Compiladores: Análise Léxica

Definição

- É a primeira fase de um compilador.
- É responsável pela leitura da entrada como um fluxo de caracteres convertendo-os num fluxo de *tokens* a ser analisado pelo analisador sintático (*parser*).
- Também chamados de *scanners*, entregam juntamente com cada *token* o seu *lexema* e os atributos associados.

Definição

- Os *tokens* usualmente são conhecidos pelo seu lexema (sequência de caracteres que compõe um único *token*) e atributos adicionais.
- Os *tokens* podem ser entregues ao *parser* como tuplas na forma <a, b,..., n > assim a entrada:

  a = b + 3

- poderia gerar as tuplas:

  <**id**, a> <=,> <**id**, b> <+,> <**num**, 3>

- note que alguns *tokens* não necessitam atributos adicionais.

Objetivos

- Remoção de espaços em branco e comentários.
- Identificação de constantes.
- Identificação de palavras-chave (construções da linguagem).
- Reconhecimento dos identificadores.
- Separação do *parser* da representação da entrada.

Remoção de Espaços em Branco e Comentários

- Espaços em branco (*whitespaces*) são os brancos, tabulações e quebras de linha.
- Comentários são textos inseridos no programa para melhorar seu entendimento.
- Se o analisador léxico "elimina" os espaços em branco e os comentários, isto é, não os envia para o *parser*, então:
  - o *parser* não precisa efetuar seu tratamento (o que representa alguma otimização) e
  - gramática não precisa considerar sequer sua existência (tal inclusão é bastante trabalhosa).

Identificação de Constantes

- Embora dígitos possam ser repassados separadamente para o *parser*, sabemos de antemão que dígitos agrupados tem significado distinto.
- Simplificado a gramática do *parser* o analisador léxico pode identificar e agrupar os dígitos encontrados como unidades autônomas entregando ao *parser* um *token* especial (p.e. **NUM** ou **VAL**) e seu valor.

Identificação de Palavras-Chave

- A grande maioria das linguagens utiliza cadeias fixas de caracteres como elementos de identificação de construções particulares da linguagem (comandos de decisão, seleção, repetição, pontuação etc.).
- Tais cadeias são as chamadas palavras-chave da linguagem e também devem ser identificadas pelo analisador léxico.

Reconhecimento de Identificadores

- Variáveis, vetores, estruturas, classes etc. são usualmente distinguidas entre si através de identificadores, ou seja, de nomes arbitrários designados pelo programador.
- Os identificadores usualmente possuem regras de formação e devem ser distinguidos das palavras-chave da linguagem (tal tarefa é bastante simplificada quando as palavras-chave são reservadas).

Separação do *Parser* da Entrada

- Como o analisador léxico efetua as seguintes operações:
  - leitura caracteres da entrada;
  - determinação dos lexemas e seus atributos;
  - entrega dos *tokens* (lexemas + atributos) para o *parser*.
- Então o *parser* não precisa conhecer detalhes sobre a entrada (origem, alfabeto, símbolos especiais, operadores complexos).

Tipos de *Tokens*

- As linguagens de programação usualmente distinguem certos tipos ou classes de *tokens*:
  - palavras-chave
  - operadores
  - identificadores
  - constantes
  - literais
  - símbolos de pontuação

Tokens, Padrões e Lexemas

- Um mesmo *token* pode ser produzido por várias cadeias de entrada.
- Tal conjunto de cadeias é descrito por uma regra denominada *padrão*, associada a tais *tokens*.
- O padrão reconhece as cadeias de tal conjunto, ou seja, reconhece os *lexemas* que são padrão de um *token*.

Tokens, Padrões e Lexemas

- A declaração C seguinte:

  int k = 123;

- Possui vária subcadeias:
  - *int* é o lexema para um *token* tipo **palavra-reservada**.
  - = é o lexema para um *token* tipo **operador**.
  - *k* é o lexema para um *token* tipo **identificador**.
  - *123* é o lexema para um *token* do tipo número **literal** cujo atributo valor é 123.
  - ; é o lexema para um *token* tipo **pontuação**.

Padrões

- São regras que descrevem o conjunto possível de lexemas que podem representar uma certa classe ou tipo de *token*.
- Em alguns casos o padrão é extremamente simples e restritivo:
  - o padrão para o *token* **float** é a própria cadeia "float", assim como **int**, **char**, **double** ou **void**.
- Em outros é um conjunto de valores:
  - o padrão para operadores relacionais em C é o conjunto {<, <=, >, >=, == e !=}

Padrões

- Usualmente os padrões são convenções determinadas pela linguagem para formação de classes de *tokens*:
  - **identificadores**: letra seguida por letras ou dígitos.
  - **literal**: cadeias de caracteres delimitadas por aspas.
  - **num**: qualquer constante numérica.

Reconhecimento de *Tokens*

- Existe considerável dificuldade no reconhecimento dos *tokens* conforme a linguagem:
  - Fortran:

    DO 5 I = 1.25

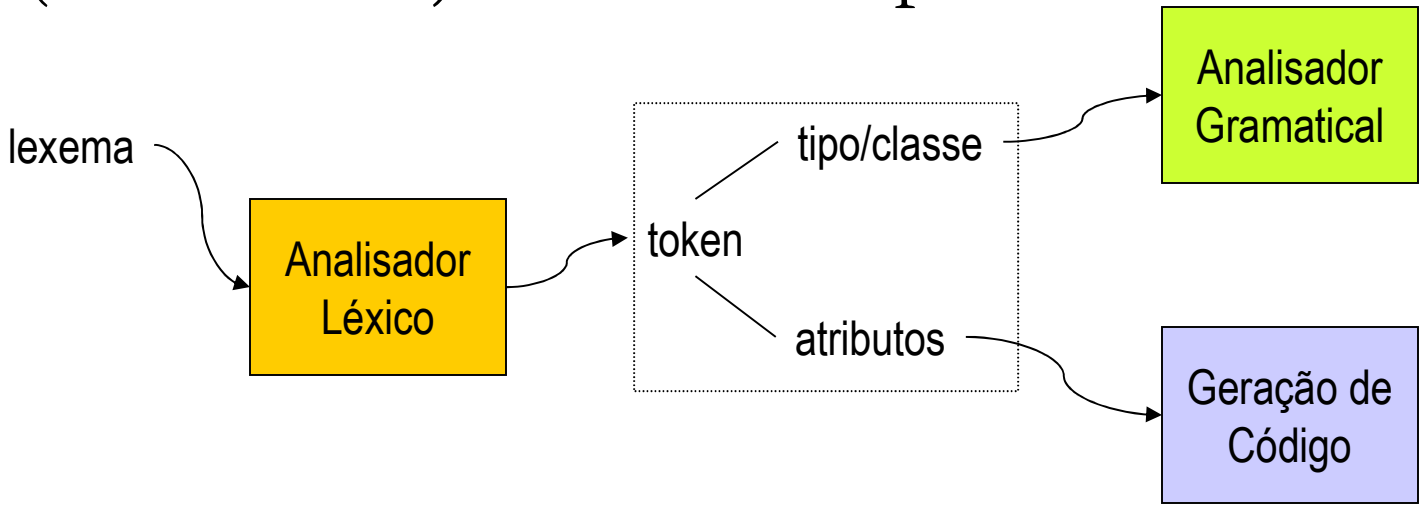
    DO 5 I = 1,25
  - PL/I:

    if then then then = else; else else = then;
  - C:

    a = b+++c;

Atributos dos *Tokens*

- Quando um lexema é reconhecido por um padrão, a analisador léxico deve providenciar outras informações adicionais (os atributos) conforme o padrão.



Atributos dos *Tokens*

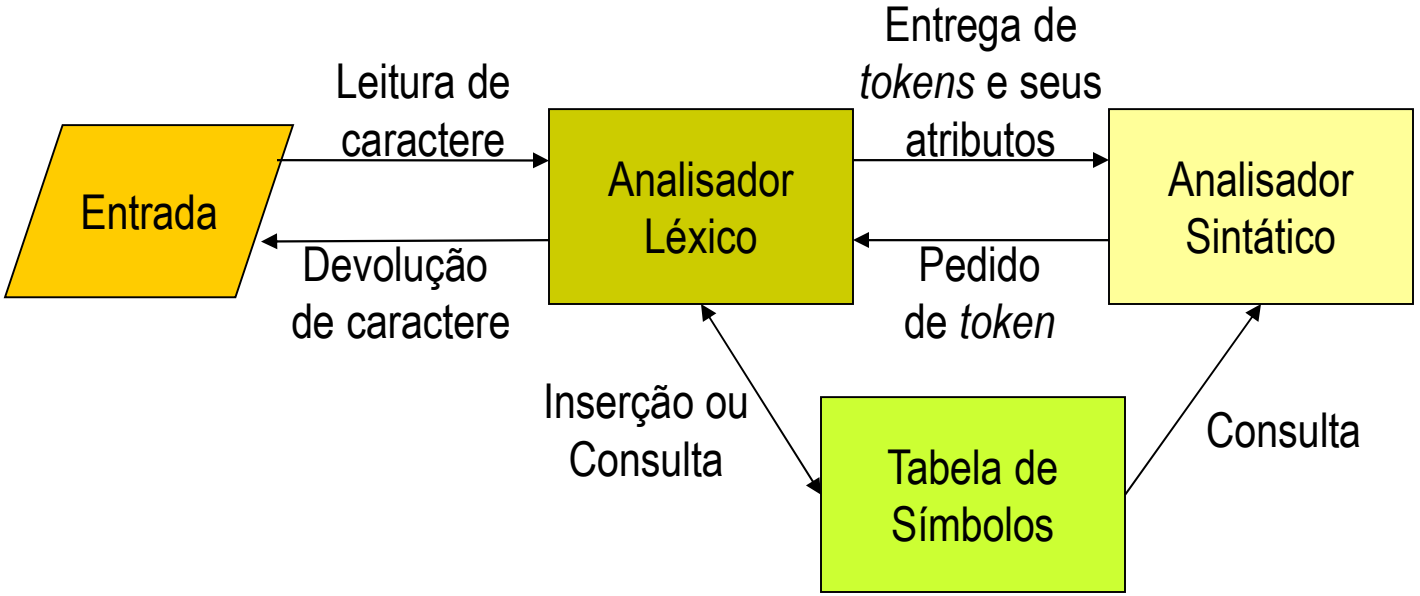
- Para a análise gramatical (sintática) basta conhecer-se o tipo do *token* (p.e. **NUM**).
- Já na fase de tradução (geração de código) o valor do *token* (seu atributo) é essencial.
- Na prática os *tokens* tem um único atributo que é um apontador para sua entrada na tabela de símbolos, onde as demais informações são armazenadas.

Estrutura Funcional

- Podem ser implementados num único passo (que efetua tanto a leitura da entrada como a separação dos *tokens* e seus atributos).
- Podem ser implementados em dois passos: um de varredura ou *scanning* (que efetua a leitura da entrada) e outro de análise léxica propriamente dita (que separa os *tokens* e seus atributos).

Interação do Analisador Léxico e o *Parser*

- Um analisador léxico interage com a entrada e com o *parser* (analisador sintático ou gramatical):



Divisão entre Análise Léxica e Análise Gramatical

Justifica-se tal divisão pois:

- Permite a simplificação do projeto de uma destas duas fases. Por exemplo:
  - É mais simples a extração de comentários e brancos pelo analisador léxico que sua incorporação na gramática.
- Leva a um projeto global de linguagem mais simples.

Divisão entre Análise Léxica e Análise Gramatical

- Permite tratar com maior eficiência as operações realizadas sobre a entrada através do uso de técnicas especializadas.
- Maior portabilidade pois peculiaridades do alfabeto de entrada, anomalias da entrada e representações especiais são isoladas na análise léxica.

Características do Funcionamento

- Em algumas situações o analisador léxico não consegue determinar o *token* apenas com o caractere corrente.
- Operadores relacionais do tipo <, <=, >, >=, != e == exigem a leitura de um caractere adicional.
- Quando tal caractere é o '=' então o par de caracteres lido é o lexema do *token*, caso contrário tal caractere deve ser "devolvido" a entrada pois pertence ao próximo *token*.

Características do Funcionamento

- O analisador léxico e o *parser* formam um par produtor-consumidor.
- Os *tokens* podem ser guardados num *buffer* até que sejam consumidos permitindo que tanto o analisador léxico quanto o *parser* pudessem ser executados em paralelo.
- Usualmente o *buffer* armazena apenas um *token* (simplificando a implementação) de forma que o analisador léxico opere apenas quando o *buffer* estiver vazio (operação por demanda).

Características de Funcionamento

- É muito conveniente que a entrada seja buferizada, i.e., que a leitura da entrada seja fisicamente efetuada em blocos enquanto que logicamente o analisador léxico consome apenas um caractere por vez.
- A buferização facilita os procedimentos de devolução de caracteres.

Erros Léxicos

- Poucos erros podem ser detectados durante a análise léxica dada a visão restrita desta fase, como mostra o exemplo:

  fi (a == f(x)) *outro_cmd*;

  - *fi* é palavra chave **if** grafada incorretamente?
  - *fi* é um identificador de função que não foi declarada faltando assim um separador (';') entre a chamada da função *fi* e o comando seguinte (*outro_cmd*)?

Erros Léxicos

- Ainda assim o analisador léxico pode não conseguir prosseguir dado que a cadeia encontrada não se enquadra em nenhum dos padrões conhecidos.
- Para permitir que o trabalho desta fase prossiga mesmo com a ocorrência de erros deve ser implementada uma estratégia de recuperação de erros.

Recuperação de Erros Léxicos

- Ações possíveis:
  - (1) remoção de sucessivos caracteres até o reconhecimento de um *token* válido (modalidade pânico).
  - (2) inserção de um caractere ausente.
  - (3) substituição de um caractere incorreto por outro correto.
  - (4) transposição de caracteres adjacentes.

Recuperação de Erros Léxicos

- Tais estratégias poderiam ser aplicadas dentro de um escopo limitado (denominado erros de distância mínima).

  ... while (a<100) { fi (a==b) break else a++; } ...

- Estas transformações seriam computadas na tentativa de obtenção de um programa sintaticamente correto (o que não significa um programa correto, daí o limitado número de compiladores experimentais que usam tais técnicas).

Enfoques de Implementação

- Existem 3 enfoques básicos para construção de um analisador léxico:
  - Utilizar um gerador automático de analisadores léxicos (tal como o compilador LEX, que gera um analisador a partir de uma especificação).
  - Escrever um analisador léxico usando um linguagem de programação convencional que disponha de certas facilidades de E/S.
  - Escrever um analisador léxico usando linguagem de montagem.

Enfoques de Implementação

- As alternativa de enfoque estão listadas em ordem crescente de complexidade e (infelizmente) de eficiência.
- A construção via geradores é particularmente adequada quando o problema não esbarrar em questões de eficiência e flexibilidade.
- A construção manual é uma alternativa atraente quando a linguagem a ser tratada não for por demais complexa.