Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Relatório Trabalho Prático

Teoria dos Compiladores

Relatório Trabalho Prático

André Luiz dos Reis - 201965004A

Lucca Oliveira Schröder - 201765205C

Juiz de Fora - MG

2022

1. Analisador léxico

Expressões regulares

Token	Expressão regular
identificador	[a - z] [a- Z-0-9-_]*
tipo	[A - Z] ([a - z] | [A - Z] | [0 - 9] | [_])*
numeroDecimal	[0 - 9]* [.] [0 - 9]+
numeroInteiro	[0 - 9]+
chacter	[‘] + [.] + [‘] | [‘] + [\ ] + [.] + [‘]

P.S.: [.] faz referência a qualquer tipo de caractere que possa ser usado

Autômato para os tokens da linguagem

A keyword faz referência a todas palavras reservadas (true, false, null, bool, etc), a todos os símbolos (“==”, “<”, “]”, etc) e também a todas operações matemáticas (“+”, “-”, etc).

Estrutura de Dados usadas

A geração do analisar léxico deu-se junto ao analisador lexico, a partir da mesma ferramenta utilizada.

2. Analisador sintático

2.1 Escolha de ferramentas

Como não geramos o analisador sintático de forma manual, utilizamos uma ferramenta para isso. Foi utilizado o ANTLR, que é uma ferramenta para geração automática de parser e analisador descendente recursivo e utiliza uma estratégia para encontrar a derivação mais à esquerda. Fizemos o seu download pelo link https://www.antlr.org/download.html e implementamos no nosso trabalho utilizando o antlr-4.8-complete.jar.

2.2 Estratégia de implementação do analisador sintático

Foi criado o arquivo lang.g4, onde são informadas as regras de gramática e léxicas. E com o uso da ferramenta ANTLR é possível integrar tanto o analisador léxico quanto o sintático.

Agora, compilando o antlr-4.8-complete.jar e com o uso da nossa classe de teste é feito o passo a passo até a criação do parser pelo ANTLR. Primeiramente, é criado uma stream de caracteres a partir de um arquivo de entrada, em seguida é passado para gerar o analisador léxico (classe langLexer), esse analisador recebe como parâmetro o stream de caracteres e esse stream é usado para criar um stream de token, e por fim, esse stream de token é passado como parâmetro para poder criar o parser. E após essa criação, podemos passar o parser em conjunto com o nome da regra, gerando o analisador descendente recursivo e dessa forma é retornado a árvore de derivação.

3. Compilação e Execução

Para a execução poderá ser utilizado o script **run.sh** passando dois parâmetros, quando necessários, separados por espaços. A compilação será feita à partir da classe principal LangCompiler.

Exemplo: run.sh -bs

4. Estrutura de Dados

A classe principal é a LangCompiler.

O projeto está dividido em dois pacotes: ast e parser. No pacote AST estão os dados necessários para a construção da árvore gerada pela analisador sintático e que não serão foco desse trabalho; a classe MySuperNode foi implementada par que o trabalho compilasse na estrutura fornecida pelo professor da disciplina.

Já no pacote parse, estão as classes responsáveis criação e validação léxica e sintática dos arquivos, além da classe responsável pela realização de testes automatizados. Foram implementadas as classes:

- MyParseAdaptor: responsável por chamar os métodos da biblioteca ANTLR e realizar a análise sintática e lexica do arquivo informado.
- SyntaxError: criada para validar os erros gerados pelo ANTLR

A estrutura de regras da gramática foi escrita no arquivo lang.g4 presente no pacote parser.