Interpretador para a Linguagem lang

Dupla

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

Arquivos Fonte

- Fat.lan e DivMod.lan arquivos de teste para testar o interpretador;
- Alteração no arquivo LangCompiler.java para utilizar o interpretador;
- TesteInterpretador.java para rodar bateria de testes do interpretador;
- Criação da pasta certoErrado para conter todos os exemplos que passam do sintático, mas falham no interpretador. Deixando os que estão certos no interpretador na pasta certo.
- lang.g4 arquivo utilizado pela ferramenta para gerar o parser e o analisador léxico;
- Arquivos gerados pela ferramenta ANTLR:
    - lang.interp;
    - lang.tokens;
    - langLexer.interp;
    - langLexer.tokens;
    - langLexer.java;
    - langParser.java;
    - langBaseVisitor.java;
    - langVisitor.java.
- Foram criadas também várias classes que representam a estrutura da AST(todas terminam com .java):
    - Add - para armazenar somas;
    - And - para armazenar AND lógicos;
    - ArraySeletor - para demonstrar o acesso a um elemento de array em determinada posição;
    - Attr - para armazenar atribuições de variáveis;
    - BinOP - classe abstrata para facilitar operações binárias;
    - Btype - classe abstrata para facilitar a definição de tipos;
    - Call - para chamadas de função como comando;
    - CallExpr - para chamadas de função como expressão;
    - Cmd - classe abstrata para facilitar chamadas de comandos;
    - CmdList - para listas de comandos;
    - Data - para armazenar informações do tipo data definido;
    - Decl - para armazenar informações dos tipo de data;

    - DataSeletor - para demonstrar o acesso a um elemento de data com determinado identificador;
    - Div - para armazenar divisões;
    - Eq - para armazenar equivalências;
    - Expr - classe abstrata para facilitar expressões;
    - False - Literal false;
    - Func - para armazenar definição de funções;
    - If - para armazenar condicionais;
    - Iterate - para armazenar laços;
    - LChar - Literal caracter;
    - Lt - para armazenar operação lógica de menos que;
    - Lvalue - para armazenar ID e seus seletores;
    - Mod - para armazenar operações de módulo;
    - Mul - para armazenar operações de multiplicação;
    - Neg - para armazenar negações;
    - NEq - para armazenar desigualdade;
    - New - para declaração dinâmica;
    - NFloat - Literal float;
    - NInt - Literal Inteiro;
    - Not - para negação;
    - Null - para resultados vazios;
    - Param - para parâmetros de funções;
    - Print - para impressões;
    - Program - para inicializar os programas;
    - Read - para leituras;
    - Return - para retornos de funções;
    - Seletor -
    - Sub - para operações de subtração;
    - SuperNode - classe abstrata que deriva todas as outras;
    - Tipo - classe abstrata para identificação de tipos de variáveis;
    - True - Literal True;
    - TyBool - para identificar tipos boolean;
    - TyChar - para identificar tipos char;
    - TyFloat - para identificar tipos float;
    - TyID - para identificar tipos declarados durante a execução;
    - TyInt - para identificar tipos int.
- Na construção dos Visitors, foi criada uma classe abstrata base Visitor.java que declara os métodos de visita de cada uma das classes da AST, citadas acima, a classe VisitorInterpretador.java que implementa todos os métodos definidos por Visitor.java que percorre a árvore de derivação gerada pelo ConstroiASTVisitor.java que define os metodos do Visitor do Antlr para gerar a AST.

Estrategia Utilizada

Para implementar o interpretador utilizamos o padrão de projeto visitor através de uma interpretação direta da árvore AST.

Estruturas Utilizadas

- Foi criado uma variável que representa o ambiente (env), que é uma pilha de hashmaps de Strings (nomes) e Objects (que podem ser qualquer um dos objetos tipos aceitos pela linguagem lang);
- Também foi utilizada uma variável funcs que é uma hashmap de Strings (nomes) e Func (a classe de funções da AST) para armazenar as funções do programa;
- A variável tipos é semelhante ao funcs, mas para armazenar as datas do programa, através de uma hashmap de Strings (nomes) e Data (a classe de funções da AST)
- “retornos” e “operands” sendo para armazenar resultado de avaliações de expressão, para retornos de função através de um ArrayList de objetos e para avaliação de expressões através de uma pilha de objetos respectivamente.

Representação da AST

Nós Abstratos	**Extends**
SuperNode	-
Cmd	SuperNode
Btype	SuperNode
Expr	SuperNode
BinOp	Expr
Seletor	SuperNode

SuperNode	**Nós Abstratos que herdam**	**Nós filhos**
Program	SuperNode	Data e Func

Data	SuperNode	Decl
Decl	SuperNode	Tipo e Lvalue
Tipo	SuperNode	Btype
Lvalue	Expr	Seletor
TyChar	Btype	-
TyInt	Btype	-
TyFloat	Btype	-
TyBool	Btype	-
TyID	Btype	-
ArraySeletor	Seletor	-
DataSeletor	Seletor	-
Func	SuperNode	Param, Tipo(retornos) e Cmd
Param	SuperNode	Tipo
CmdList	Cmd	Cmd
Print	Cmd	Expr
If	Cmd	Expr e Cmd
Iterate	Cmd	Expr e Cmd
Return	Cmd	Expr
Call	Cmd	Expr e Lvalue
Attr	Cmd	Lvalue e Expr
Add	BinOp	Expr
Sub	BinOp	Expr
Div	BinOp	Expr
Mul	BinOp	Expr
Mod	BinOp	Expr
Lt	BinOp	Expr

And	BinOp	Expr
Eq	BinOp	Expr
NEq	BinOp	Expr
Not	Expr	Expr
Neg	Expr	Expr
True	Expr	-
False	Expr	-
Null	Expr	-
NFloat	Expr	-
NInt	Expr	-
LChar	Expr	-
New	Expr	Expr e Tipo
CallExpr	Expr	Expr

Compilação

Todos os comandos a seguir devem ser executados dentro do terminal dentro da pasta lang. O uso do ANTLR adiciona um comando para gerar os arquivos: lang.interp, lang.tokens, langLexer.interp, langLexer.tokens, langLexer.java e langParser.java a partir do arquivo lang.g4, entretanto este comando gera mais arquivos desnecessários que foram removidos, devido a isso enviamos aqueles arquivos ao invés de enviar apenas o lang.g4 e ter mais uma etapa de execução de comandos.

O primeiro comando para compilar todos os arquivos .java encontrados na pasta do programa seria este:

- javac -cp .:antlr-4.8-complete.jar ast/*.java parser/*.java visitors/*.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 errado:

- java -classpath .:antlr-4.8-complete.jar lang.LangCompiler -bsm