Conceitos Básicos de Compiladores

ÉFREN L. SOUZA

Roteiro

1. Processadores de Linguagens
2. A Estrutura de um Compilador
3. Análise Léxica
4. Análise Sintática
5. Análise Semântica
6. Geração de Código Intermediário
7. Otimização de Código
8. Geração de Código
9. Bibliografia

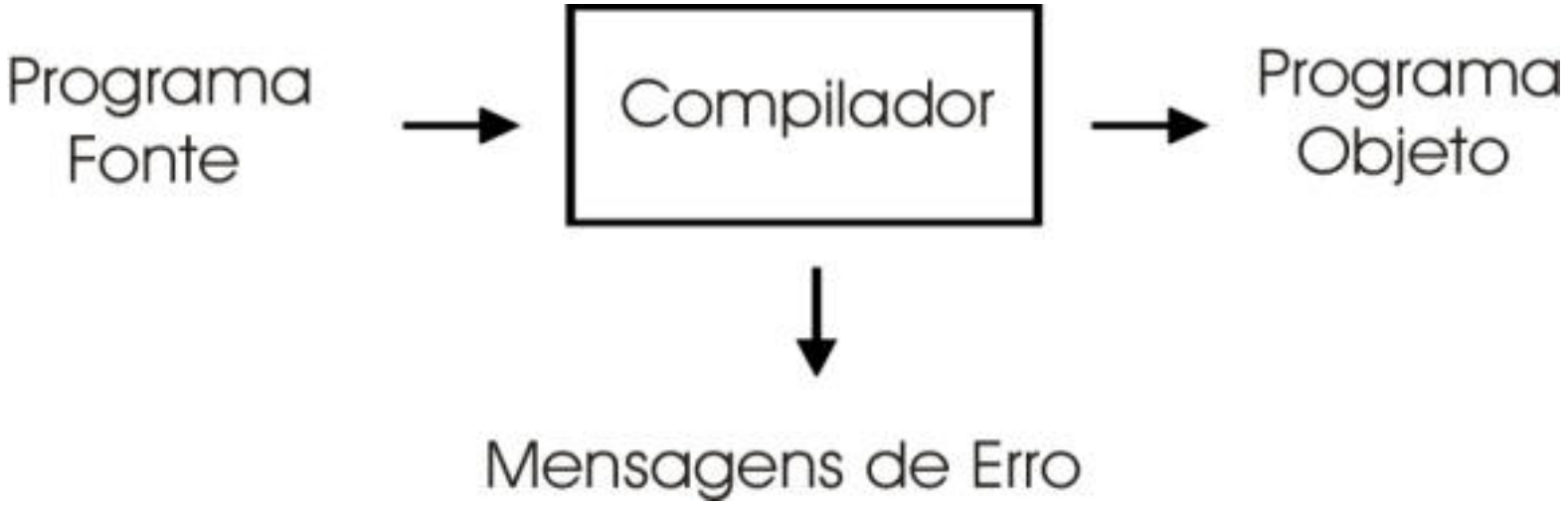
Processadores de Linguagens

Linguagens de Programação

- Linguagens de programação são notações para descrever computações
- Todo *software* é escrito em uma linguagem de programação
- Antes de executar um programa, ele precisa ser traduzido para um formato que possa ser entendido pelo computador
- Os programas que fazem essa tradução são os **compiladores**

Compilador

- Programa que recebe como entrada um programa em uma linguagem de programação (fonte) e o traduz para um programa equivalente em outra linguagem (objeto)
- Uma função do compilador é relatar erros no programa fonte

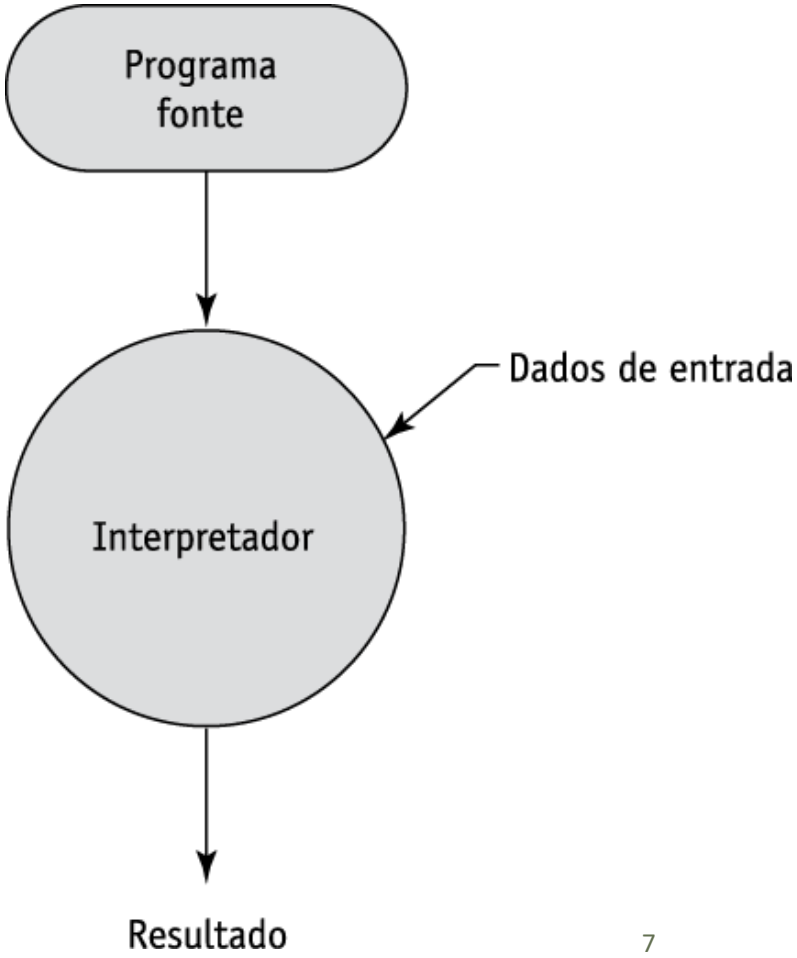


Programa objeto executável

- Se o programa objeto é uma linguagem de máquina, ele pode ser chamado para processar entradas e gerar saída

Interpretador

- Não produz um programa objeto como resultado
- Executa as operações especificadas no programa fonte sobre as entradas fornecidas



Compilador x Interpretador

COMPILADORES

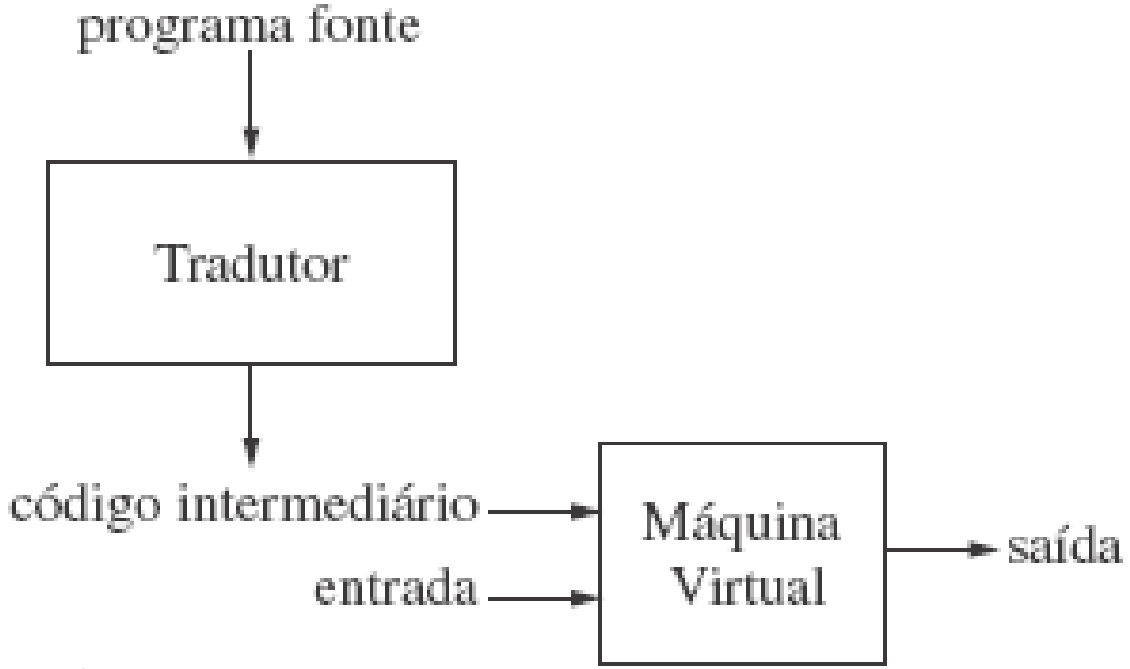
- O programa compilado executa mais rápido
- Protege o código fonte original
- Correção no código requer nova compilação

INTERPRETADORES

- A execução do programa é mais lenta
- O código fonte original fica acessível
- Correções podem ser realizadas mais rápido

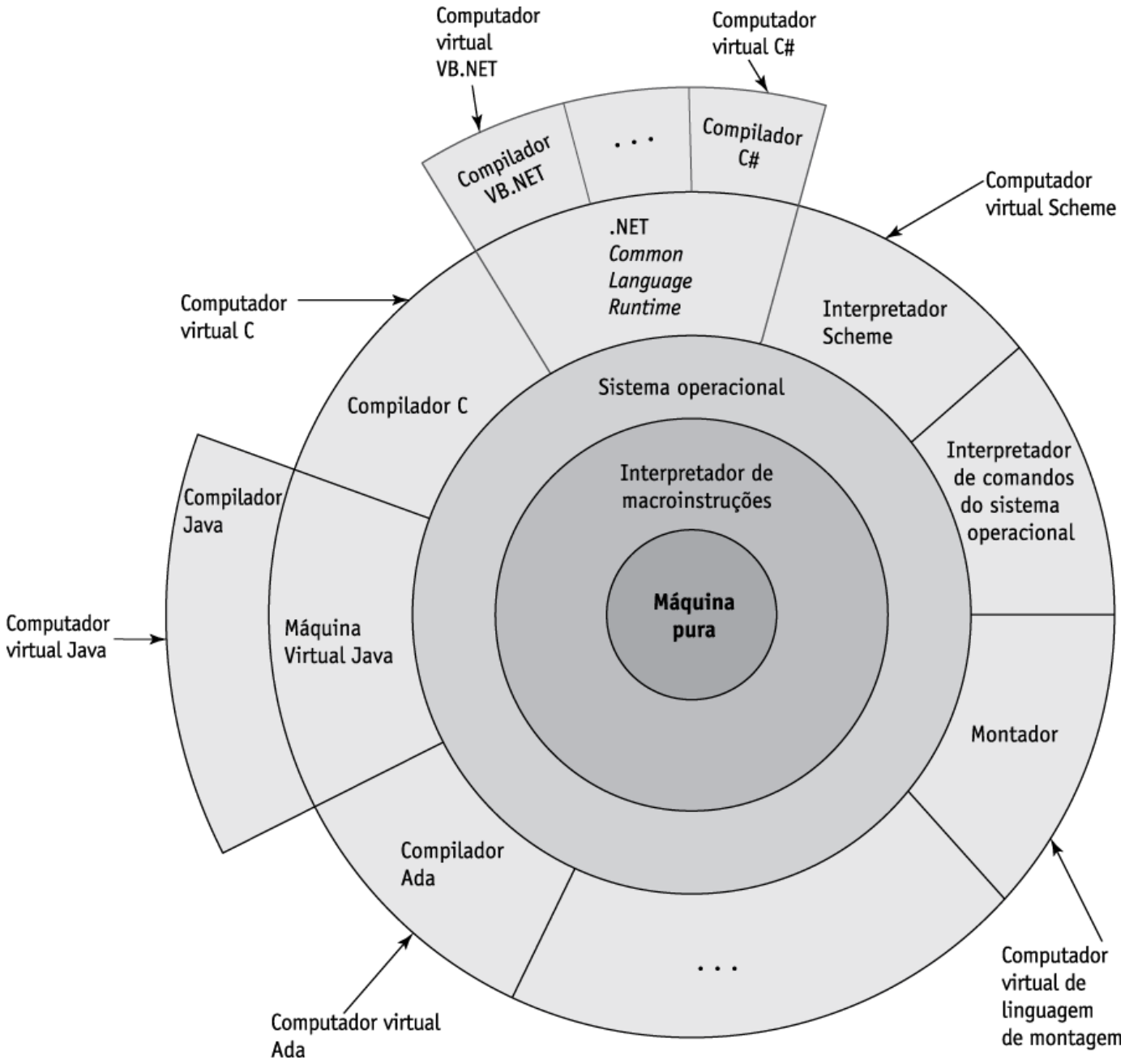
Compilador Híbrido

- Um meio termo entre os compiladores e os interpretadores
- Uma linguagem de alto nível é traduzida para uma linguagem intermediária que permite fácil interpretação
- Mais rápido do que interpretação pura



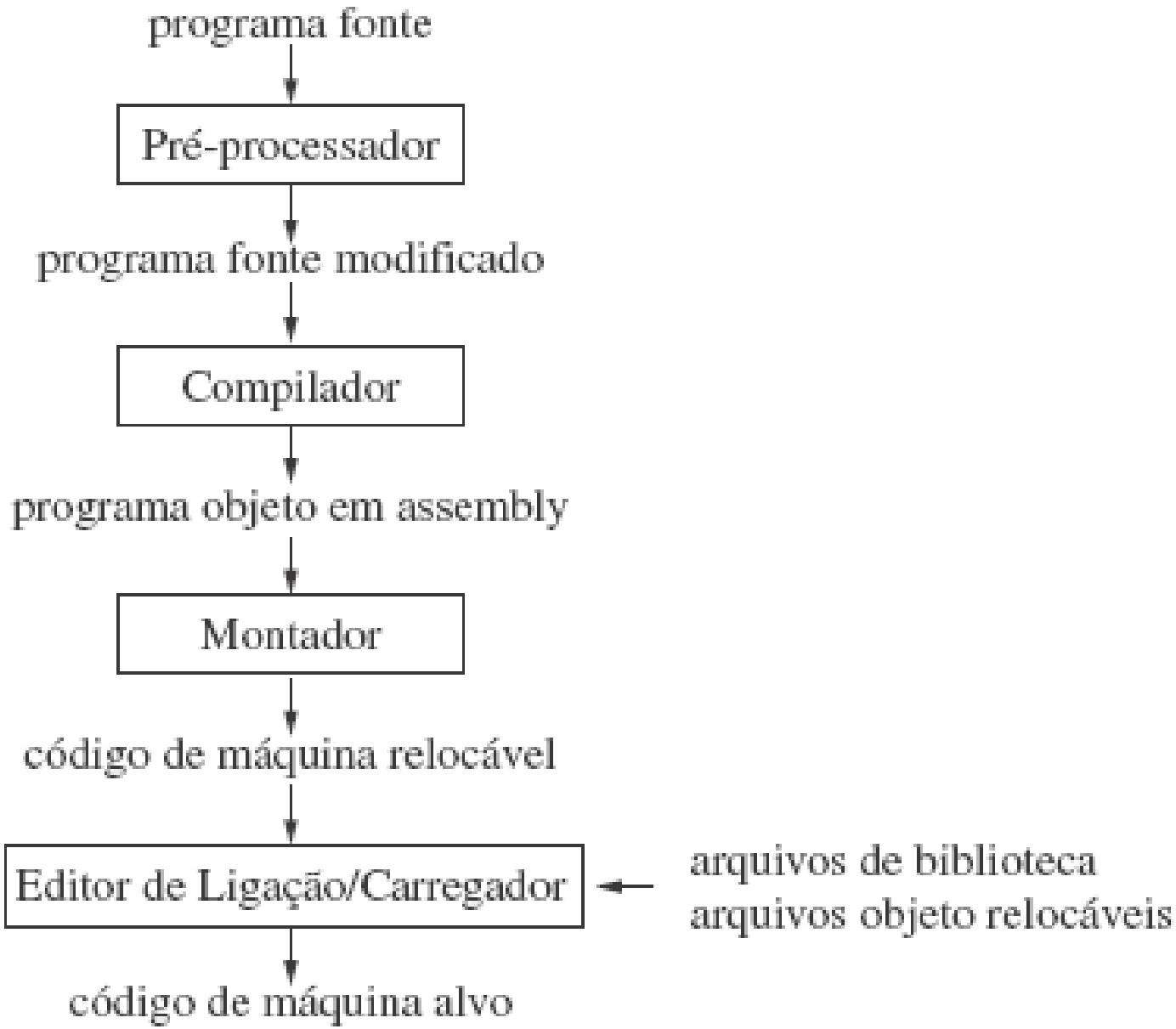
Interface em camadas

- O SO e as implementações de linguagem ficam em camadas superiores à interface de linguagem de máquina
- As linguagens .Net e Java têm um compilador híbrido



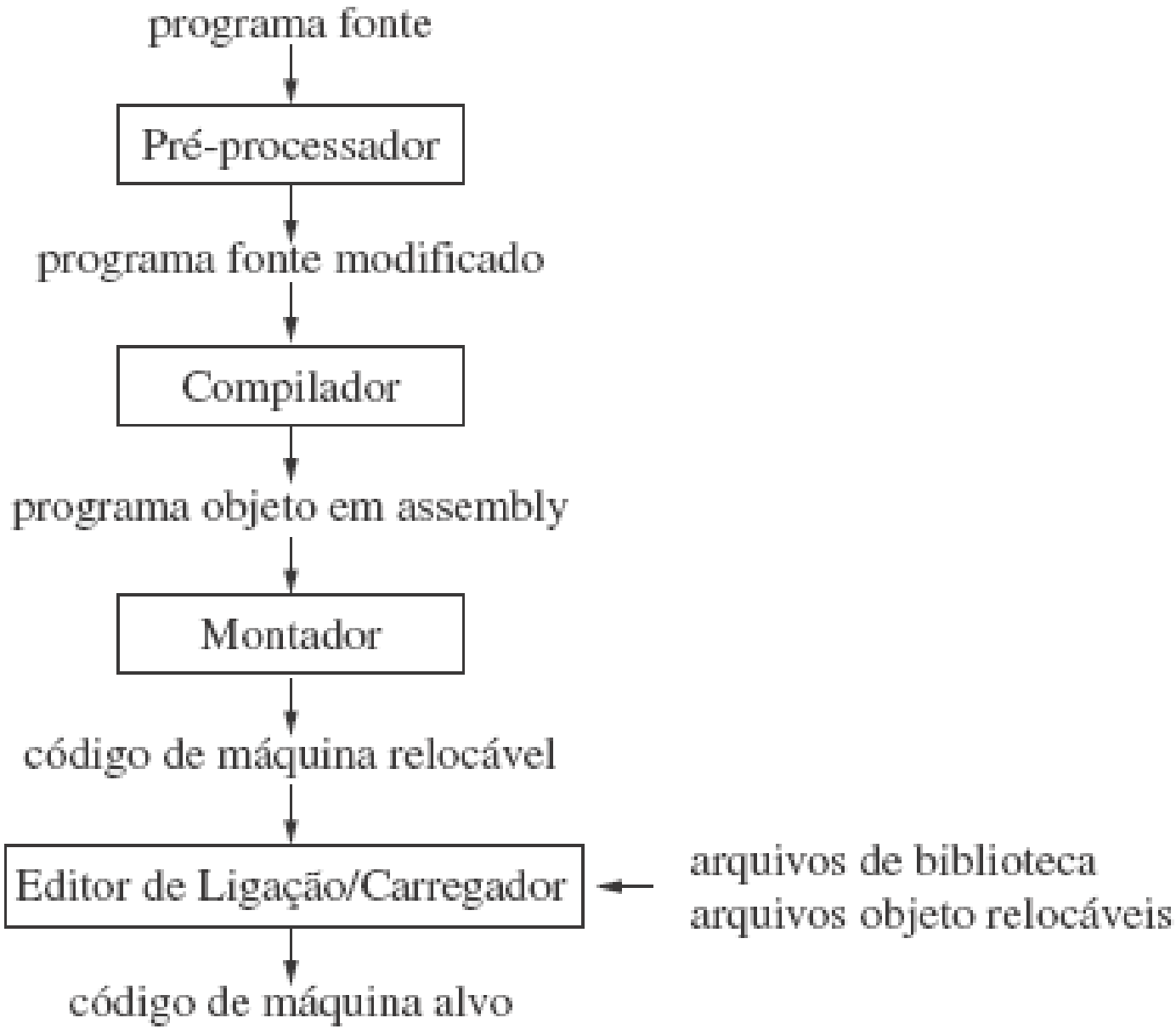
Além do compilador...

- Outros programas podem ser necessários para gerar um programa objeto executável



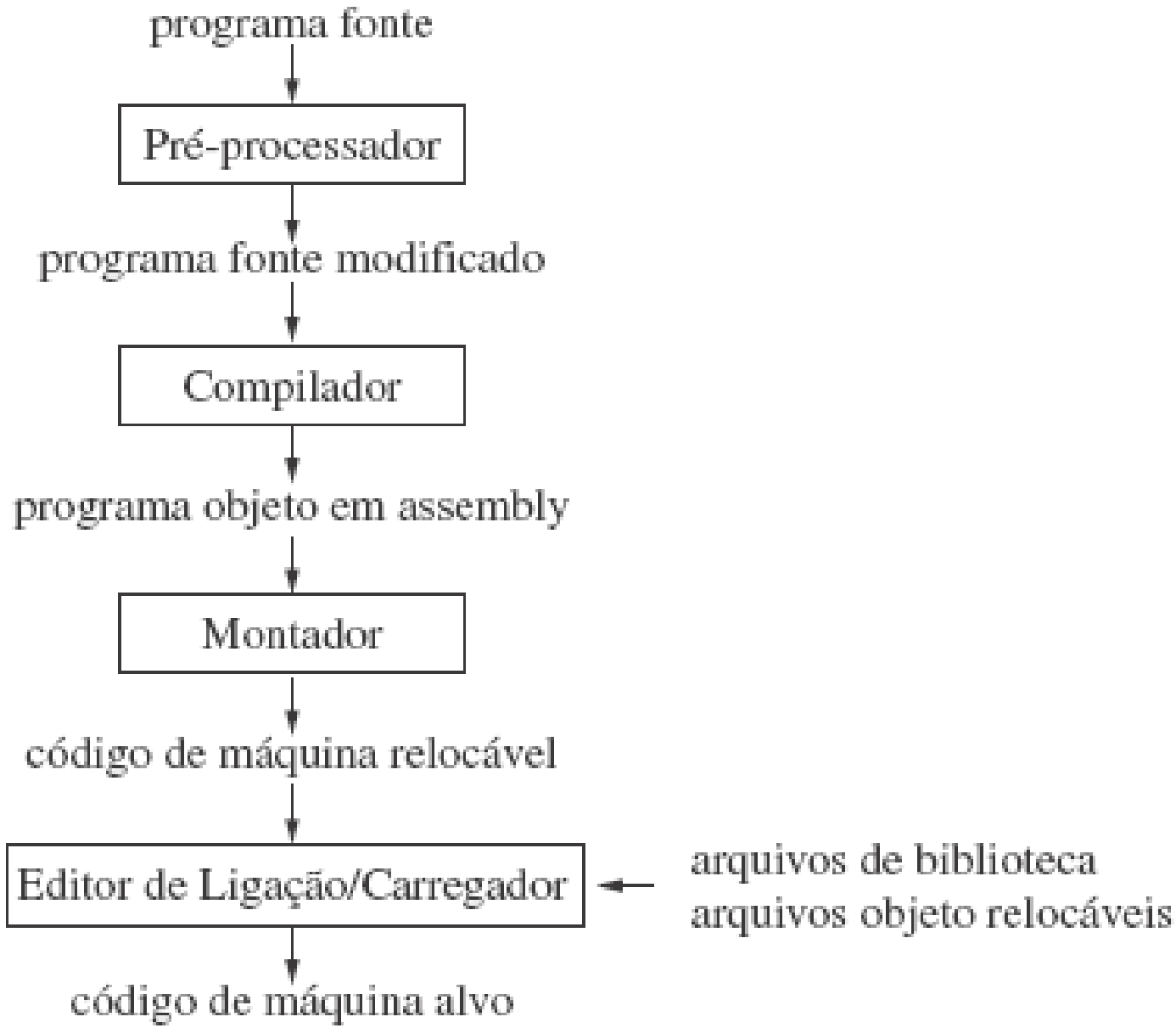
Pré-processador

- Programas podem estar divididos em módulos
- O pré-processador coleta e reúne os módulos do programa fonte
- Também pode expandir macros



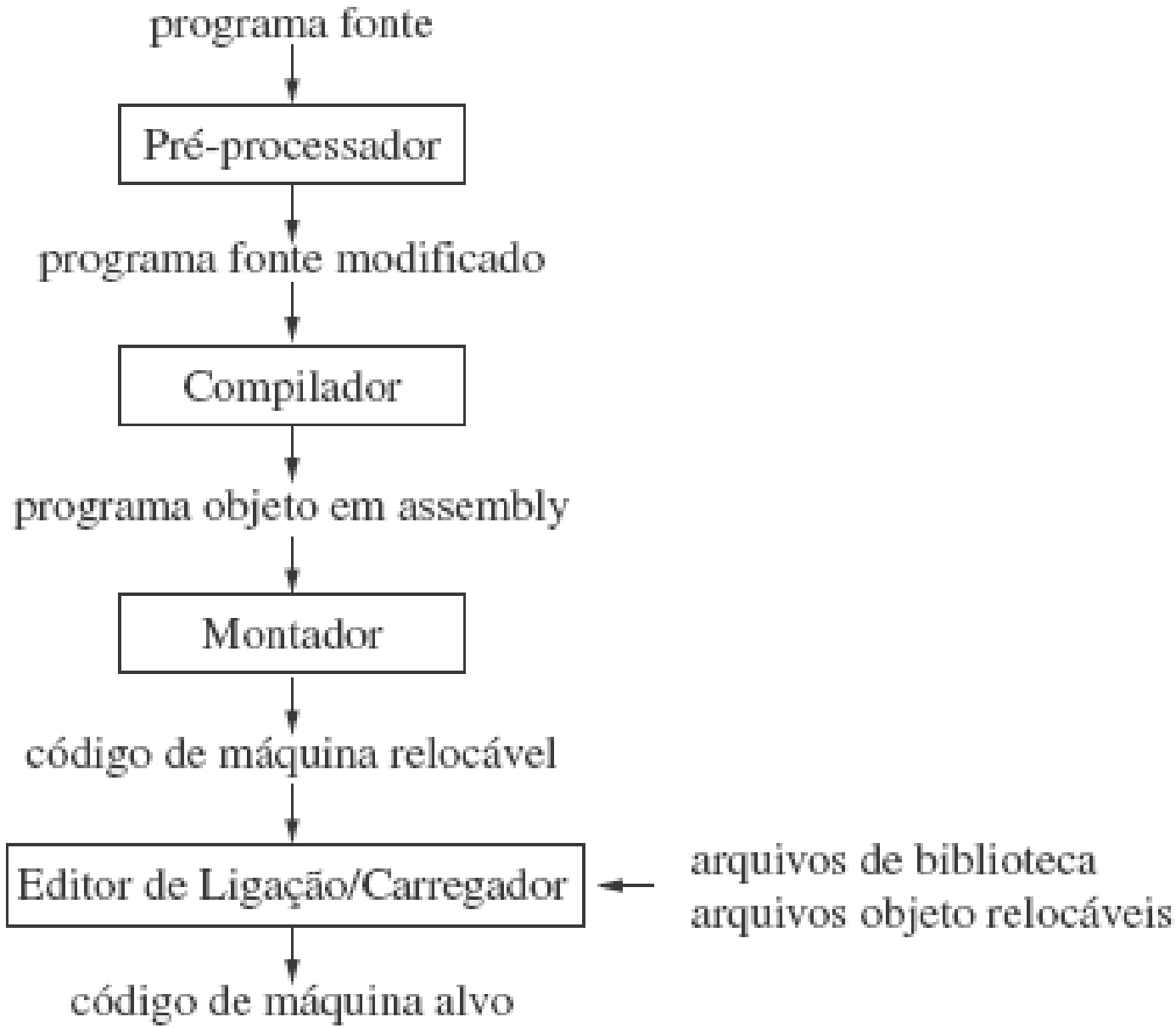
Compilador

- Recebe o programa fonte modificado e o transforma em linguagem simbólica (assembly)



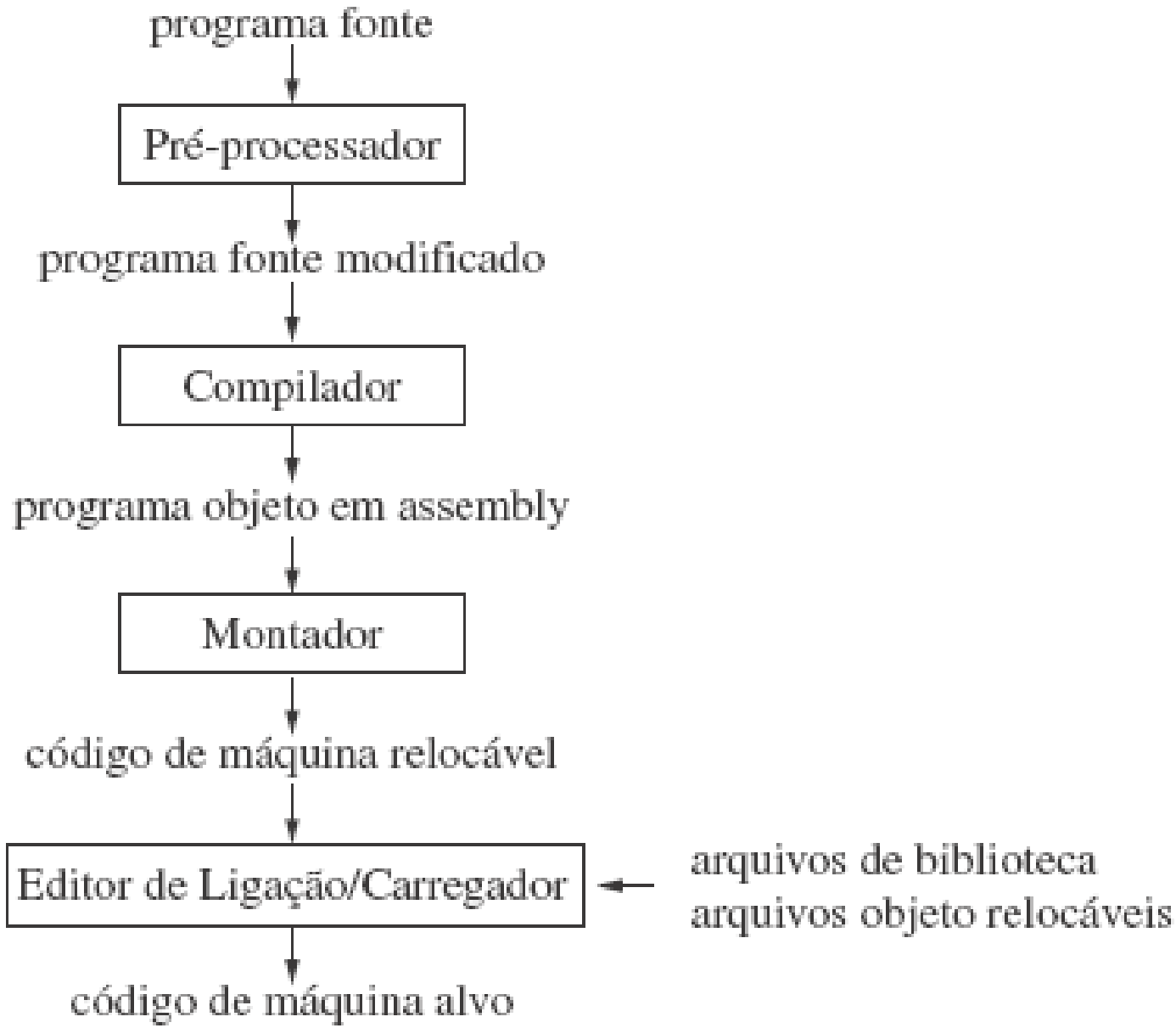
Montador (assembler)

- Gera o código de máquina a partir do código assembly
- Código de máquina relocável não possui referências externas



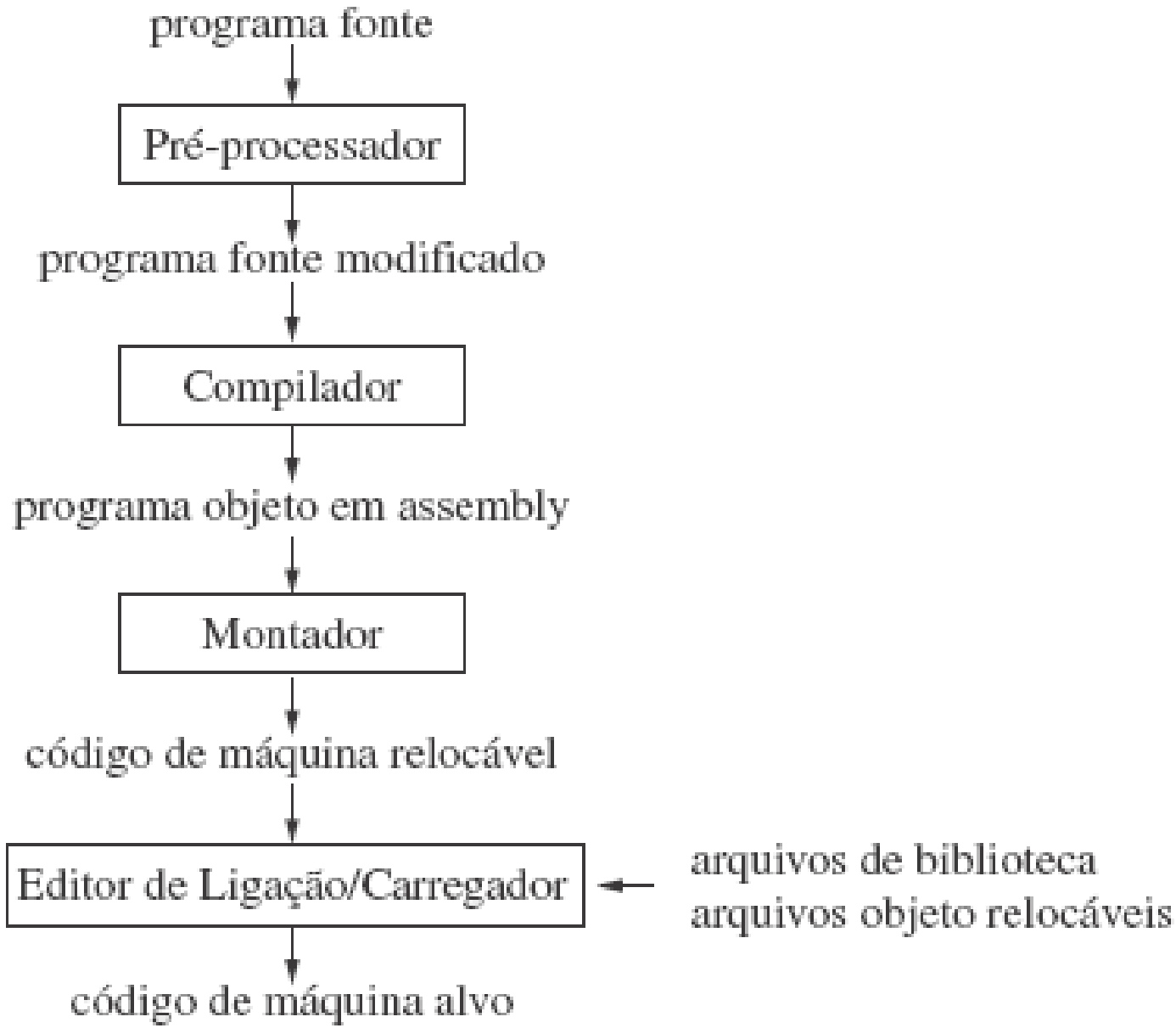
Ligação (*linker*)

- O código de máquina pode precisar de arquivos externos (bibliotecas)
- O ligador une o código de máquina com as bibliotecas necessárias



Carregador (*loader*)

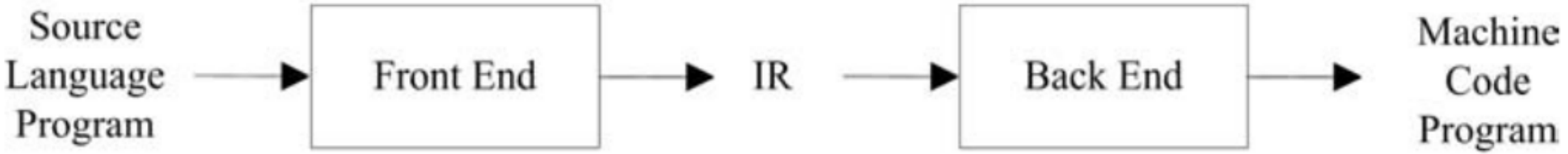
- Reúne na memória todos os arquivos necessários para a execução



A Estrutura de um Compilador

Partes de um compilador

- Um compilador é dividido em duas grandes partes
  - Análise (*front-end*)
  - Síntese (*back-end*)



Análise (I)

- Também chamada de *front-end*
- Divide o programa fonte em suas partes constituintes
- Impõe uma estrutura gramatical a essas partes constituintes
- Gera uma representação intermediária da linguagem

Análise (II)

- Oferece mensagens de erro esclarecedoras
  - Erro de sintaxe
    - Construções que fogem das regras da linguagem
    - Parênteses sem correspondência
  - Erro de semântica
    - Variável não declarada
    - Operações entre tipos diferentes

Análise (III)

- Coleta e guarda informações sobre o código em uma Tabela de Símbolos

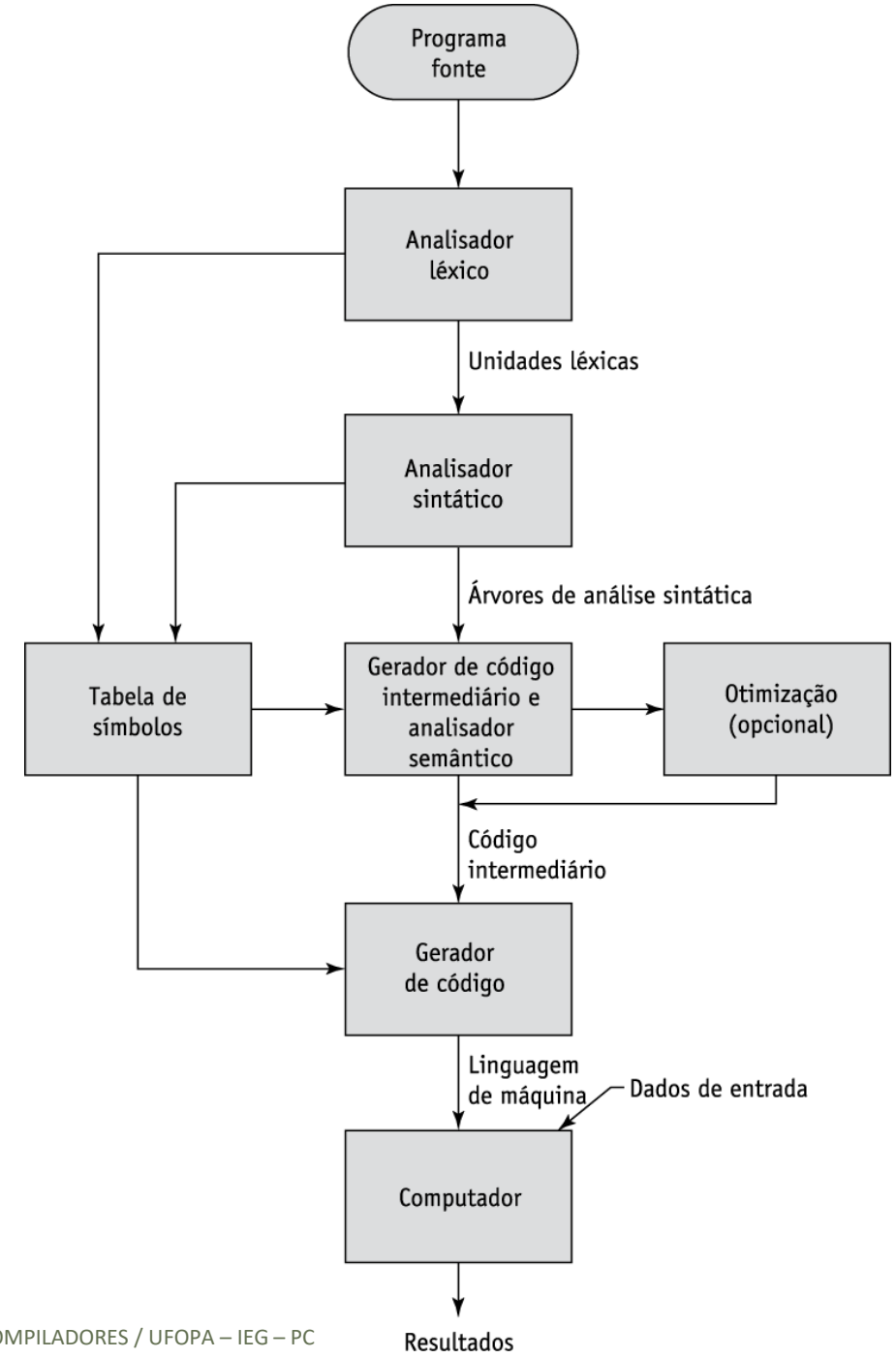
- Fornece para a etapa de **Síntese** a **representação intermediária do código** e a **tabela de símbolos**

Síntese

- Também chamada de *back-end*
- Constrói o programa objeto a partir da representação intermediária e da tabela de símbolos
- Opcionalmente, antes de construir o programa, pode passar por uma fase de otimização

Processo de compilação

- Sequencia de passos que transformam uma representação do programa fonte em outra
- Algumas fases podem ser agrupadas
- A tabela de símbolos é usada durante todo o processo de compilação



Análise Léxica

Análise Léxica

- Também chamada de leitura ou *scanning*
- Lê o fluxo de caracteres do código e os agrupa em sequências significativas (lexemas)
- Para cada lexema é produzido um *token*
- Os *tokens* são passados para a fase de análise sintática

Tokens

- O *token* é uma estrutura com o formato $\langle nomeToken, valorAtributo \rangle$
- O nome do *token* é um símbolo abstrato usado para identificar uma classe de *tokens*
- O valor do atributo pode ser um lexema ou uma referência para a tabela de símbolos
- Algumas vezes o valor do atributo pode ser omitido

Identificando *tokens* (I)

- O trecho de código
  - `position = initial + rate * 60`

- Gera os lexemas
  - $\langle id, 1 \rangle \langle = \rangle \langle id, 2 \rangle \langle + \rangle \langle id, 3 \rangle \langle * \rangle \langle 60 \rangle$

Identificando *tokens* (II)

- $\langle id, 1\rangle\langle =\rangle\langle id, 2\rangle\langle +\rangle\langle id, 3\rangle\langle *\rangle\langle 60\rangle$
- O identificador `position` é mapeado para o lexema $\langle id, 1\rangle$
  - id é um símbolo que significa identificador
  - O valor 1 é um endereço da tabela de símbolos que possui informações sobre o identificador, como nome e tipo
- A atribuição é mapeada para o *token* $\langle =\rangle$
  - O valor do atributo é omitido, pois não é necessário

Análise Sintática

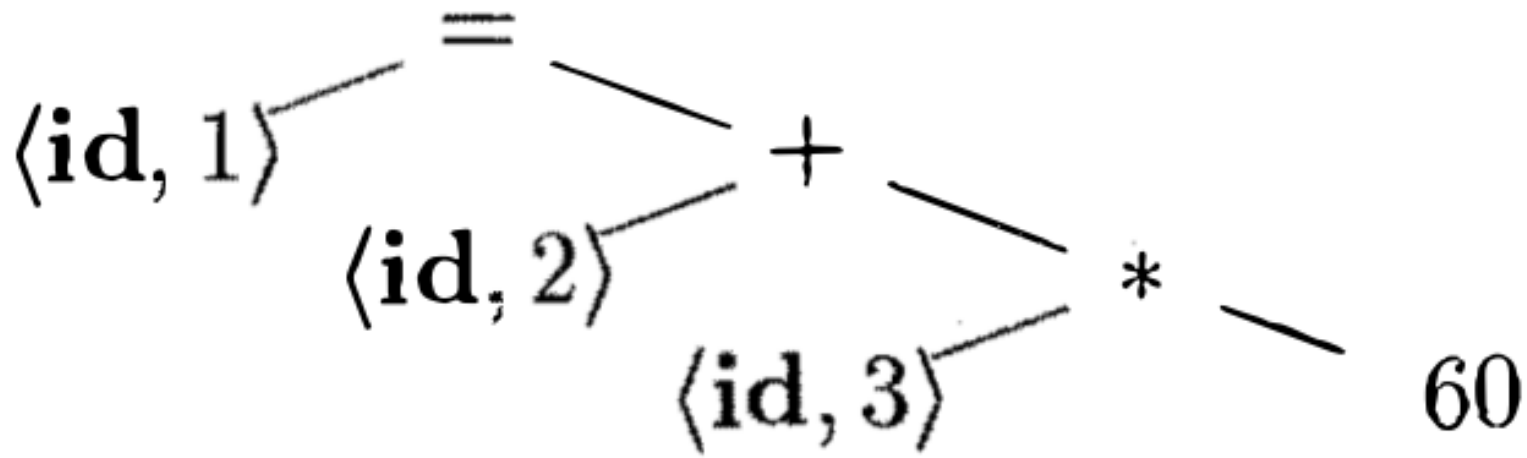
Análise Sintática

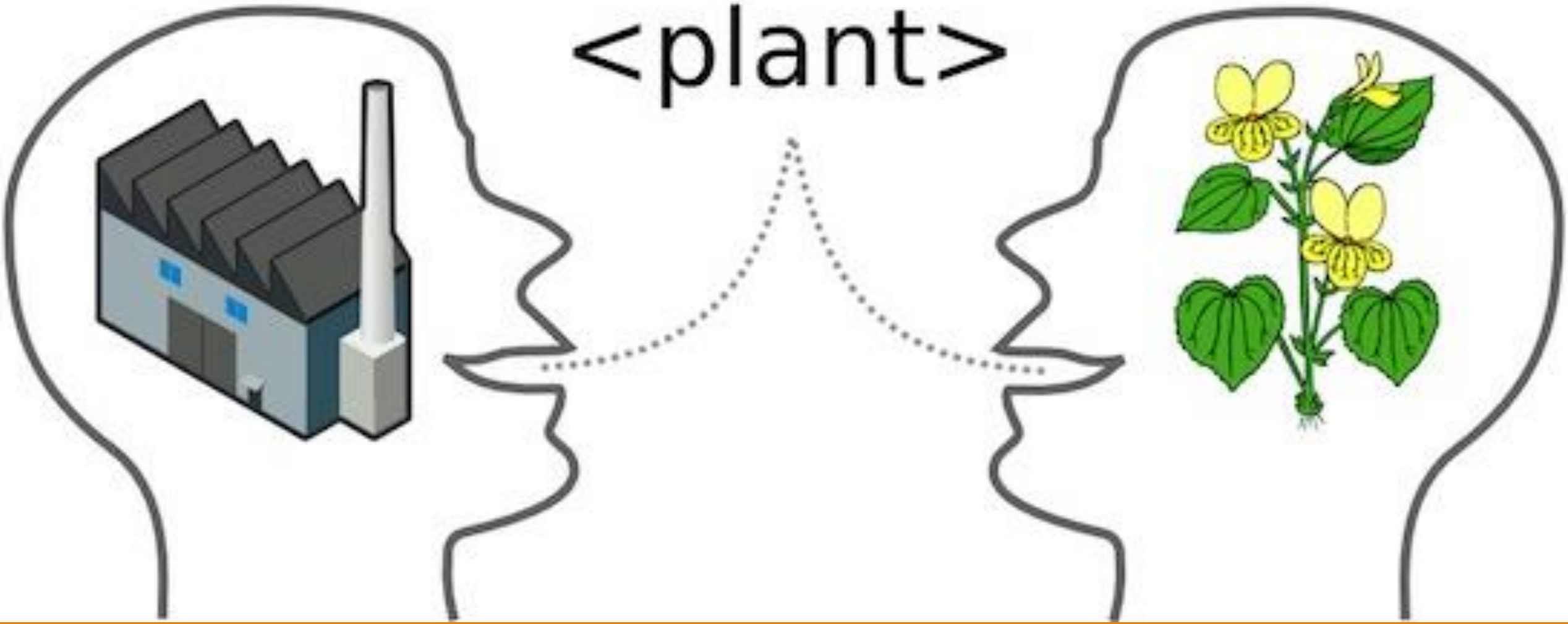
- Usa os *tokens* para gerar uma representação do código chamada **árvore sintática**
  - Gramáticas livres de contexto são usadas para especificar essa árvore

- As próximas fases de compilação usam essa árvore para gerar o programa objeto

Árvore Sintática

- Cada nó interno representa uma operação
- Cada nó folha representa um argumento da operação
- Mostra a ordem em que as operações devem ser executadas





Análise Semântica

Análise Semântica (I)

- Usa a árvore sintática e a tabela de símbolos para verificar a consistência semântica da linguagem fonte

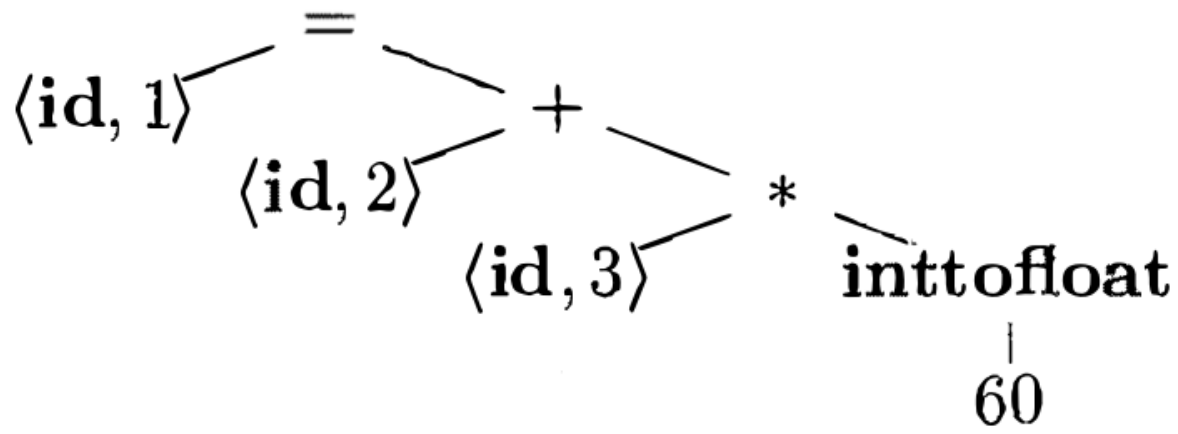
- Uma das suas funções é a verificação de tipos
  - Verifica se os operadores possuem operandos apropriados

Análise Semântica

- Conversões de tipos (coerções)
  - O operador de soma pode ser aplicado a dois inteiros
  - O operador de soma pode ser aplicados a dois reais
  - Na soma aplicada entre um inteiro e um real, o compilador pode converter o inteiro para real

Ajuste semântico

- A análise semântica pode fazer ajustes na árvore sintática
- Considerando que `position`, `initial` e `rate` tenham sido declaradas como reais
  - A análise semântica acrescenta um nó na árvore para converter o literal inteiro para real



Geração de Código Intermediário

Código Intermediário

- O código intermediário deve ser
  - Fácil de produzir
  - Fácil de traduzir para a linguagem alvo
- Facilita a otimização
- Uma representação intermediária bastante utilizada é o **código de três endereços**

Código de três endereços

- É uma sequência de instruções com no máximo três operandos cada

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Código de três endereços (I)

- Aspectos dessa representação
  - Cada atribuição possui apenas um operador, assim a ordem das instruções determina a ordem de execução das operações
  - Algumas instruções possuem menos de três operandos
  - O compilador precisa criar nomes temporários para guardar os valores computados

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Otimização de Código

Otimização

- Fase opcional
- Faz mudanças no código intermediário para torná-lo melhor
- Explorar as otimizações ao máximo exige muito tempo
- Existem otimizações simples que melhoram o código significativamente sem demandar muito tempo

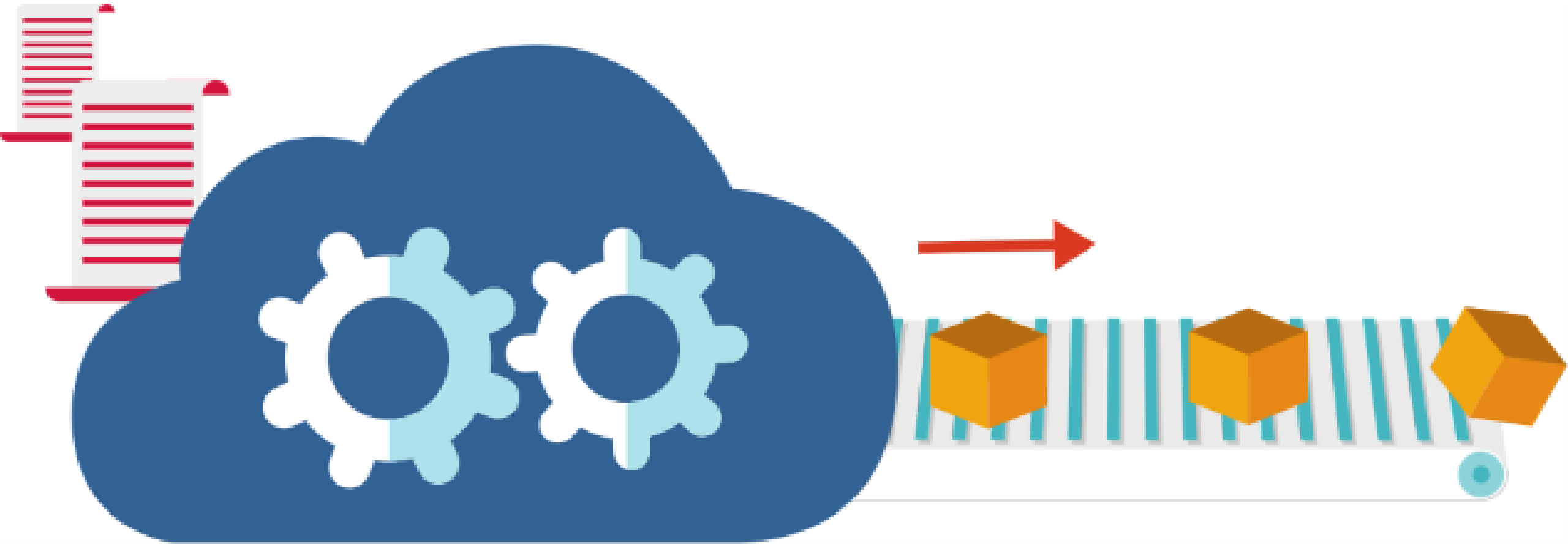
Exemplo de otimização

- A operação de conversão de tipo pode ser substituída diretamente pelo literal do tipo
- Nomes temporários desnecessários podem ser removidos

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

otimização

```
t1 = id3 * 60.0
id1 = id2 + t1
```



Geração de Código

Geração de Código

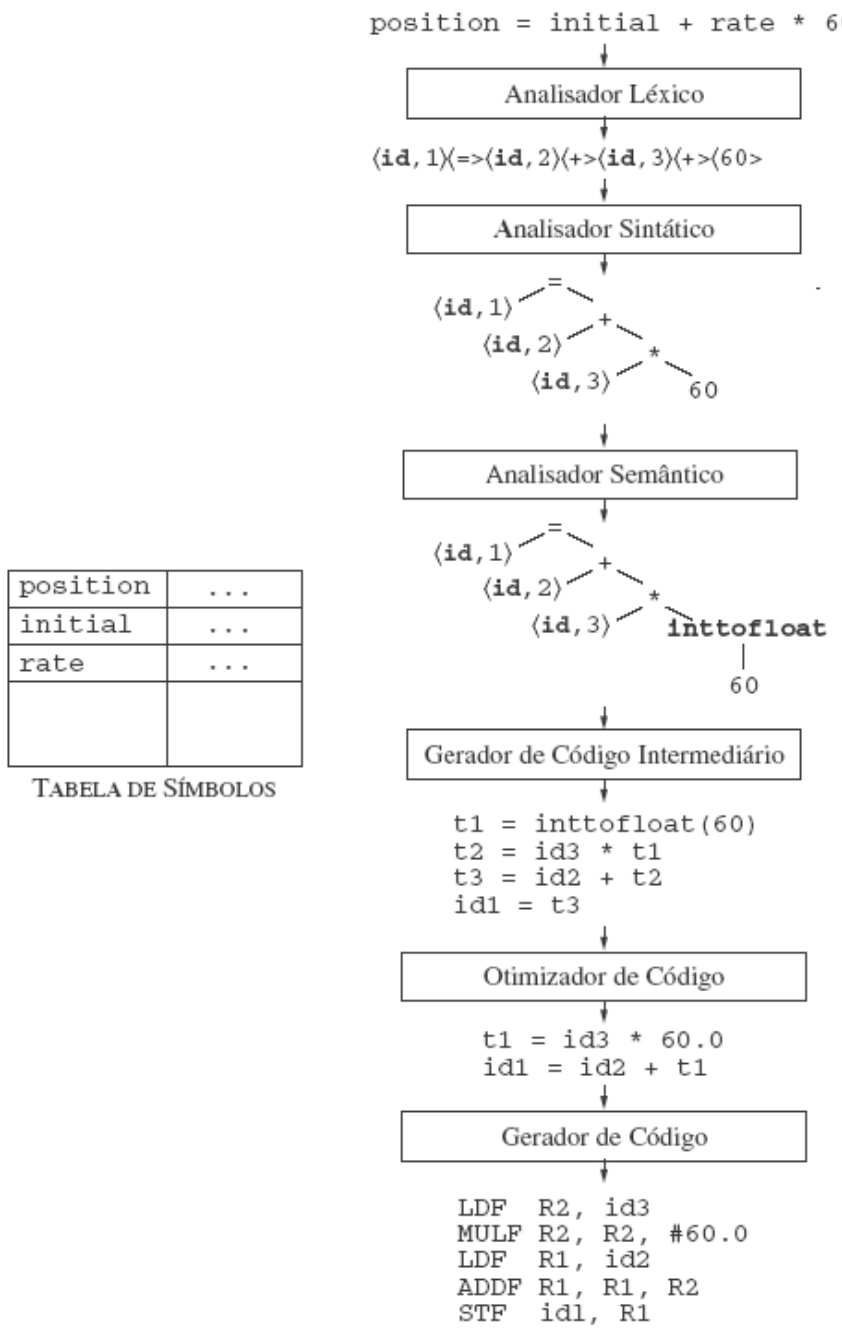
- Recebe o código intermediário e mapeia para o linguagem objeto
- Se a linguagem objeto for código de máquina, deve-se selecionar registradores para as variáveis usadas

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Geração de Código

```
LDF   R2, id3
MULF  R2, R2, #60.0
LDF   R1, id2
ADDF  R1, R1, R2
STF   id1, R1
```

Visão Geral



position	...
initial	...
rate	...

TABELA DE SÍMBOLOS

Bibliografia

[1] AHO, A. V.; SETHI, R.; ULLMAN, J. D. Compiladores: princípios, técnicas e ferramentas. Rio de Janeiro: LTC, 1995.

[2] LOUDEN, K.C. Compiladores: Princípios e Práticas. Cengage, 2006.

[3] TOSCANI, S.S.; PRICE, A.M.A. Implementação de Linguagens de Programação: Compiladores. Bookman, 3ª Edição, 2008.