DANILO CHAGAS CLEMENTE, LUCAS ALVARENGA LOPES, LARA RAMOS LINHARES, MARCO TÚLIO AMARAL

Relatório técnico

Analisador Léxico

Lavras – MG

02/04/2024

Visão geral da linguagem: Nome e descrição das características gerais da linguagem.

A linguagem ZooLogic adota características inspiradas na linguagem C/C++ em seus quesitos sintáticos e semânticos. Já sobre sua composição lexical, é utilizado termos relacionados ao mundo animal, incluindo habitat, nome de animais e outras características que compõem a fauna e o ambiente.

GitHub do Projeto: https://github.com/Am4ral/ZooLogic/tree/main

Definição Léxica da Linguagem: lexemas aceitos, classes de lexemas e seus padrões de identificação.

Padrão	**Tipo de Lexema**	**Sigla**
Função main()	selva	MAIN
Função	arvore	FUNC
Saída (cout)	lhama	ESC
Controle (return)	desmatamento	RET
Entrada (cin)	porco	LER
Controle (if)	cobra	IF
Controle (else)	cauda	ELSE
Controle (elif)	caudaCobra	ELIF
Tipo de Dado (bool)	boi	TIPO
Tipo de Dado (float)	pato	TIPO
Tipo de Dado (int)	indio	TIPO
Tipo de Dado (string)	centopeia	TIPO
Tipo de Dado (char)	pe	TIPO
Estrutura de Repetição (loop)	formiga	FOR
Estrutura de Repetição	baleia	WHILE
Operadores Aritméticos	*,/,+,-, %	OP_ARIT
Operadores Relacionais	<,<=,>=,>,=,<>	OP_REL

Operador de Atribuição	=	OP_ATR
Operadores Condicionais	&&, ||, ==	OP_COND
Sequência de letras	[a-zA-Z]	fragment LETRA
Sequências de dígitos (sem vírgula)	NÚMERO INTEIRO	fragment DIGITO
Sequências de dígitos (com vírgula)	NÚMERO REAL	NUM
Sequências de letras e números que começam com uma letra	VARIÁVEL	VAR
Delimitador	'('	AP
Delimitador	')'	FP
Delimitador	'{'	AC
Delimitador	'}'	FC
Delimitador	‘;’	PV
Delimitador	‘ “ ‘	AASP
Comentário	‘/ /’	COMEN

Exemplos de uso da linguagem: Implementação dos algoritmos de Fatorial e Soma dos N Termos da Sequência de Fibonacci.

```
indio selva() {
  indio n

  indio fatorial;

  lhama << "Digite um numero (0 <= n <= 12) " << endl;
  porco << n << endl;

  fatorial = 1;
  formiga (indio i = 1; i <= n; i++) {
    fatorial *= i;
  }

    vai dar erro aqui ::::

  cout << n << "!=" << fatoral << endl;

  desmatamento 0;
}
```

```
arvore indio fibonacci(indio n) {
    cobra (n == 0 || n == 1) {
        desmatamento n;
    }
    cauda{
        desmatamento fibonacci(n - 1) + fibonacci(n - 2);
    }
}

arvore indio somaFibonacci(indio n) {
  indio soma = 0;
  formiga (indio i = 0; i < n; i++) {
    soma += fibonacci(i);
  }
  desmatamento soma;
}

selva() {
indio n;

lhama << "Digite um numero ";
porco >> n;

lhama << "O " << n << "numero da sequencia de Fibonacci eh " << fibonacci(n) << endl;

desmatamento 0;
}
```

```
arvore indio somaFibonacci(indio n) {
    indio soma = 0;

    formiga (indio i = 0; i < n; i++) {
        soma += fibonacci(i);
    }

    desmatamento soma;
}


arvore indio fibonacci(indio n) {
    cobra (n == 0 || n == 1) {
        desmatamento n;
    }

    cauda {
        desmatamento fibonacci(n - 1) + fibonacci(n - 2);
    }
}


selva() {
    indio n;

    lhama << "Digite um numero ";
    porco >> n;

    lhama << "A soma dos " << n << " primeiros termos da sequencia de Fibonacci eh " << somaFibonacci(n) << endl;

    desmatamento 0;
}
```

Implementação do Analisador Léxico: Descrição de cada etapa da implementação do analisador léxico, incluindo a criação da gramática e descrição de arquivos no caso de uso de geradores de analisadores, descrição do programa desenvolvido para execução do analisador léxico. São esperadas descrições dos artefatos e capturas de tela.

1. Definição Léxica da Gramática:

Primeiramente, definimos a gramática da linguagem Zoologic, identificando os tokens (lexemas) e suas regras de formação. Isso inclui palavras-chave, símbolos especiais, identificadores, números, etc. Essa definição foi feita em um arquivo *GrammarZooLogic.g4* em classes de tokens e os identificadores foram criados por fragmentos de regra.

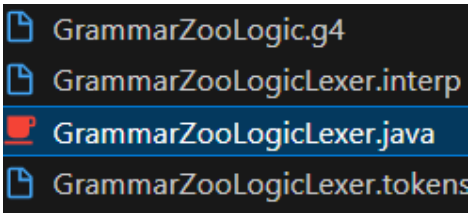
```
grammar GrammarZooLogic;

MAIN: 'selva';
FUNC: 'arvore';
IF: 'cobra';
ELSE: 'cauda';
ELIF: 'caudaCobra';
RET: 'desmatamento';
FOR: 'formiga';
WHILE: 'baleia';
TIPO: 'indio' | 'pato' | 'boi' | 'pe' | 'centopeia';
AP: '(';
FP: ')';
AC: '{';
FC: '}';
ASP: '"';
PV: ';';
COMEN: '//';
ESC: 'lhama';
LER: 'porco';
VAR: LETRA(DIGITO|LETRA)*;
NUM: DIGITO+('.'DIGITO+)?;
fragment DIGITO: [0-9];
fragment LETRA: [a-zA-Z];
OP_ARIT: '+'|'-'|'*'|'/'|'%';
OP_REL: '<'| '>'| '>='| '<='| '==' | '!=';
OP_COND: '&&' | '||';
OP_ATR: '=';
WS: [ \r\t\n]+ ->skip;
ErrorChar: . ;
```

2. Escolha da Ferramenta:

Optamos por utilizar o ANTLR (ANother Tool for Language Recognition) devido à sua eficiência e flexibilidade na geração de analisadores léxicos e sintáticos.

3. Geração do Analisador Léxico:

Utilizamos o ANTLR em Java para gerar o analisador léxico a partir da gramática definida. Passamos o arquivo *GrammarZooLogic.g4* que possui as definições da gramática para o ANTLR e foi gerado um analisador léxico a partir do arquivo *.g4*.



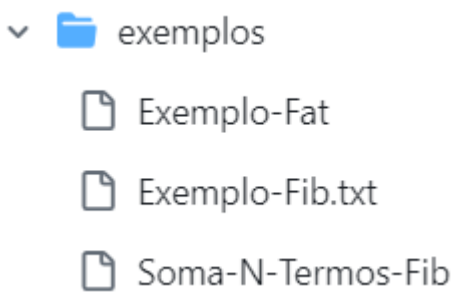
4. Desenvolvimento do Programa de Execução:

Desenvolvemos um programa em Java que utiliza o analisador léxico gerado pelo ANTLR para analisar o código fonte em Zoologic. Este programa pode ler o código fonte, tokeniza-lo usando o analisador léxico e fornecer os tokens reconhecidos como saída.

```java
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.Token;

import java.io.IOException;

public class ExemploLexer {

    public static void  main (String[] args){
        String filename = "F:\\Faculdade\\5 periodo\\Compiladores\\ZooLogic\\exemplos\\Exemplo-Fat";
        try{
            CharStream input = CharStreams.fromFileName(filename);
            GrammarZooLogicLexer lexer = new GrammarZooLogicLexer(input);
            Token token;
            while (!lexer._hitEOF){
                token = lexer.nextToken();
                System.out.println("Token: <Classe: "+lexer.getVocabulary().getSymbolicName(token.getType()) +" ,Lexema: "+ token.getText() +">");
                }

        } catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

5. Teste e Depuração:

Testamos o analisador léxico com três arquivos teste de código em Zoologic para garantir que os lexemas sejam corretamente reconhecidos e classificados. Realizamos depuração conforme necessário para corrigir quaisquer erros ou falhas de reconhecimento.



Casos de Teste: Apresentação do resultado da execução dos algoritmos de exemplo com e sem erros léxicos.

Exemplo Com Erro | Exemplo-Fat:

```
1     indio selva() {
2       indio n
3
4       indio fatorial;
5
6       lhama << "Digite um numero (0 <= n <= 12) " << endl;
7       porco << n << endl;
8
9       fatorial = 1;
10      formiga (indio i = 1; i <= n; i++) {
11        fatorial *= i;
12      }
13
14        vai dar erro aqui ::::
15
16      cout << n << "!=" << fatoral << endl;
17
18      desmatamento 0;
19    }
```

```
Token: <Classe: VAR ,Lexema: fatorial>
Token: <Classe: OP_ARIT ,Lexema: *>
Token: <Classe: OP_ATR ,Lexema: =>
Token: <Classe: VAR ,Lexema: i>
Token: <Classe: PV ,Lexema: ;>
Token: <Classe: FC ,Lexema: }>
Token: <Classe: VAR ,Lexema: vai>
Token: <Classe: VAR ,Lexema: dar>
Token: <Classe: VAR ,Lexema: erro>
Token: <Classe: VAR ,Lexema: aqui>
Token: <Classe: ErrorChar ,Lexema: :>
Token: <Classe: ErrorChar ,Lexema: :>
Token: <Classe: ErrorChar ,Lexema: :>
Token: <Classe: ErrorChar ,Lexema: :>
Token: <Classe: VAR ,Lexema: cout>
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: VAR ,Lexema: n>
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: ASP ,Lexema: ">
Token: <Classe: OP_REL ,Lexema: !=>
Token: <Classe: ASP ,Lexema: ">
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: OP_REL ,Lexema: <>
Token: <Classe: VAR ,Lexema: fatoral>
```

Caso Sem Erro | Exemplo-Fat:

Sem a linha sublinhada em verde, temos o mesmo arquivo sem erro