Gerador de Código para a Linguagem lang

Dupla

- Eduardo Vieira Marques Pereira do Valle - 201665554C
- Matheus Brinati Altomar - 201665564C

Arquivos Fonte

- TestCompilador.java para testar os arquivos da localizacao testes/semantica/certo e gerar os códigos em java na pasta codigoGeradoJava.
- Foram criados 2 novos arquivos, “JavaVisitor”.java e “java”.stg. O primeiro é um visitor que preenche o template definido no segundo.
- Alem de alterar o arquivo TypeVisitor.java para gerar e disponibilizar as estruturas env e funcoes

Estratégias Utilizadas

- Para reproduzir o comando iterate de lang em Java, utilizamos: um for em que o valor do interator e inicializado com o valor da expressão passado é vai diminuindo até chegar a 0.
- Para a questão de retorno múltiplo, utilizamos duas funções, para cada função que tenha retorno, e um vetor global de objetos para os retornos, enquanto tem um local na primeira função. A primeira é a função de retorno void de lang apenas “traduzida” para Java, enquanto a segunda recebe um parâmetro a mais que é a posição do vetor de retornos. Essa segunda função então chama a primeira, adiciona os retornos ao vetor global de retornos, seleciona o retorno do índice que foi passado, limpa o vetor global de retornos e, por fim, retorna o valor apropriado de tipo Object. Sempre que é visitado um return os valores de sua expressão são inseridos no vetor local da função e depois desse processo são inseridos no vetor global(para impedir que o vetor global fosse alterado antes da avaliação de todas as expressões do retorno).
- Quando ocorre a chamada da função como expr é chamado a sua função auxiliar e seu retorno e transformado para o tipo que deveria ser de acordo com os retornos da função
- Já quando ocorre a chamada da função como comando, se tiverem variáveis para anexar os valores dos retornos elas recebem os valores do vetor global de acordo com o seu tipo, nesse processo é zerado o vetor global.
- Para representar data utilizamos classes com atributos públicos.

- Para a expressão “new” permitimos a criação de vetores mesmo que não fosse especificado seu tamanho, considerando ele sempre com a primeira dimensão 1, caso não fosse um vetor nem um data retornamos o valor default para os atributos primitivos.

Estruturas Utilizadas

- private STGroup groupTemplate para inicializar o template java.stg;
- type, stmt, expr templates para representar tipo, comando e expressão respectivamente;
- List<ST> funcs, params, datas, decls lista de templates para representar funções, parametros, datas e declarações;
- ArrayList<STyFunc> funcoes para poder ter o tipo de retorno correto quando ocorre a chamada de função como expressão;
- String fileName nome do arquivo e da classe;
- HashMap<String, STipo> localEnv para ter o tipo das variáveis locais da função atual;
- ArrayList<HashMap<String, STipo>> env para ter o escopo das funções em ordem de visita do nó função;
- private FileWriter file para representar o arquivo a ser impresso o código.

Compilação

Todos os comandos a seguir devem ser executados dentro do terminal dentro da pasta lang. O primeiro comando para compilar todos os arquivos .java encontrados na pasta do programa seria este:

- javac -cp .:antlr-4.8-complete.jar:ST-4.3.1.jar ast/*.java parser/*.java visitors/*.java tipos/*.java LangCompiler.java -d .

Por fim, basta usar o comando abaixo para executar os arquivos gerados pelo comando anterior, a fim de testar os arquivos encontrados na pasta certo:

- java -classpath .:antlr-4.8-complete.jar:ST-4.3.1.jar lang.LangCompiler -bgc