada pelo ANTLR, presente também no último trabalho. Para cada nó foi implementado uma classe específica representando as características que foram pré-determinadas ao representarmos a gramatica da linguagem no arquivo **lang.g4**, presentes na pasta AST. Vale ressaltar que algumas alterações foram feitas na gramática a fim de facilitar a implementação, porém as mesmas não alteraram o sentido representado pela mesma.

O comportamento do interpretador foi implementado no arquivo **VisitorInterpretator.java** seguindo o padrão *visitor*. Para a bateria de testes foram utilizados os mesmos arquivos do trabalho anterior. Vale ressaltar que, pela existência de arquivos sintaticamente corretos porém logicalmente incorretos, os mesmos foram separados na pasta **certos_Errados** pois o interpretador não consegue os interpretar corretamente .

## 4 Execução do Programa

Para a execução do programa foi criado um makefile que possui os seguintes comandos:

- **make** - utilizado para fazer a limpeza dos arquivos .class gerados na pasta 'lang' e depois realizar a compilação de todos os arquivos .java.
- **make clean** - utilizado para fazer a limpeza dos arquivos .class gerados na pasta 'lang'.
- **make execute** - utilizado para realizar a execução da bateria de testes.

A bateria de testes gerada pelo make execute é dos arquivos da pasta 'certa'. Para realizar os testes dos arquivos da pasta 'errado' deve-se alterar o valor da variável **okSrcs** na classe **TestParser.java** para **"testes/sintaxe/errado/"**.

Para executar os arquivos da pasta **certos_Errados** basta alterar o valor da variável **okSrcs** na classe **TestParser.java** para **"testes/sintaxe/certos_Errados/"**. Vale lembrar a necessidade de se pressionar Enter ao realidar testes em arquivos contendo a função Read.