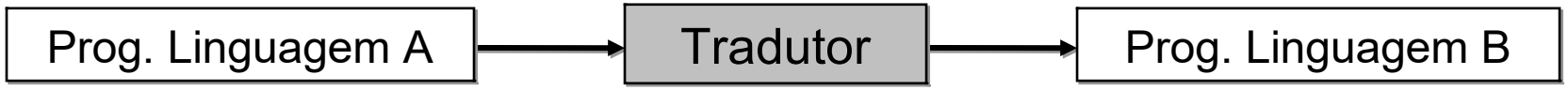
Introdução à Compilação

Prof. Leandro Magno

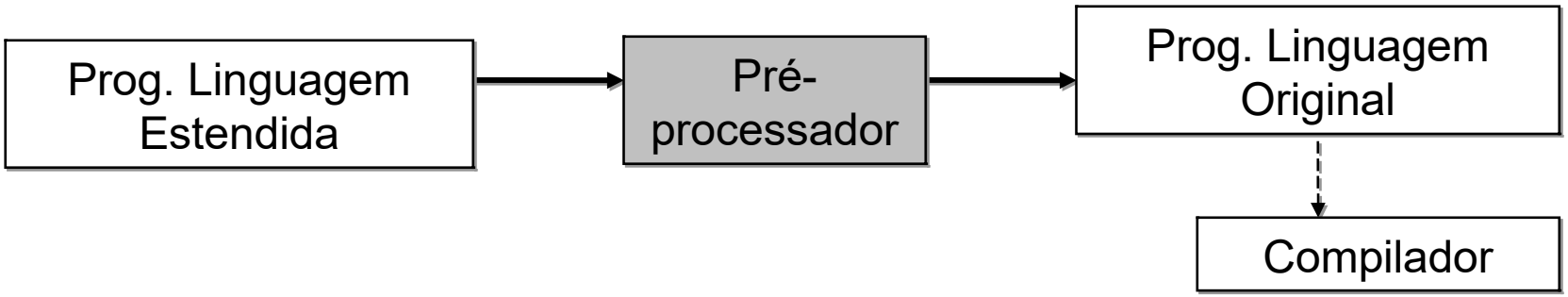
Slides adaptados a partir do material cedido pelos professores Heloise Manica Paris Teixeira, Yandre M. G. e Costa e Profa. Valéria D. Feltrim (DIN – UEM)

Tipos de Tradutores

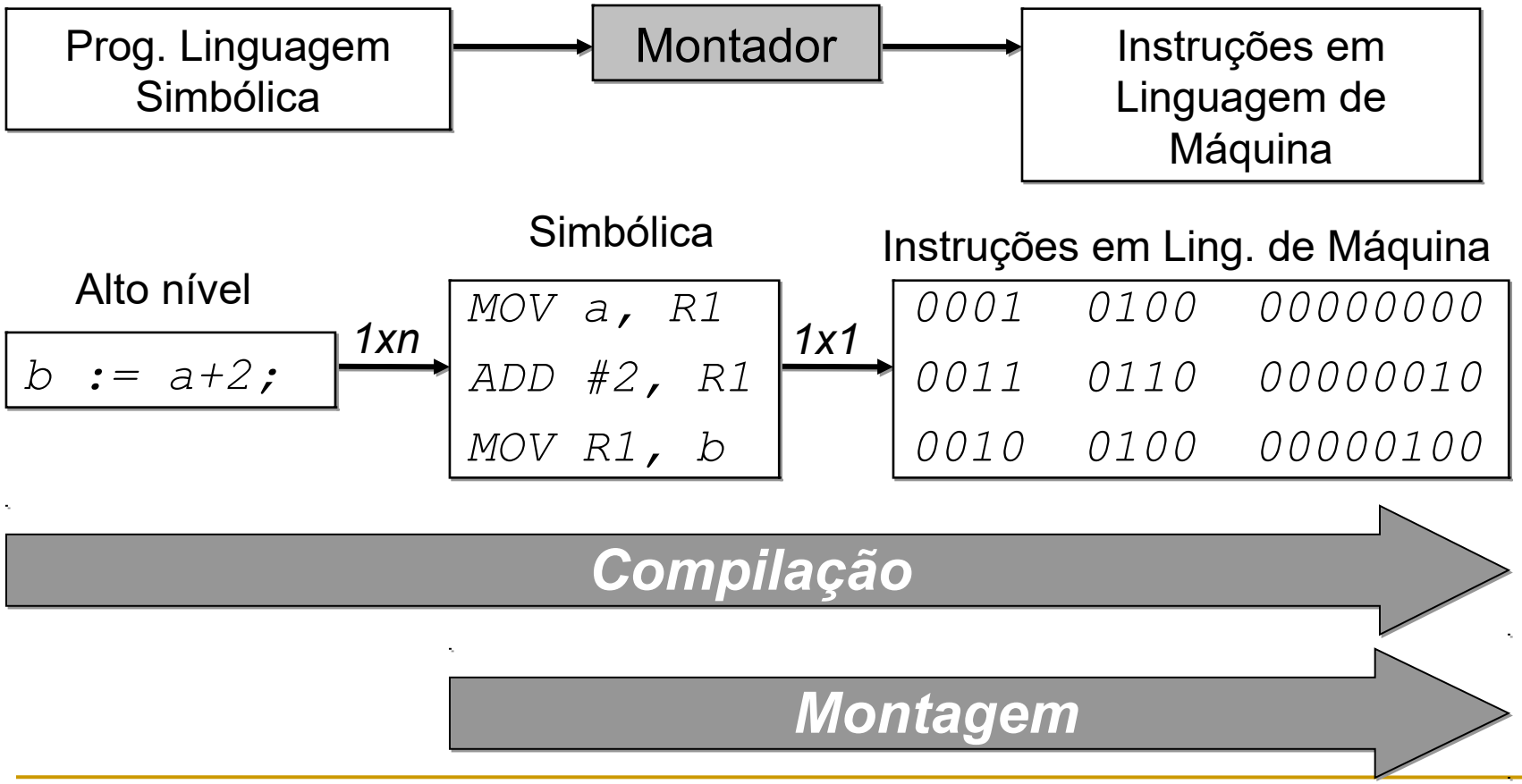
- **Tradutor**: programa que transforma um programa fonte escrito numa linguagem em um programa equivalente escrito em uma linguagem diferente;



- **Pré-processador**: programa que transforma um programa escrito em uma linguagem estendida em um programa equivalente escrito em linguagem original;

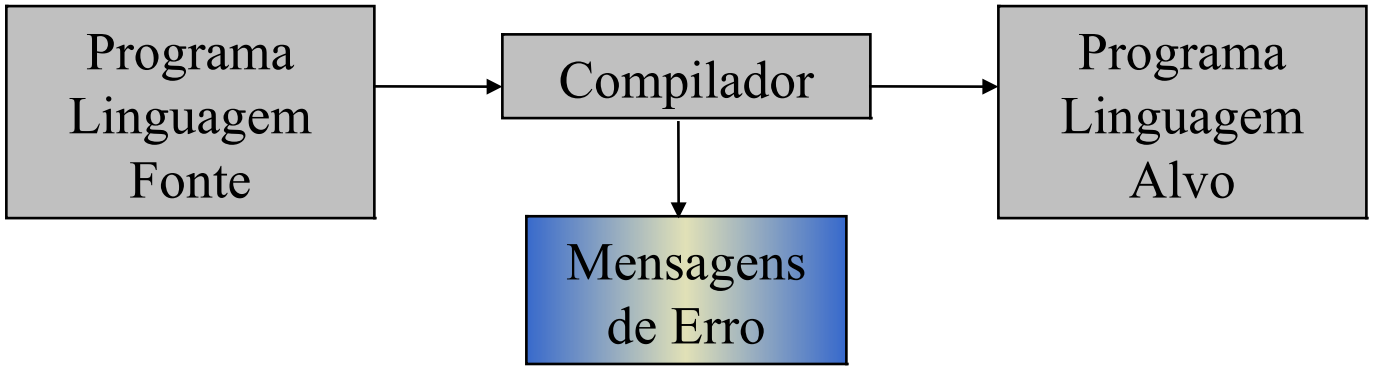


- **Montador**: tradutor que transforma um programa escrito em uma linguagem simbólica (de baixo nível) em instruções equivalentes em linguagem de máquina;



Compilador

- é um programa que traduz o código fonte escrito em uma linguagem de mais alto nível para outra linguagem de mais baixo nível;
- a linguagem original é chamada de linguagem fonte, e a linguagem final é chamada de linguagem destino ou alvo;



- **Interpretador**:
  - tradutor que funciona em tempo de execução.
  - Estes programas traduzem programas codificados em linguagem de alto nível para um código intermediário e o coloca em execução.
  - executam instrução por instrução do código-fonte

Interpretadores vs. compiladores

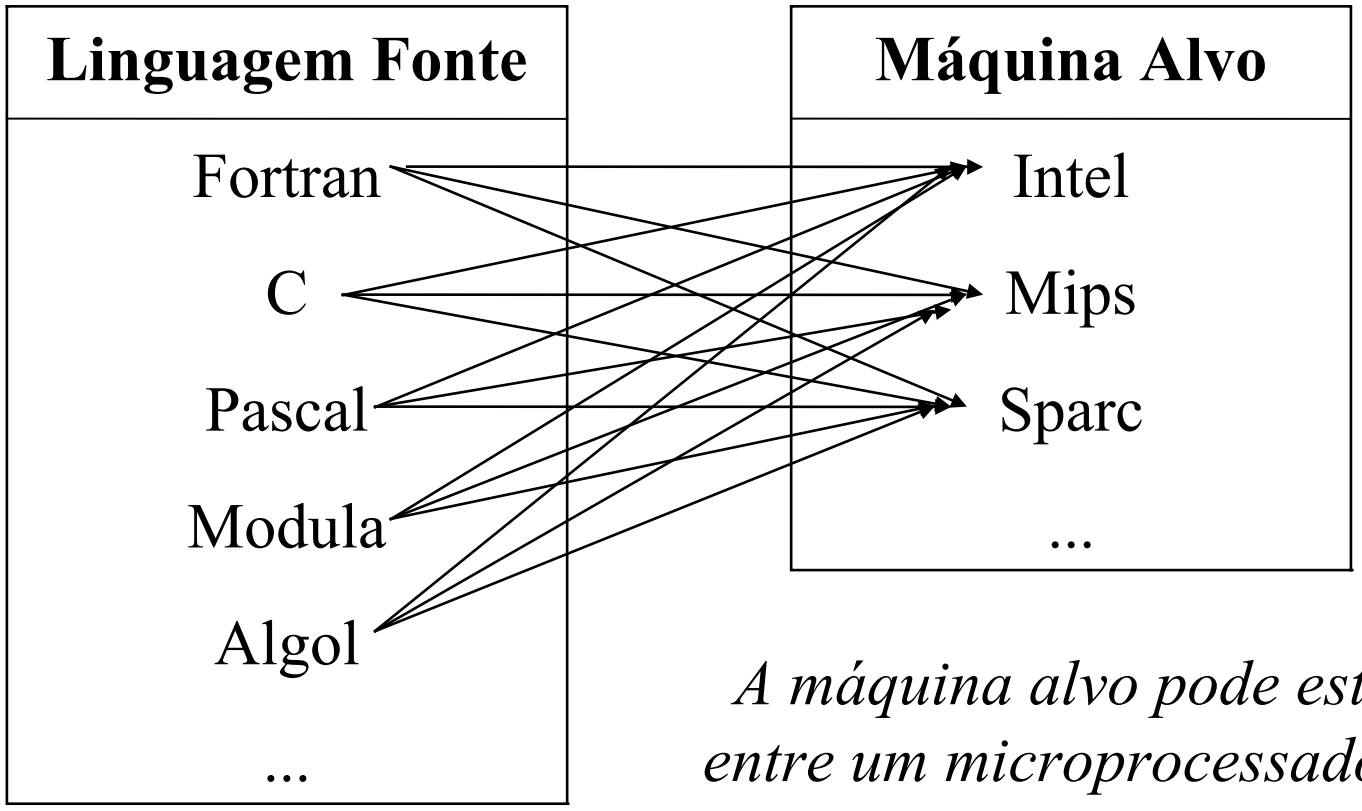
- **Qual a diferença entre compilação e interpretação?**

- Interpretadores
  - Menores que os compiladores
  - Mais adaptáveis a ambientes computacionais diversos
  - Tempo de execução maior
  - Ex. Javascript, Python, Perl

- Compiladores
  - Compila-se uma única vez, executando-se quantas vezes se queira
  - Tempo de execução menor
  - Ex. C, Pascal, Delph, etc...

- Compiladores híbridos (Java)
  - Compila-se para um código intermediário/virtual, que, por sua vez, é interpretado por uma máquina virtual

Variedade de compiladores



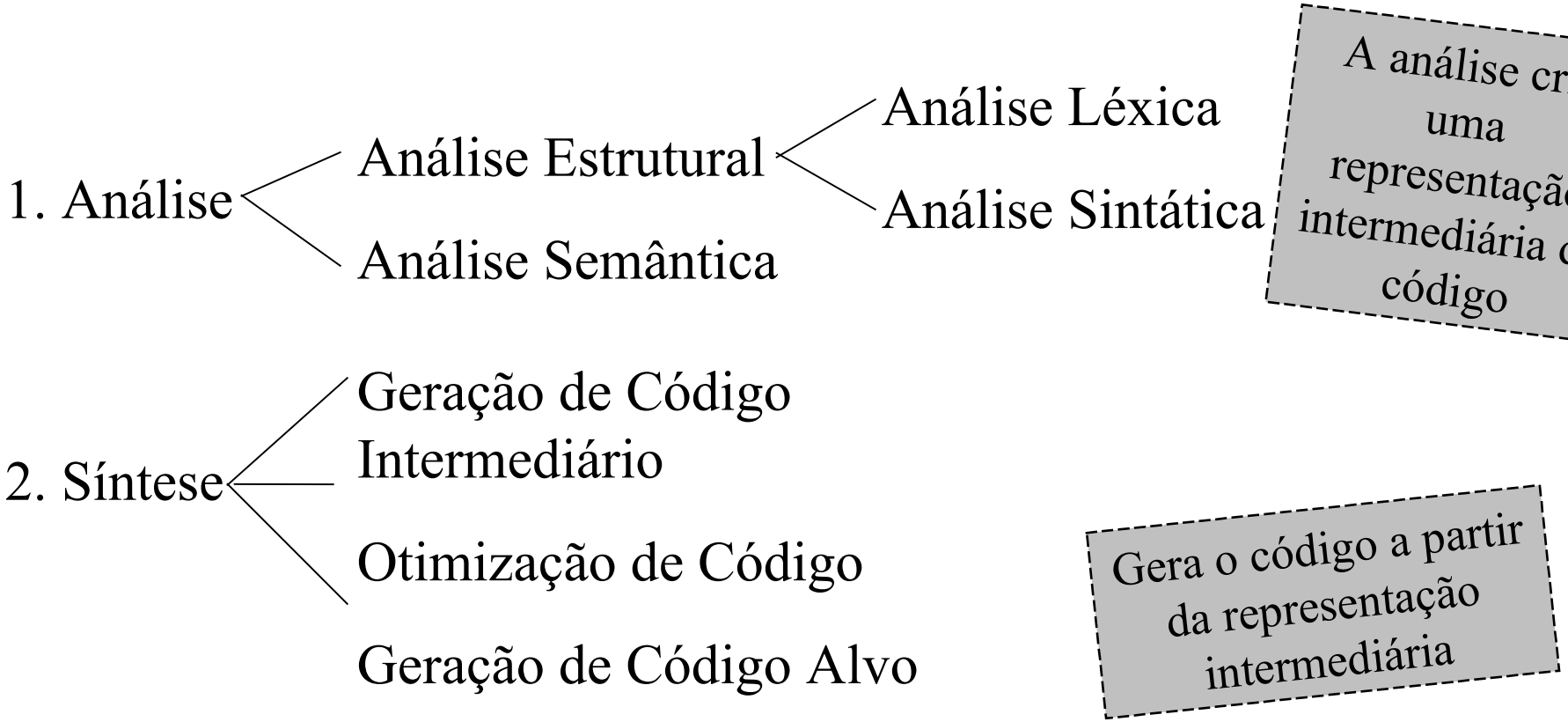
A máquina alvo pode estar entre um microprocessador e um supercomputador

Motivos para desenvolver um compilador?

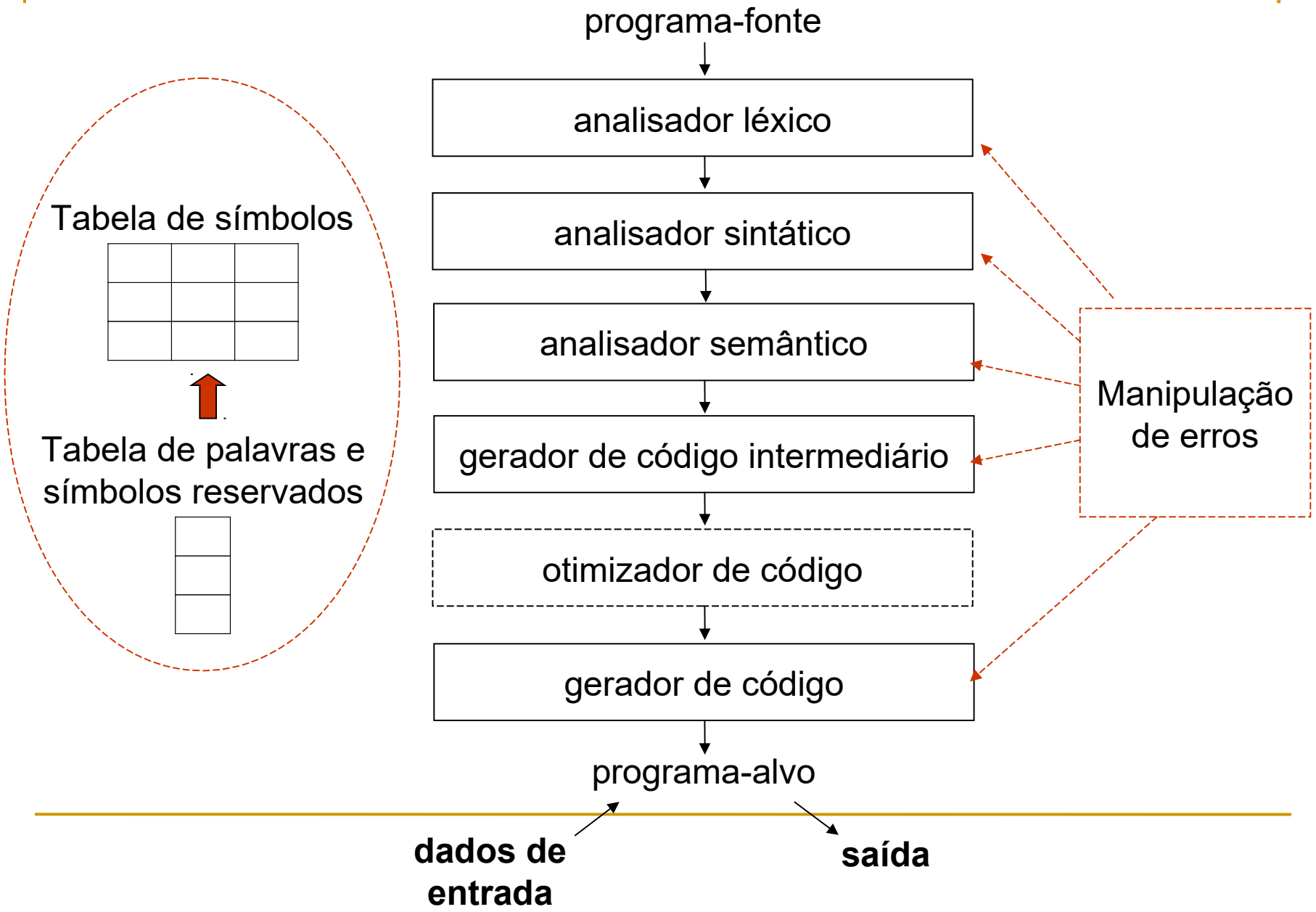
- Em que situações é necessário construir um novo compilador?
  - Criação de uma nova Linguagem
  - Extensão de uma linguagem existente
  - Surgimento de uma nova máquina
  - Desempenho do compilador existente

Estrutura Geral de um Compilador

(Modelo de compilação de Análise e Síntese)



Estrutura geral de um compilador



Fases da compilação

- Lexical: palavras (*tokens*) do programa
  - i, while, =, [, (, <, int
  - Erro: j@
- Sintática: combinação de tokens que formam o programa
  - comando_while → while (expressão) comandos
  - Erro: while (expressão comandos
- Semântica e contextual: adequação do uso
  - Tipos semelhantes em comandos (atribuição, por exemplo),
  - uso de identificadores declarados
  - Erros: i="1"
- Geração de código: especificidades da máquina-alvo e sua linguagem
  - Alocação de memória, uso de registradores
  - Erro: a[1000000000]

Estruturas da compilação

- Tabela de Símbolos
  - É uma estrutura de dados com algoritmos apropriados para a manipulação de seus dados (listas, árvores, arranjos, ...);
  - A estrutura de dados que manipula esta tabela deve permitir rápido armazenamento ou recuperação dos dados.
  - Guarda informações sobre os identificadores:
    - Nome
    - Endereço
    - Tipo

Estruturas da compilação

- Como saber durante a compilação de um programa o tipo e o valor dos identificadores, escopo das variáveis, número e tipo dos parâmetros de um procedimento, etc.?
  - Tabela de símbolos

Identificador	**Classe**	**Tipo**	**Valor**	**...**
i	var	integer	1	...
fat	proc	-	-	...
...				

Estruturas da compilação

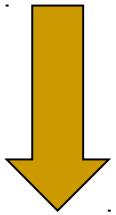
- Como diferenciar palavras e símbolos reservados (while, int, :=) de identificadores definidos pelo usuário?
  - Tabela de palavras e símbolos reservados

int
while
:=
...

Analisador léxico

- Identifica no arquivo fonte os símbolos pertencentes à linguagem;
- Reconhecimento e classificação dos tokens
  - Expressões regulares, autômatos

x:=x+y*2



<x,id_1> <:=,:=> <x,id_1> <+,op> <y,id_2> <*,op> <2,num>

Analisador sintático

- Verificação da formação do programa
  - Com base em uma gramática livre de contexto
  - Exemplo: regras abaixo definem o comando **WHILE (Pascal)**

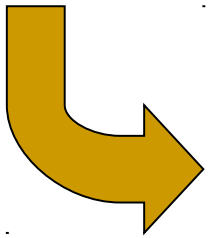
```
comando       → comandoWhile
              | comandoIf
              | comandoAtrib,
              | ...
comandoWhile → WHILE expr_bool DO comando;
expr_bool → expr_arit < expr_arit
            | expr_arit > expr_arit
            | ...
expr_arit → expr_arit * termo
          | termo
          | ...
termo → expr_arit
      | NÚMERO
      | IDENTIFICADOR
```

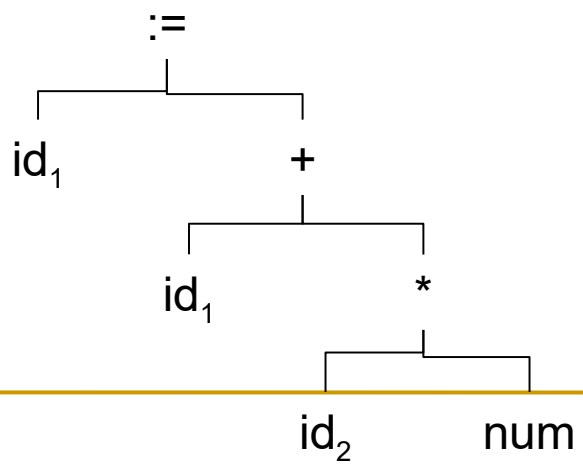
Analisador sintático

- Verificação da formação do programa
  - Com base em uma gramática livre de contexto
  - Representação explícita ou não da árvore de derivação

<x,id_1> <:=,:=> <x,id_1> <+,op> <y,id_2> <*,op> <2,num>

comando_atribuição → id_1 := id_1 op id_2 op num

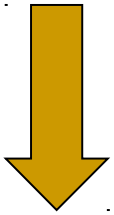




Analisador semântico

- Verificação do uso adequado

$$id_1 := id_1 \text{ op } id_2 \text{ op num}$$



$$(id_1)_{int} := (id_1 \text{ op } id_2 \text{ op num})_{int}$$

$$\text{busca\_tabela\_símbolos}(id_1)=\text{TRUE}$$

$$\text{busca\_tabela\_símbolos}(id_2)=\text{TRUE}$$

Analisador semântico

- Exemplos de verificações semânticas:
  - Compatibilidade de tipos

    ... var A: boolean; B: real;

    … A:=B+0,5;
  - Duplicidade de identificadores:

    Var A, A, B: integer;
  - Compatibilidade entre parâmetros formal e atual:

    procedure X (a, b: integer);

    begin

    ...

    end;

    ...

    X(a, b, c);

Gerador de código intermediário

- Gera um conjunto de instruções, equivalentes ao programa fonte, para uma máquina hipotética.
  - Ex.: `A:=(B+C) * (D+E)`
    - Gera-se quádruplas:
      - `(+, B, C, T1)`
      - `(+, D, E, T2)`
      - `(*, T1, T2, A)`
- A maior diferença entre o código intermediário e o código objeto é que o intermediário não especifica detalhes de baixo nível de implementação, tais como endereços de memória e registradores, entre outros.

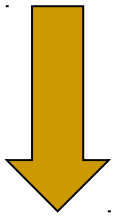
Gerador de código intermediário

- Esta representação intermediária tem as seguintes vantagens:
  - Possibilita a otimização do código intermediário, a fim de obter código objeto final mais eficiente;
  - Resolve, de maneira gradual, problemas da passagem do código fonte para objeto

Gerador de código intermediário

- Geração de código intermediário/preliminar

$id_1 := id_1 \text{ op } id_2 \text{ op num}$



Variáveis temporárias geradas pelo compilador

$temp1 := id_2 * 2$

$temp2 := id_1 + temp1$

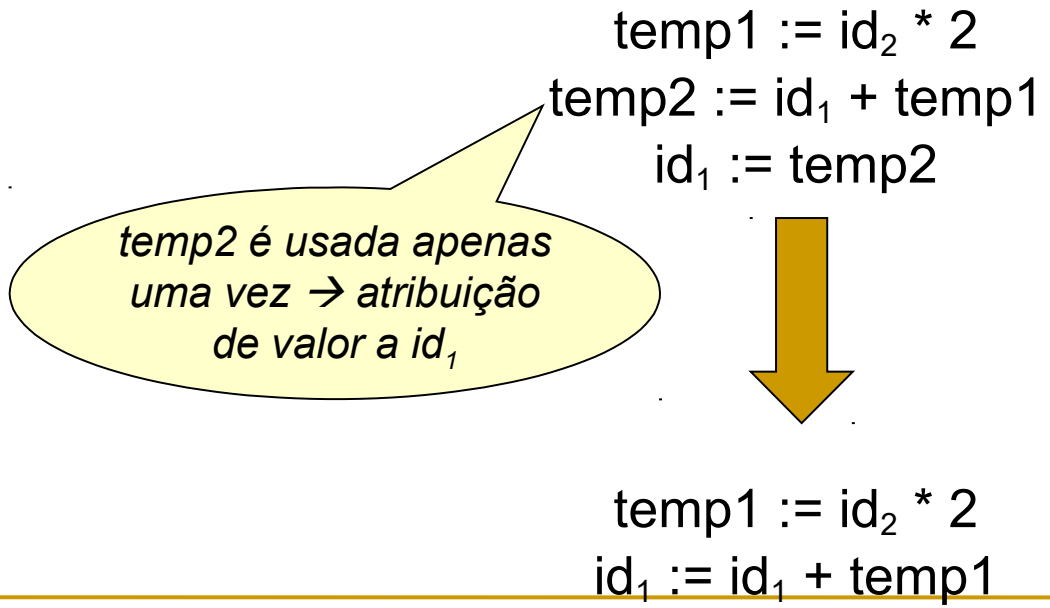
$id_1 := temp2$

Código de três endereços

(tipo de código no qual cada instrução deve ter, no máximo, três operandos)

Otimizador de código Intermediário

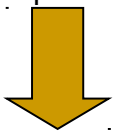
- Função:
  - melhorar o código intermediário de forma que ocupe menos espaço e/ou “execute” mais rápido.
- Exemplo



Gerador de código Objeto

- Sua principal função é gerar o código equivalente ao programa fonte para uma máquina real, a partir do código intermediário otimizado.
- Geração do código para a máquina-alvo

temp1 := id_2 * 2
id_1 := id_1 + temp1

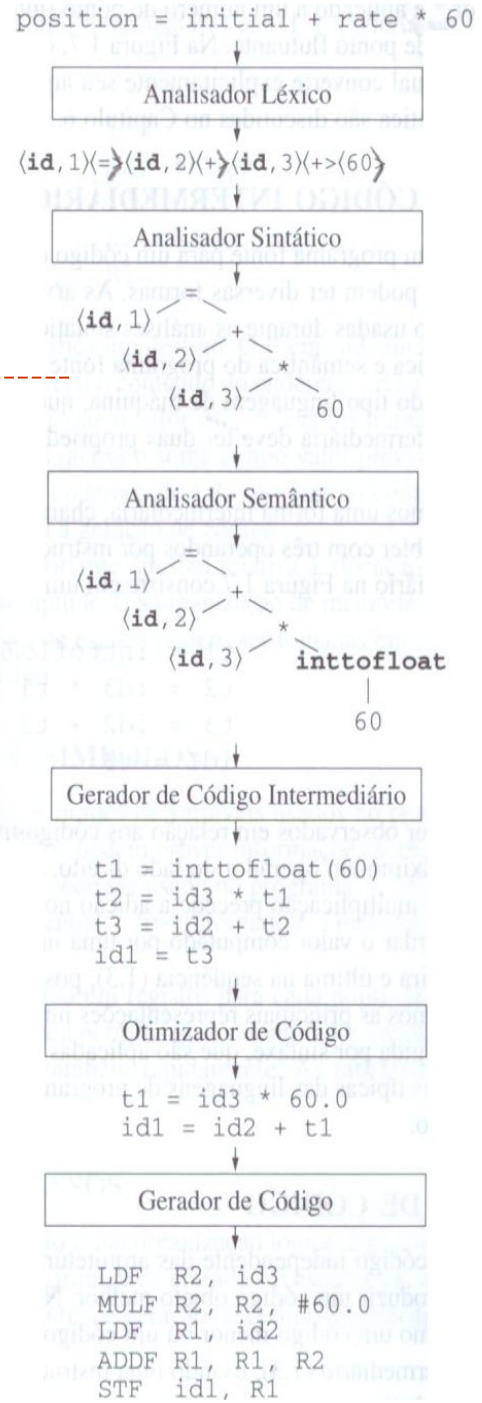


MOV id_2 R1
MULT 2 R1
MOV id_1 R2
ADD R1 R2
MOV R2 id_1

Tokens e seus lexemas

Árvore sintática
(implícita ou explícita)

Tabela de
símbolos <u>única</u>



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

TABELA DE SÍMBOLOS

Passagens

- Passagem: leitura de um arquivo de entrada e escrita de um arquivo de saída
- Em relação a passos de compilação, como são classificados os compiladores?
  - Compiladores com várias passagens
  - Compilador de uma única passagem

Passagens

- Compiladores com várias passagens (n-passos)

- Vantagens
  - Menor utilização da memória do computador, já que cada passo exerce apenas uma parte das funções de todo o compilador.
  - Maior possibilidade de se efetuar otimizações
  - Os projetos e implementações das várias partes do compilador são mais independentes.
- Desvantagens
  - Maior volume de entrada/saída (tipicamente acesso a disco ou à rede)
  - Em geral há aumento do tempo de compilação.
  - Aumento do projeto total, com necessidade de introdução das linguagens intermediárias.

Passagens

- Compilador de uma única passagem
  - O programa alvo ou objeto já é o produto final do compilador, ou seja, o programa em linguagem de máquina desejado.
    - código-fonte → código-alvo
  - Todo processo de compilação em memória: o código-alvo é gerado enquanto o código-fonte é lido.
  - Possíveis problemas:
    - Falta de memória

Exercícios

- Em que situações é necessário construir um novo compilador ?
- Qual a diferença entre compilação e interpretação?
- Porque programas compilados em geral apresentam maior desempenho?
- Em relação a passos de compilação, como são classificados os compiladores?
- Defina token.
- Qual(is) a(s) principais função(ões) do analisador léxico, sintático e semântico?
- Considere a linguagem de programação Pascal e dê pelo menos um exemplo para cada tipo de erro: léxico, sintático e semântico.
- Explique as etapas que constituem o processo de compilação e seu inter-relacionamento.