Teoria da Computação e Compiladores

Introdução aos Conceitos de Compiladores

Prof. Dr. Fernando Kakugawa

fernando.kakugawa@animaeducacao.com.br

Universidade Anhembi Morumbi

Propriedades de um Bom Compilador

- Deve obedecer completamente a especificação da Linguagem
- Deve ser capaz de manipular programas de tamanho essencialmente arbitrário
  - Desde que exista memória disponível para isto
- Velocidade de compilação é uma propriedade importante mas não essencial
  - Prioridade gerar código correto

Propriedades de um Bom Compilador

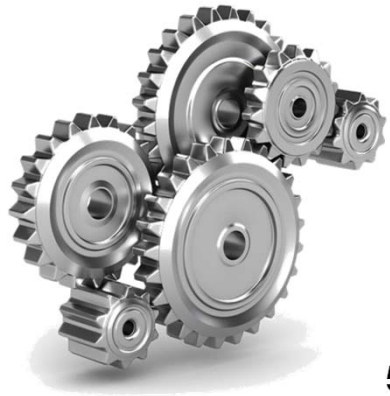
- O tamanho de um compilador não é mais problema
  - Máquinas atuais com alta capacidade de armazenamento
- Caráter Amistoso
  - Bons relatórios de erros
- Velocidade e Tamanho do código gerado
  - Dependem do propósito do compilador

História da Construção dos Compiladores

- 1945 a 1960: geração de código
  - Maquinas peculiares
  - Desenvolvimento de linguagens lento
  - Problema
    - Como gerar código assembly para uma determinada máquina
  - Programação em assembly em alta
  - Linguagens de programação de alto nível vistas com suspeita

História da Construção dos Compiladores

- Percepção
  - Se os compiladores de alto nível não gerassem código tão eficiente quanto os códigos desenvolvidos pelos programadores a mão com assembly, estes não teriam sucesso
- Surgimento do 1o Compilador Fortran
  - Ênfase do compilador na geração de código otimizado

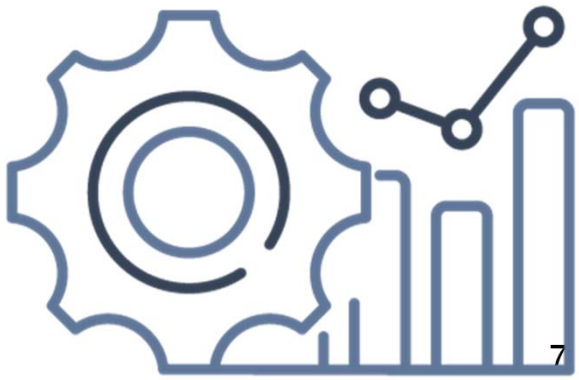


História da Construção dos Compiladores

- 1960 a 1975: análise
  - Proliferação das linguagens de programação
  - Visão dos projetistas
    - Existência de um compilador para uma linguagem era mais importante do que compilador que gerasse código eficiente
  - Ênfase do desenvolvimento deslocado do back-end para o front-end
  - Estudos sobre linguagens formais
    - Revelam técnicas eficazes que podem ser utilizadas no desenvolvimento de front-ends (analisadores)

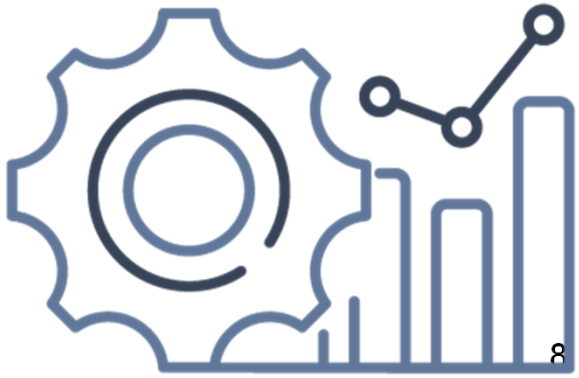
História da Construção dos Compiladores

- 1975 até a época atual : geração de código e otimização de código
  - Diminuição
    - Número de linguagens
    - Qtde de máquinas peculiares
  - Redução da necessidade de compiladores
  - Busca por compiladores profissionais
    - Confiáveis
    - Eficientes (código gerado e uso)
    - Interface agradável
  - Surgem novos paradigmas de programação
    - Orientação a objetos
    - Funcional
    - Lógica
    - Distribuída

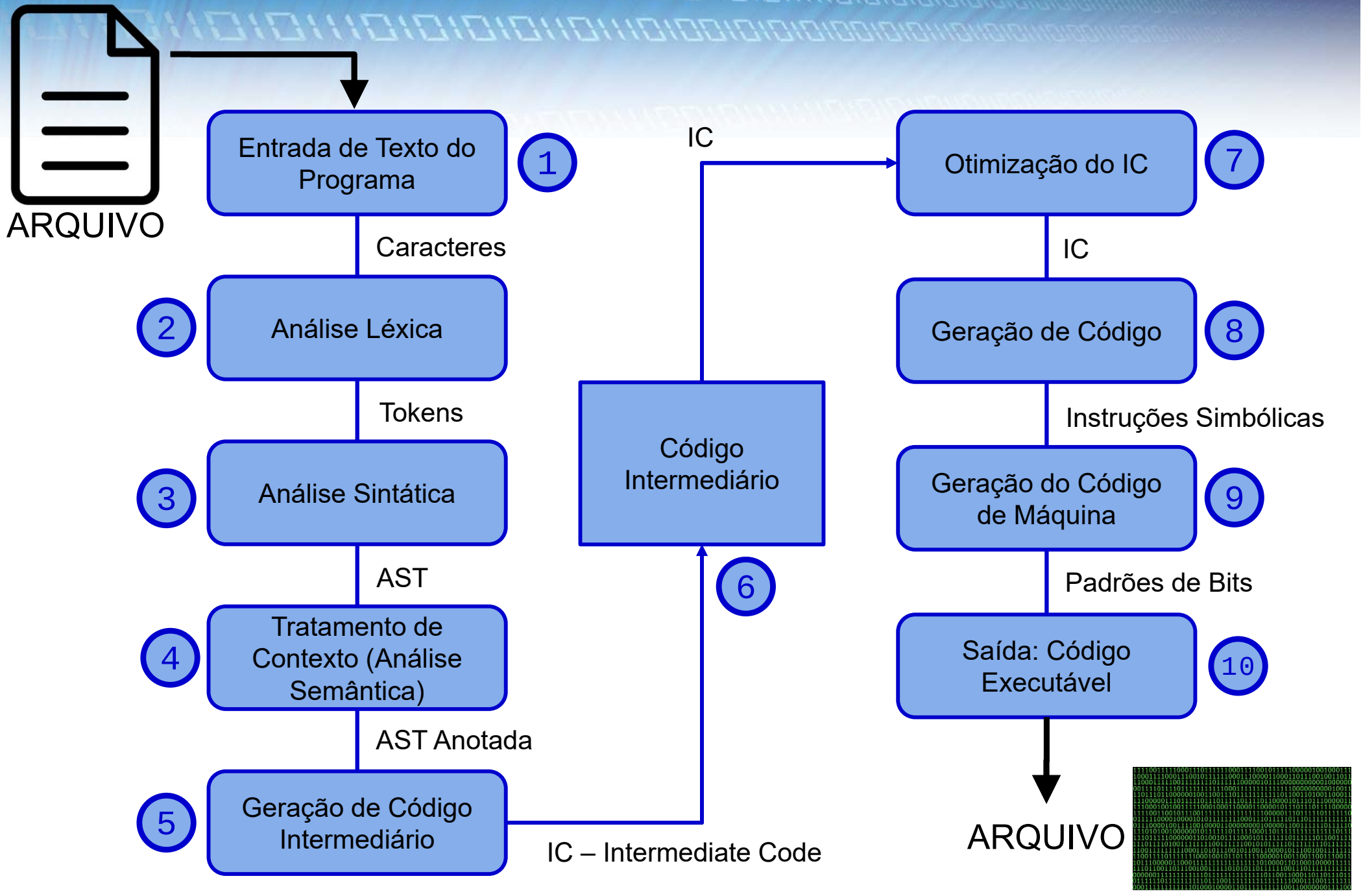


História da Construção dos Compiladores

- Maior esforço para
  - Alocação e desalocação de dados
  - Chamadas de procedimentos remotos
  - Extensão de listas
- Pois estes recursos são inseridos pelo compilador na geração do executável
- Objetivo não é mais “como compilar” e sim “o que compilar”



Estrutura de um Compilador



Estrutura de um Compilador

1. Módulo de Entrada de Texto do Programa
   - Lê o arquivo e o converte no fluxo de caracteres
   - Faz a inclusão de arquivos
     - Include C, C++
     - Import Java
2. Módulo de Análise Léxica
   - Isola o fluxo de caracteres identificando tokens
   - Determina a classe e representação destes tokens
   - Pode ser escrito a mão ou através de geradores (utilizam descrições dos símbolos)

Estrutura de um Compilador

- Exemplo Léxico

  Considere o trecho de programa Pascal:

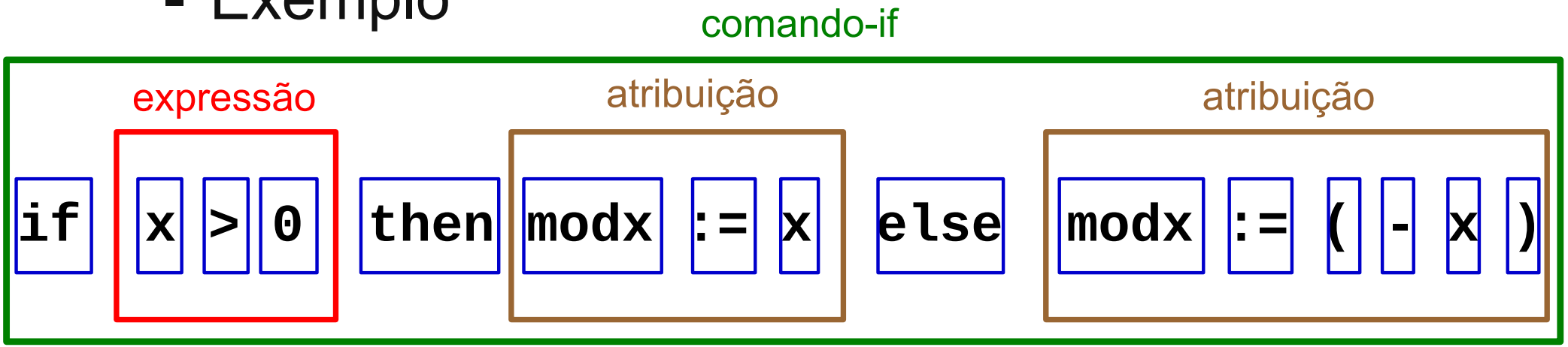
```
if x > 0 then              {x eh positivo}
      modx := x
else                       {x eh negativo}
      modx := (-x)
```

- Identificando os tokens

Estrutura de um Compilador

3. Módulo Análise Sintática

- Converte o fluxo de Símbolos (tokens) em uma árvore sintática abstrata (AST), verificando se a sintaxe do programa está correta
- Exemplo



Estrutura de um Compilador

4. Módulo Análise Semântica

- Reúne informações de diversas partes do programa e cria anotações nos nós da AST
- Exemplos:
  - Tipo de declarações
  - Identificar chamadas de rotinas locais e remotas

Estrutura de um Compilador

5. Módulo de Geração de Código Intermediário (IC)
   - Converte construções específicas da linguagem em construções mais gerais
   - Projetista do compilador define como será esta representação mais geral
     - Propriedade: deve ser razoavelmente direto gerar o código de máquina a partir desta representação

Estrutura de um Compilador

- Exemplos de conversões
  - Substituir um laço while por teste, rótulo e saltos em linguagens imperativas (assembly)
  - Determinar qual método chamar para um dado objeto em linguagens de vinculação dinâmica (Java)
  - Substituir uma regra da linguagem Prolog por uma rotina que faça a busca e o retrocesso apropriado

Estrutura de um Compilador

6. Módulo de Otimização de Código Intermediário

- Executa um pré-processamento no código intermediário
  - Objetivo: melhorar o desempenho do código gerado pelo módulo de geração de IC
  - Exemplos de otimizações
    - Substituição de uma constante pelo valor representado por ela
    - Substituição da chamada de uma função pelo seu corpo

Estrutura de um Compilador

- Exemplo de Otimizações
  - Considere a sequência de comando de atribuição da primeira coluna

Fonte	Código Intermediário Original	Código Intermediário Otimizado

`w:=(a+b)+c;`	`t1:=a+b`
	`t2:=t1+c`
	`w:=t2`
		`t1:=a+b`
		`t2:=t1+c`
		`w:=t2`

`x:=(a+b)*d;`	`t3:=a+b`
	`t4:=t3*d`
	`x:=t4`
		`t4:=t1*d`
		`x:=t4`

`y:=(a+b)+c;`	`t5:=a+b`
	`t6:=t5+c`
	`y:=t6`
		`y:=t2`

`z:=(a+b)*d+e`	`t7:=a+b`
	`t8:=t7*d`
	`t9:=t8+e`
	`z:=t9`
		`t9:=t4+e`
		`z:=t9`

Estrutura de um Compilador

- Exemplo de otimizações
  - Retirada de comandos *invariantes de loop*

```
for i:=1 to n do begin
   pi := 3.1416;
   pi4 := pi / 4;
   d[i] := pi4 * r[i] * r[i];
end;
```

Estrutura de um Compilador

7. Módulo Geração de Código

- Reescreve a AST em um lista linear de instruções da máquina de destino em forma mais ou menos simbólica
- Para isto:
  - Seleciona instruções para segmentos da AST
  - Aloca registradores para conter dados
  - Organiza as instruções de forma apropriada

Estrutura de um Compilador

8. Módulo de Geração de Código de Máquina

- Converte as instruções simbólicas de máquina em padrões de bits correspondentes
- Determina os endereços de máquina do programa e dos dados
- Utiliza tabelas de constantes e relocação

Estrutura de um Compilador

- Exemplos de Otimização de um comando de atribuição

```
x := a + b * c,
```

gera o código:

```
1.  Load b          { t1 := b*c }
2.  Mult c
3.  Store t1
4.  Load a          { t2 := a+t1 }
5.  Add t1
6.  Store t2
7.  Load t2         { x := t2 }
8.  Store x
```

Estrutura de um Compilador

9. Código Otimizado

- a instrução 7 é desnecessária e pode ser retirada: copia para o acumulador o valor de t2, que já se encontra lá
- (após a remoção da instrução 7) a instrução 6 é desnecessária e pode ser retirada: o valor da variável t2 nunca é utilizado
- (considerando que a soma é comutativa) as instruções 4 e 5 podem ser trocadas por 4' e 5', preparando novas alterações:

```
4′  Load t1
5′  Add a
```

- As instruções 3 e 4' são desnecessárias e podem ser retiradas (pelas mesmas razões que 6 e 7 acima)
- O código final após as transformações é consideravelmente melhor que o original:

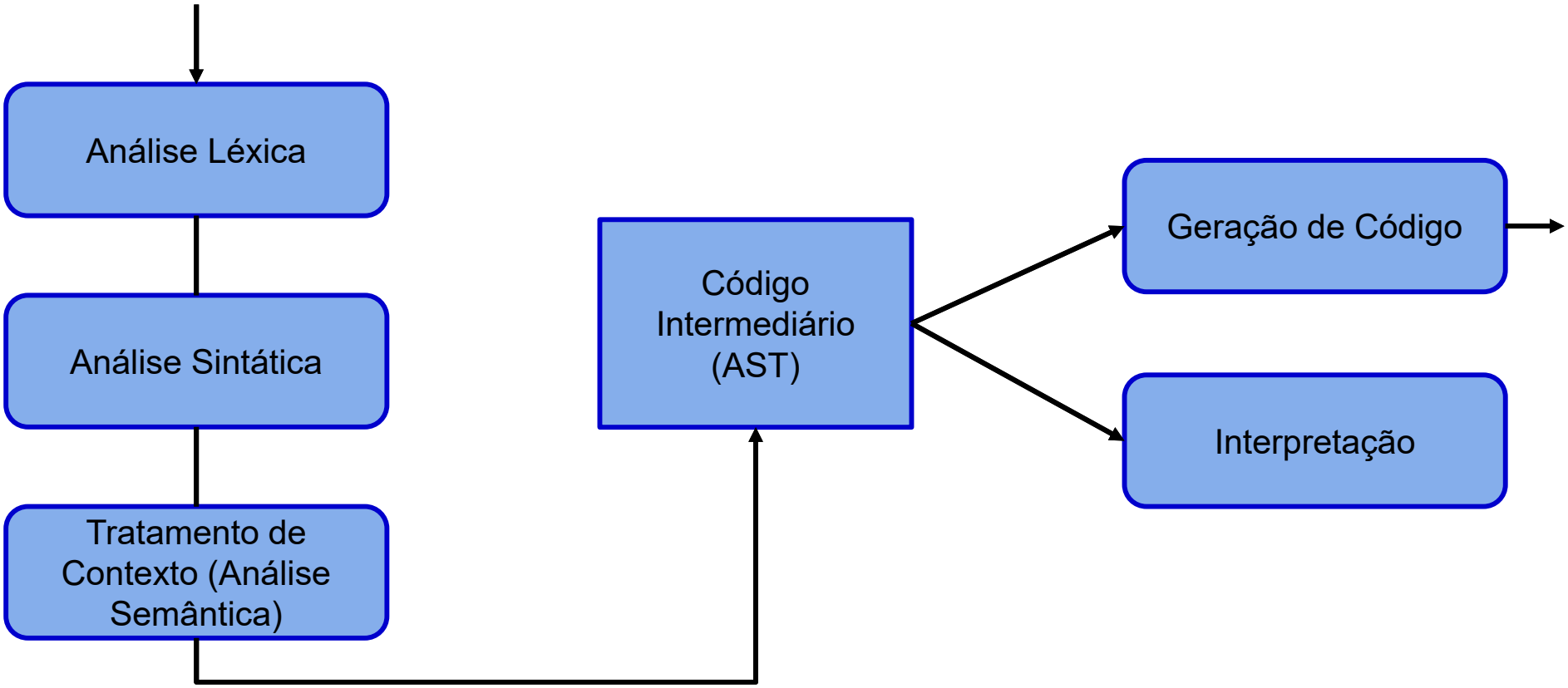
```
1  Load b
2  Mult c
5´ Add a
8  Store x
```

Estrutura de um Compilador

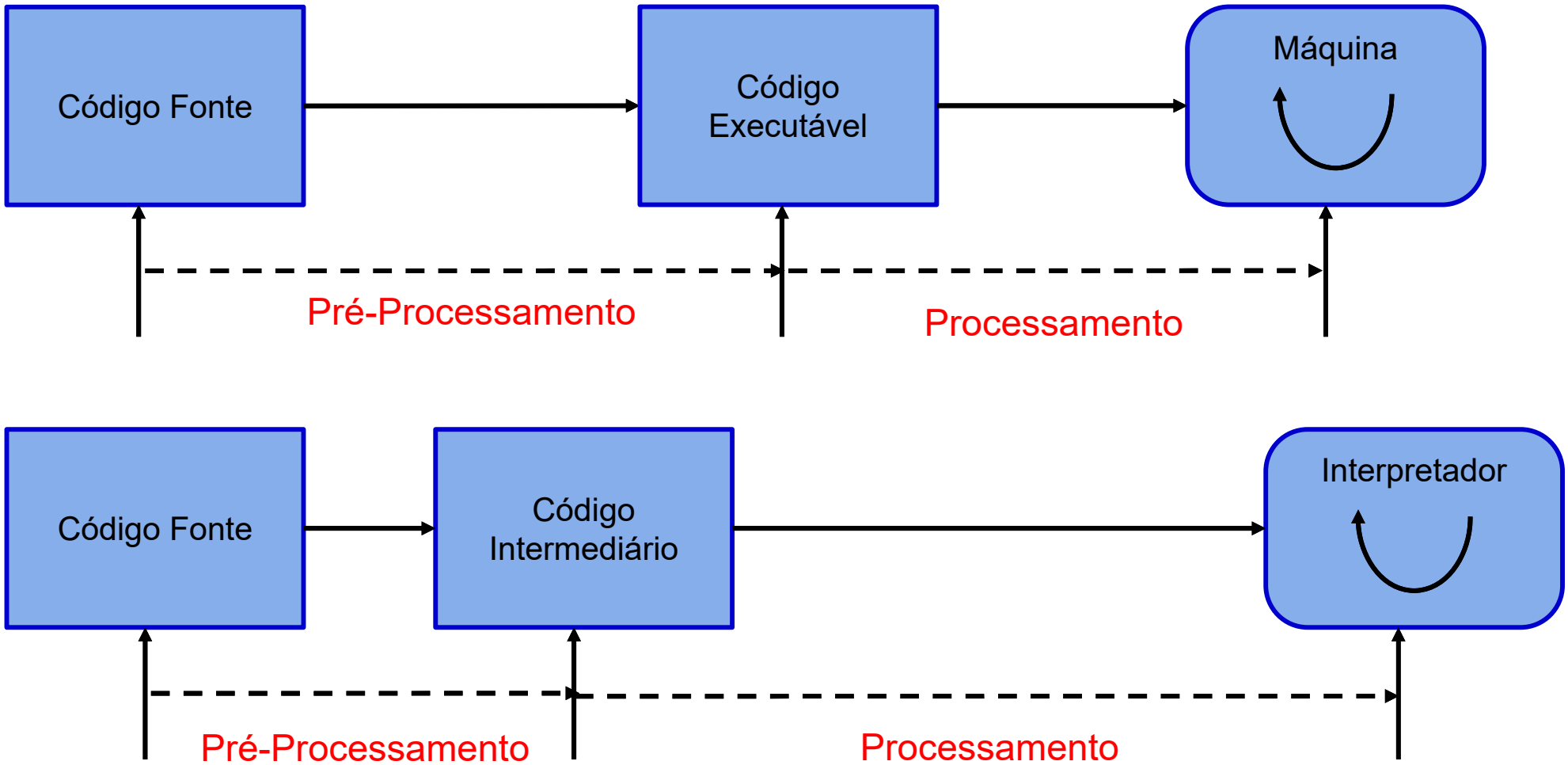
10.Módulo de Saída do Código Executável

- Combina
  - Instruções de máquinas codificadas
  - Tabelas de constantes
  - Tabelas de relocação
  - Cabeçalhos iniciais e finais
- A fim de formar o arquivo fonte com o código executável

Estrutura Compilador X Interpretador



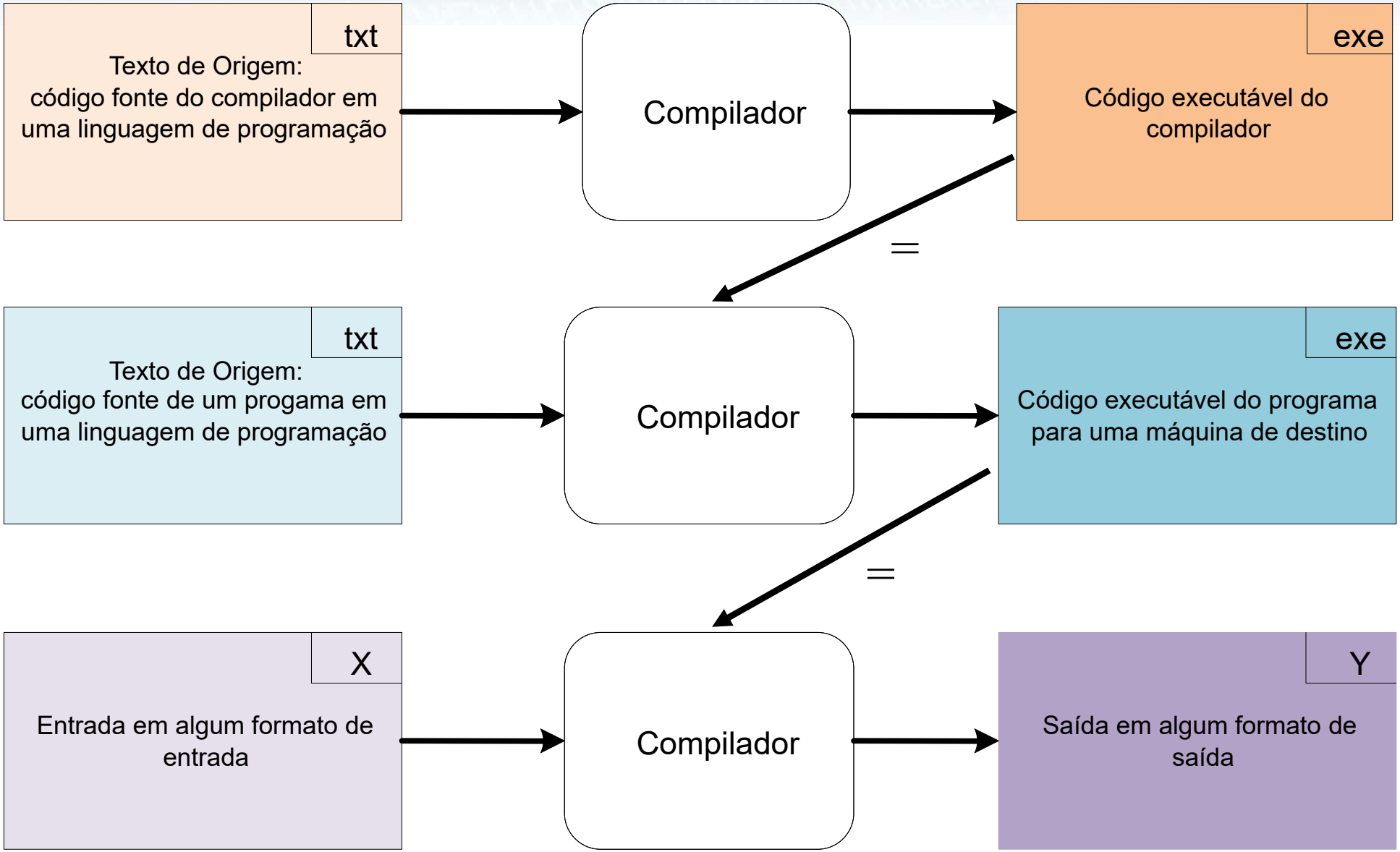
Estrutura Compilador X Interpretador



Estrutura Compilador X Interpretador

- Na compilação:
  - O tempo de análise e compilação de programas é considerável
  - A forma intermediária resultante, código binário executável específico da máquina, é de baixo nível
  - A execução do programa é relativamente rápida
- Na interpretação:
  - O tempo de análise e compilação de programas é de mínimo a moderado
  - A forma intermediária resultante, alguma estrutura de dados específica do sistema, é nível alto a médio
  - A execução do programa é relativamente lenta

Exemplos de Aplicações



Observações

- O conceito de um compilador está associado à ideia de geração de um programa executável
- Entretanto outras aplicações:
  - Implementação de um outro compilador
    - Linguagem C++ e Java
  - Conversão de um arquivo de um formato para outro
    - ASCII para EBCDIC
    - GIF para JPEG

Importante

- Na compilação a entrada possui uma propriedade chamada de semântica (significado) que deve ser preservado no processo de compilação
- Exemplo:
  - Se no código fonte ocorre a soma de dois números o programa executável deve somar dois números e não multiplicá-los.

Material elaborado por:

Prof. Dr. Augusto Mendes Gomes Jr.

augusto.gomes@animaeducacao.com.br

Prof. Dr. Fernando Kakugawa

fernando.kakugawa@animaeducacao.com.br