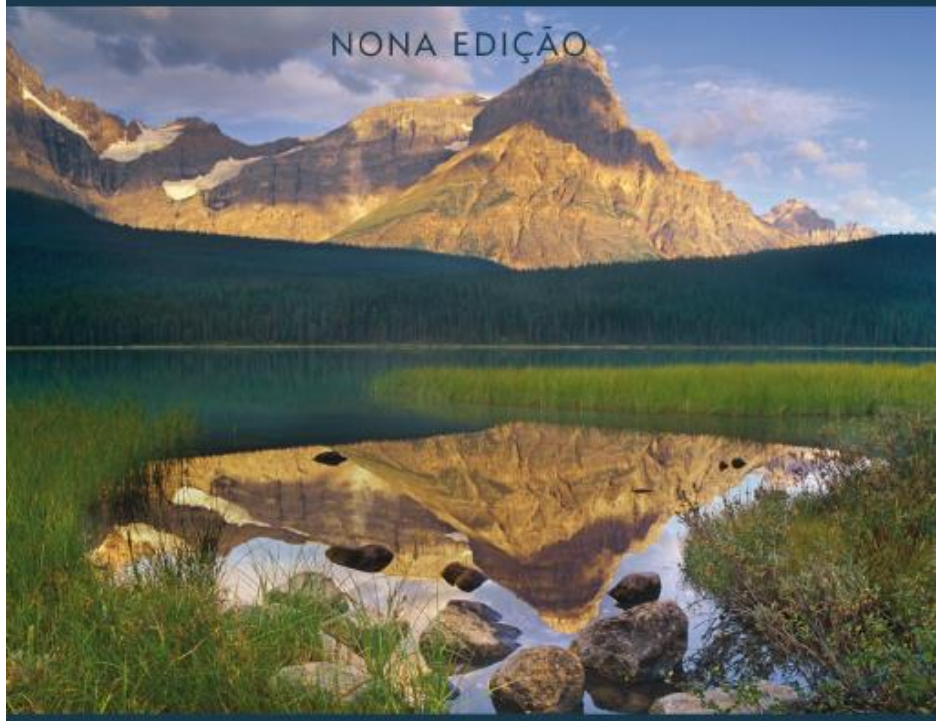
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

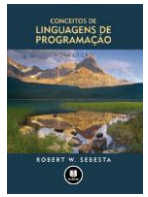


ROBERT W. SEBESTA

bookman

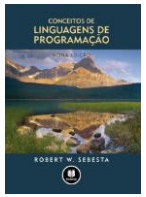
Capítulo 1

Aspectos Preliminares



Motivação

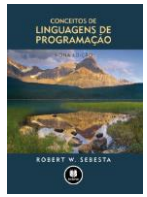
- Linguagens de Programação são usadas por um analista de sistemas em:
    - Projetos disciplinares
    - Estágio e Trabalho (Empresas)
    - Pós-graduação
- Existem dezenas de linguagens de programação
- Linguagens com características diferentes



Motivação

- Quais características de uma linguagem de programação são importantes?
- Como comparar estas características?
- Qual linguagem de programação usar?
- Como comparar diversas linguagens?
- Como estudar novas linguagens?
- Como projetar novas linguagens?
- Qual a melhor linguagem para uma determinada aplicação?

Paradigmas de Programação

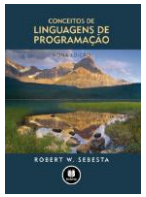


Objetivos

- Análise crítica de paradigmas e linguagens de programação

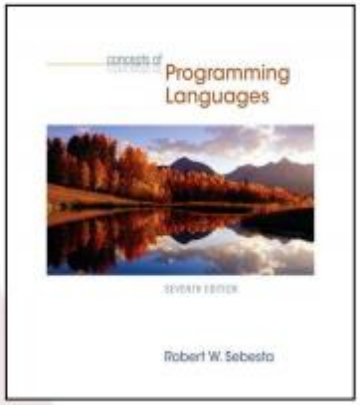
- Estudo dos conceitos gerais de linguagens de programação: valores, tipos, bindings, escopo, memória, entre outros.

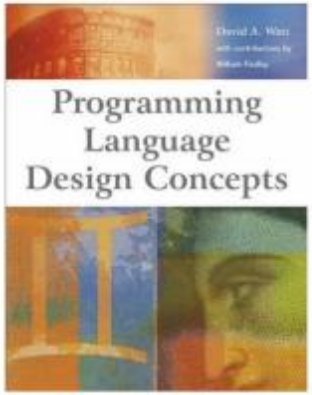
- Visão geral dos paradigmas imperativo, funcional, orientado a objetos, lógico e concorrente

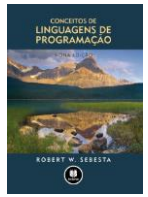


Bibliografia

- Livro-texto
  - Robert W. Sebesta. *Concepts of Programming Languages*. Addison Wesley, seventh edition. 2005.
- Livros complementares
  - *Robert W. Sebesta. Conceitos de Linguagens de Programação. Bookman, quinta edição. 2002.*
  - *David Watt. Programming Language Design Concepts, John Wiley & Sons, 2004.*

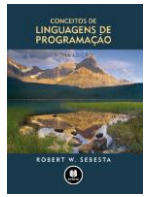






Linguagens de Programação

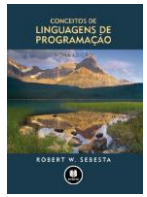
- Definições:
    - Programa
    - Algoritmo
    - Linguagem de programação
    - Paradigma de linguagens



Programa

- Um conjunto completo de comandos estruturados de forma a especificar um algoritmo – (BCS, A Glossary of Computing Terms).

- Algoritmo – conjunto de regras e de procedimentos lógicos, perfeitamente definidos, que levam à solução de um problema em um número finito de etapas

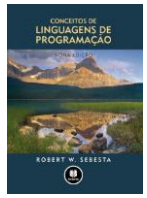


Linguagens de Programação

- Uma linguagem é um conjunto de regras sistemáticas para a comunicação de ideias

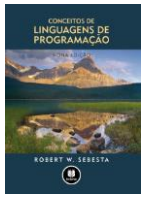
- Linguagens de programação são instrumentos para facilitar a comunicação entre humanos e computadores a fim de solucionar problemas

- Assim, linguagens de programação têm o objetivo de representar alguma informação por meio de uma sequência de símbolos



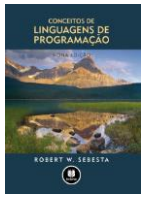
Características de uma linguagem de Programação

- Voltada para descrição de algoritmos (programas)
- Implementável (executável) nos computadores tradicionais
- Gramática e significado bem definidos (não permite sentenças com significado duvidoso)



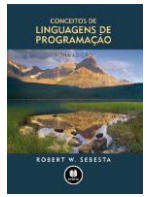
Requisitos de uma Linguagem de Programação

- Universal – todo problema que pode ser resolvido pelo computador tem que ter uma solução programável na LP
- Tem que ser implementável num computador
- Natural para expressar problemas em um certo domínio de aplicação
- Tem que ser capaz de ter uma implementação aceitável, em termos de eficiência



Motivações para Linguagens diferentes

- Propósitos diferentes
- Avanços tecnológicos
- Interesses comerciais
- Cultura e background científico

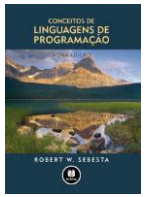


Paradigmas de Programação

- Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns

- Cada linguagem apresenta uma maneira particular de modelar o que é um programa.

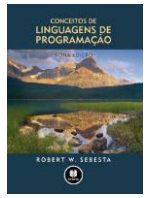
    - Cada paradigma agrupa linguagens que representam programas de forma semelhante



Paradigma de Programação

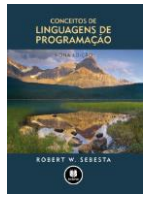
- A escolha de um determinado paradigma influencia a forma com que uma aplicação real é modelada do ponto de vista computacional

- Principais paradigmas:
    - Imperativo
    - Funcional
    - Orientado a Objetos
    - Lógico
    - Concorrente



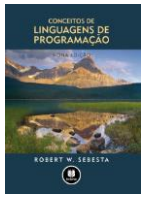
Paradigma de Programação

- Imperativo
    - Procedural – Pascal, FORTRAN, COBOL, C, ADA, Python
    - Orientado a Objetos (Smalltalk, objectPascal, C++, Java), Python
- Declarativo
    - Funcional – LISP, ML, Haskell, Python
    - Lógico – Prolog
- Concorrente
    - Paralelo – n processadores + 1 memória (ADA, concurrent Pascal)
    - Distribuído – n processadores + m memórias (Java)
- Paradigmas Híbridos
    - OOLP (Object-Oriented Logic Programming)



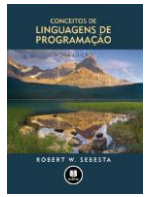
Tópicos do Capítulo 1

- Razões para estudar conceitos de linguagens de programação
- Domínios de programação
- Critérios de avaliação de linguagens
- Influências no projeto de linguagens
- Categorias de linguagens
- Trade-offs no projeto de linguagens
- Métodos de implementação
- Ambientes de programação



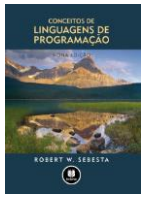
Razões para estudar conceitos de linguagens de programação

- Capacidade aumentada para expressar ideias
- Embasamento melhorado para escolher linguagens apropriadas
- Habilidade aumentada para aprender novas linguagens
- Melhor entendimento da importância da implementação
- Melhor uso de linguagens já conhecidas
- Avanço geral da computação



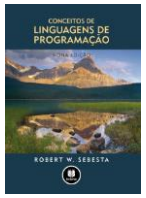
Razões para estudar os conceitos de linguagens de programação

- Aumento da capacidade de expressar ideias
  - Capacidade intelectual ⇔ poder de se expressar
  - Facilidades de algumas linguagens podem ser simuladas em outras linguagens



Razões para estudar os conceitos de linguagens de programação

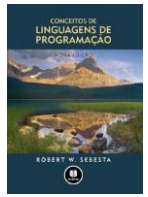
- 1. Capacidade aumentada para expressar ideias
- A linguagem adotada impõe limitações
- Conhecer uma variedade de recursos
- Aprender novas construções
- E se o programador for obrigado a usar uma determinada linguagem?
- Simular construções de outras linguagens
- Exemplo: tipo booleano em C



Razões para estudar os conceitos de linguagens de programação

- Maior embasamento para a escolha de linguagens apropriadas
    - Escolha mais consciente
    - Como escolher a linguagem de um novo projeto?
    - Linguagens surgem ao longo do tempo
    - Ex.: C# (2000)

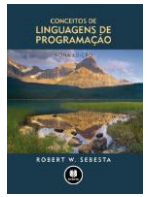
- Capacidade aumentada para aprender novas linguagens
    - Consolidação dos conceitos das linguagens



Razões para estudar os conceitos de linguagens de programação

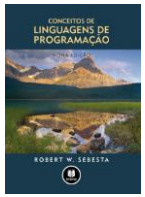
- Entender melhor a importância da implementação
  - Uso de uma linguagem de forma mais inteligente
  - Melhor entendimento da importância da implementação
- Aspectos de implementação que afetam esses conceitos
- Pode levar a habilidade de usar de maneira mais eficiente
- Entender as escolhas entre as construções e as consequências

- Aumento da capacidade de projetar novas linguagens
- Avanço global da computação



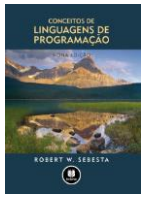
Domínios de programação

- Aplicações científicas
  - Grande número de computações de aritmética de ponto flutuante; uso de matrizes
  - Fortran
- Aplicações empresariais
  - Produz relatório, usa números decimais e caracteres
  - COBOL
- Inteligência artificial
  - Símbolos em vez de números manipulados; uso de listas ligadas
  - LISP
- Programação de sistemas
  - Precisa de eficiência por causa do uso contínuo
  - C
- Software para a Web
  - Eclética coleção de linguagens: de marcação (como XHTML), de scripting (como PHP), de propósito geral (como Java)



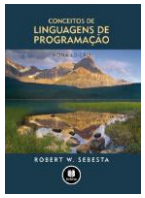
Critérios de avaliação de linguagens

- **Legibilidade**: facilidade com a qual os programas podem ser lidos e entendidos
- **Facilidade de escrita**: facilidade com a qual uma linguagem pode ser usada para criar programas para um dado domínio
- **Confiabilidade**: conformidade com as especificações
- **Custo**: o custo total definitivo de uma linguagem



Elegibilidade

- Facilidade com a qual os programas podem ser lidos e entendidos
- Antes de 1970: eficiência e legibilidade de máquina
- Após, facilidade de manutenção



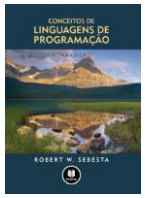
Elegibilidade

- **1.1 Simplicidade geral**
  - Uma linguagem com muitas construções é difícil de aprender
  - Multiplicidade de recursos
    - Ex.: count = count + 1

      count += 1

      count++

      ++count
  - Sobrecarga de operadores
    - Apesar de útil, pode não fazer sentido para quem ler
    - Ex.: operador + para vetores

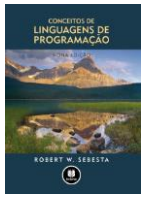


Elegibilidade

1.2 Ortogonalidade

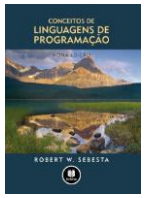
- Conjunto pequeno de construções que podem ser combinado a um número pequeno de formas
- Cada combinação é legal e significativa
- Um recurso é independente do contexto
- A falta de ortogonalidade leva a exceções
- Ex. Na linguagem C:
  - Vetores não podem ser retornados por funções e registros sim.
  - Um membro de uma estrutura pode ser de qualquer tipo, exceto void
  - Vetores podem ser qualquer tipo exceto void
  - Parâmetros são passados por valor, exceto vetores

11/02/2015



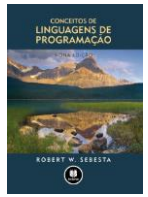
Elegibilidade

- **1.3 Tipos de dados**
  - Presença de mecanismos para definir tipos de dados
  - O que é mais claro?
    - flag=1 ou flag=true
- **1.4 Projeto da sintaxe**
  - Formato dos identificadores: Fortran 77 permite 6 caracteres
  - Palavras especiais (while, for, if)
    - { } ou end if / end loop



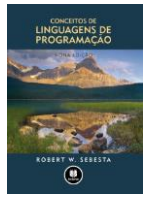
Facilidade de Escrita

- A capacidade de criar programas para um domínio
- **2.1 Simplicidade e ortogonalidade**
  - Número menor de construções
- **2.2 Suporte à abstração**
  - Habilidade de definir estruturas complicadas ignorando alguns detalhes
  - Subprogramas
  - Abstração de estruturas de dados
- **2.3 Expressividade**
  - cont++ ou for



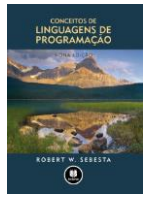
Critério de avaliação: confiabilidade

- Verificação de tipos
  - Testes para detectar erros de tipos
- Tratamento de exceções
  - Interceptar erros em tempo de execução e tomar medidas corretivas
- Utilização de apelidos
  - Nomes distintos que podem ser usados para acessar a mesma célula de memória
- Legibilidade e facilidade de escrita
  - Uma linguagem que não oferece maneiras naturais para expressar os algoritmos requeridos irá necessariamente usar abordagens não naturais, reduzindo a confiabilidade



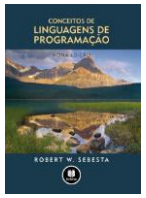
Critério de avaliação: custo

- Treinar programadores para usar a linguagem
- Escrever programas (proximidade com o propósito da aplicação em particular)
- Compilar programas
- Executar programas
- Sistema de implementação da linguagem: disponibilidade de compiladores gratuitos
- Confiabilidade baixa leva a custos altos
- Manter programas



Critério de avaliação: outros

- Portabilidade
  - A facilidade com a qual os programas podem ser movidos de uma implementação para outra
- Generalidade
  - A aplicabilidade a uma ampla faixa de aplicações
- Bem definida
  - Em relação à completude e à precisão do documento oficial que define a linguagem



Desafio

O triângulo de Pascal é um triângulo numérico infinito formado por números binomiais $\binom{n}{k}$ onde n representa o número da linha e k representa o número da coluna, iniciando a contagem a partir do zero.

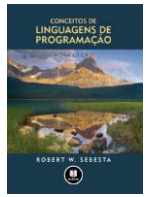
O triângulo foi descoberto pelo matemático chinês Yang Hui, e 500 anos depois várias de suas propriedades foram estudadas pelo francês Blaise Pascal.

Cada número do triângulo de Pascal é igual à soma do número imediatamente acima e do antecessor do número de cima.

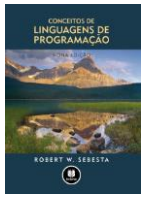
Crie um programa que tenha como entrada um número inteiro e imprima o Triângulo de Pascal até a linha determinada por esse número.

Obs: **NÃO** se deve usar recursão, array nem nenhuma outra estrutura parecida.

```
Digite até que linha você deseja que o Triângulo seja impresso:
7 //Entrada do usuário

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
```

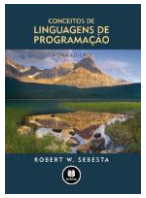



```
#include <stdio.h>

int main() {
 int colunas,j;
 int anterior;

 int linhas;
 do {
scanf("%d",&linhas);
 } while (linhas < 0);

 anterior = 1;
 printf("%d\n",anterior);

 if (linhas) {
int linha_atual = 1;
colunas = 2;
linha_atual = 1;
for(j=1;j<colunas && linha_atual <= linhas;) {

  if(j==1) printf("1 ");
  anterior = anterior*(linha_atual-j+1)/j;
  printf("%d ", anterior);
  j++;
  if (j == colunas) {
            anterior = 1;
            linha_atual++;
            colunas = linha_atual+1;
            j=1;
            puts("");
  }
}

 }
}
```



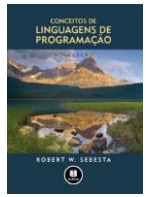
Influências no projeto de linguagens

- Arquitetura de computadores
  - Linguagens são projetadas considerando a principal arquitetura de computadores, chamada de arquitetura de *von Neumann*
- Metodologias de projeto de programas
  - Novas metodologias de desenvolvimento de software (por exemplo, desenvolvimento de software orientado a objeto) levaram a novos paradigmas de programação e, por extensão, a novas linguagens de programação

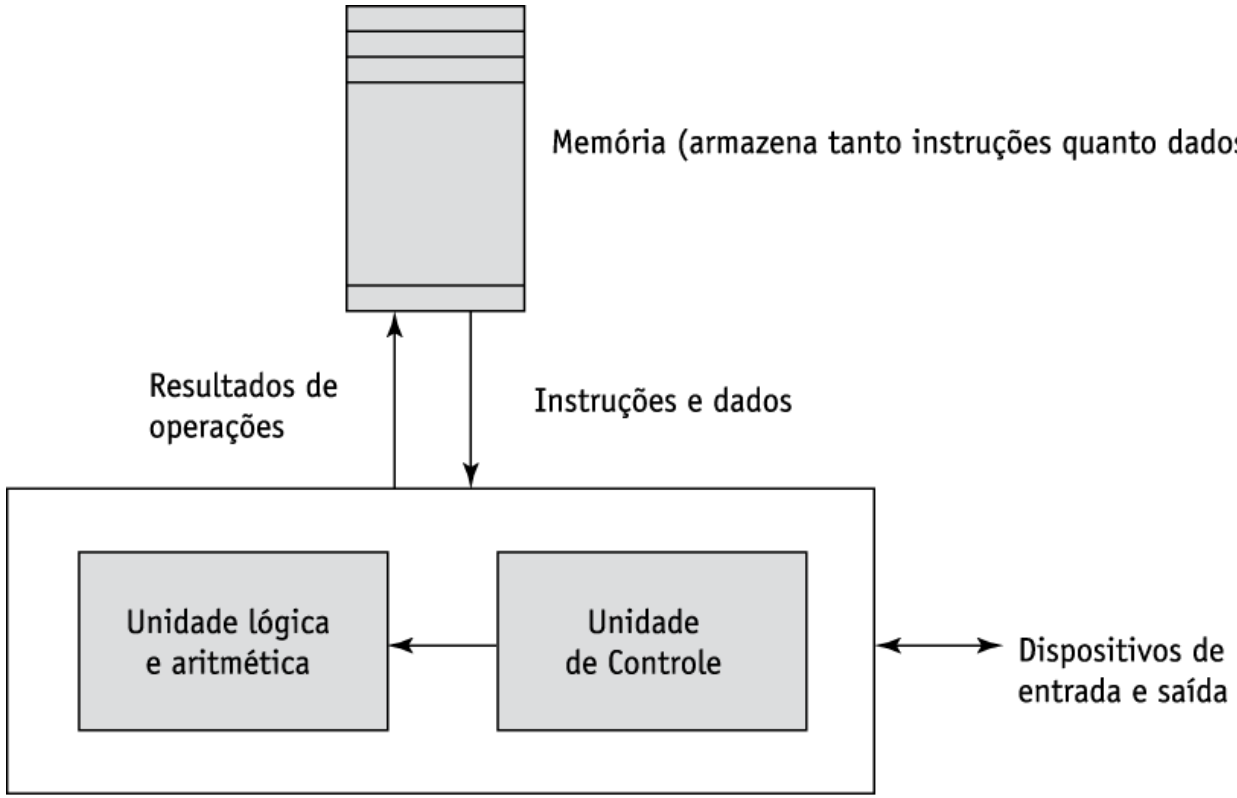


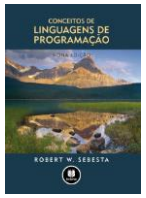
Influências na arquitetura de computadores

- Principal arquitetura de computadores: von Neumann
- Linguagens imperativas, mais populares, por causa dos computadores von Neumann
  - Dados e programas armazenados na memória
  - A memória é separada da CPU
  - Instruções e dados são canalizadas a partir da memória para CPU
  - Base para linguagens imperativas
    - Variáveis modelam as células de memória
    - Sentenças de atribuição são baseadas na operação de envio de dados e instruções
    - Iteração é eficiente



Arquitetura *Von Neumann*

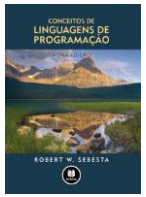




Arquitetura *Von Neumann*

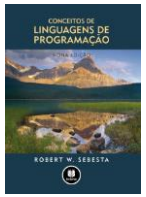
- Ciclo de obtenção e execução (em um computador com arquitetura von Neumann)

```
inicialize o contador de programa
repita para sempre
  obtenha a instrução apontada pelo contador de programa
  incremente o contador de programa
  decodifique a instrução
  execute a instrução
fim repita
```



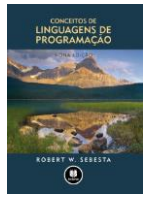
Influências na metodologia de programa

- Anos 1950 e começo dos 1960: Aplicações simples; preocupação com a eficiência da máquina
- Final dos anos 60: Eficiência das pessoas se tornou importante; legibilidade, melhores estruturas de controle
  - Programação estruturada
  - Projeto descendente (*top-down*) e de refinamento passo a passo
- Final dos anos 70: Da orientação aos procedimentos para uma orientação aos dados
  - Abstração de dados
- Meio dos anos 80: Programação orientada a objetos
  - Abstração de dados + herança + vinculação dinâmica de métodos



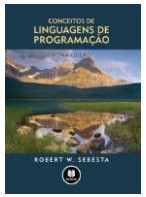
Categorias de linguagens

- Imperativa
  - Características centrais são variáveis, sentenças de atribuição e de iteração
  - Inclui linguagens que suportam programação orientada a objeto
  - Inclui linguagens de *scripting*
  - Inclui as linguagens visuais
  - Exemplos: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Funcional
  - Principais meios de fazer os cálculos é pela aplicação de funções para determinados parâmetros
  - Exemplos: LISP, Scheme
- Lógica
  - Baseada em regras (regras são especificadas sem uma ordem em particular)
  - Example: Prolog
- De marcação/programação híbrida
  - Linguagens de marcação estendida para suportar alguma programação
  - Exemplos: JSTL, XSLT



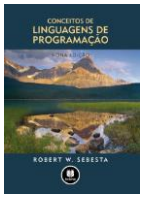
Trade-Offs no projeto de linguagens

- Confiabilidade x custo de execução
  - Exemplo: Java exige que todas as referências aos elementos de um vetor sejam verificadas para garantir que os índices estão em suas faixas legais
- Legibilidade x facilidade de escrita
  - Exemplo: APL inclui um poderoso conjunto de operadores (e um grande número de novos símbolos), permitindo que computações complexas sejam escritas em um programa compacto, com o custo de baixa legibilidade
- Facilidade de escrita (flexibilidade) x confiabilidade
  - Exemplo: Ponteiros de C++ são poderosos e flexíveis, mas não são confiáveis



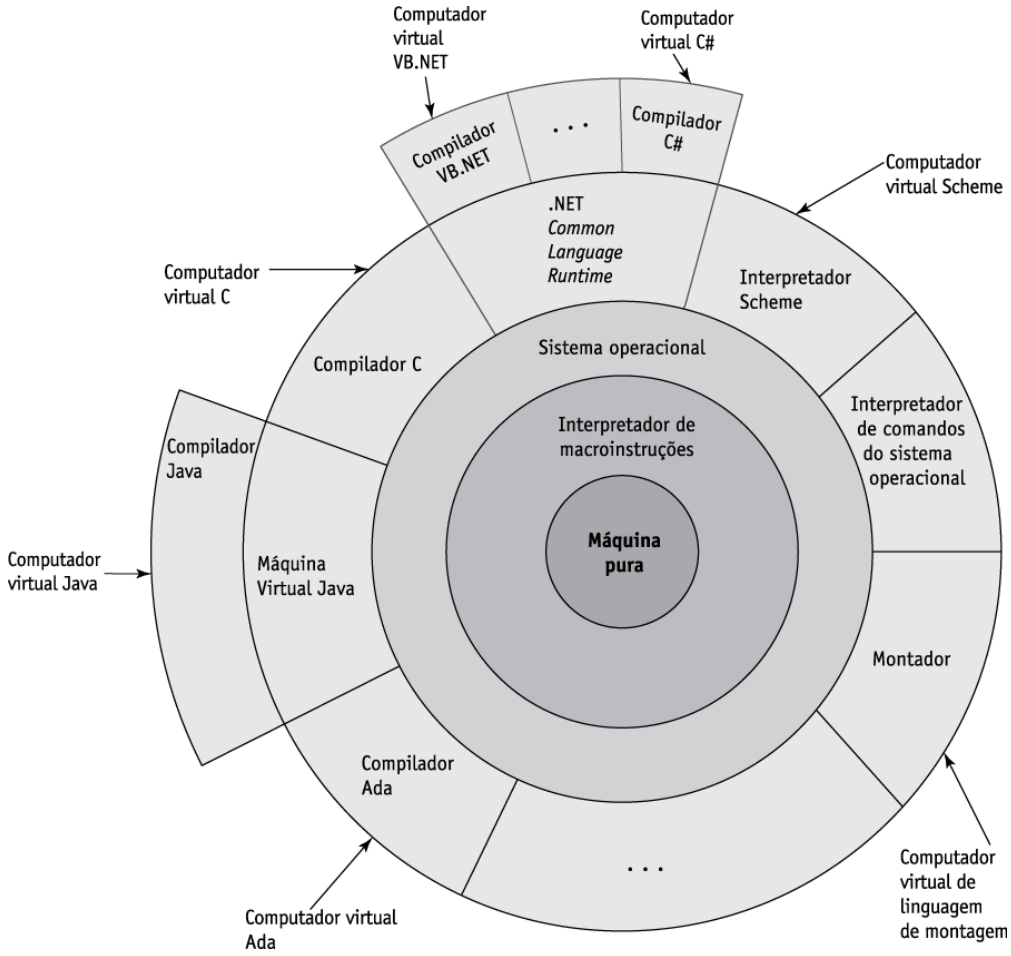
Métodos de implementação

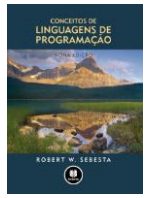
- Compilação
  - Programas são traduzidos para linguagem de máquina
- Interpretação pura
  - Programas são interpretados por outro programa chamado interpretador
- Sistemas de implementação híbridos
  - Um meio termo entre os compiladores e os interpretadores puros



Visão em camadas de um computador

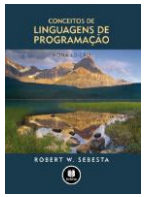
O Sistema operacional e as implementações de linguagem são colocados em camadas superiores à interface de linguagem de máquina de um computador



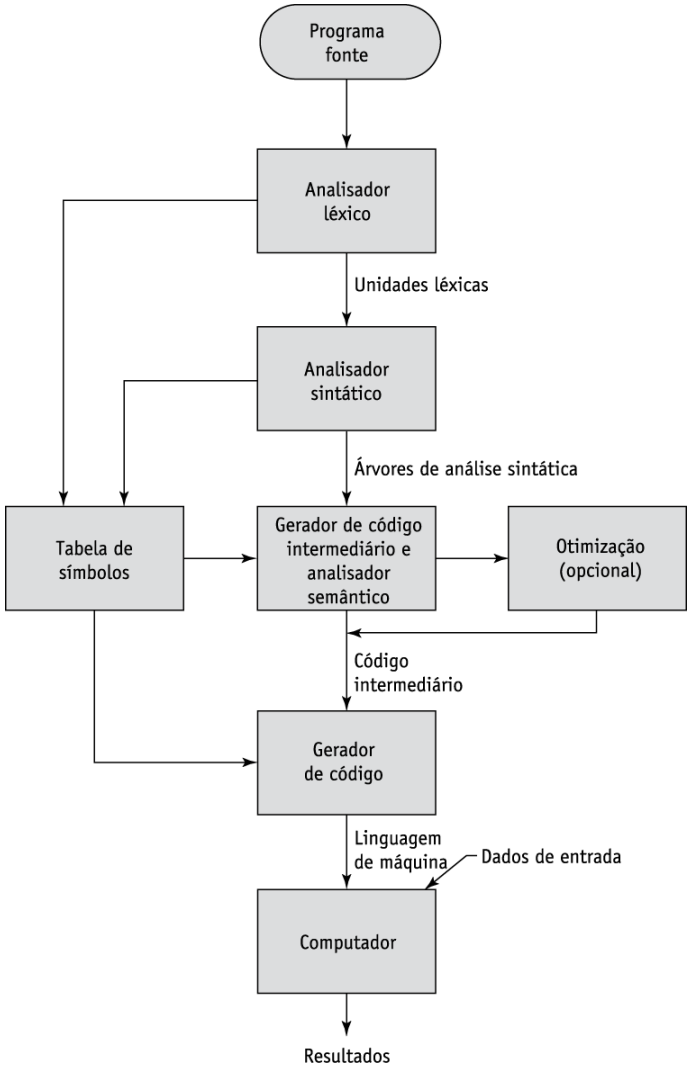


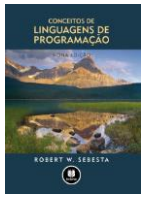
Compilação

- Traduz programas (linguagem de fonte) em código de máquina (linguagem de máquina)
- Tradução lenta, execução rápida
- Processo de compilação tem várias fases:
  - análise léxica: agrupa os caracteres do programa fonte em unidades léxicas
  - análise sintática: transforma unidades léxicas em árvores de análise sintática (*parse trees*), que representam a estrutura sintática do programa
  - análise semântica: gera código intermediário
  - geração de código: código de máquina é gerado



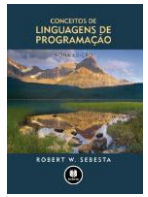
O processo de compilação





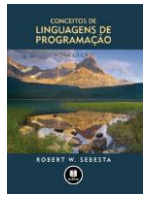
Terminologias de compilação adicionais

- **Módulo de carga** (imagem executável): o código de usuário e de sistema juntos
- **Ligação e carga**: o processo de coletar programas de sistema e ligá-los aos programas de usuário



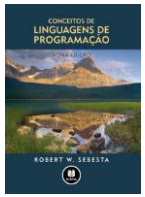
Gargalo de *Von Neumann*

- A velocidade de conexão entre a memória de um computador e seu processador determina a velocidade do computador
- Instruções de programa normalmente podem ser executadas mais rapidamente do que a velocidade de conexão, o que resulta em um gargalo
- Conhecido como *gargalo de von Neumann*; é o fator limitante primário na velocidade dos computadores

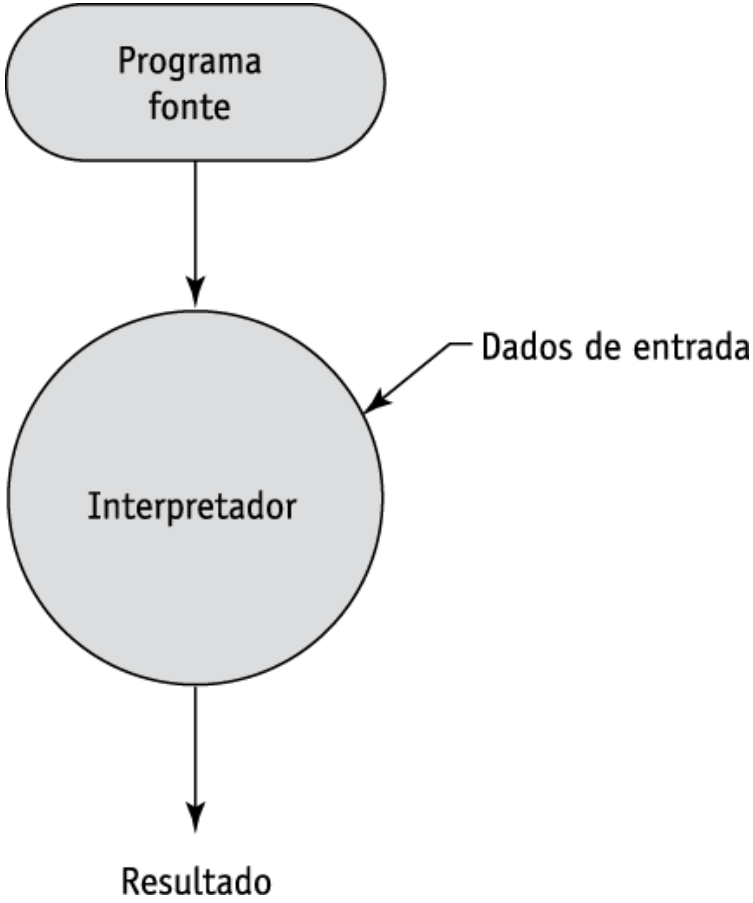


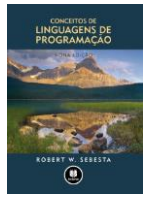
Interpretação pura

- **Sem tradução**
- Fácil implementação de programas (mensagens de erro em tempo de execução podem referenciar unidades de código fonte)
- Execução mais lenta (tempo de execução de 10 a 100 vezes mais lento do que nos sistemas compilados)
- Geralmente requer mais espaço
- Raramente usada em linguagens de alto nível
- Volta significativa com algumas linguagens de *scripting* para a Web (como JavaScript e PHP)



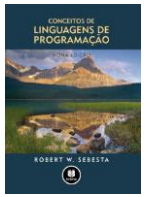
Processo de interpretação pura



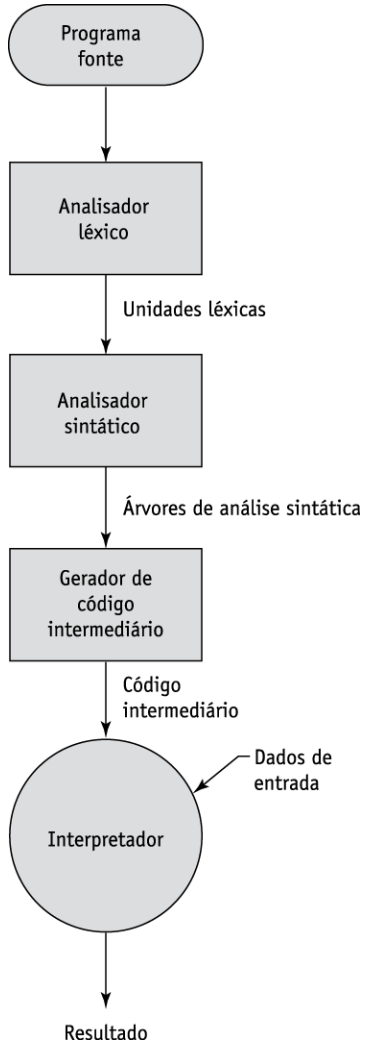


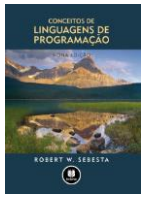
Sistemas de implementação híbridos

- Um meio termo entre os compiladores e os interpretadores puros
- Uma linguagem de alto nível é traduzida para uma linguagem intermediária que permite fácil interpretação
- Mais rápido do que interpretação pura
- Exemplos
  - Programas em Perl eram parcialmente compilados para detectar erros antes da interpretação
  - As primeiras implementações de Java eram todas híbridas; seu formato intermediário, *byte code*, fornece portabilidade para qualquer máquina que tenha um interpretador de *bytecodes* e um sistema de tempo de execução associado (juntos, são chamados de Máquina Virtual Java)



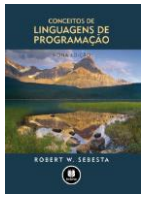
Sistema de implementação híbrido





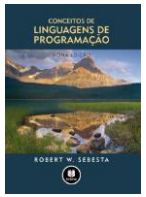
Sistemas de implementação *Just-in-Time*

- Inicialmente traduz os programas para uma linguagem intermediária
- Então, compila os métodos da linguagem intermediária para linguagem de máquina quando esses são chamados
- A versão em código de máquina é mantida para chamadas subsequentes
- Sistemas JIT são agora usados amplamente para programas Java
- As linguagens .NET também são implementadas com um sistema JIT



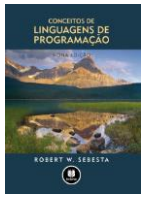
Pré-processadores

- As instruções de pré-processador são comumente usadas para especificar que o código de outro arquivo deve ser incluído
- Um pré-processador é um programa que processa um programa imediatamente antes de o programa ser compilado para expandir macros embutidos
- Um exemplo conhecido: pré-processador de C
  - expande `#include, #define` e macros similares



Ambientes de programação

- Coleção de ferramentas usadas no desenvolvimento de software
- UNIX
  - Um ambiente de programação mais antigo
  - Agora bastante usado por meio de uma interface gráfica com o usuário (GUI) que roda sobre o UNIX
- Microsoft Visual Studio .NET
  - Grande e complexo
- Usado para desenvolver software em qualquer uma das cinco linguagens .NET
- NetBeans
  - Usado primariamente para o desenvolvimento de aplicações Web usando Java, mas também oferece suporte a JavaScript, Ruby e PHP



Resumo

- O estudo de linguagens de programação é valioso por diversas razões:
  - Aumenta nossa capacidade de usar diferentes construções ao escrever programas
  - Permite que escolhamos linguagens para os projetos de forma mais inteligente
  - Torna mais fácil o aprendizado de novas linguagens
- Critérios mais importantes para a avaliação de linguagens:
  - Legibilidade, facilidade de escrita, confiabilidade e custo geral
- As principais influências no projeto de linguagens são a arquitetura de máquina e as metodologias de projeto de software
- Os principais métodos de implementar linguagens de programação são a compilação, a interpretação pura e a implementação híbrida