Especificação da Fase 2

Bacharelado em Ciência da Computação
Universidade Federal de São Carlos, Campus Sorocaba
Compiladores

1 Segunda Fase

Nesta segunda fase vocês realizarão a análise léxica, sintática e semântica para a linguagem descrita pela gramática da Seção 3.

2 Entrega

Data: 21/06/19.

Entregue um arquivo compactado .zip com o formato de nome: F01_Nome1_Nome2_Nome3_Nome4.zip, contendo apenas os arquivos de extensão .java e testes realizados, respeitando a seguinte estrutura:

- F01_Nome1_Nome2_Nome3_Nome4/AST: Classes da árvore sintática.
- F01_Nome1_Nome2_Nome3_Nome4/Lexer: Classes do analisador léxico.
- F01_Nome1_Nome2_Nome3_Nome4/Testes: Arquivos de teste (Opcional).
- F01_Nome1_Nome2_Nome3_Nome4/Compiler.java: Arquivo principal e analisador sintático.
- F01_Nome1_Nome2_Nome3_Nome4/Main.java: Arquivo que contém a função void main().

Em nome poder ser NomeSobrenome, para diferenciar alunos com nomes iguais.

Use o arquivo Main.java do compilador 9 (da apostila do Zé), que toma como argumento um arquivo com o código a ser analisado e imprime a saída para tela.

Seu Compiler.java deve implementar um método compile(), chamado em Main.java.

Adicione um cabeçalho com nome e RA dos integrantes do grupo em todos os arquivos *.java que forem entregues.

Deve ser possível compilar esta estrutura via linha de comando com javac.

3 Gramática

Na gramática, são terminais:

- Id, um identificador;
- LiteralInt, um número inteiro entre 0 e 2147483647;
- LiteralString, uma sequência de caracteres entre " e ".

Há duas funções pré-definidas, "write" e "writeln", cuja sintaxe é dada pela regra da gramática FuncCall. Elas tomam uma ou mais expressões como argumentos e as imprimem na saída padrão. Apenas expressões dos tipos "Int" e "String" podem ser impressas. "writeln" imprime "\r\n" ao final de todos os argumentos.

Há algumas produções não fatoradas, como `AssignExprStat` (não se sabe se teremos uma atribuição ou apenas uma expressão) e `ExprPrimary` (não se sabe se teremos apenas um Id, uma variável, ou Id seguido de "(", uma chamada de função).

`Program ::= Func` $\{Func\}$

```
Func ::= "function" Id [ "(" ParamList ")" ] ["->" Type ] StatList
ParamList ::= ParamDec {","ParamDec}
ParamDec ::= Id ":" Type
Type ::= "Int" | "Boolean" | "String"
StatList ::= "{" {Stat} "}"
Stat ::= AssignExprStat | ReturnStat | VarDecStat | IfStat | WhileStat
AssignExprStat ::= Expr [ "=" Expr ] ";"
ReturnStat ::= "return" Expr ";"
VarDecStat ::= "var" Id ":" Type ";"
IfStat ::= "if" Expr StatList [ "else" StatList ]
WhileStat ::= "while" Expr StatList
Expr ::= ExprAnd {"or"ExprAnd}
ExprAnd ::= ExprRel {"and"ExprRel}
ExprRel ::= ExprAdd [ RelOp ExprAdd ]
RelOp ::= "<" | "<=" | ">" | ">=" | "==" | "!="
ExprAdd ::= ExprMult {("+" | "-")ExprMult}
ExprMult ::= ExprUnary {("*" | "/")ExprUnary}
ExprUnary ::= [ ( "+" | "-" ) ] ExprPrimary
ExprPrimary ::= Id | FuncCall | ExprLiteral
ExprLiteral ::= LiteralInt | LiteralBoolean | LiteralString
LiteralBoolean ::= "true" | "false"
FuncCall ::= Id "(" [ Expr {","Expr} ] ")"
```

4 Análise Semântica

Um programa nessa gramática deve ter pelo menos uma função chamada "main", que é onde começa a execução. Esta função não deve ter parâmetros nem tipo de retorno.

O tipo do retorno de uma função deve ser dado após "->", sendo opcional. Algumas funções são pré-definidas:

- readInt, que retorna um inteiro lido da entrada;
- readString, que retorna uma linha lida da entrada;
- print(expr), que imprime expr na saída padrão. expr pode ser do tipo Int ou String
- println(expr), mesmo que print(expr) seguido da impressão de \r \n.

Cada variável deve estar associada a um tipo básico específico e só deve receber valores deste tipo.

Quando houver uma atribuição, lembre-se de verificar se os dois lados da expressão tem o mesmo tipo.

Para que sejam utilizadas no código, as variáveis precisam ter sido declaradas e quando forem utilizadas em operações, deve-se respeitar o tipo com que foram declaradas.

Devido a regra do AssignExprStat, podemos ter expressões como:

2 = funcao();

Essas expressões são possíveis sintaticamente, mas são incorretas semanticamente e seu compilador deve emitir erro.

Para que ocorra a chamada de função/procedimento, elas precisam antes terem sido definidas.

A quantidade e tipos de parâmetros na chamada de função/procedimento devem ser iguais aos utilizados na declaração das mesmas.

5 Mensagens de Erro

Nesta fase as mensagens devem ser significativas, principalmente para informar precisamente o erro semântico identificado.

Use o formato o formato a seguir para as mensagens de erro:

\n<nome do arquivo>:<número da linha de erro>:<mensagem de erro>\n<linha do código com erro>

Em <nome do arquivo> não é para conter o caminho do arquivo!

Em <linha do código com erro> pode manter o padrão do compilador, de imprimir o código próximo ao ponto onde o token estava no momento que o erro ocorreu.

O compilador pode continuar lançando exceção java.lang.RuntimeException, mas a mensagem de erro deve estar no formato citado.