Compiladores

- Prof. Raimundo Santos Moura
- Notas de aula do Professor André Santos (http://www.cin.ufpe.br/~if688)
- Requisitos:
  - Conhecimento de programação C, C++, Java ou Python

1

Construção de Compiladores

- Livro-Texto:
  - AHO, Alfred V.; ULLMAN, Jeffrey D.; SETHI, R. Compiladores: princípios, técnicas e ferramentas. Rio de Janeiro: LTC, 1995.

2

Motivação

- Conhecimento das estruturas e algoritmos usados na implementação de linguagens: noções importantes sobre uso de memória, eficiência, etc.
- Aplicabilidade frequente na solução de problemas que exigem alguma forma de tradução entre linguagens ou notações.
- Implementação de linguagens para um domínio específico.
- Geradores e analisadores de código.

3

Motivação

- A disciplina de compiladores faz uso de um grande número de conceitos estudados em outras disciplinas do curso: linguagens de programação, algorítmos, linguagens formais, arquitetura, engenharia de software.

4

Contexto Histórico

- Demanda por linguagens de mais alto nível que linguagem de máquina e assembler.
- Nos anos 1950, compiladores eram programas notadamente difíceis de se escrever.
- Avanço teórico e de técnicas e ferramentas de implementação tornaram possível implementar compiladores muito mais facilmente.

5

Classificações: Gerações

① Linguagens de máquina
② Linguagens de montagem (*Assembly languages*)
③ Fortran, Cobol, Lisp, C, C++, C#, Java, Python
④ SQL, Postscript (*Domain Specific Languages*)

6

Compilador

- Programa que lê um programa escrito em uma linguagem (fonte) e o traduz para uma outra linguagem (destino), reportando erros quando eles ocorrem.
- Compilador x Tradutor

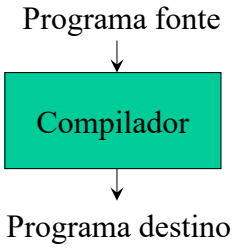
10

Linguagens

- Linguagem fonte: C, Pascal, Java, Fortran, etc.
- Linguagem destino: linguagem de máquina (assembler) de um processador, de uma máquina virtual (e.g. Java ou .NET), ou qualquer outra linguagem (e.g. C).

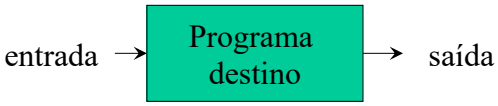
11

Processadores de Linguagens: Compilador



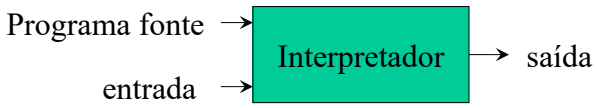
12

Execução de um programa



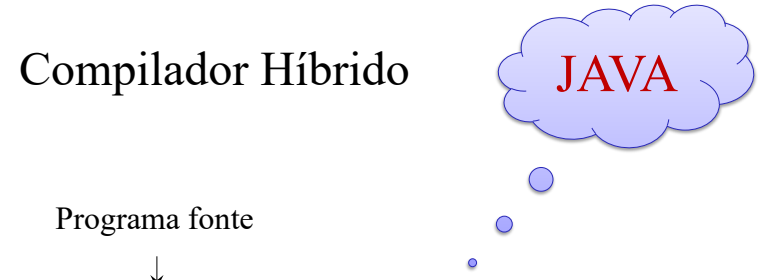
13

Processadores de Linguagens: Interpretador



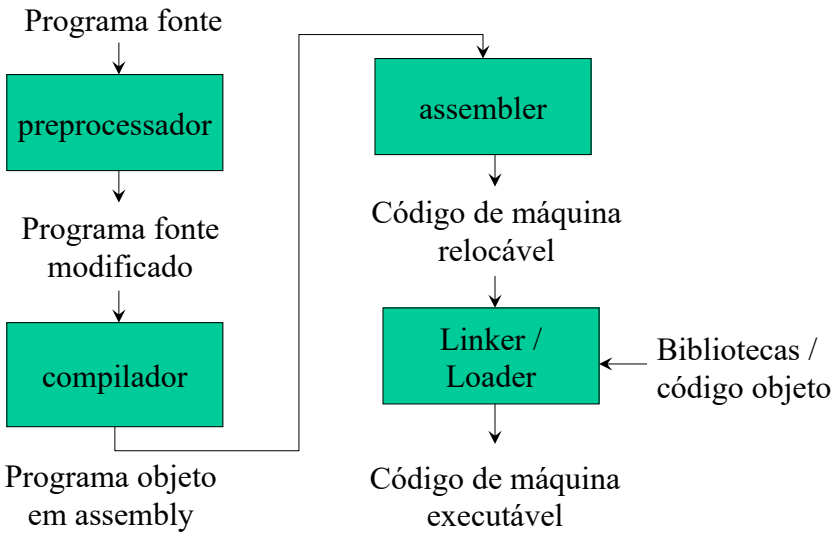
14

Compilador Híbrido



15

Sistema de processamento de uma linguagem



16

Programas auxiliares do processo de compilação

- Preprocessadores: processam macros, incluem arquivos, suportam compilação condicional e extensão de linguagens.
- Assemblers: servem como uma pequena abstração da arquitetura da máquina-destino. São na realidade um tradutor /compilador simples, de dois passos, que gera código relocável.

17

Programas auxiliares do processo de compilação (cont.)

- Carregadores (*loaders*) e linkeditores (*linkers*)
  - Ajustam o código relocável, resolvem referências externas.

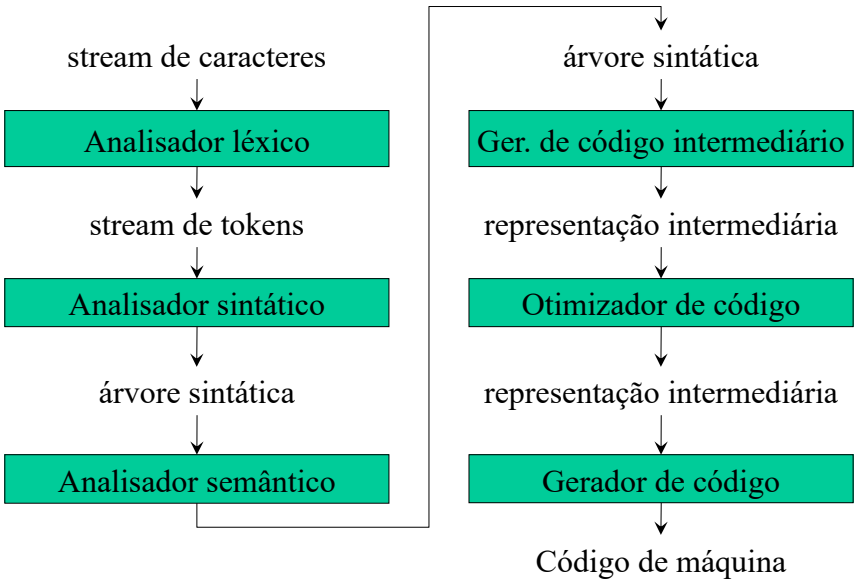
18

Compilação: Análise e Síntese

- Análise: quebra o código fonte em suas partes, e cria uma representação intermediária do programa.
  Verifica erros e constrói tabela de símbolos.
- Síntese: Constroi o programa-destino a partir da representação intermediária.

19

Fases de um compilador



20

Análise: *front-end* do compilador
(até geração de código intermediário)

Síntese: *back-end* do compilador

21

Análise do programa fonte

- Análise léxica – lê a seqüencia de caracteres e a organiza como *tokens* – sequencias de caracteres com algum significado
- Análise sintática – agrupa caracteres ou *tokens* em uma estrutura hierárquica com algum significado
- Análise semântica – verifica se os componentes de um programa se encaixam de forma a ter um significado adequado.

22

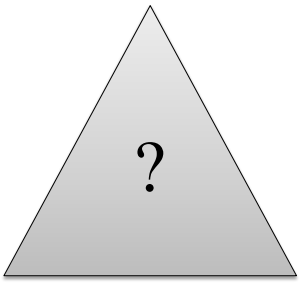
Programas baseados em análise

- Editores de programa dirigidos à sintaxe (comuns nos IDEs, como no Eclipse e Visual Studio)
- Pretty-printers
- Interpretadores

23

Exemplo do processo de compilação

- Livro-texto, exemplos 1.1, 1.2 e 1.3



24

Análise léxica ou *Scanning*

- Lê os caracteres de entrada e agrupa-os em sequências chamadas *lexemas*.
- Para cada lexema o analisador léxico produz como saída um *token* da forma

  <*nome do token*, *valor do atributo*>

  que é passado para a fase seguinte.

25

Exemplo

- position = initial + rate * 60

<**identificador**, 1>
<=>
<**identificador**, 2>
<+>
<**identificador**, 3>
<*>
<**number**, 60>

26

Tabela de Símbolos

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

27

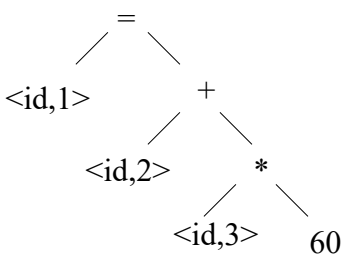
Análise sintática ou *parsing*

- A partir dos *tokens* cria uma estrutura em árvore (*árvore sintática*) que representa a estrutura gramatical do programa.

28

Árvore Sintática

- position = initial + rate * 60



29

Análise semântica

- Verifica o programa em relação a possíveis erros semânticos e guarda informações adicionais

30

Exemplo: verificação de tipos

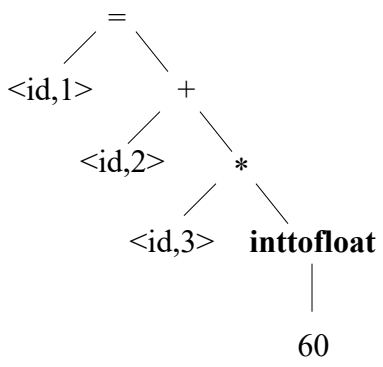
A expressão

```
x = x + 3.0
```

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

31

Análise Semântica



32

Código intermediário

- Idealmente deve ser fácil de produzir e também de traduzir para a linguagem-destino.
- Na prática, gera-se código para uma máquina abstrata.
- Exemplo: ***Three-address-code***: usa três operandos por instrução, cada um como se fosse um registrador.

33

Código intermediário - exemplo

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

34

Otimização de código

- Realiza transformações no código visando melhorar sua performance em aspectos de tempo de execução, uso de memória, tamanho do código executável etc.

35

Otimização de código - exemplo

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

```
t2 = id3 * 60.0
id1 = id2 + t2
```

36

Geração de código

- Realiza a alocação de registradores (se necessária) e a tradução do código intermediário para a linguagem-destino.

37

Geração de código

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

38

Tabela de Símbolos

- Estrutura de dados usada para guardar identificadores e informações sobre eles:
  - alocação de memória,
  - tipo do identificador,
  - escopo (onde é válido no programa)
  - se for um procedimento ou função: número e tipo dos argumentos, forma de passagem dos parâmetros e tipo do resultado.

39

Tabela de Símbolos

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

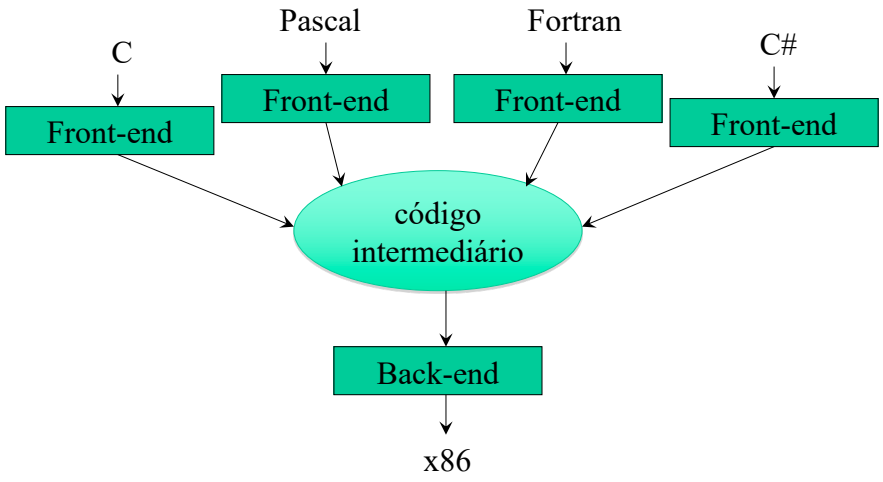
40

Organizando fases em passos

- Fases: visão lógica de um compilador
- Em uma implementação as fases podem ser agrupadas em um ou mais *passos*
- Passos: número de vezes em que se passa pelo mesmo código.
- Por exemplo, em um assembler são necessários pelo menos dois passos.

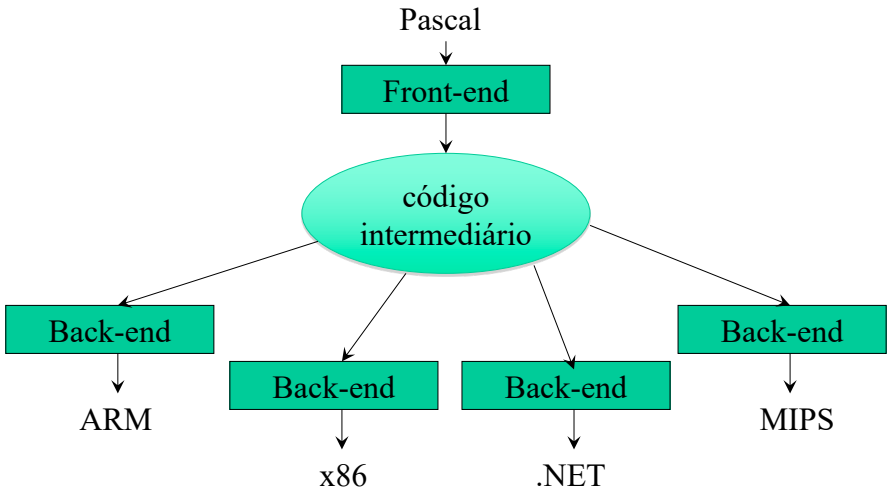
41

Separação entre front-end e back-end para criação de múltiplos compiladores



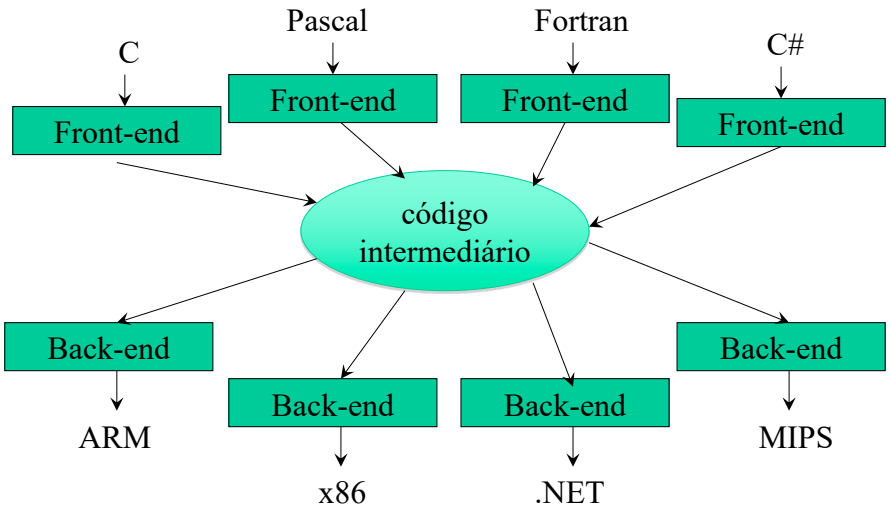
42

Separação entre front-end e back-end para criação de múltiplos compiladores



43

Separação entre front-end e back-end para criação de múltiplos compiladores



44

Ferramentas auxiliares para a construção de compiladores

- *Scanner generators*, baseados em expressões regulares. Exemplo: lex
- *Parser generators*, baseados em gramáticas livres de contexto. Exemplo: yacc
- *Mecanismos de tradução dirigida por sintaxe*, geram rotinas para navegar na *parse tree* e gerar código intermediário
- *Geradores de gerador de código* (*template matching*)
- Mecanismos de análise de fluxo de dados (essencial para a otimização de código)
- *Toolkits de construção de compiladores*

45