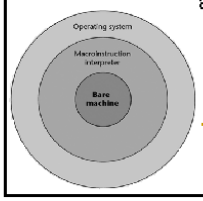
Modelos de linguagens de Programação

-- Aula 02 –

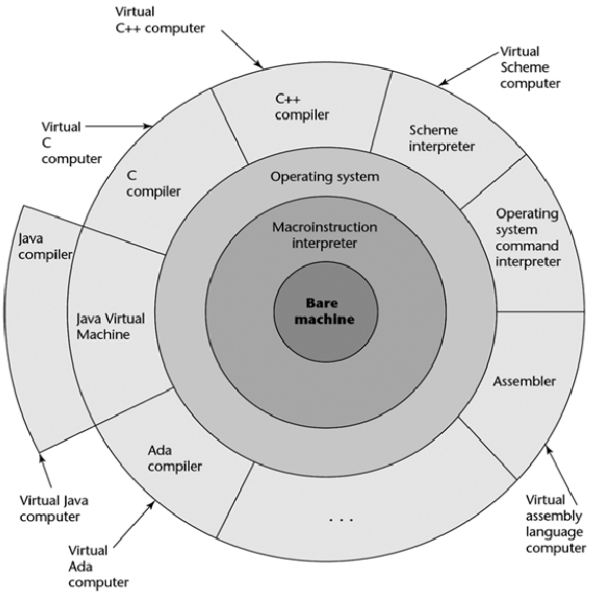
Detalhamento sobre compilação e interpretação

Máquinas reais

- Aceitam um conjunto de instruções executáveis (linguagem de máquina)
- Uma LP de alto nível poderia ser diretamente executável nelas, mas:
  - A complexidade e o custo de implementação desta máquina não se justifica (exemplo do modelo Japonês)
  - Possuiria flexibilidade reduzida
  - O modelo atual possui diversas camadas:
    - núcleo HW → interpretador de macro-instruções → SO → LPs de alto nível
    - há gerenciamento de recursos do sistema em nível mais alto que a linguagem de máquina → Sistemas operacionais (E/S, gerenciamento de arquivos, etc.)

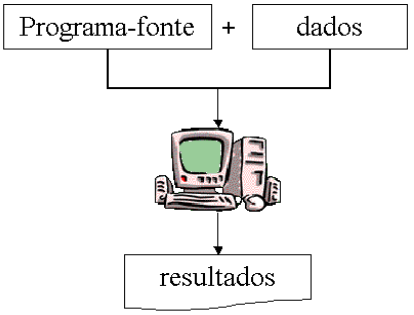


Modelo de camadas



Execução de programas

Como se dá a execução de programas nesse contexto?

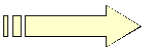


- A notação usada no programa pode ser incompatível com o conjunto de instruções executáveis
- Solução: compilação, simulação (MV)

Computador e linguagens

- Um computador pode ser representado por:
  - uma **máquina virtual**, capaz de executar operações mais abstratas (representadas através de uma linguagem de programação)
  - uma **máquina real**, capaz de executar um determinado conjunto de operações concretas (expressas em linguagem de máquina)

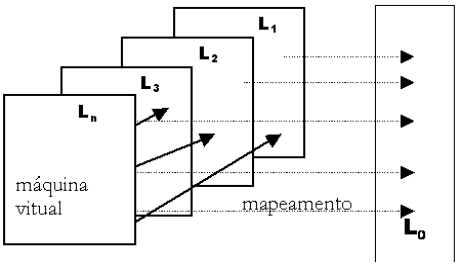
L_1: máquina Assembly
L_2: máquina C
L_3: máquina Pascal
L_4: máquina virtual Java



L_0: máquina real

Linguagens e Máquinas

- Cada máquina representa um conjunto integrado de estruturas de dados e algoritmos
- Cada máquina é capaz de armazenar e executar programas em uma linguagem de programação L1, L2, L3,...Ln.



Máquina reais e máquinas virtuais

- Máquina real: conjunto de hardware e sistema operacional capaz de executar um conjunto próprio de instruções (plataforma de execução)
- O elo de ligação entre a máquina abstrata e a máquina real é o processador da linguagem
- Processador da linguagem: programa que traduz as ações especificadas pelo programador usando a notação própria da linguagem em uma forma executável em um computador
- Máquina abstrata (virtual): combinação de um computador e um processador de linguagem

7

Projeto de Linguagens de Programação

- **Questões de projeto**:
  - Qual a finalidade da LP? O uso é geral ou específico?
  - Qual o domínio de aplicação?
  - Qual é a sua principal diferença em relação a outras LP existentes?
- **Questões de implementação**:
  - Qual é seu paradigma principal?
  - Quais são suas raízes? É nova ou estende uma existente?
  - Como será feita a tradução da linguagem?
  - A plataforma de execução é homogênea ou heterogênea?

8

Formas de execução

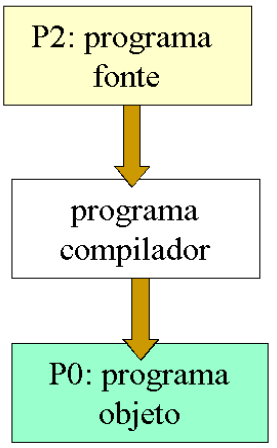
- Compilação
- Interpretação

Compilação

- **Compilador**: programa em linguagem de máquina
- **Programa alvo**: chamado de *código objeto* se seu formato é entendido pelo SO

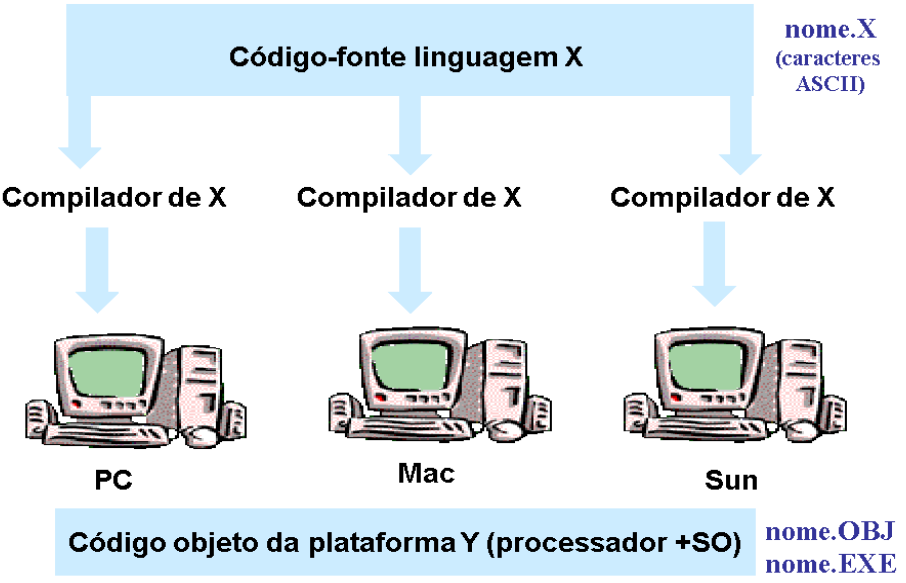
Programa Compilador

- Compiladores aceitam como entrada um programa escrito em linguagem de programação e produzem um programa equivalente em outro código
- Programa objeto: linguagem de máquina, linguagem Assembly ou linguagem intermediária
- Sendo P2 o programa fonte e P0 o programa objeto, P2 e P0 devem ser funcionalmente equivalentes



11

Compiladores x plataformas



Compiladores: características

- Compiladores se baseiam na noção de código executável permanente, diretamente executável na plataforma de destino
- **Características**:
  - **Eficiência**: programas mais rápidos
  - **Confiabilidade**: mais espaço e tempo para verificações do código fonte

O que dificulta a compilação?

- Vinculação tardia de:
  - nomes a objetos (regras de escopo)
  - tipos a objetos/nomes (regras de tipo)
  - programas a código (classes dinâmicas em Java, novas funções criadas durante a execução em Scheme)

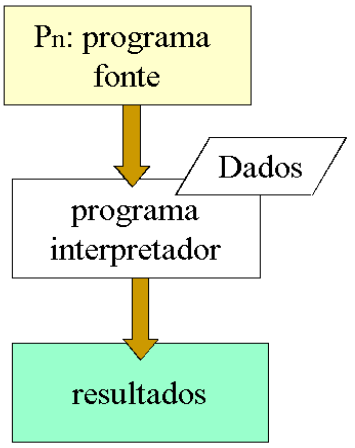
Interpretação

- Interpretador =*máquina virtual*
  - capaz de "executar" código de alto nível
  - cada instrução é traduzida para linguagem de máquina imediatamente antes de ser executada

15

Interpretador

- Realiza o processo de execução a partir de um programa fonte escrito em uma linguagem de programação ou *código intermediário* e mais o conjunto de dados de entrada exigidos pelo programa
- Pode produzir código executável, mas não produz programa objeto persistente
- Exemplos: Basic, LISP, Smalltalk



16

Por que interpretadores podem ser interessantes?

- Se baseiam na noção de código intermediário, não diretamente executável na plataforma de destino
- **Características**:
  - **Simplicidade**: programas menores e menos complexos que compiladores, fácil *startup* a partir do fonte
  - **Portabilidade**: o mesmo código pode ser aceito como entrada em qualquer plataforma que possua um interpretador
  - **Depuração**: pelo fato de terem acesso direto ao código do programa (e não ao código objeto) podem fornecer ao desenvolvedor mensagens de erro mais acuradas
  - **Flexibilidade**: vinculação tardia de objetos, nomes e dados, regras menos estritas de tipos

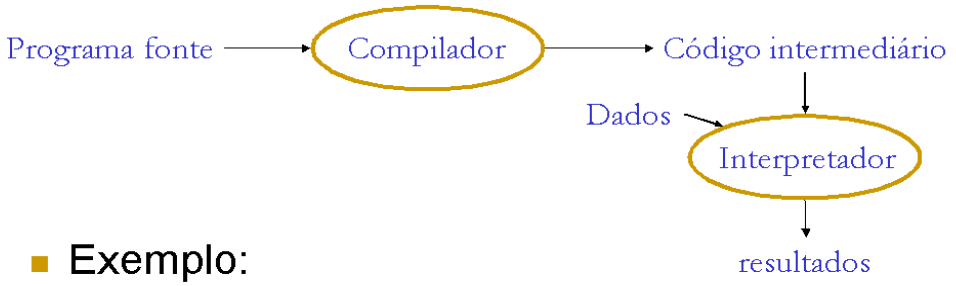
17

Compilação vs Interpretação

- **Compilação pura**
  - geração de código executável
  - depende da plataforma de execução
  - tradução lenta X execução rápida
  - transformação mais apurada do código
  - código intermediário não guarda muita semelhança com o código fonte
  - menor flexibilidade, maior eficiência
- **Interpretação pura**
  - não gera código executável
  - é independente de plataforma
  - a execução é lenta
  - a transformação é puramente mecânica, sem grandes avaliações
  - oferece mais flexibilidade e melhor diagnóstico (alteração dinâmica e instantânea)

18

Tradução híbrida (compilação com interpretação)



- Exemplo:
  - Java
    - código intermediário = bytecodes
    - interpretador = JVM (Java Virtual Machine)

19

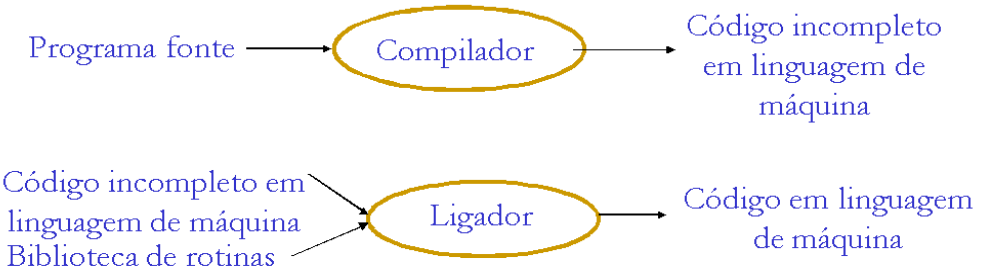
Tradução híbrida: características

- Geração de código intermediário
- Independente de plataforma de execução
- Tradução menos lenta X execução não muito rápida
- LP dita compilada se este passo é o mais significativo
- LP é dita interpretada se o passo de compilação é simples, sem grandes transformações do código fonte

21

Estratégias práticas diversas

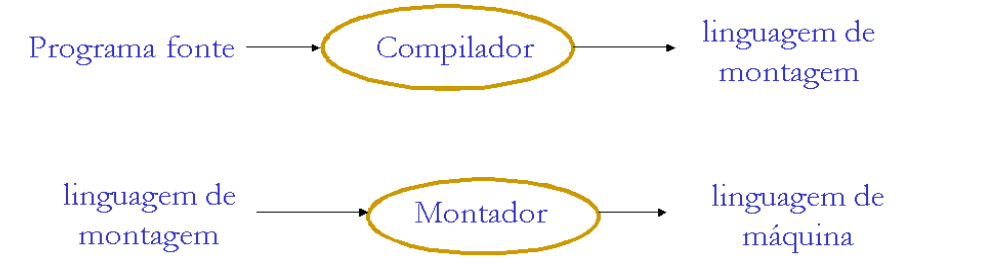
- Uso de Ligador (linker)
  - Liga o código com bibliotecas de sub-rotinas
    - funções matemáticas, I/O, etc.
    - "extensões" do conjunto de instruções do HW



22

Estratégias práticas diversas

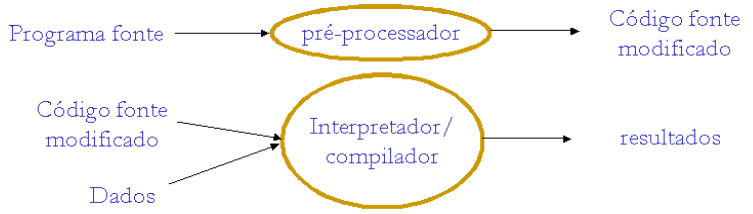
- Uso de Linguagem de Montagem
  - Facilita depuração
  - Isola o compilador do HW e/ou do SO
  - Montador é mais simples que o compilador e pode ser compartilhado por vários compiladores



23

Estratégias práticas diversas

- Uso de pré-processadores
  - Programa que faz conversões entre linguagens de programação de alto nível similares ou para formas padronizadas de uma mesma linguagem de programação
  - Possibilita a utilização de extensões da linguagem (macros ou mesmo novas construções sintáticas) utilizando os processadores da linguagem original
  - Realiza compilação condicional (remove trechos de código)



24

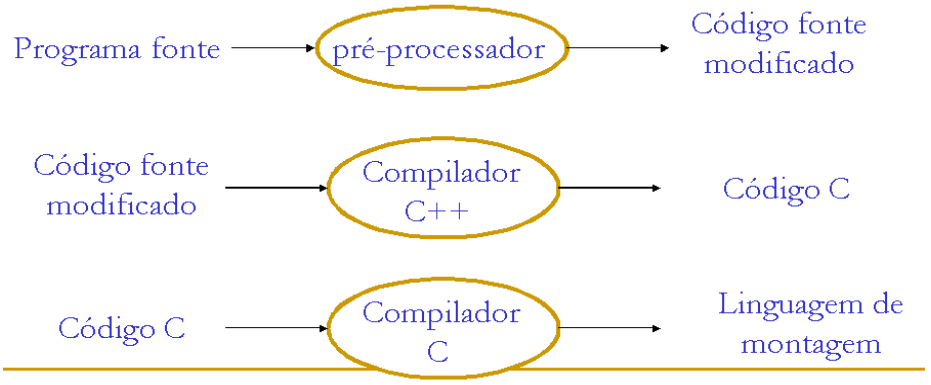
Preprocessador: exemplo em C#

```
#define Dutch

using System;

public class Preprocessor {
  public static void Main() {
    #if Dutch
      Console.WriteLine("Hallo Wereld");
    #else
      Console.WriteLine("Hello World");
    #endif
  }
}
```

25

Estratégias práticas diversas

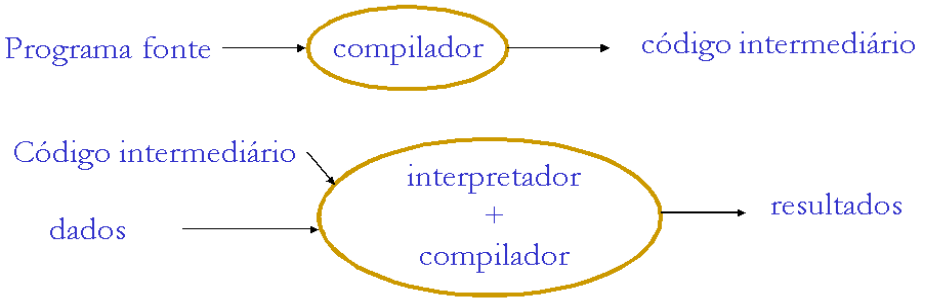
- Uso de linguagens intermediárias de alto-nível
  - Exemplo: compilador AT&T para C++



26

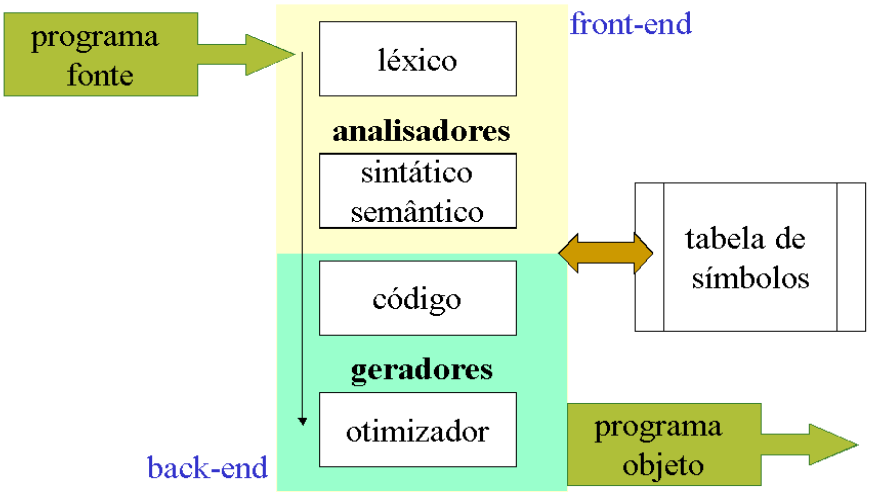
Estratégias práticas diversas

- Compilação e Interpretação juntas



- Exemplo: Java (JDK 5.0) – JIT
  - código intermediário = bytecodes
  - interpretador = JVM (Java Virtual Machine)
  - compilação adaptativa durante execução

Componentes de compiladores



28

Compiladores: fluxo de execução

29

Analisador léxico

- Analisador léxico (*scanner/tokenizador*):
  - Tem por objetivo separar os símbolos individuais da linguagem (*tokens*): identificadores, palavras-chave, operadores, etc.
  - Popula a tabela de símbolos

Exemplo

```
Procedure verifica(i,j,n,m: integer; var maior: integer);
    var k: longint;
    begin
      k:=i+(j-1)*n-1;
      if k < m then
         maior := m;
      else
         maior := k;
    end;
```

- Análise léxica retornaria os *tokens:*
  - Procedure - procedimento
  - verifica - identificador
  - (- parêntese
  - i - identificador
  - j - identificador
  - : :

- Observação:
  - Exemplo de erro léxico:
    - Procedure ***89verifica*** (...

    [Não há tokens iniciando com números seguidos de caracteres]

Tabela de símbolos

- Tem por objetivo manter um mapeamento entre os identificadores usados no programa e suas propriedades
- Exemplo: `x := a * (b + c) ;`

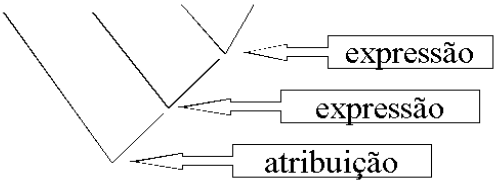
(1) := (2) * ((3) + (4))

nome	tipo	escopo
x	real	...
a	real	...
b	int	...
c	int	...

32

Analisador sintático

- Analisador sintático (*parser*):
  - tem por objetivo descobrir a estrutura de cada construção do programa: declaração, expressão, atribuição, if, while, etc.
  - aplica a gramática da linguagem para formar a árvore de análise a partir da seqüência de átomos (*tokens*)
  - exemplo: `x := a * (b + c) ;`

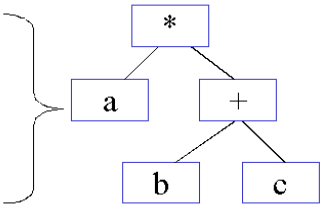


33

Analisador semântico

- Análise semântica:
  - Usa a árvore de análise para compreender as ações envolvidas e gerar uma representação interna do programa (refinar a tabela de símbolos)
  - Faz avaliação de símbolos
  - Pode executar uma ação correspondente: gerar código
- Exemplo:

```
x := a * (b + c);
```




```
(+, b, c, t1)
(*, a, t1, t2)
(:=, t2, x)
```


Geradores

- **Gerador de código**:
  tem por objetivo produzir código funcionalmente equivalente de cada construção do programa
  - Exemplo:

```
x := a * (b + c);
```

```
(+, b, c, t1)
(*, a, t1, t2)
(:=, t2, x)
```

```
Load b
Add c
Sto t1
.....
```

Geradores

- **Otimizador de código**:
  tem por objetivo aplicar um conjunto de técnicas sobre o código objeto para torná-lo mais eficiente
  - Exemplos:
    - Eliminação de expressões redundantes
    - Substituição de funções "in-line"
    - Reorganização de comandos
    - etc.

Erros de Compilação e execução

- Erros em programas podem ser classificados de acordo com o momento (ou a possibilidade) da detecção durante a compilação:
  - erro léxico (detectado pelo scanner/tokenizador)
  - erro sintático (detectado pelo parser)
  - erro semântico estático (detectado pela análise semântica)
  - erro semântico dinâmico (detectado no código gerado)
  - erro que o compilador não consegue detectar nem consegue gerar código que o detecte

Bibliografia

- Sebesta. Robert W. Conceitos de Linguagens de Programação. Bookman: Porto Alegre, 2000.
- Definição de compiladores na Wikipedia: http://en.wikipedia.org/wiki/Compiler