Alunos : André Penso e Ernesto Taborda
Disciplina: Compiladores
Professor: Cleiton Marques

Trabalho Final – Criação de um Analisador Léxico e Sintático - Ferramentas de Análise - Etapa 1

Resumo

Analisador Léxico (Lexer)

O analisador léxico é a primeira etapa da compilação. Sua função é ler o código-fonte como uma sequência de caracteres e transformá-lo em tokens, que são unidades léxicas significativas como palavras-chave (int, float), identificadores (idade, nome), operadores (=, +) e símbolos (;, ,).

Além disso, o léxico também:

- Ignora espaços, tabulações e comentários.
- Detecta erros léxicos, como caracteres inválidos ou identificadores malformados (ex: 2variavel).
- Classifica cada token com seu tipo e posição (linha e coluna), o que ajuda na análise sintática.

Exemplo de entrada:

int idade = 30;

Tokens gerados:

- INT, IDENTIFICADOR(idade), IGUAL, NUM_INT(30), PVIRG

Analisador Sintático (Parser)

O analisador sintático é a **segunda etapa da compilação**. Ele recebe os tokens do léxico e **verifica se a sequência está de acordo com a gramática da linguagem** — ou seja, se a estrutura do código está correta.

Ele trabalha como um verificador de **ordem e combinação**: não basta que os tokens estejam certos, eles precisam **estar na ordem certa**.

Funções do parser:

- Identifica **declarações válidas** (como int idade = 30;).
- Detecta **erros de sintaxe**, como int = 30;, que é inválido porque não tem um identificador antes do =.
- Garante que regras como **fim com ponto e vírgula**, **lista separada por vírgulas** e **tipos permitidos** estejam sendo respeitadas.

Estrutura dos Arquivos

- **AnalisadorLexico.flex**
  Arquivo que define o analisador léxico. Nele estão especificados os padrões das expressões regulares que reconhecem palavras-chave (int, float, char), identificadores, números, literais e operadores. Também trata erros léxicos, como identificadores inválidos que começam com números.
- **AnalisadorSintatico.cup**
  Contém a gramática usada pelo analisador sintático gerado com CUP. Define os tipos, estruturas de declaração, lista de identificadores, atribuições e valores. É responsável por validar a estrutura das declarações conforme as regras da linguagem.
- **AnalisadorLexico.java e AnalisadorSintatico.java**
  Arquivos gerados automaticamente pelas ferramentas JFlex e CUP, a partir dos arquivos .flex e .cup. Esses arquivos implementam os analisadores propriamente ditos.
- **sym.java**
  Classe que contém os símbolos terminais utilizados pela gramática. Também é gerada automaticamente pelo CUP.
- **TesteLexico.java**
  Classe de teste utilizada para verificar apenas o analisador léxico. Lê o arquivo teste.txt e imprime os tokens identificados com seus valores, linha e coluna.
- **teste.txt**
  Arquivo de entrada com exemplos de declarações de variáveis. É usado para testar o funcionamento dos analisadores.
- **Arquivos .class**
  Arquivos compilados das classes Java (.java) para execução no interpretador da JVM.

- **jflex-full-1.8.1.jar e java-cup-11b.jar**
  Bibliotecas necessárias para a geração e execução dos analisadores. São usadas para processar os arquivos .flex e .cup.

Como Compilar e Executar o Projeto

Pré-requisitos

- Java JDK instalado (recomendado: versão 8 ou superior)
- Terminal ou prompt de comando (cmd, PowerShell, etc.)
- Arquivos java-cup-11b.jar e jflex-full-1.8.1.jar disponíveis na mesma pasta do projeto

Etapa 1 – Gerar o Analisador Léxico com JFlex

java -jar jflex-full-1.8.1.jar AnalisadorLexico.flex

Etapa 2 – Gerar o Analisador Sintático com CUP

java -jar java-cup-11b.jar -parser AnalisadorSintatico -symbols sym AnalisadorSintatico.cup

Etapa 3 – Compilar os Arquivos Java

javac -cp ".;java-cup-11b.jar" AnalisadorLexico.java AnalisadorSintatico.java sym.java TesteLexico.java

Etapa 4 – Rodar o Analisador Léxico

java -cp ".;java-cup-11b.jar" TesteLexico

Etapa 5 – Rodar o Analisador Sintático Completo

java -cp ".;java-cup-11b.jar" AnalisadorSintatico

OBS: Foi necessário inserir um código no final do AnalisadorLexico.java

```
public static void main(String[] args) throws Exception {
    AnalisadorSintatico parser = new AnalisadorSintatico(
        new AnalisadorLexico(new java.io.FileReader("teste.txt"))
    );
    parser.parse();
    System.out.println("✓ Código válido.");
}
```

Explicação do código

new java.io.FileReader("teste.txt")

Abre o arquivo teste.txt, que contém uma ou mais linhas com declarações de variáveis (ex: int x = 10;).

new AnalisadorLexico(...)

Cria o analisador léxico (scanner), que vai ler o texto caractere por caractere e gerar tokens como INT, IDENTIFICADOR, IGUAL, etc.

new AnalisadorSintatico(...)

Cria o analisador sintático (parser), que recebe os tokens gerados pelo léxico e tenta combiná-los com a gramática definida em .cup, verificando se as estruturas estão corretas.

parser.parse();

Chama o método principal do CUP que faz a análise sintática.

Se tudo estiver correto, a execução continua normalmente.

Se houver erro (como falta de ;, uso de tipo inválido, etc.), será lançada uma exceção.

System.out.println(" Código válido.");

Essa linha é exibida somente se o código for sintaticamente correto segundo sua gramática CUP.

Se o código estiver incorreto, ela não será exibida, e será mostrado o erro gerado pelo parser.

As entradas devem ser feitas pelo arquivo teste.txt

Exemplos de Entradas Válidas

```
int x;
float y = 2.5;
char c = 'A';
char nome = "Ana";
int a, b, c;
```

Exemplos de Entradas Inválidas

```
double d;         // tipo não suportado
int = 30;         // faltando identificador
123abc = 10;      // identificador inválido
```

Especificações do Analisador Léxico:

O tipo int aceita qualquer número inteiro positivo (regex: [0-9]+).

O tipo float aceita qualquer número real com casas decimais, sem limite definido. Por exemplo: 3.14, 0.0001, 123.456789.

O tipo char aceita apenas um único caractere entre aspas simples. Exemplo válido: 'A'. Exemplo inválido: 'AB'.

Cadeias de caracteres (para char tipo string) são aceitas com aspas duplas e podem ter qualquer tamanho, mas não há uma limitação definida pelo analisador.