Universidade Federal de Juiz de Fora - UFJF - Departamento de Ciência da Computação

Teoria dos Compiladores - 2021/1

Alunos: Arthur de Freitas Dornelas 201735004
Vinicius da Cruz Soranço 201735003

1. Introdução

Nesta etapa do trabalho, foi desenvolvido um Interpretador para a linguagem **lang**, dessa forma, foi utilizado a linguagem Java e o auxílio da ferramenta ANTLR 4.

2. Interpretador

Para implementação do Interpretador, utilizamos a classe **Intrepreter.java**, que funciona basicamente como uma sobrescrita da classe LangBaseVisitor que é gerada pelo ANTLR. Dessa forma, realizamos as modificações de cada função da classe base, onde é possível trabalhar com AST gerada pela ferramenta, para se adequar a interpretar os requisitos propostos da linguagem lang.

Portanto, como estruturas de dados nessa classe, utilizamos basicamente três:

Stack - Pilha de HashMap que é usada para armazenar a memória do programa, dessa forma, toda vez que é gerado um novo Interpreter pelo construtor temos o empilhamento do HashMap.

HashMap - Utilizado para armazenar as atribuições que são feitas nas variáveis, funções e classes.

ArrayList - Utilizado para armazenar os parâmetros passados e as expressões que serão retornadas.

Em geral, essa é a criação das 4 estruturas globais de utilização, porém dentro das sobrescritas das funções temos a utilização de estruturas auxiliares para funcionamento interno, onde são adaptadas para as particularidades da linguagem.

3. Compilação

Para executar o interpretador basta entrar na raiz do projeto e rodar o *build.sh* pela linha de comando ou executar os seguintes comandos em sequência:

- *java -jar ./lang/lexer/grammar/antlr-4.8-complete.jar -o lang/lexer/grammar/antlr -package lang.lexer.grammar.antlr -visitor ./lang/lexer/grammar/Lang.g4 -Xexact-output-dir*

- *javac -cp .:lang/lexer/grammar/antlr-4.8-complete.jar lang/LangCompiler.java*

- *java -cp .:lang/lexer/grammar/antlr-4.8-complete.jar lang/LangCompiler -bsm*

Com isso, o interpretador vai ser gerado, e o nosso código vai ser compilado e executado, imprimindo os testes na tela.

4. Testes

Para rodar os testes, criamos a classe **TestInterpreter.java**, que tem como objetivo rodar os testes propostos e retornar os que tiveram erros e os que foram aceitos. Utiliza do analisador sintático já implementado e chama a função do Interpreter mandando a AST como parâmetro e o vocabulário.
No final a classe imprime a quantidade de erros e acertos dos testes feitos.