Aula: Análise Léxica em Compiladores

1. Contextualização: O Processo de Compilação

A compilação é o processo pelo qual um código-fonte, escrito em uma linguagem de alto nível, é traduzido para uma representação de baixo nível, geralmente código de máquina. Esse processo é estruturado em fases distintas, sendo a **análise léxica** a primeira etapa do front-end de um compilador. Seu objetivo é converter a sequência de caracteres do código-fonte em uma sequência de **tokens**, que são as unidades léxicas fundamentais para as fases subsequentes.

Formalmente, a análise léxica atua como uma função de mapeamento: dado um fluxo de caracteres C = {c1, c2, ..., cn}, ela produz uma sequência de tokens T = {t1, t2, ..., tm}, onde m ≤ n, eliminando informações irrelevantes, como espaços e comentários.

---

2. Definição e Objetivos da Análise Léxica

A análise léxica é realizada pelo **analisador léxico** (ou *lexer*), um componente que processa o código-fonte caractere por caractere, agrupando-os em tokens conforme regras pré-definidas. Um **token** é uma tupla composta por:

- **Classe léxica**: a categoria do token (ex.: palavra-chave, identificador, operador, literal numérico, símbolo).
- **Lexema**: a sequência exata de caracteres reconhecida (ex.: “while”, “x”, “=”).
- (Opcionalmente) **Atributos**: informações adicionais, como o valor numérico de um literal ou a posição na tabela de símbolos.

O *lexer* também desempenha um papel de filtragem, removendo elementos sintaticamente irrelevantes, como espaços em branco, tabulações e comentários, que não contribuem para a semântica do programa.

Exemplo Prático

Considere o código-fonte:

```
int x = 42;
```

O analisador léxico produz a seguinte sequência de tokens, representados como tuplas (classe léxica, lexema, [atributos opcionais]):

1. (PALAVRA-CHAVE, “int”)
2. (IDENTIFICADOR, “x”)
3. (OPERADOR, “=”)
4. (LITERAL-NUMÉRICO, “42”, 42)
5. (SÍMBOLO, “;”)

---

3. Fundamentos Teóricos e Implementação

O funcionamento do analisador léxico é baseado em **autômatos finitos determinísticos (AFD)**, que modelam o reconhecimento de padrões no código-fonte. Um AFD é definido por:

- Q: conjunto finito de estados.
- Σ: alfabeto de entrada (caracteres do código).
- δ: função de transição (δ: Q × Σ → Q).
- q0: estado inicial.
- F: conjunto de estados finais (que aceitam um token).

Os padrões reconhecidos pelo AFD são descritos por **expressões regulares**, que especificam as classes léxicas. Por exemplo:

- Identificadores: [a-zA-Z][a-zA-Z0-9]*
- Números inteiros: [0-9]+
- Palavras-chave: “if” | “while” | “int” | ...

Exemplo de Autômato

Para reconhecer números inteiros ([0-9]+):

- Q = {q0, q1}
- Σ = {0, 1, 2, ..., 9} ∪ {outros}
- δ(q0, [0-9]) = q1, δ(q1, [0-9]) = q1, δ(q1, outros) = erro

- F = {q1}

Ao processar “123”:

1. q0 → 1 → q1
2. q1 → 2 → q1
3. q1 → 3 → q1
4. q1 (aceitação → token (LITERAL-NUMÉRICO, “123”, 123)).

Na prática, ferramentas como **Lex** e **Flex** convertem expressões regulares em AFDs, gerando código eficiente para o *lexer*.

---

4. Integração com o Processo de Compilação

O analisador léxico funciona como uma interface entre o código-fonte e o **analisador sintático** (*parser*), que consome os tokens para verificar a correção gramatical do programa conforme a gramática formal da linguagem (geralmente uma gramática livre de contexto). O *lexer* é invocado sob demanda pelo *parser*, retornando um token por vez.

Além disso, o *lexer* interage com a **tabela de símbolos**, armazenando identificadores e seus atributos para uso em fases posteriores (ex.: análise semântica). Erros léxicos, como lexemas inválidos (ex.: “@x” em C), são detectados nesta fase, interrompendo a compilação com mensagens de erro.

Exemplo de Erro Léxico

Código: `int 42x = 10;`

- O lexema “42x” viola a expressão regular para identificadores ([a-zA-Z][a-zA-Z0-9]*), pois inicia com um dígito. O *lexer* sinaliza um erro: “identificador inválido”.

---

5. Exercício Prático

Analisem o seguinte trecho de código em C e identifiquem os tokens gerados pelo analisador léxico. Para cada token, especifiquem a classe léxica e o lexema:

```
while (x <= 10) {
    x = x + 1;
}
```

Solução

1. (PALAVRA-CHAVE, “while”)
2. (SÍMBOLO, “(”)
3. (IDENTIFICADOR, “x”)
4. (OPERADOR, “<=”)
5. (LITERAL-NUMÉRICO, “10”, 10)
6. (SÍMBOLO, “)”)
7. (SÍMBOLO, “{”)
8. (IDENTIFICADOR, “x”)
9. (OPERADOR, “=”)
10. (IDENTIFICADOR, “x”)
11. (OPERADOR, “+”)
12. (LITERAL-NUMÉRICO, “1”, 1)
13. (SÍMBOLO, “;”)
14. (SÍMBOLO, “}”)

Este exercício ilustra a segmentação precisa realizada pelo *lexer*, ignorando espaços e quebras de linha.

---

6. Conclusão

A análise léxica é uma etapa essencial no pipeline de compilação, responsável por transformar o código-fonte em uma representação tokenizada, facilitando a análise sintática e semântica subsequentes. Sua implementação combina teoria de autômatos e expressões regulares, oferecendo uma base robusta para o processamento de linguagens formais.

Na próxima aula, exploraremos a **análise sintática**, que utiliza os tokens gerados para construir uma representação estrutural do programa, como uma árvore sintática. Há questões ou aspectos que desejam aprofundar?

Notas Adicionais

- **Literatura Recomendada:** “Compilers: Principles, Techniques, and Tools” (Aho, Sethi, Ullman) – Capítulo 3.
- **Tópicos Avançados (Opcional):** Otimização de AFDs, integração com *lookaheads* no *parser*, tratamento de ambiguidades léxicas.