Modelos de Linguagens de Programação

-- Aula 02 --

Tópicos

- Modelos e paradigmas de linguagens: revisão e detalhamento
- Propriedades e características desejáveis em linguagens de programação
- Compilação e interpretação (visão geral)
- Questões
- Bibliografia

Modelos/paradigmas de computação

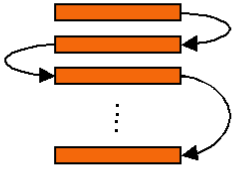
- IMPERATIVO
  - COMO executar
  - próximas do hardware
  - + desempenho
  - Von Neumann (arquitetura)
- DECLARATIVO
  - O QUE executar
  - próximas do pensamento natural
  - eficiência dependente dos compiladores

Modelos/paradigmas de programação

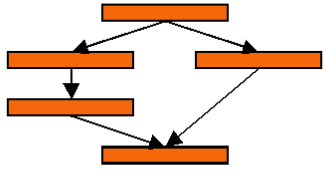
- DECLARATIVO
  - Funcional:
    - ênfase em valores computados por funções
  - Lógico:
    - ênfase em axiomas lógicos
- IMPERATIVO
  - Estruturado:
    - sequências de comandos realizam transformações sobre dados
    - Orientado a procedimentos
  - Orientado a Objetos:
    - organização dos dados domina
    - comandos explícitos realizam transformações sobre dados

Modelos de execução

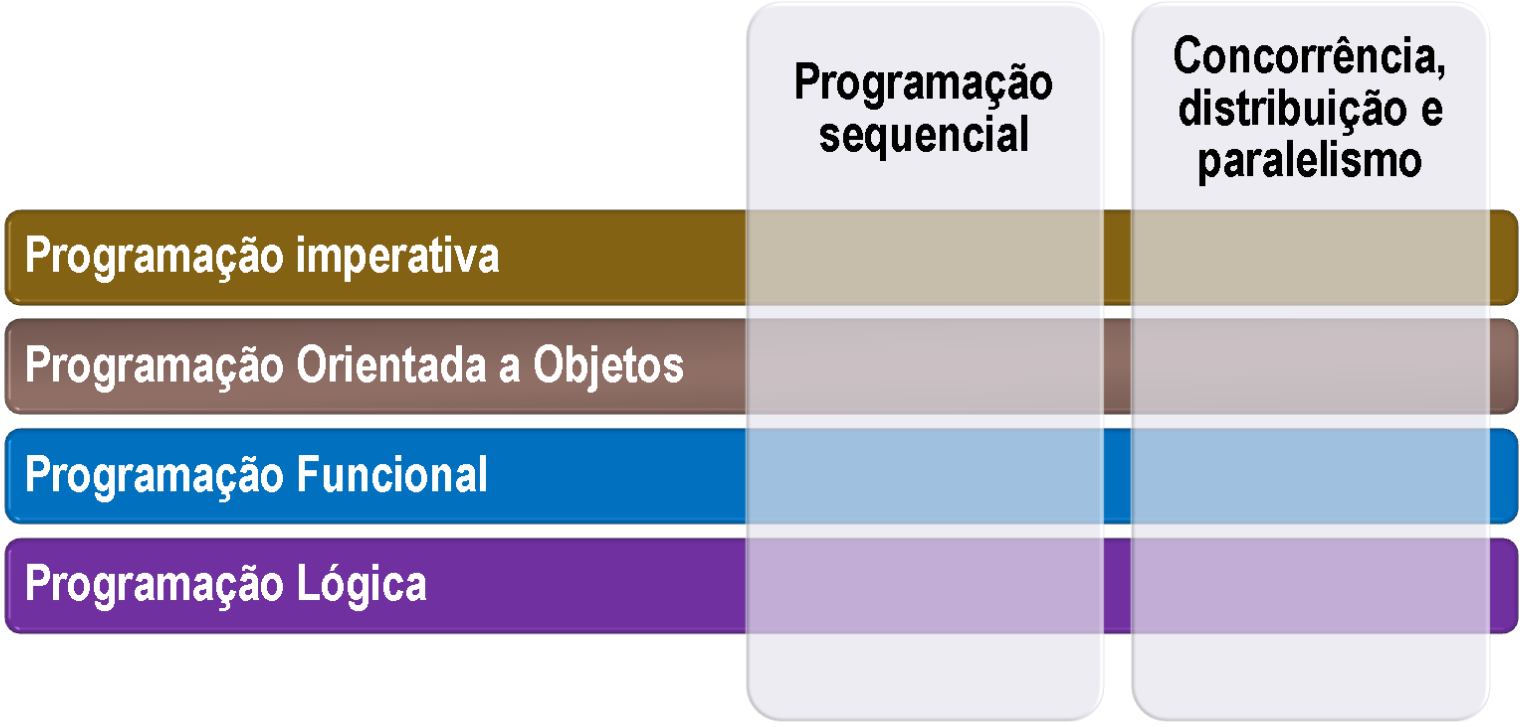
- SEQUENCIAL
  - computação realizada após o término da anterior
  - controle de fluxo de execução é interno ao programa (e.g., sequência, seleção, iteração, invocação)



- CONCORRENTE
  - múltiplas computações podem ser executadas simultaneamente
  - computações paralelas (múltiplos processadores compartilham memória)
  - computações distribuídas (múltiplos computadores conectados por uma rede de comunicação)



Modelos: relação



Tópicos

- Modelos e paradigmas de linguagens: revisão e detalhamento
- Propriedades e características desejáveis em linguagens de programação
- Compilação e interpretação (visão geral)
- Questões
- Bibliografia

Como escolher uma linguagem?

- Uso de critérios e características que apóiem uma avaliação correta e baseada em argumentos técnicos
- Lembrar que o tempo do profissional é fundamental (crise do software)

Razões comuns (mas erradas)

- Fanatismo
- Preconceito
- Inércia
- Medo da mudança
- Modismo
- Pressões comerciais
- Conformismo

Aspectos técnicos e econômicos

- Nível de abstração
- Modelagem de dados e de processos
- Portabilidade
- Confiabilidade
- Eficiência
- Escalabilidade
- Modularidade
- Reusabilidade
- Legibilidade
- Disponibilidade de compiladores e ferramentas
- Familiaridade (custo-benefício do aprendizado)

Critérios e características principais

- Critérios: **(aquilo considerado importante dentro de um contexto)**
  - Legibilidade
  - Redigibilidade
  - Confiabilidade
- Características: **(propriedades distintas que diferenciam quantitativamente ou qualitativamente uma coisa de outra)**
  - Simplicidade
  - Ortogonalidade
  - Portabilidade
  - Confiabilidade
  - Expressividade
  - Reusabilidade
  - Estruturas de controle e Tipos de dados e estruturas
  - (projeto de) Sintaxe
  - (suporte a) Abstração

Critério: legibilidade (e redigibilidade)

- Facilidade de ler e escrever programas: legibility, writability
- Influi no desenvolvimento, depuração e manutenção de programas

Contraexemplos:

- falta de comandos estruturados de controle (e.g., uso de goto)
- vocabulário com diferentes comportamentos:
  - e.g., operador * em linguagem C
    *p = (*p) * q; (multiplicar e/ou retornar o conteúdo da memória)
- efeitos colaterais (não explícitos):
  - e.g., operador ++ em linguagem C
    x++; (retorna o valor de x, mas também incrementa a variável)

Critério: legibilidade (e redigibilidade)

Contraexemplos (continuação):

- Uso de marcadores de bloco genéricos (e.g., C, C++):

```
while (x > 0){
   if (x % 2  == 0){
      for(y=1;x<10;x++){
         :
      }
   }
}
```

Num programa muito grande, você saberia dizer **qual comando o } está fechando?**

- Uso de marcadores de bloco opcionais (e.g., C, C++):

```
if (x > 1)
   if (x == 2)
      x = 3;
else x = 4;
```

Tem-se a impressão de que o **else** é do **if** mais externo!

Critério: legibilidade (e redigibilidade)

- Fatores que melhoram a legibilidade e a redigibilidade:
  - abstração de dados e de processos
  - modularização de programas
  - comandos de controle estruturados
  - tipos e estruturas de dados (adequados) (boolean x inteiro, registros...)
  - documentação de código
  - convenções léxicas, de sintaxe e de semântica (identificadores, palavras especiais e reservadas...)

OBS: devem ser oferecidos pelas linguagens e utilizados pelo programador

Critério: confiabilidade

- Mecanismos que facilitem a produção de programas que atendam às suas especificações (sejam confiáveis):
  - Tipagem forte:
    - assegura que utilização dos tipos seja compatível com a sua definição
    - evita operações perigosas
  - Tratamento de exceções:
    - especificam como proceder em caso de comportamento não usual
    - possibilitam o diagnóstico e o tratamento em tempo de execução
  - Redução do Aliasing (uso de apelidos/sinônimos):
    - impedir com que dois ou mais métodos ou nomes façam referência à mesma célula de memória
    - contraexemplos: uniões e ponteiros em C

Característica: simplicidade

- A representação de cada conceito deve ser simples de aprender e de dominar:
  - Simplicidade sintática
  - Simplicidade semântica

Característica: simplicidade

- Simplicidade sintática
  - representação feita de modo preciso, sem ambiguidades

Contraexemplos:

```
a++; a=a+1; a+=1; ++a;
```

(incrementos em C, desde que usados de forma independente)

Característica: simplicidade

- Simplicidade semântica
  - a representação deve ter significado independente de contexto

Contraexemplo:

`1 + 2` versus `“alo” + “você”`

(sobrecarga: a semântica pode ser (re)definida pelo programador)

Característica: ortogonalidade

- Possibilidade de combinar entre si, sem restrições, os componentes básicos da LP (sem produzir efeitos anômalos)
- Comportamento previsível no uso dos conceitos
- Quanto menor o número de exceções aos padrões regulares, mais ortogonal é a linguagem
- Ortogonalidade aumenta regularidade (legibilidade, simplicidade), mas também a complexidade

Característica: ortogonalidade

Exemplos:

- permitir a aplicação de operações em qualquer tipo de dado e esperar que o resultado seja coerente
- permitir combinações de estruturas de dados

Contraexemplos:

- não permitir que um array seja usado como parâmetro de um procedimento
- operador de soma não ser permitido para bytes em Java

Ortogonalidade versus Complexidade

- Preocupações:
  - É possível pensar na semântica de todas as possibilidades: úteis, legais?
  - Qual seu custo de implementação?
  - Flexibilidade = exceções!

Característica: estruturas de controle

- De fluxo, repetição, etc.
- Quanto mais variedade (e especificidade), melhor

Exemplos:

- `if...then...else...; switch(...) case...;`
- `do ... while ...; while ...; repeat ... until ...;`
- `for ...; foreach ...;`

Contraexemplo:

- instruções `goto`

Característica: tipos e estruturas

- Variedade de tipos e estruturas de dados
- Existência de facilidades adequadas para a sua definição

Exemplos:

- enumerações,
- tipos booleanos,
- registros
- coleções genéricas

Característica: portabilidade

- Multiplataforma:
  - executar em diferentes plataformas sem a necessidade de maiores adaptações (sem exigências especiais de hardware/software)
  - e.g., aplicação compatível com Unix e Windows
- Longevidade:
  - ciclo de vida útil do software e o do hardware não precisam ser síncronos
  - possiblidade de usar o mesmo software após uma mudança de hardware

Característica: reusabilidade

- Reutilização do código em diversas aplicações
- Aumenta a produtividade

Mecanismos:

- parametrização de subprogramas
- modularização
- algoritmos genéricos (e.g., *templates*)
- bibliotecas (disponibilidade e facilidade de criação)
- componentes (disponibilidade e facilidade de criação)
- frameworks que sigam *design patterns*

Característica: expressividade

- Representação clara e simples de dados e procedimentos a serem executados pelo programa
- Poder dos operadores
- Proximidade com o homem
- Linguagens mais modernas:
  - incorporam apenas um conjunto básico de representações de tipos e comandos
  - aumentam o poder de expressividade com bibliotecas de componentes (baseando-se inclusive em polimorfismo, herança)
  - e.g., C++, Java, C# e Ruby

Características versus propriedades

	Legibilidade	Escritabilidade	Confiabilidade
Simplicidade	X	X	X
Ortogonalidade	X	X	X
Estruturas de controle	X	X	X
Tipos e estruturas de dados	X	X	X
Sintaxe	X	X	X
Suporte a abstração		X	X
Expressividade		X	X
Checagem de tipos			X
Restrições de *aliasing*			X
Tratamento de exceções			X

Tópicos

- Modelos e paradigmas de linguagens: revisão e detalhamento
- Propriedades e características desejáveis em linguagens de programação
- Compilação e interpretação (visão geral)
- Questões
- Bibliografia

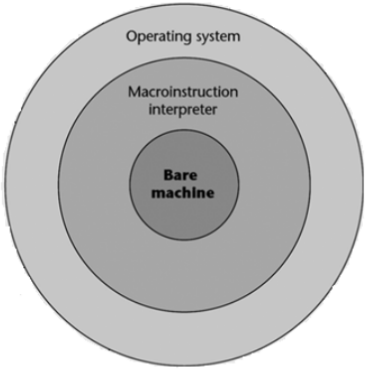
Definição de Linguagem de Programação

- Sintática: uma linguagem de programação é uma notação utilizada pelo programador para especificar ações a serem executadas por um computador

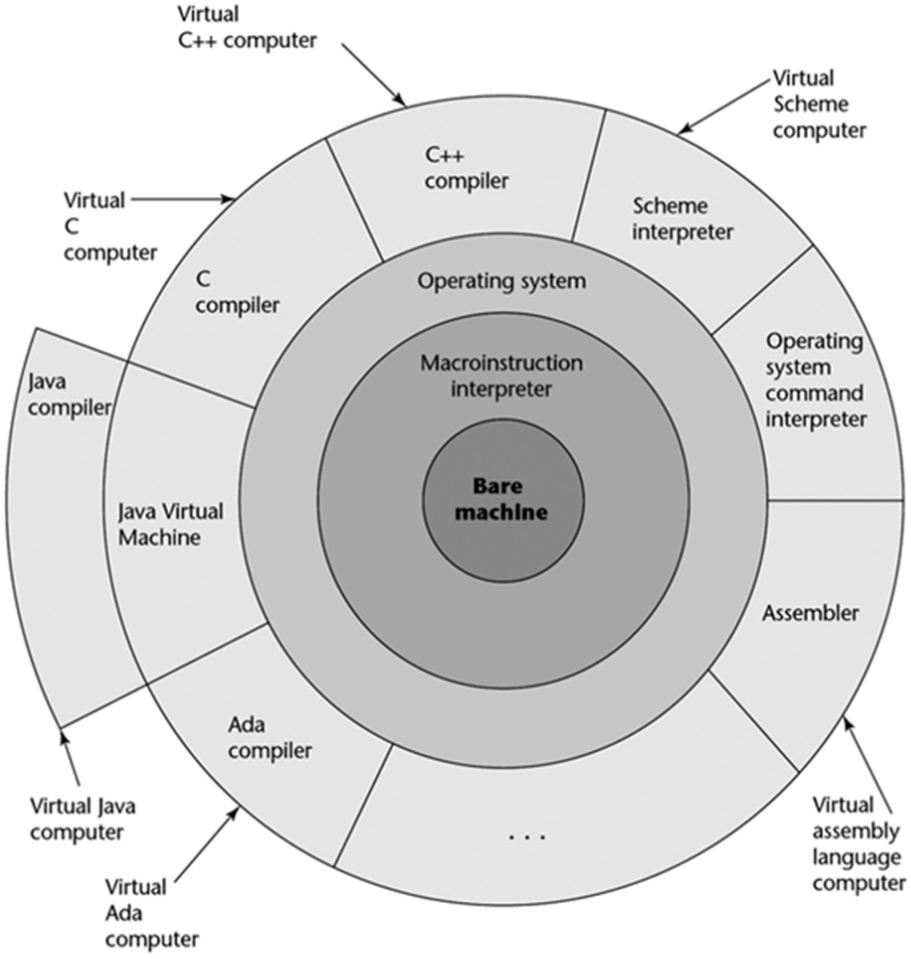
- Semântica: Uma linguagem de programação compreende um conjunto de conceitos que um programador usa para resolver problemas de programação

Máquinas reais

- Aceitam um conjunto de instruções executáveis (linguagem de máquina)
- Uma LP de alto nível poderia ser diretamente executável nelas?
  - Sim, 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.)



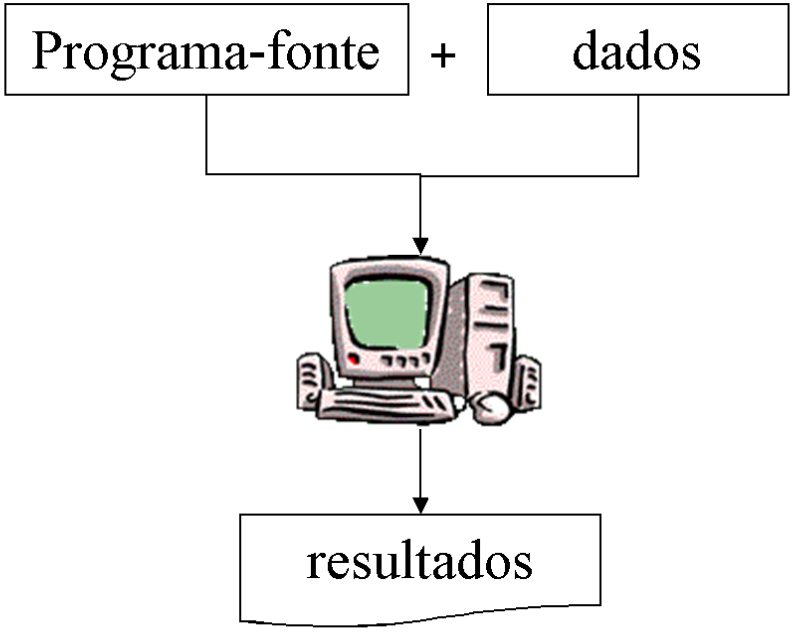
Modelo de camadas



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), interpretação



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?

Compiladores: características

- 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

Compiladores versus plataformas

Código-fonte linguagem X

nome.X
(caracteres ASCII)

Compilador de X

Compilador de X

Compilador de X

PC

Mac

Sun

Código objeto (alvo) da plataforma Y (processador +SO)

nome.OBJ
nome.EXE

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 (não produz programa objeto persistente)
  - Se baseiam na noção de código intermediário, não diretamente executável na plataforma de destino

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
- independente de plataforma
- execução lenta
- transformação puramente mecânica, sem grandes avaliações
- oferece mais flexibilidade e melhor diagnóstico (alteração dinâmica e instantânea)

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