Paradigmas de Programação

Aula 02 – Introdutória sobre Linguagens de Programação

Professora: Milene Serrano

Agenda

- √ **Aula de HOJE:**
  - √ Introdução ao Conceito de **Paradigma**;
  - √ Linguagens de Programação;
  - √ Conceitos Matemáticos em Linguagens de Programação;
  - √ Representação em Linguagens de Programação;
  - √ Hierarquia de Linguagens de Programação;
  - √ Especificação de Linguagens de Programação;
  - √ Analisadores e Outros em Linguagens de Programação;
  - √ Considerações Finais;
  - √ Exercícios, e
  - √ Leitura Complementar.

- √ **Próxima Aula:**
  - √ Visão Geral sobre Paradigmas de Programação.

Introdução ao Conceito de Paradigma

Introdução ao Conceito de Paradigma

"Um modelo conceitual que permite enxergar o mundo ou focar em um dado problema de forma específica, orientada por conceitos e princípios bem definidos, padronizados".

Agora, Paradigma de Programação

"Um padrão/modelo conceitual que orienta soluções de projeto e implementação".

- ✓ Ou seja, uma forma ou base para se estabelecer como os elementos que compõem um programa estão organizados e interagem entre si.

Enxergar o mundo e representá-lo em um modelo visando a sua futura implementação ...

Linguagens de Programação

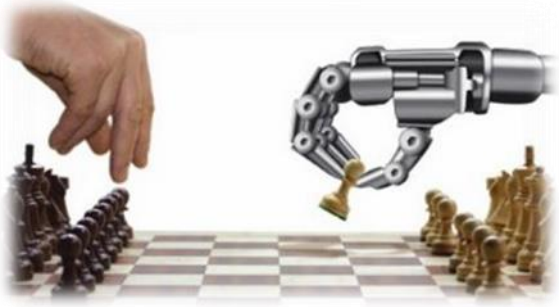
Primeiramente, Linguagem...

"Acredita-se que a profundidade de nossa capacidade intelectual seja influenciada pelo poder expressivo da linguagem em que comunicamos nossos pensamentos." [Sebesta]

- ✓ Ou seja, a linguagem é uma forma de comunicação...

Linguagens de Programação

- Na Engenharia de Software, uma Linguagem de Programação representa um conjunto de recursos que podem ser compostos para construir programas, permitindo a comunicação Homem-Máquina.

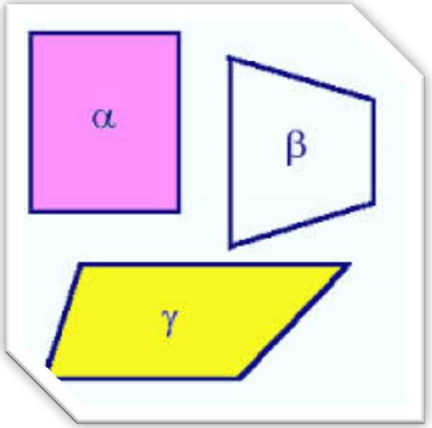


- Além disso, uma Linguagem de Programação estabelece um conjunto de regras para lidar com essa composição de recursos, conduzindo para a produção de programas de qualidade.

Conceitos Matemáticos em Linguagens de Programação

Conceitos Matemáticos

- Alguns conceitos matemáticos associados à definição de uma Linguagem de Programação:
  - Símbolo;
  - Sentença;
  - Tamanho de uma Sentença, e
  - Alfabeto.

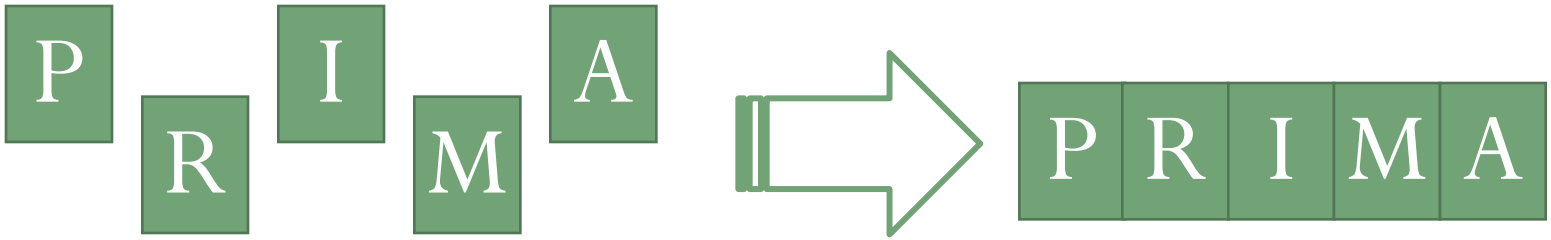


Conceitos Matemáticos

- Um símbolo é uma entidade abstrata básica, sem definição formal.
  - Exemplos: letras, dígitos, e outros.
- Símbolos são ordenáveis lexicograficamente. Portanto, podem ser comparados com base em igualdade ou precedência.
- Por exemplo:
  - As letras do alfabeto são ordenadas: A < B < C < ... < Z.
- Utilidade:
  - Usá-los como elementos atômicos em definições de linguagens.

Conceitos Matemáticos

- Uma sentença é uma sequência finita de símbolos.
- Por exemplo:



Símbolos **Sentença**

- Vale ressaltar que as sentenças vazias são sentenças constituídas por nenhum símbolo.

Conceitos Matemáticos

- O tamanho de uma sentença w, denotado |w|, é dado pelo número de símbolos que compõem w.
- Portanto, o tamanho da sentença PRIMA é 5.
- E o tamanho da sentença vazia é 0.

Conceitos Matemáticos

- Um alfabeto, denotado por V, é um conjunto finito de símbolos.
- Assim, considerando os símbolos, por exemplo, dígitos e letras, tem-se os seguintes alfabetos:
  - Vbinário = {0, 1};
  - Vvogais = {a, e, i, o, u}.
- Um conjunto vazio também pode ser considerado um alfabeto.
- O **fechamento reflexivo** de um alfabeto V, denotado por V*, é o conjunto de todas as sentenças que podem ser formadas com os símbolos de V, inclusive a sentença vazia.

Conceitos Matemáticos

- O **fechamento transitivo** de um alfabeto V, denotado por V^+, é dado por $V^* - \{\varepsilon\}$.

- Seja V o alfabeto dos dígitos binários, V = {0, 1}:

  - O **fechamento reflexivo** de V é:

    $V^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...\}$.

  - O **fechamento transitivo** de V é:

    $V^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...\}$

Representação em Linguagens de Programação

Representação

- Uma linguagem L é um conjunto de sentenças formadas por símbolos tomados de algum alfabeto V.

- Assim, por exemplo, o conjunto de sentenças válidas da língua portuguesa poderia ser definido como um subconjunto de {a, b, c,..., z}$^{+}$.

Representação

- Uma linguagem é **finita**, quando é composta por um conjunto finito de sentenças.

- Seja V = {a, b} um alfabeto, **L1** e **L2** linguagens definidas conforme segue:
  - **L1** = {w | w ϵ V* ∧ |w| < 3}, ou seja, L1 é uma linguagem constituída por todas as sentenças de tamanho menor que 3 formadas por símbolos de V. Portanto:

    L1 = {ε, a, b, aa, ab, ba, bb}

  - **L2** = {ε} **//linguagem constituída pela sentença vazia**

Representação

- Uma linguagem é **infinita**, quando é composta por um conjunto infinito de sentenças.

- Seja V = {a, b} um alfabeto, L1 e L2 linguagens definidas conforme segue:
  - L1 = {w | w ϵ V* ^ |w| MOD 2 = 0}, ou seja, L1 é uma linguagem constituída por todas as sentenças de tamanho par formadas por símbolos de V. Portanto, L1 = {ε, aa, ab, ba, bb, aaaa, aaab, aaba, ...}

PS: MOD é uma operação que representa o resto da divisão inteira.

  - L2 = {w | w é um palíndromo}. Portanto L2 = {ε, a, b, aa, bb, aba, baab, ...}

PS: Palíndromo é uma palavra que tem a mesma leitura da esquerda para a direita e vice-versa.

Representação

- Como representar uma linguagem **finita**?
  - Uma linguagem finita, composta por um conjunto finito de sentenças, pode ser definida através da enumeração das sentenças constituintes ou através de uma descrição algébrica.

- Como representar uma linguagem **infinita**?
  - Uma linguagem infinita, composta por um conjunto infinito de sentenças, deve ser definida, idealmente, através de **representação finita**.

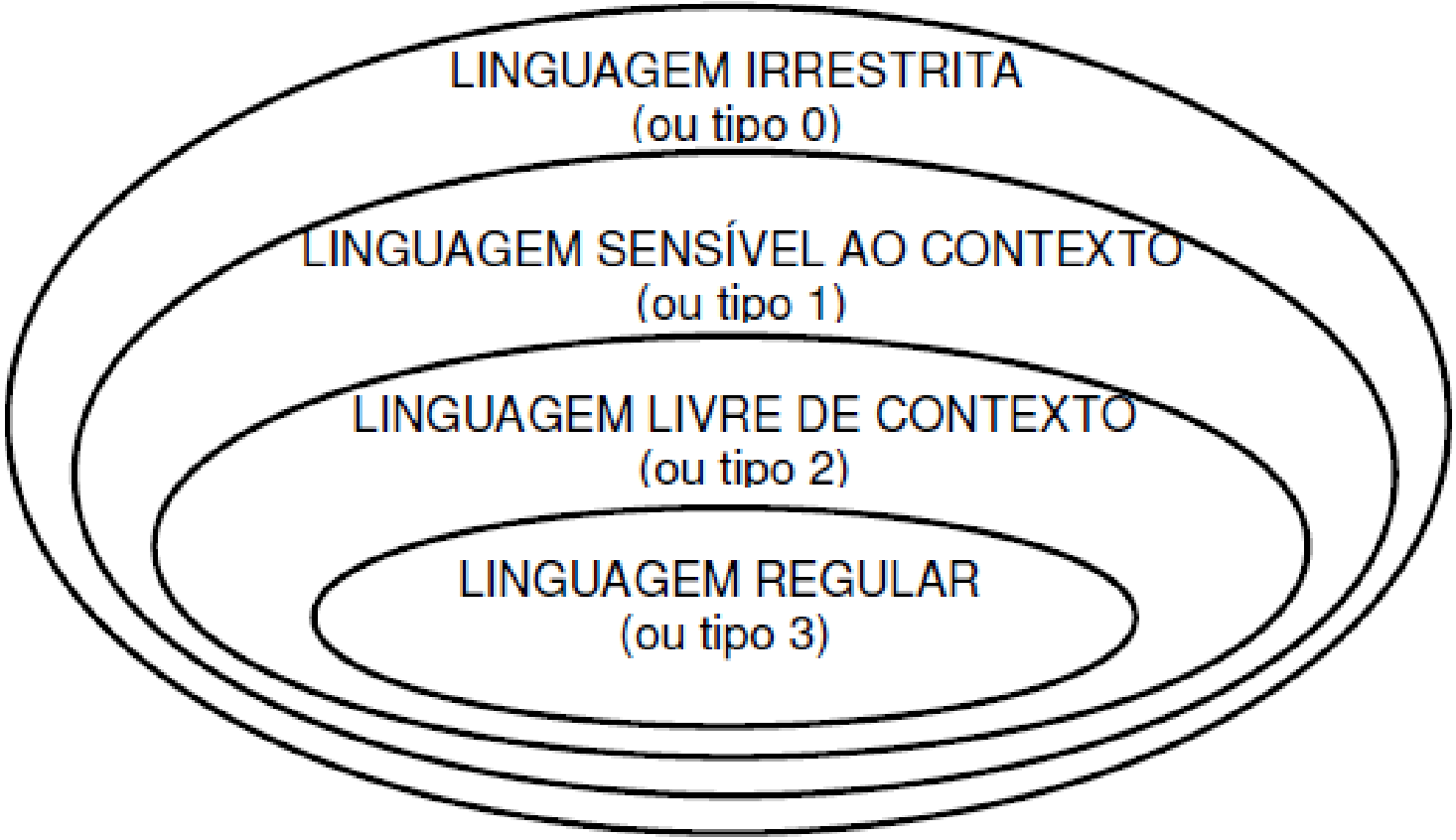
Representação

- **Reconhecedores** ou **Sistemas Geradores** são dois tipos de representações finitas.
  - Um reconhecedor é um dispositivo formal usado para verificar se uma determinada sentença pertence ou não à linguagem.
    - **Exemplos:** autômatos finitos, autômatos de pilha e Máquinas de Turing.
  - Um sistema gerador é um dispositivo formal usado para gerar de forma sistemática as sentenças de uma dada linguagem.
    - **Exemplos:** gramáticas.
- Todo reconhecedor e todo sistema gerador pode ser representado por um algoritmo.

Hierarquia de Linguagens de Programação

Hierarquia

- Noam Chomsky definiu uma hierarquia de linguagens como modelos para linguagens naturais:



Hierarquia

- As **linguagens regulares** constituem um conjunto de linguagens bastante simples. Essas linguagens podem ser reconhecidas por autômatos finitos, geradas por gramáticas regulares, e facilmente descritas por expressões simples, chamadas expressões regulares.

- Segundo MENEZES (1997), algoritmos para reconhecimento e geração de linguagens regulares são de grande eficiência, fácil implementação e pouca complexidade.

- HOPCROFT & ULLMAN (1979) afirmam que vários são os problemas cujo desenvolvimento pode ser facilitado pela conversão da especificação com base em termos de expressões regulares.

Hierarquia

- A importância das **linguagens livres de contexto** reside no fato de que especificam adequadamente as estruturas sintáticas das linguagens de programação.

- Essas linguagens podem ser reconhecidas por autômatos de pilha e geradas por gramáticas livre de contexto (GLC).

- A maioria das linguagens de programação pertence ao conjunto das linguagens livre de contexto e pode ser analisada por algoritmos eficientes.

Hierarquia

- Segundo MENEZES (1997), as **linguagens sensíveis ao contexto** e **linguagens irrestritas** *"permitem explorar os limites da capacidade de desenvolvimento de reconhecedores ou geradores de linguagens"*.

- Essas linguagens podem ser, respectivamente, reconhecidas por Máquinas de Turing limitadas e geradas por gramáticas sensíveis ao contexto(GSC), e reconhecidas por Máquinas de Turing e geradas por gramáticas irrestritas.

Hierarquia

Linguagens sem restrições. Existem versões de Cálculo Lambda que exploram tais aspectos...

Linguagens voltadas para lidar com problemas de decisão do tipo PSPACE-completo...

Teoria de autômatos: linguagem formal e gramática formal			
Hierarquia Chomsky	**Gramática**	**Linguagem**	**Reconhecedor**
Tipo-0	Irrestrita	Recursivamente enumerável	Máquina de Turing
--	--	Recursiva	Máquina de Turing que sempre para
Tipo-1	Sensível ao contexto	Sensível ao contexto	Autômato linearmente limitado
Tipo-2	Livre de contexto	Livre de contexto	Autômato com pilha
Tipo-3	Regular	Regular	Autômato finito

As mais simples de todas, cuja solução é mais direta e óbvia...

Linguagens de programação em geral ...

Especificação de Linguagens de Programação

Especificação

- √ Já sabemos que uma linguagem L é qualquer subconjunto de sentenças sobre um alfabeto V.
- √ Mas, qual subconjunto é esse? Como defini-lo?
- √ Uma gramática é um dispositivo formal usado para definir qual subconjunto de V* forma determinada linguagem.
- √ Portanto, uma gramática define uma estrutura sobre um alfabeto de forma a permitir que apenas determinadas combinações de símbolos sejam consideradas sentenças.

Especificação

√ O que é GRAMÁTICA?

"É um sistema gerador de linguagens; é um sistema de reescrita; é uma maneira finita de descrever uma linguagem; é um dispositivo formal usado para especificar de maneira finita e precisa uma linguagem infinita."

Especificação

- Formalmente, uma gramática G é definida como sendo uma quádrupla G = (N, T, P, S), onde:
  - N é um conjunto finito de símbolos denominados símbolos não-terminais, usados na descrição da linguagem;
  - T é um conjunto finito de símbolos denominados símbolos terminais, os quais são os símbolos propriamente ditos;
  - P é conjunto finito de pares (α,β) denominados regras de produção (ou regras gramaticais) que relacionam os símbolos terminais e não-terminais;
  - S é o símbolo inicial da gramática pertencente a N, a partir do qual as sentenças de uma linguagem podem ser geradas.

Especificação

- √ As regras gramaticais são representadas por $\alpha ::= \beta$ ou $\alpha \rightarrow \beta$, onde α e β são sentenças sobre V, com α envolvendo pelo menos um símbolo pertencente a V_N.

- √ O significado de uma regra de produção $\alpha \rightarrow \beta$, é α produz β ou α é definido por β.

- √ Uma sequência de regras de produção da forma:
  - √ $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, ..., \alpha \rightarrow \beta_n$

- √ Podemos abreviar como uma única produção na forma:
  - √ $\alpha \rightarrow \beta_1 | \beta_2 | ... B_n$

Especificação

- Exemplo, a linguagem dos números inteiros (sem sinal) é gerada pela seguinte gramática

  G:

  $G = (N, T, P, S)$ — **Quádrupla**

  onde

  $V_N = \{N, D\}$ — **Símbolos Não-Terminais**

  $V_T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ — **Símbolos Terminais**

  $P = \{$

  Regras de Produção

  $N \rightarrow D\ N \mid D$

  $D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

  $\}$

  $S = N$ — **Símbolo Inicial da Gramática**

Analisadores e Outros em Linguagens de Programação

Analisadores e Outros

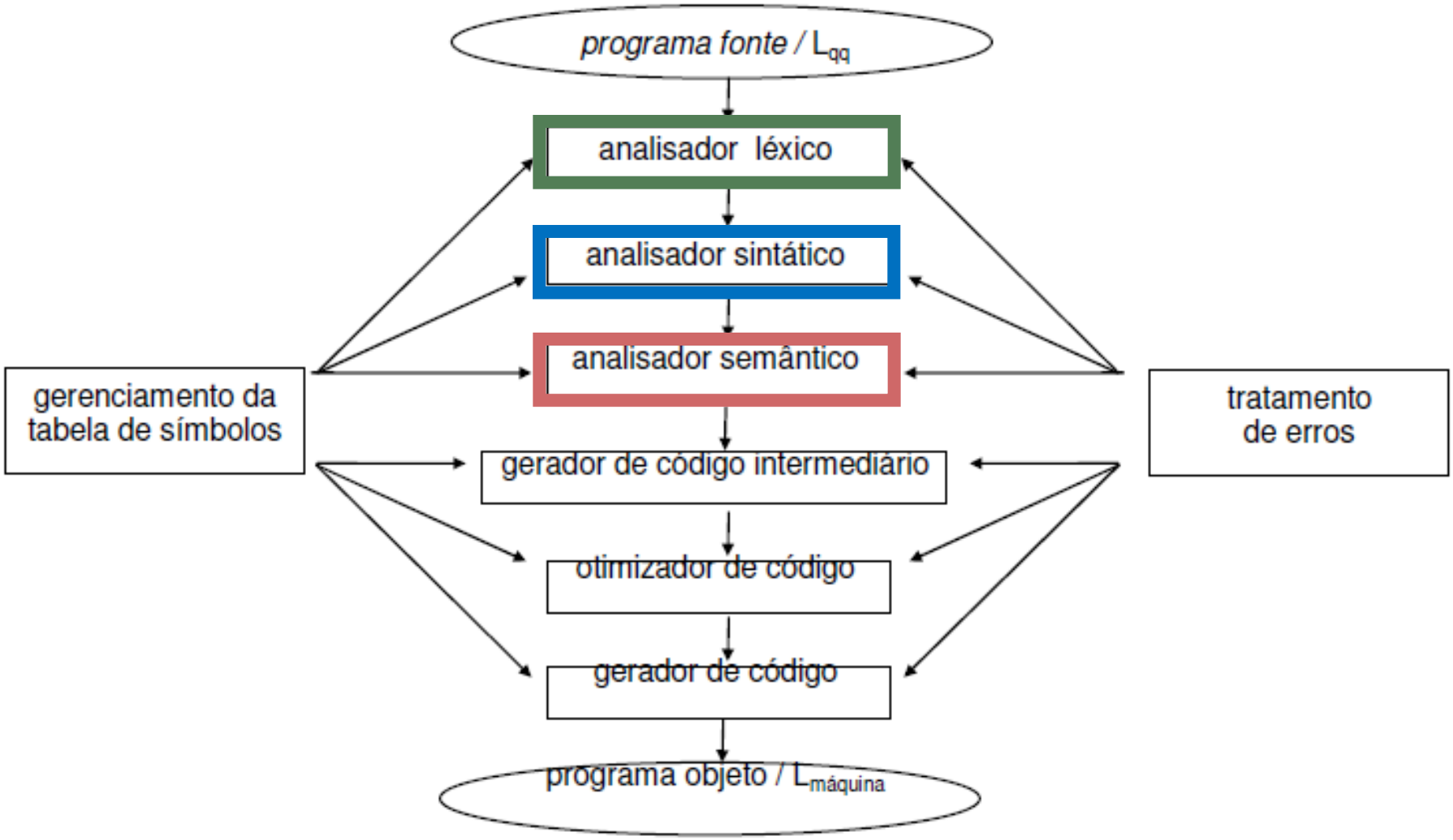
- O objetivo de um compilador é traduzir as sequências de caracteres que representam o programa fonte em código executável, ou seja, traduz um programa escrito em linguagem fonte para um programa escrito em linguagem de máquina.

Analisadores e Outros

- ✓ Nessa tradução, dentre outros detalhes, são necessários:

  - Analisadores Léxicos;
  - Analisadores Sintáticos, e
  - Analisadores Semânticos.

Analisadores e Outros



Analisadores e Outros

- O **analisador léxico** separa a sequência de caracteres que representa o programa fonte em entidades ou *tokens*, símbolos básicos da linguagem.

- Durante a análise léxica, os *tokens* são classificados como palavras reservadas, identificadores, símbolos especiais, constantes de tipos básicos (ex. inteiro real, literal, dentre outros), entre outras categorias.

Analisadores e Outros

- Considere a sequência de caracteres:

SOMA := SOMA + 35

- Esses caracteres podem ser agrupados, pelo analisador léxico, em 5 entidades:

(VALOR)	(CLASSE)
SOMA	identificador
:=	comando de atribuição
SOMA	identificador
+	operador aritmético de adição
35	constante numérica inteira

- Um *token* consiste de um par ordenado (valor, classe). A classe indica a natureza da informação contida em valor.

- Outras funções atribuídas ao analisador léxico são: ignorar espaços em branco e comentários, e detectar erros léxicos.

Analisadores e Outros

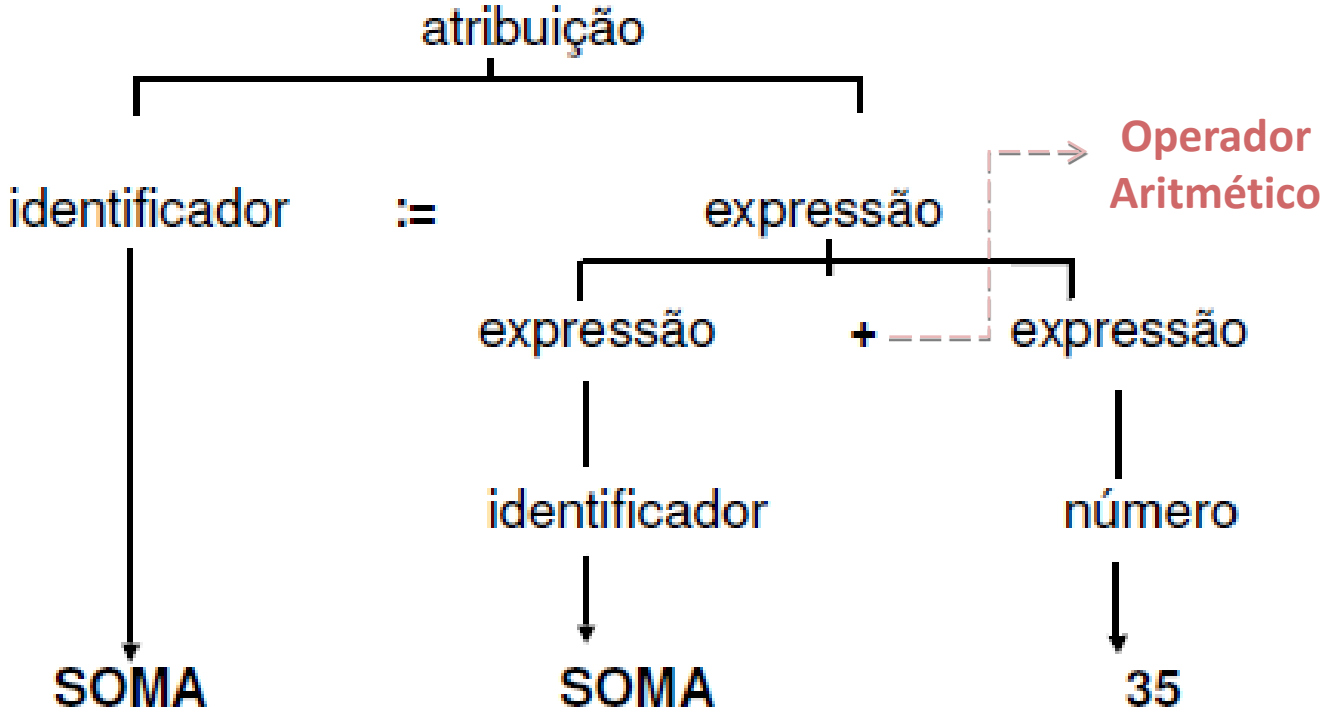
- O **analisador sintático** agrupa os *tokens* fornecidos pelo **analisador léxico** em estruturas sintáticas, construindo a árvore sintática correspondente.

- Para isso, utiliza uma série de regras de sintaxe, que constituem a gramática da linguagem fonte.

- É a gramática da linguagem que define a estrutura sintática do programa fonte.

Analisadores e Outros

- Por exemplo, para a lista de *tokens* exemplificados no **analisador léxico** anterior, o **analisador sintático** construiria a árvore de derivação:



Analisadores e Outros

- O analisador sintático tem também por tarefa o reconhecimento de erros sintáticos, que são construções do programa fonte que não estão de acordo com as regras de formação de estruturas sintáticas como especificado pela gramática.

Analisadores e Outros

- O **analisador semântico** utiliza a árvore sintática determinada pelo **analisador sintático** para:
  - identificar operadores e operandos das expressões;
  - reconhecer erros semânticos;
  - fazer verificações de compatibilidade de tipo;
  - analisar o escopo das variáveis, e
  - fazer verificações de correspondência entre parâmetros atuais e formais.

Analisadores e Outros

- Por exemplo, para o comando de atribuição **SOMA:= SOMA + 35**, é necessário fazer a seguinte análise:
  - **O identificador SOMA foi declarado?**
    - Em caso negativo, erro semântico.
  - **O identificador SOMA é uma variável?**
    - Em caso negativo, erro semântico.
  - **Qual o escopo da variável SOMA? Local ou Global?**
    - Essa informação foi especificada?
  - **Qual o tipo da variável SOMA?**
    - O valor atribuído no lado direito do comando de atribuição é compatível?

Analisadores e Outros

- ✓ Não existe uma fronteira definida entre o que deve ser tratado pelo **analisador sintático** e o que deve ser tratado pelo **analisador semântico**, cabendo ao programador do compilador a escolha, segundo suas preferências (*expertise*).

Analisadores e Outros

- Um **gerador de código intermediário** gera uma versão intermediária do programa fonte.

- Por exemplo, considerando a atuação do **gerador de código intermediário** que gera uma instrução para cada operador na árvore sintática anteriormente apresentada, temos, como possível resultado, o **código intermediário**:

  $temp_1$:= 35

  $temp_2$:= SOMA + $temp_1$

  SOMA:= $temp_2$

Analisadores e Outros

- Um **otimizador de código** melhora o código intermediário de tal forma que o programa objeto resultante seja mais rápido em tempo de execução.

- Por exemplo, considerando o código intermediário anteriormente comentado...

- O código intermediário:

  $temp_1$:= 35
  $temp_2$:= SOMA + $temp_1$
  SOMA:= $temp_2$

ANTES DA OTIMIZAÇÃO

- Poderia ser otimizado para:

  SOMA:= SOMA + 35

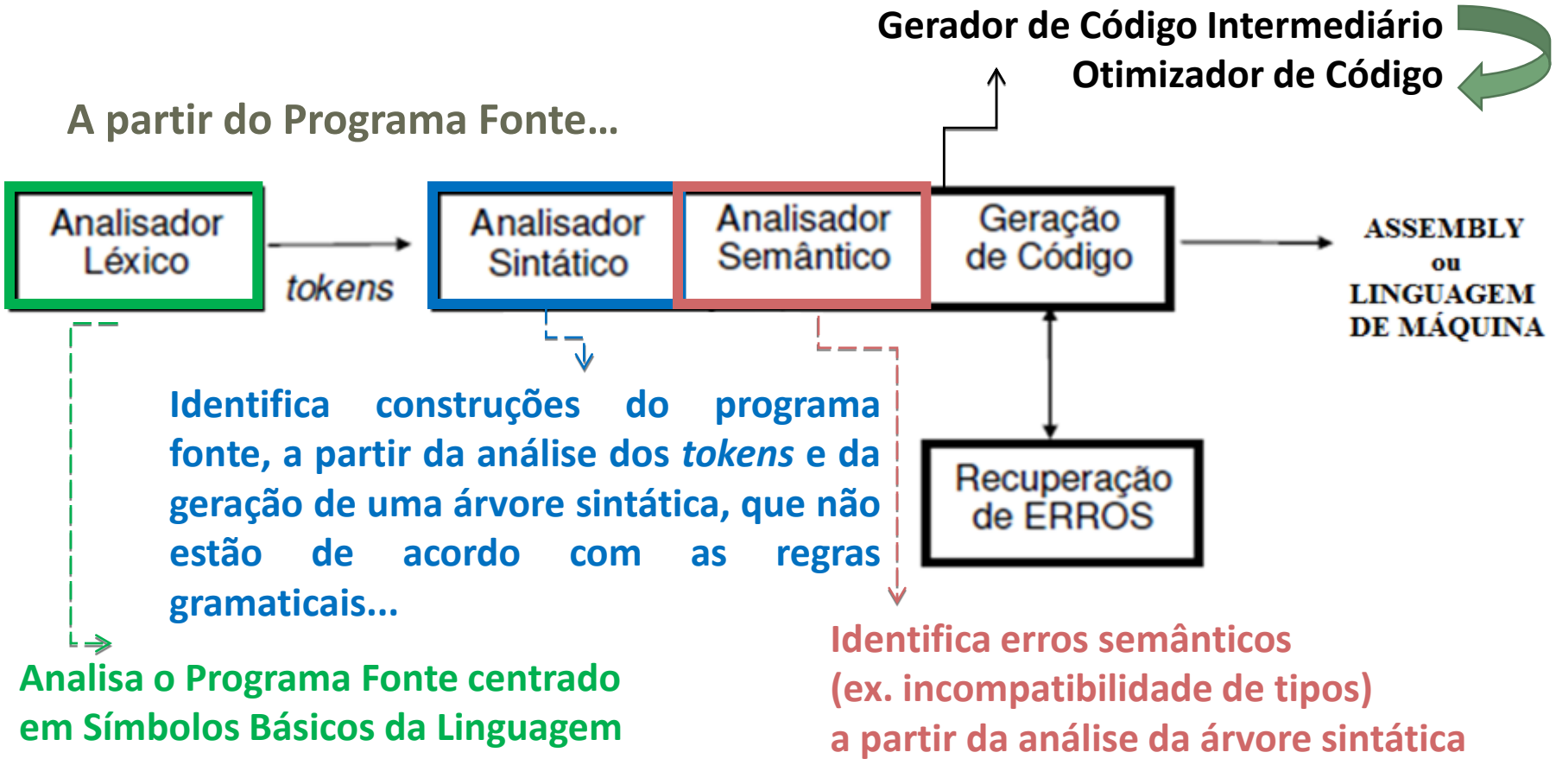
DEPOIS DA OTIMIZAÇÃO

Analisadores e Outros

- Um **gerador de código** permite a geração do código para o programa objeto, consistindo, normalmente, de código em linguagem *assembly* ou de código em linguagem de máquina:
  - **MOV AX, [soma] %** → copia o conteúdo do endereço de memória correspondente ao rótulo SOMA para o registrador AX;
  - **ADD AX, 35 %** → soma o valor constante 35 ao conteúdo do registrador AX, e
  - **MOV [soma], AX %** → copia o conteúdo do registrador AX para o endereço de memória correspondente ao rótulo “soma”.

Analisadores e Outros

✓ Em resumo, um compilador pode ser representado por:



Considerações Finais

Considerações Finais

- ✓ A aula de hoje conferiu uma visão geral sobre paradigmas, linguagens, e fundamentos básicos de compiladores.

Vamos exercitar?

Exercícios

Exercício 01

- √ Especifique a gramática que gera a linguagem das letras do alfabeto.

Exercício 01

- Resposta:

  G:

  $G = (N, T, P, S)$

  onde

  $V_N = \{N, D\}$

  $V_T = \{a, b, c, d, e, ..., z\}$

  $P = \{$

  $N \rightarrow D$

  $D \rightarrow a \mid b \mid c \mid d \mid e \mid ... \mid z$

  $\}$

  $S = N$

Exercício 02

- √ Especifique a gramática que gera a linguagem dos meses do ano.

Exercício 02

√ Resposta:

G:

$G = (N, T, P, S)$

onde

$V_N = \{N, D\}$

$V_T = \{Jan, Fev, Mar, Abr, ..., Dez\}$

P = {

$N \rightarrow D$

$D \rightarrow Jan \mid Fev \mid Mar \mid Abr \mid ... \mid Dez$

}

S = N

Exercício 03

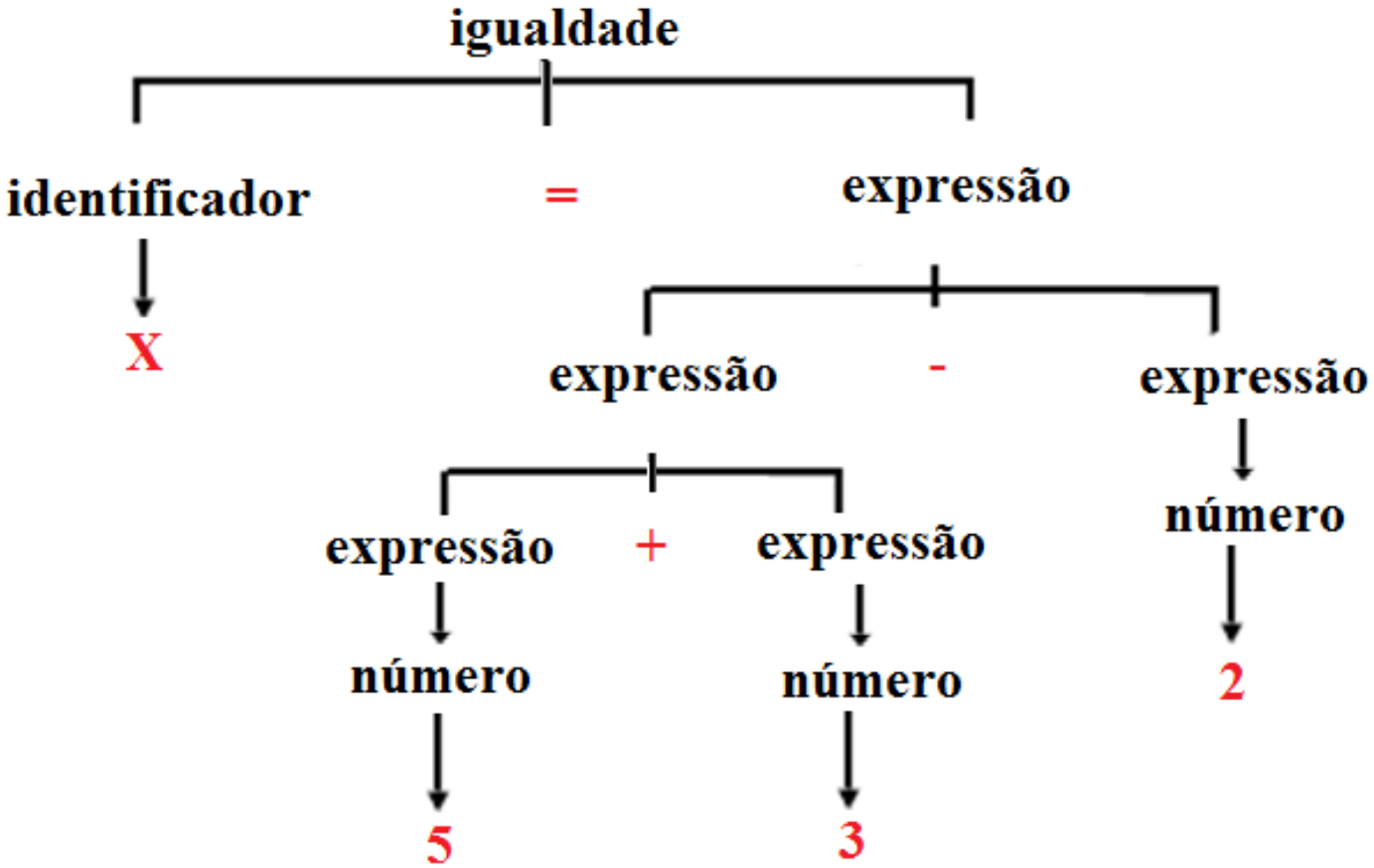
√ Considere:

X= 5 + 3 - 2

√ Construa a respectiva árvore sintática…

Exercício 03

√ Possível Resposta:



Exercício 04

- Considere o seguinte código gerado pelo gerador intermediário de código:

  $temp_1$:= 20

  $temp_2$:= MULTIPLICACAO * $temp_1$

  MULTIPLICACAO := $temp_2$

  $temp_3$:= 50

  $temp_4$:= SOMA+ $temp_3$

  $temp_5$:= MULTIPLICACAO

  SOMA := SOMA+ $temp_5$

- Procure otimizar o código...

Exercício 04

√ Possíveis Respostas:

MULTIPLICACAO := MULTIPLICACAO * 20
SOMA:= SOMA + 50
SOMA:= SOMA + MULTIPLICACAO

OU

MULTIPLICACAO := MULTIPLICACAO * 20
SOMA:= SOMA + 50 + MULTIPLICACAO

Leitura Complementar

Leitura Complementar

- √ [EBRARY] Scott, M. L. Programming Language Pragmatics, eISBN: 9780080515168, 2nd edition, 915 pages, editor: Morgan Kaufmann, Saint Louis, MO, USA, November 2005.

- √ [OPEN ACCESS] Introduction to Computer Science Programming Paradigms, Stanford Graduate School of Education (Stanford University), Stanford, CA, February 2013.

http://see.stanford.edu/see/lecturelist.aspx?coll=2d712634-2bf1-4b55-9a3a-ca9d470755ee

http://videolectures.net/stanfordcs107s08_programming_paradigms/

https://www.udemy.com/cs-107-programming-paradigms/

- √ [EBRARY] Tremblay, J. P. and Sorenson, P. G. Theory and Practice of Compiler Writing, pISBN: 9788178001837, 813 pages, editor: Global Media, Hyderabad, IND, 2008.

Leitura Complementar

- √ [EBRARY] Singh, R., Sharma, V. and Varshney, M. Design and Implementation of Compiler, eISBN: 9788122428650, 423 pages, editor: New Age International, Daryaganj, Delhi, IND, 2009.

- √ [OPEN ACCESS] Compiler Tools, February 2013.

http://dinosaur.compilertools.net/

- √ [OPEN ACCESS] Bumblebee Software, February 2013.

http://www.bumblebeesoftware.com/index.htm

FIM!!!

Dúvidas?

Professora: Milene Serrano
Sala 27 / UED