Projeto de Compiladores

FIR – Faculdade Integrada do Recife

João Ferreira

26 e 27 de fevereiro de 2007

Agenda da Aula

- Revisão
  - Linguagem de Programação
  - Tradutores
  - Compilador
- As Fases de Um Compilador

Linguagem de Programação

- Comunicação entre o programador e o computador
- Notação formal para expressar algoritmos
- Computador
  - Linguagem de baixo nível (bits)
- Humanos
  - Linguagem baseada em instruções de alto nível (comandos)

Tradução Alto/Baixo Nível

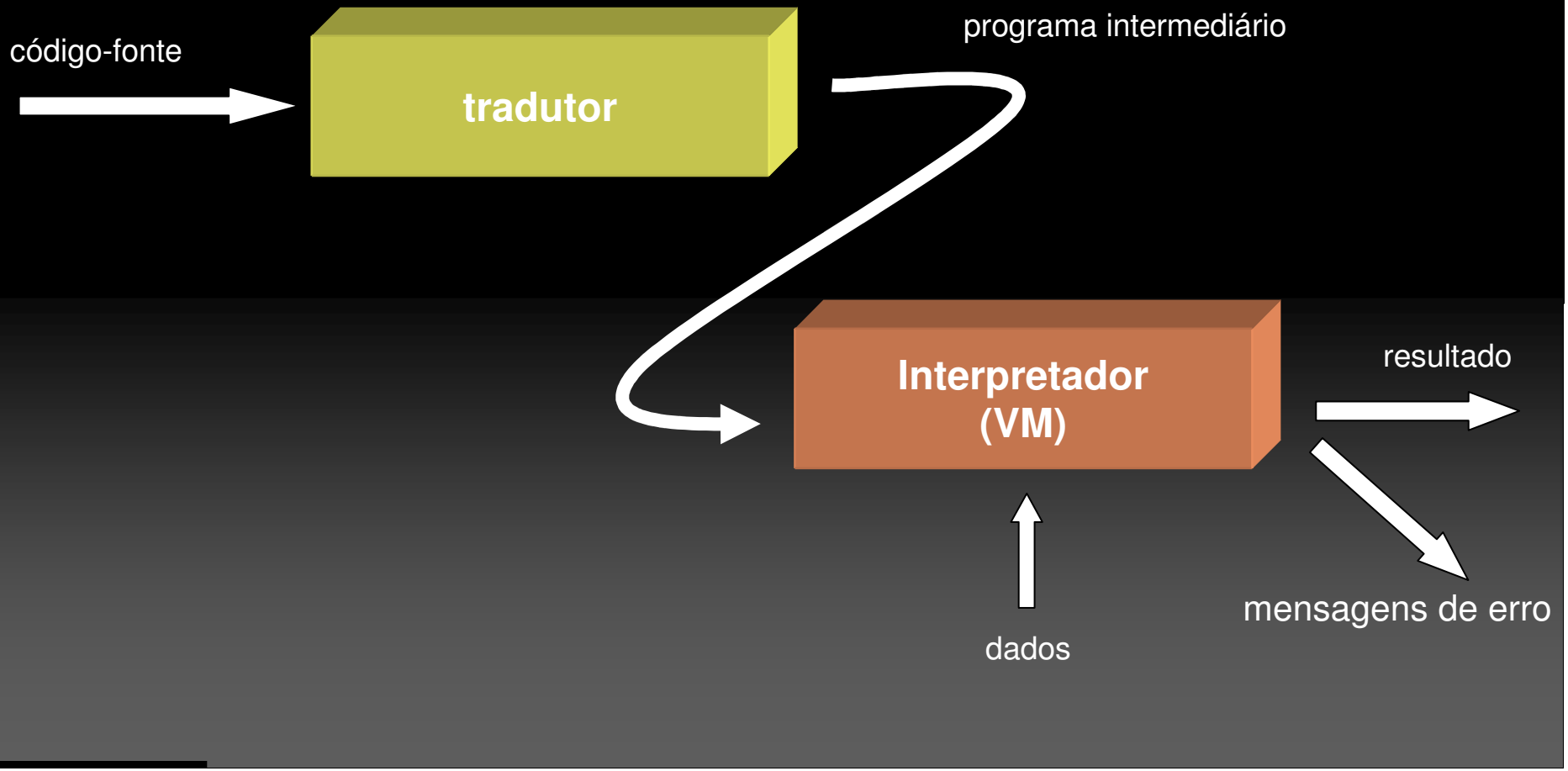
Tradutor

- Compiladores
  - Performance superior
  - Otimização do código
  - Baixa portabilidade
- Interpretador
  - Não gera código executável
  - Geralmente tem implementação mais simples
  - Geralmente são menores
  - Alta portabilidade
- JIT – Just in Time Compilation

Compilador

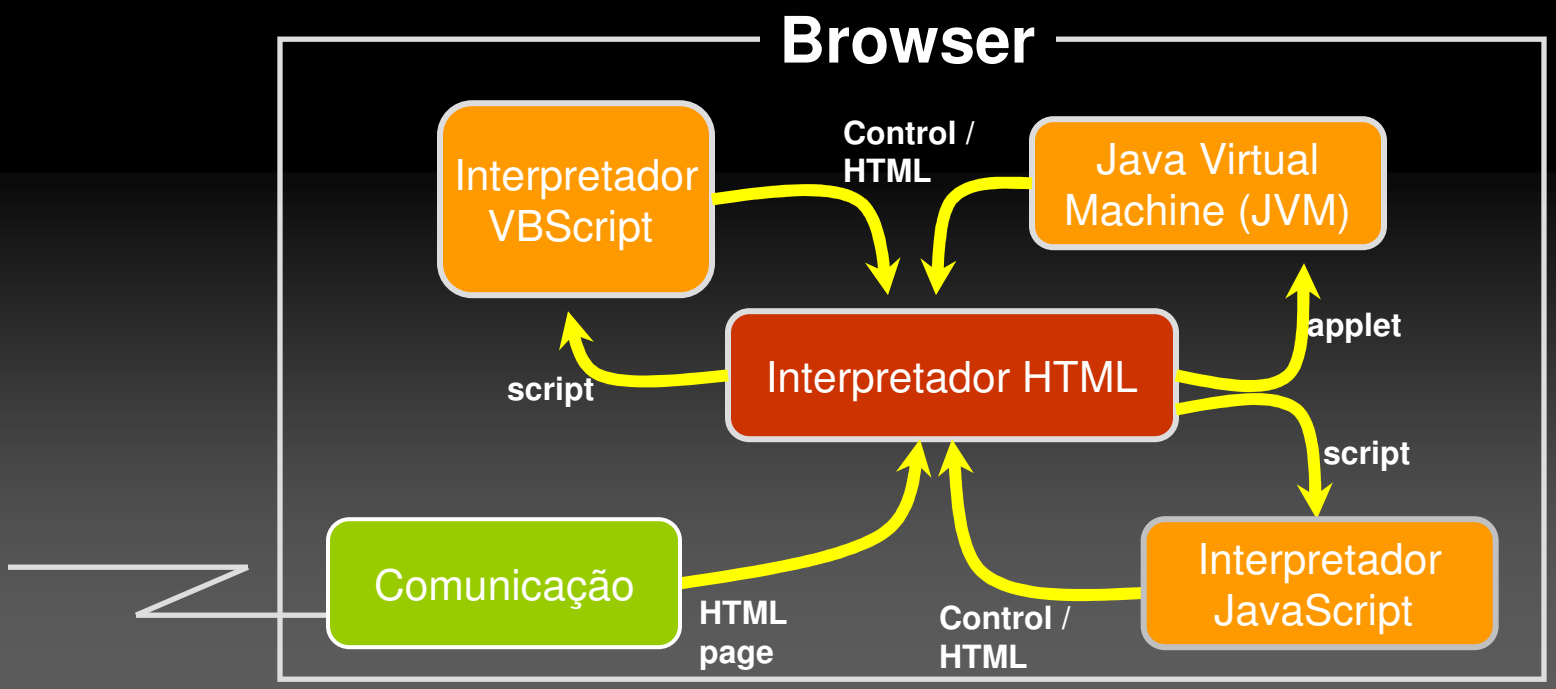

código-fonte
compilador
programa objeto
mensagens de erro
dados
programa objeto
resultado

Interpretador

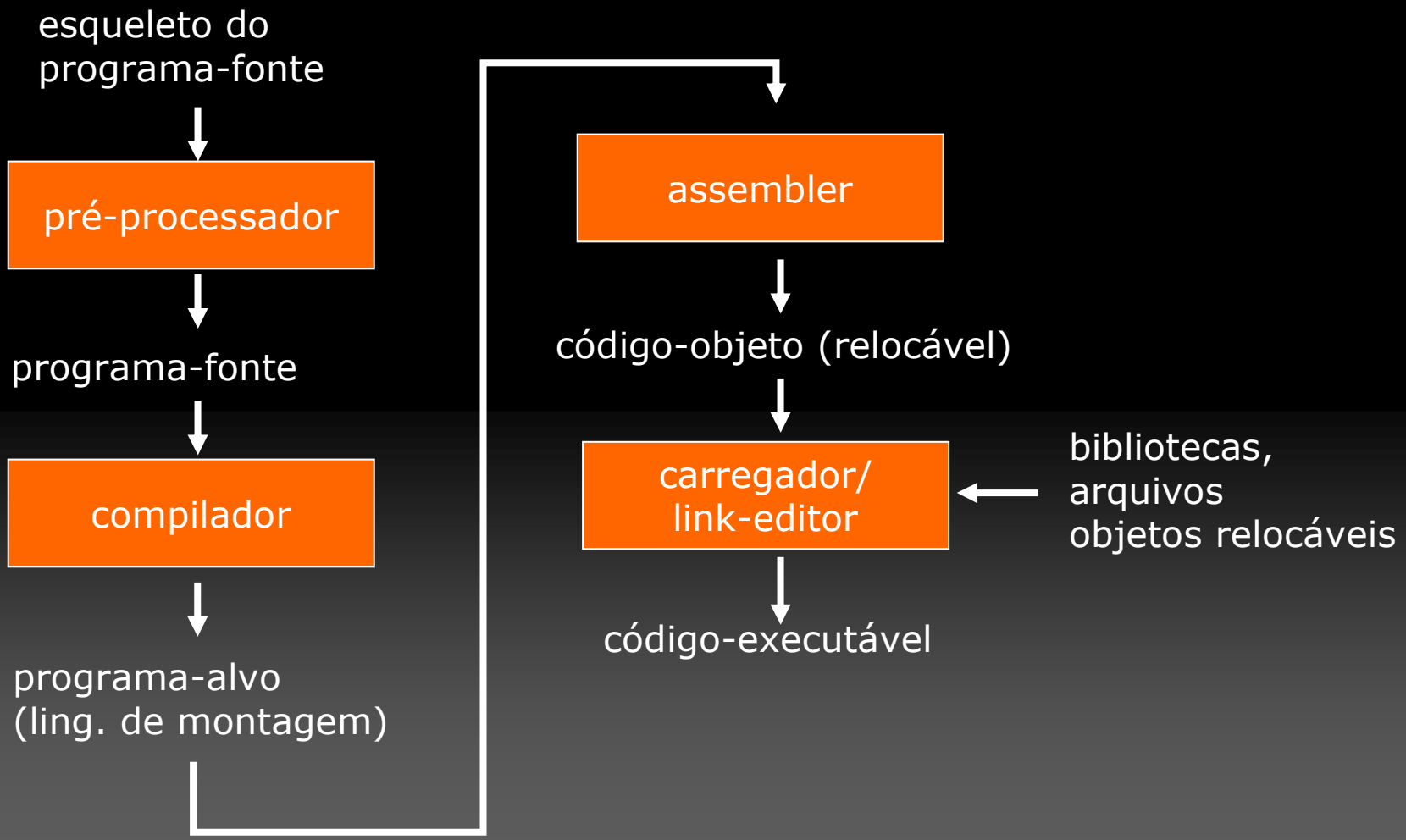


Enquanto isso na Web...

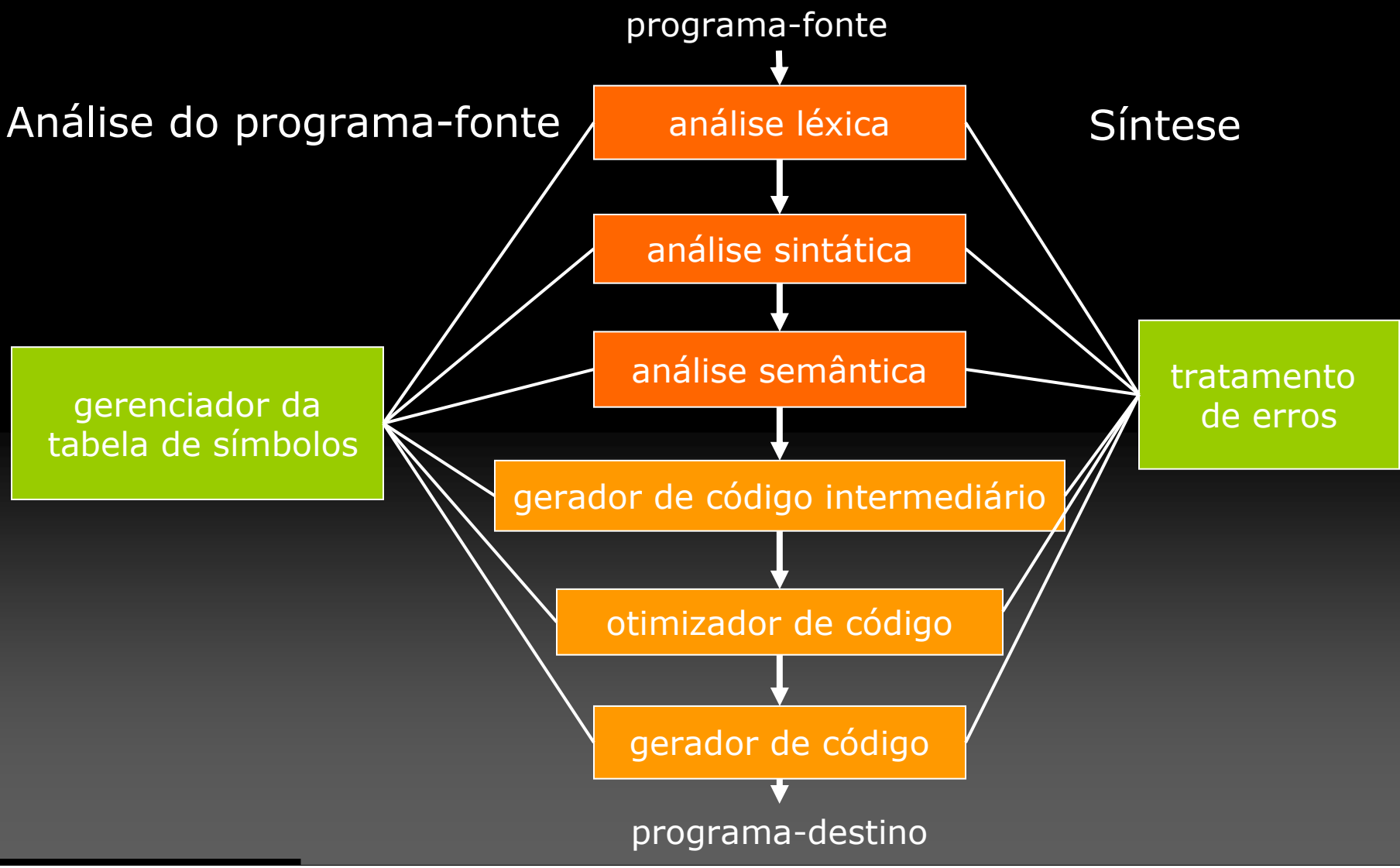
- Várias linguagens são usadas para adicionar conteúdo dinâmico
  - Uma boa parte é executada simultaneamente!



Contexto de Um Compilador



As Fases de Um Compilador



Gerenciamento da Tabela de Símbolos

- Informalmente é chamado de “fase”
- Estrutura de dados para armazenamento e recuperação de identificadores
- Informações sobre
  - Alocação de Memória
  - Tipo dos Identificadores
  - Escopo dos Identificadores
  - (Procedimento/Função) Número e Tipo dos Argumentos e Retorno
  - (Procedimento/Função) Forma de Passagem de Parâmetros
- Nem sempre as informações são preenchidas ao mesmo tempo

Gerenciamento da Tabela de Símbolos

position := initial + rate * 60

Tabela de Símbolos

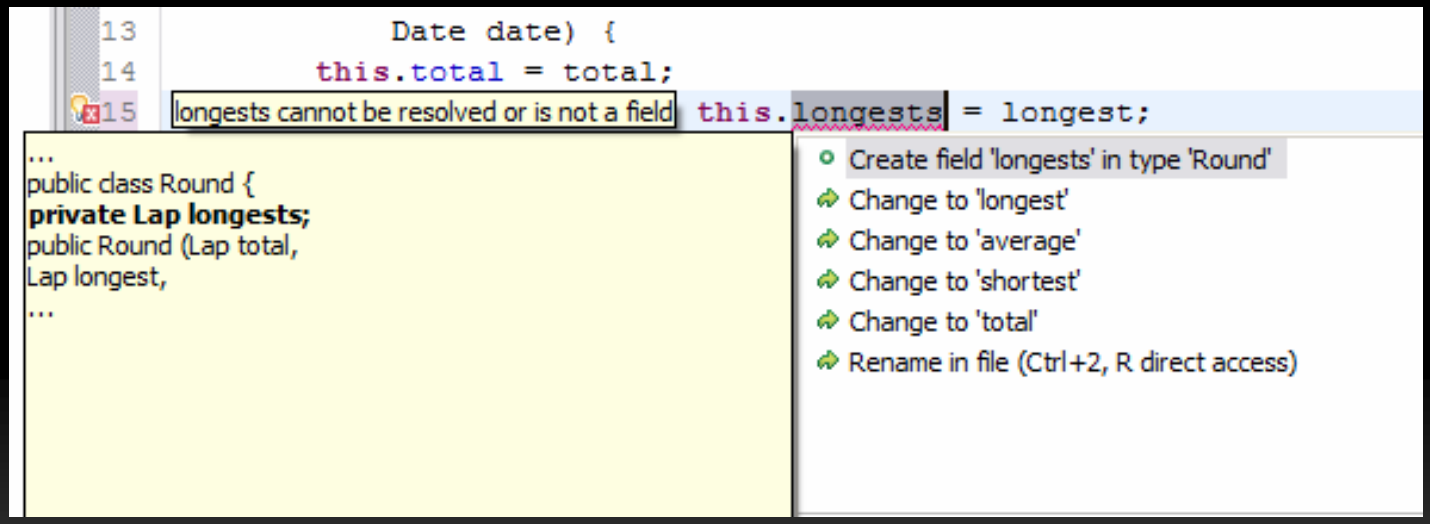
	identificador	**tipo**	
1	position		...
2	initial		...
3	rate		...
...	...	...	...

Detecção e Impressão de Erros

- Informalmente é chamada de “fase”
- Erros podem ser encontrados em qualquer fase do compilador
- O ideal é que mesmo após um erro seja encontrado o compilador prossiga
- Opcionalmente o compilador pode apresentar alternativas para a correção do erro encontrado

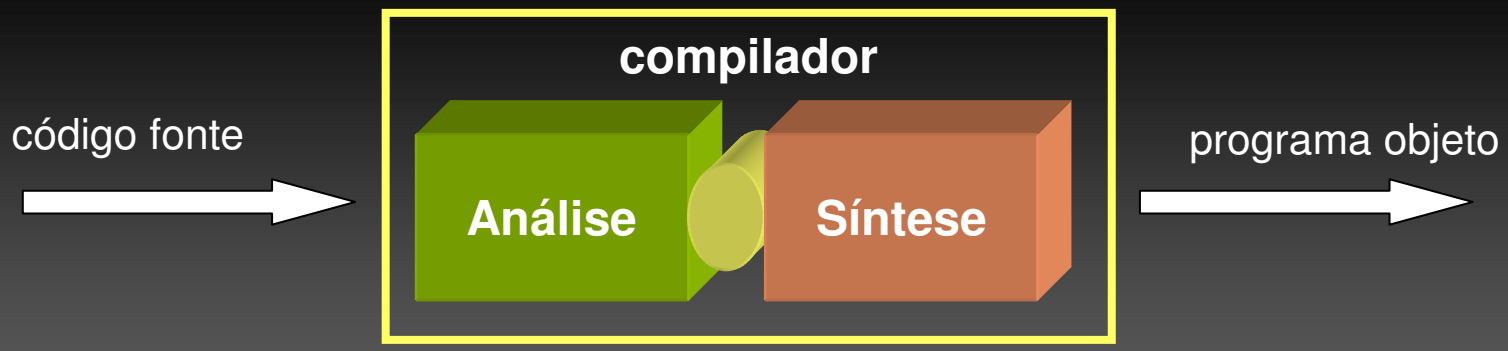
Detecção e Impressão de Erros

Eclipse



Modelo Análise-Síntese

- A compilação pode ser dividida em duas partes
  - Análise – divide e estrutura o código-fonte criando uma representação intermediária
  - Síntese – monta o código desejado a partir da representação intermediária



Análise

- É uma tarefa relativamente simples
- Divide o programa-fonte em suas partes constituintes
- Cria uma representação intermediária do programa-fonte
- Inclui possíveis mecanismos de pré-processamento

Análise

- Outras ferramentas utilizam técnicas de análise:
  - Editores de estrutura
  - Verificadores de sintaxe
  - Formatadores para impressão
- Fases
  1. Léxica
  2. Hierárquica
  3. Semântica

Análise Léxica (Scanning)

- Caracteres são lidos da esquerda para direita e agrupados em *tokens* (seqüência de caracteres que possui um significado).
  - for
  - while
  - if
- Espaços em branco são eliminados.

Análise Léxica (Exemplo)

position := initial + rate * 60

- Identificadores
  - position
  - initial
  - rate
- Operador de atribuição
  - :=
- Operadores
  - +
  - *
- Número
  - 60

	identificador	tipo	
id_1	position		...
id_2	initial		...
id_3	rate		...
...	...	...	...

Análise Sintática (Parsing)

- *Tokens* são agrupados hierarquicamente em "frases gramaticais" que serão utilizadas pelo compilador para sintetizar a saída do programa.

- As operações do código-fonte são reconhecidas e armazenadas em uma estrutura hierárquica chamada "árvore sintática"
  - Cada nó representa uma operação
  - Cada nó-filho representa os argumentos da operação

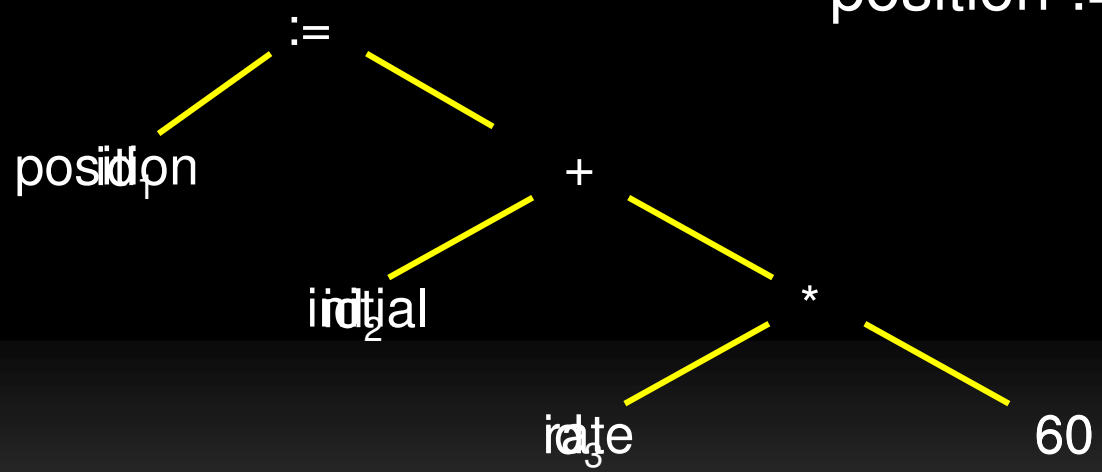
Análise Sintática (Parsing)

- Regras recursivas expressam a estrutura hierárquica do programa

- Exemplo
  - Um identificador é uma expressão
  - Um número é uma expressão
  - Se $expr_1$ e $expr_2$ são expressões, então também o serão:
    - $expr_1 + expr_2$
    - $expr_1 * expr_2$
    - $(expr_1)$

Árvore Sintática (Exemplo)

position := initial + rate * 60



Análise Semântica

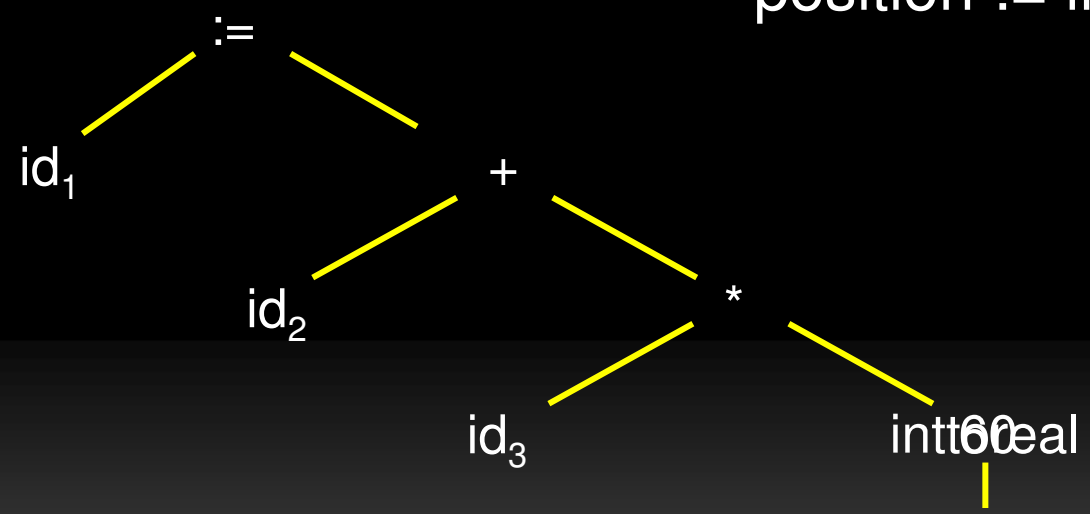
- Verificações a fim de assegurar que os componentes do programa estão agrupados corretamente de acordo com a semântica da linguagem
- Procura erros semânticos (de significado) no programa.
  - Exemplo: verificação de tipos
- A expressão

  x = x + 3.0

  está sintaticamente correta, mas pode estar semanticamente certa ou errada, dependendo do tipo de x.

Análise Semântica (Exemplo)

position := initial + rate * 60



Síntese

- Constrói o programa alvo a partir da representação intermediária produzida pela análise
- Utiliza a maior parte das técnicas especializadas
- É uma tarefa relativamente complexa

Geração de Código Intermediário

- Idealmente deve ser fácil de produzir e também de traduzir para a linguagem-destino.
- Na prática, está gerando código para uma máquina abstrata.
- Three-address-code
  - usa memória como registrador
  - uso extensivo de nomes temporários
  - usa apenas operações simples
  - pode utilizar menos do que 3 endereços

Geração de Código Intermediário (Exemplo)

position := initial + rate * 60

temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 = temp3

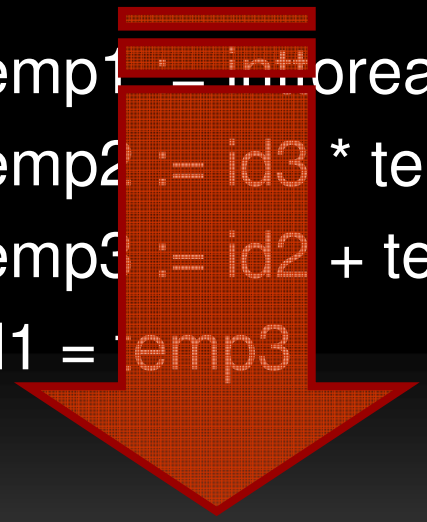
Otimização de Código

- Melhorar o código intermediário através de transformações

- Melhorar =
  - tempo de execução
  - uso de memória
  - tamanho do código executável

Otimização de Código (Exemplo)

position := initial + rate * 60



temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 = temp3

temp1 := id3 * 60.0
id1 := id2 + temp1

Geração de Código

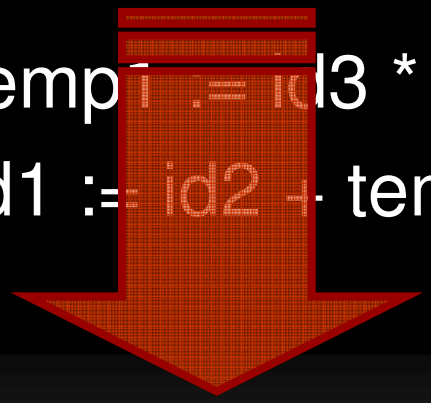
- Última fase
- Responsável por produzir o código-alvo
- Alocação de memória para cada variável usada no programa
- Cada instrução intermediária é traduzida numa seqüência de instruções de máquina

Geração de Código (Exemplo)

position := initial + rate * 60

temp1 := id3 * 60.0

id1 := id2 + temp1



```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

F – Floating-point
– Constant

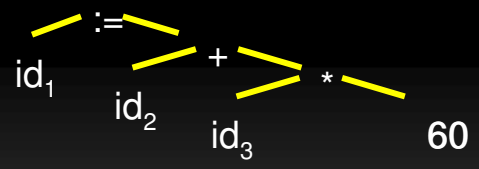
Exemplo Completo

position := initial + rate * 60

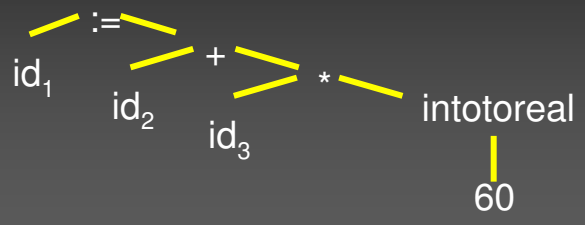
análise léxica

id1 := id2 + id3 * 60

análise sintática



análise semântica



gerador de código intermediário

temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3

otimizador de código

temp1 := id3 * 60.0
id1 := id2 + temp1

gerador de código

MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

Agrupamento de Fases

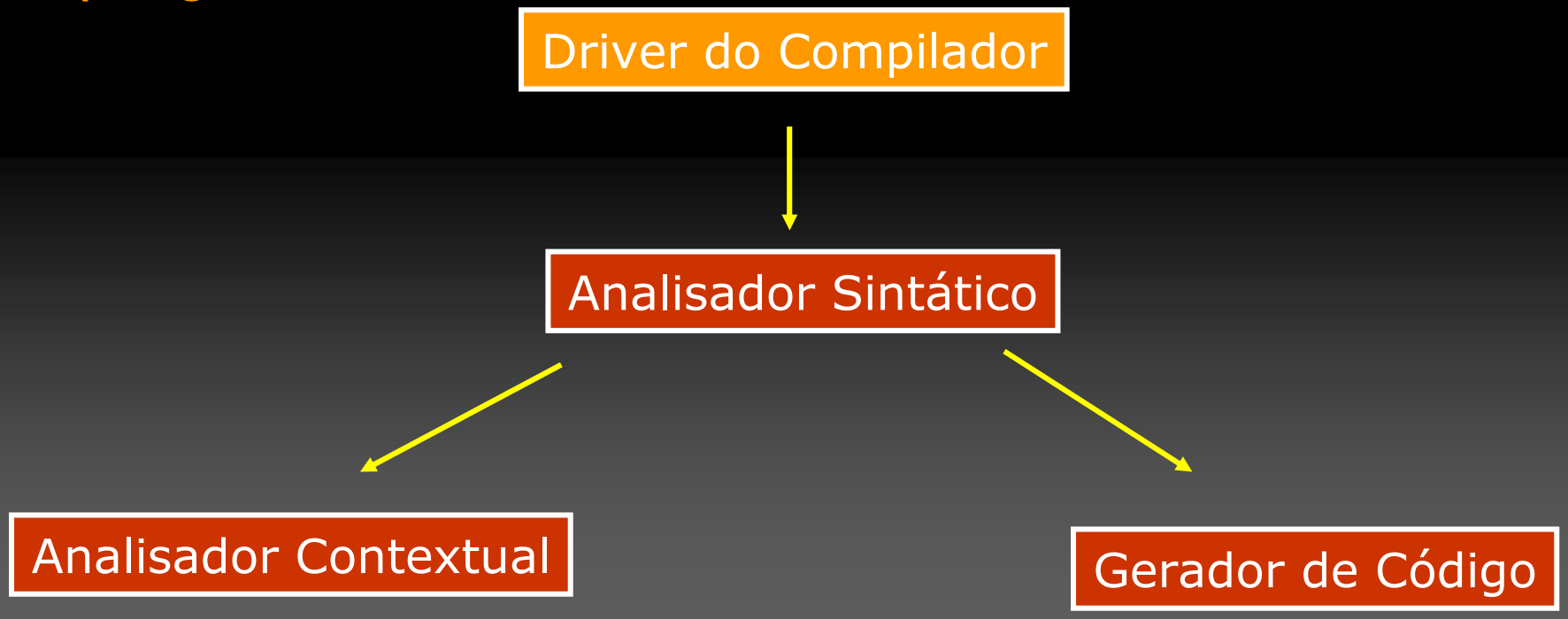
- Passo é a leitura e escrita completa do código-fonte
- Fase ≠ Passo

- Interfaces
  - Vanguarda
  - Retaguarda

- Passo
  - Compilação em Passo Único
  - Compilação em Múltiplos Passos

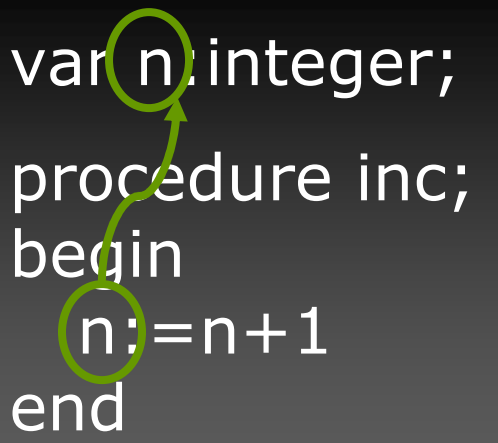
Compilação em Passo Único

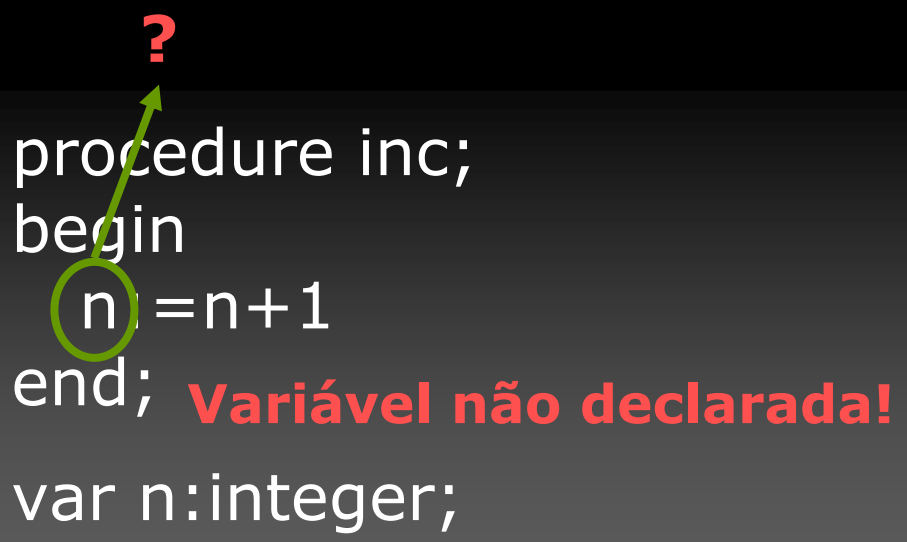
- O Analisador Sintático realiza/chama as atividades de Análise Contextual e Geração de Código à medida que lê e reconhece o programa



Compilação em Passo Único

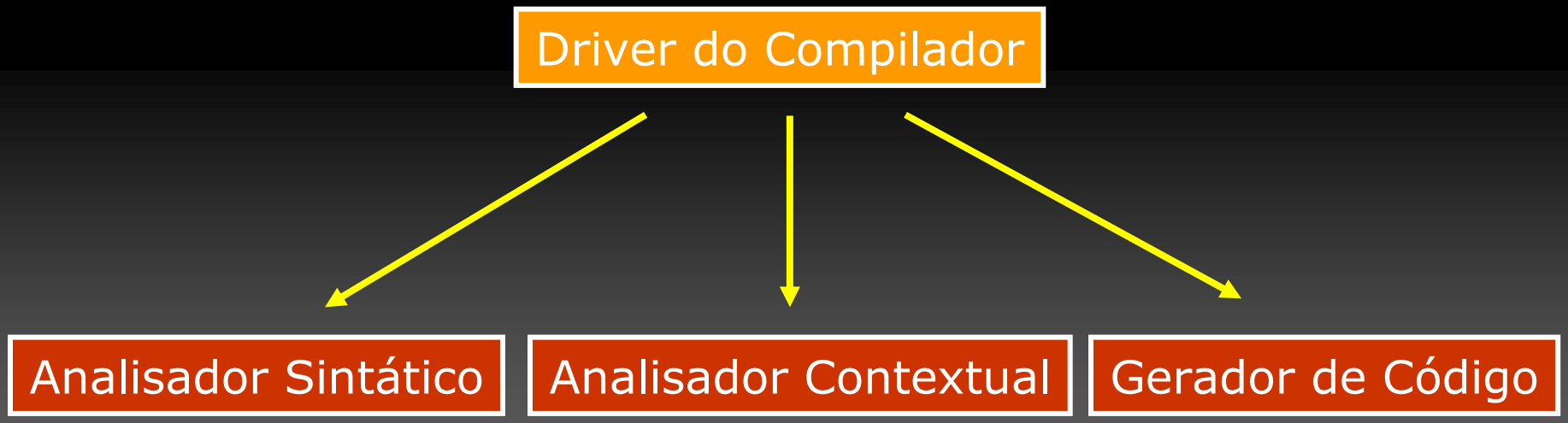
- Pascal foi criada para ser implementada através de um compilado de passo único
  - Todo identificador deve ser declarado antes de ser utilizado!





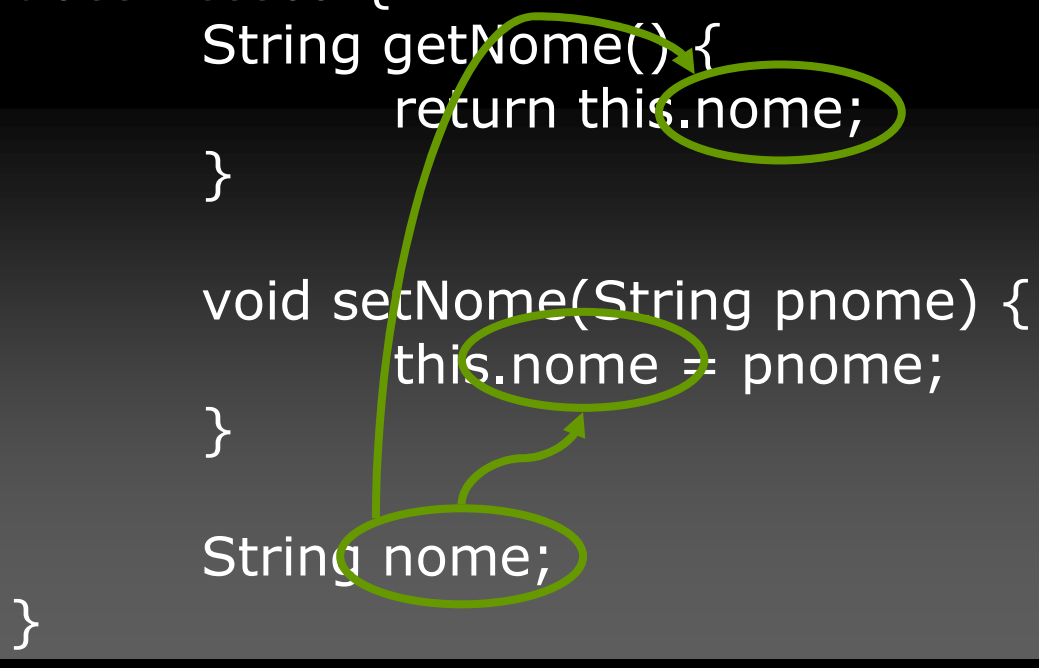
Compilação em Múltiplos Passos

- Módulo principal (driver) chama cada um dos passos responsáveis pela análise sintática, análise contextual e geração de código.



Compilação em Múltiplos Passos

- Java permite que identificadores sejam usados antes de serem declarados
  - Exige um compilador com pelo menos 2 passos



Pontos Relevantes no Projeto de Um Compilador

- Considerações
  - I/O (*input/output*) consomem tempo
  - ↓Fases = ↑Memória
- Compilador de um passo tem como vantagem
  - Velocidade
  - Espaço
- Compilador de múltiplos passos tem como vantagem
  - Modularidade
  - Flexibilidade
  - Transformações/otimizações de programas
- Características da linguagem fonte podem inviabilizar o uso de um passo único
  - Se permite o uso de uma função antes da sua definição; GOTO.

Pontos Relevantes no Projeto de Um Compilador

	Passo Único	Múltiplos Passos
Velocidade	**melhor**	**pior**
Memória	**programas maiores**	**(potencialmente) programas menores**
Modularidade	**pior**	**melhor**
Flexibilidade	**pior**	**melhor**
Otimização "Global"	**impossível**	**possível**
Linguagem	**Compiladores de passo único não são possíveis para todas as linguagens de programação**	

Programas Auxiliares do Processo de Compilação

- Preprocessador
  - Processamento de macros
  - Inclusão de arquivos
  - Compilação condicional
  - Extensão de linguagens
- Montador (a*ssembler*)
  - Abstração da arquitetura da máquina-destino. É na realidade um tradutor/compilador simples, de dois passos, que gera código relocável.
- Carregador (*loader*) e linkeditores (*linker*)
  - Ajustam o código relocável, resolvem referências externas.

Resumo

- Linguagem de Programação
- Tradutor
- Fases
  - Análise Léxica
  - Análise Sintática
  - Análise Semântica
  - Geração de Código Intermediário
  - Otimização de Código
  - Geração de Código
- Agrupamento de Fases
- Pontos Relevantes no Projeto de Um Compilador
- Programas Auxiliares

Pra Finalizar...

- Onde encontro material para estudar?
  - Livro “UFRGS” (Capítulo 1)
  - Livro do “dragão” (Capítulos 1 e 2)
  - WEB!

Só mais uma coisa...

- Tarefa de Casa
  1. Fazer um resumo sobre os principais tipos de linguagem de programação quanto a geração (1ª, 2ª , 3ª , 4ª , 5ª, ...) citando características e exemplos
  2. Fazer um resumo sobre os principais paradigmas de programação (Imperativo, Orientado a Objetos, Lógico e Funcional) citando características e exemplos
- Entrega na próxima aula

Até a próxima semana!