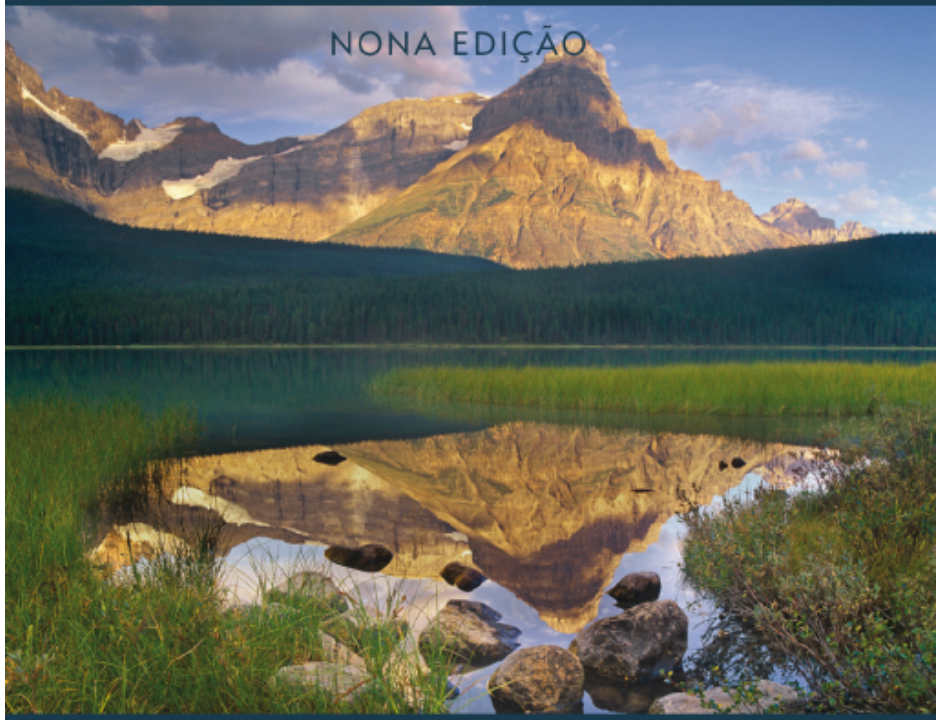
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

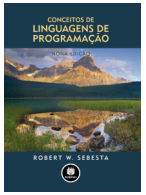


ROBERT W. SEBESTA

bookman

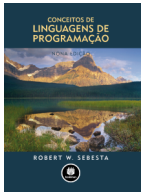
Capítulo 9

Subprogramas



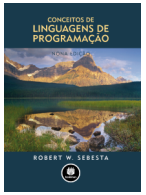
Introdução

- Dois recursos fundamentais de abstração
  - Abstração de processos
    - Desde o início da história das linguagens de programação
  - Abstração de dados
    - Desde o início dos anos 1980



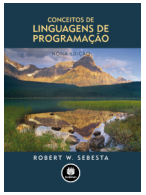
Fundamentos de subprogramas

- Cada subprograma tem um único ponto de entrada
- A unidade de programa chamadora é suspensa durante a execução do subprograma chamado
- O controle sempre retorna para o chamador quando a execução do subprograma termina



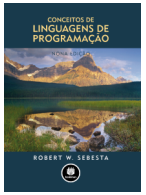
Definições básicas

- Uma ***definição de subprograma*** descreve a **interface** e as **ações** da **abstração** de um subprograma
  - Em **Python**, definições de subprogramas são **executáveis**; em todas as outras linguagens, elas são não executáveis
- Uma *chamada a subprograma* é a requisição **explícita** que diz que o subprograma deve ser **executado**
- Um ***cabeçalho*** *de subprograma* é a primeira parte da definição, incluindo o **nome**, o **tipo** de subprograma e os **parâmetros** formais
- O *perfil de parâmetros* (ou ***assinatura***) de um subprograma contém o **número**, a **ordem** e os **tipos** de seus parâmetros formais
- O ***protocolo*** é o perfil de parâmetros mais, se for uma **função**, seu tipo de **retorno**



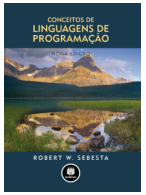
Definições básicas (continuação)

- Declarações de funções em C e C++ são chamadas de ***protótipos***
- Uma ***declaração de subprograma*** fornece o **protocolo**, mas não inclui seu corpo
- Um ***parâmetro formal*** é uma variável listada no **cabeçalho** do subprograma e **usado** nele
- Um ***parâmetro real*** representa um **valor** ou endereço usado na sentença de **chamada do subprograma**



Correspondência entre os parâmetros reais e formais

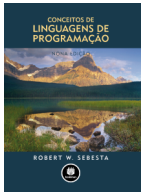
- **Posicional**
  - A **vinculação dos parâmetros** reais a parâmetros formais é por **posição**: o primeiro real é vinculado ao primeiro formal e assim por diante
  - Seguro e efetivo
- **Palavra-chave**
  - O nome do parâmetro formal a que um parâmetro real deve ser vinculado é especificado com o parâmetro real
  - *Vantagem*: Parâmetros podem aparecer em qualquer ordem, evitando erros de correspondência
  - *Desvantagem*: O usuário deve saber os nomes dos parâmetros formais



Valores padrão de parâmetros formais

- Em certas linguagens (como C++, Python, Ruby, Ada e PHP), parâmetros formais podem ter valores padrão (se nenhum parâmetro real é passado)
  - Em C++, parâmetros padrão devem aparecer por último, já que os parâmetros são posicionalmente associados

- Número variável de parâmetros
  - C# permite que os métodos aceitem um número variável de parâmetros, desde que sejam do mesmo tipo — o método especifica seu parâmetro com o modificador **`params`**
  - Em Ruby, os parâmetros reais são enviados como elementos de uma dispersão anônima e o correspondente parâmetro formal é precedido por um asterisco
  - Em Python, o real é uma lista de valores e o parâmetro formal correspondente é um nome com um asterisco
  - Em Lua, um número variável de parâmetros é representado por um parâmetro formal com reticências

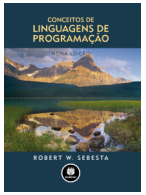


Blocos em Ruby

- Ruby inclui métodos de iteração para suas estruturas de dados
- Iteradores são implementados com blocos, que podem ser definidos por aplicações
- Os blocos podem ter parâmetros formais, que são especificados entre barras verticais; o bloco que é passado para o subprograma chamado é chamado com uma sentença `yield`

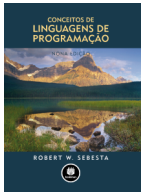
```
def fibonacci(last)
  first, second = 1, 1
  while first <= last
    yield first
    first, second = second, first + second
  end
end

puts "Fibonacci numbers less than 100 are:"
fibonacci(100) {|num| print num, " "}
puts
```



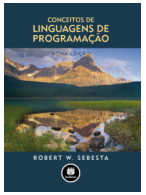
Procedimentos e funções

- Existem duas categorias de subprogramas
  - *Procedimento* são coleções de sentenças que definem computações parametrizadas
  - *Funções* se parecem estruturalmente com os procedimentos, mas são semanticamente modeladas como funções matemáticas
    - Se uma função é um modelo fiel, ela não produz efeitos colaterais
    - Na prática, muitas funções em programas têm efeitos colaterais



Questões de projeto para subprogramas

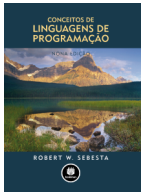
- As **variáveis locais** são alocadas estaticamente ou dinamicamente?
- As **definições de subprogramas** podem aparecer em outras definições de subprogramas?
- Que método ou métodos de **passagem de parâmetros** são usados?
- Os **tipos** dos parâmetros reais são **verificados**?
- Se os subprogramas puderem ser passados como parâmetros e puderem ser aninhados, qual é o **ambiente de referenciamento** de um subprograma passado como parâmetro?
- Os subprogramas podem ser **sobrecarregados**?
- Os subprogramas podem ser **genéricos**?



Ambientes de referenciamento local

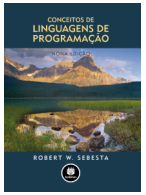
- Variáveis locais podem ser dinâmicas da pilha
  - Vantagens
    - Suporte para recursão
    - Armazenamento para variáveis locais é compartilhado entre alguns subprogramas
  - Desvantagens
    - Custo para alocação, liberação, tempo de inicialização
    - Endereçamento indireto
    - Subprogramas não podem ser sensíveis ao histórico

- Variáveis locais podem ser estáticas
  - Vantagens e desvantagens são o oposto daquelas das variáveis locais dinâmicas da pilha

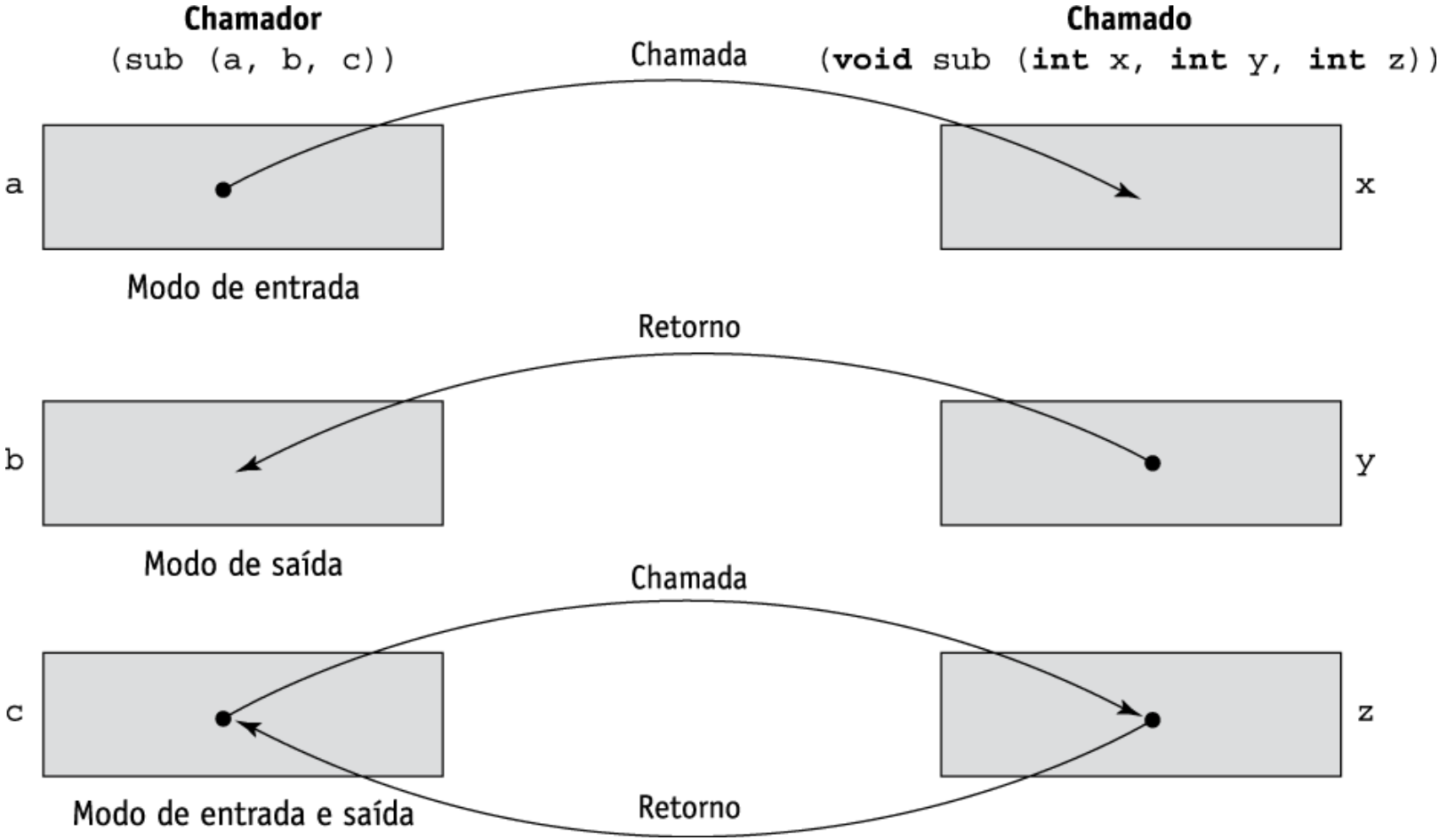


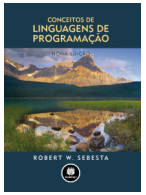
Modelos semânticos de passagem de parâmetros

- Modo de entrada
- Modo de saída
- Modo de entrada e saída



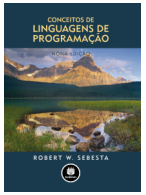
Modelos de passagem de parâmetros





Modelos conceituais de transferência e dados

- Um valor real é copiado (movimento físico)
- Um caminho de acesso é transmitido



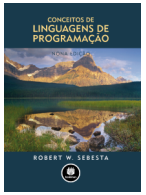
Passagem por valor

- O valor do **parâmetro real** é usado para **inicializar** o **parâmetro formal** correspondente

  – Normalmente implementada por **cópia**

  – Poderia ser implementada transmitindo um caminho de acesso para o valor do parâmetro real no chamador, mas isso requereria que o valor estivesse em uma célula com proteção contra escrita (uma que pudesse ser apenas lida)

  – *Desvantagens* (se cópias são usadas): é necessário armazenamento adicional para o parâmetro formal e o movimento pode ser custoso



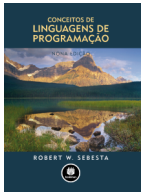
Passagem por resultado

- Quando um parâmetro é passado por **resultado**, **nenhum valor** é **transmitido** para o subprograma

- O parâmetro formal correspondente age como uma variável local, mas logo antes de o controle ser transmitido de volta para o chamador

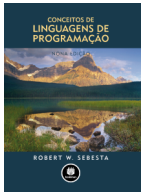
- Problema em potencial:

  `sub(p1, p1);` pode existir uma colisão entre parâmetros reais



Passagem por valor-resultado

- Modelo de implementação para parâmetros no modo de entrada e saída no qual os valores reais são copiados
- Chamada de passagem por cópia
- Parâmetros formais têm armazenamento local
- Desvantagens:
  - As mesmas da passagem por resultado
  - As mesmas da passagem por valor

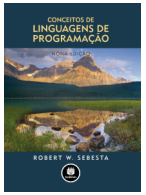


Passagem por referência

- Transmite um caminho de acesso

- Vantagem: processo de passagem é eficiente (não são necessárias cópias nem espaço duplicado)

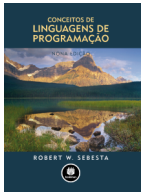
- Desvantagens
  - Acessos mais lentos (do que na passagem por valor) a parâmetros formais
  - Potenciais efeitos colaterais (colisões)
  - Apelidos podem ser criados



Passagem por nome

- Por substituição textual

- Quando os parâmetros são passados por nome, o parâmetro real é, na prática, textualmente substituído pelo parâmetro formal correspondente em todas as suas ocorrências no subprograma

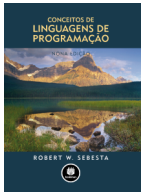


Implementando métodos de passagem de parâmetros

- Na maioria das linguagens contemporâneas, a **comunicação** via parâmetros ocorre por meio da **pilha** de tempo de execução

- Passagem por **referência** é a mais **simples** de implementar; apenas seu endereço deve ser colocado na pilha

- Um erro sutil, mas fatal, pode ocorrer com parâmetros com passagem por referência e passagem por valor-resultado: um formal correspondente a uma constante pode ser trocado erroneamente

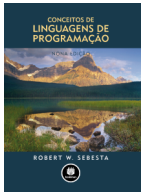


Métodos de passagem de parâmetros das principais linguagens

- C
  - Passagem por valor
  - A passagem por referência é atingida por meio do uso de ponteiros como parâmetros

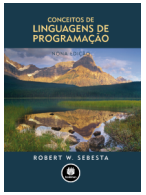
- C++
  - Inclui o tipo referência para passagem por referência

- Java
  - Todos os parâmetros têm passagem por valor
  - Parâmetros objetos têm passagem por referência



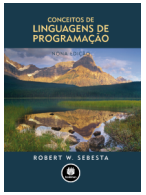
Métodos de passagem de parâmetros das principais linguagens (continuação)

- Fortran 95
  - Parâmetros podem ser declarados para serem dos modos de entrada, de saída ou de entrada e saída
- C#
  - Método padrão: passagem por valor
    - Passagem por referência é especificada precedendo um parâmetro formal e seu real correspondente com `ref`
- PHP: similar a C#
- Perl: parâmetros reais são implicitamente colocados em uma matriz pré-definida chamada de `@_`
- Python e Ruby usam passagem por atribuição (todos os valores de dados são objetos)



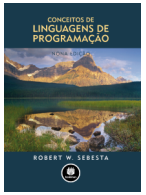
Verificação de tipos dos parâmetros

- Considerado importante para confiabilidade
- FORTRAN 77 e versão original do C: não tinham
- Pascal, FORTRAN 90, Java e Ada: sempre requerem
- ANSI C e C++: escolha é feita pelo usuário
- Linguagens relativamente **novas** como Perl, JavaScript e PHP **não requerem** verificação de tipos
- Em Python e Ruby, variáveis não têm tipos (objetos sim), então verificação de tipos dos parâmetros não é possível



Matrizes multidimensionais como parâmetros

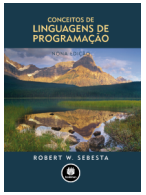
- Se uma matriz multidimensional é passada para um subprograma e o subprograma é compilado separadamente, o compilador precisa saber o tamanho declarado dessa matriz para construir a função de **mapeamento** de armazenamento



Matrizes multidimensionais como parâmetros: C e C++

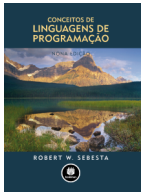
- Matriz pode ser passada como um ponteiro, e as dimensões reais da matriz podem ser incluídas como parâmetros

- A função pode avaliar a função de mapeamento de armazenamento escrita pelo usuário usando **aritmética de ponteiros** cada vez que um elemento da matriz precisar ser referenciado



Matrizes multidimensionais como parâmetros : Ada

- Ada – não é um problema
  - Matrizes com restrições – tamanho é parte do tipo da matriz
  - Matrizes sem restrições – tamanho declarado é parte da declaração do objeto

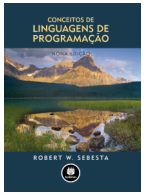


Matrizes multidimensionais como parâmetros : Java e C#

- Similar a Ada

- Matrizes são objetos; elas são todas de uma única dimensão, mas os elementos podem ser matrizes

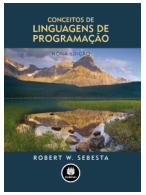
- Cada matriz herda uma constante nomeada (`length` em Java, `Length` em C#) que é configurada para o tamanho da matriz quando o objeto matriz é criado



Considerações de projeto para passagens de parâmetros

- Duas considerações importantes
  - Eficiência
  - Transferências de dados de uma via ou de duas vias

- Mas as considerações acima estão em conflito
  - **Boa programação** sugere **acesso limitado a variáveis**, o que significa **uma via**, sempre que possível
  - Mas **passagem por referência é mais eficiente** para passar **estruturas** de tamanho significativo

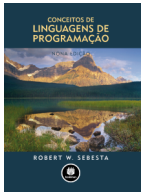


Parâmetros que são subprogramas

- Às vezes, é conveniente enviar os nomes de subprogramas como parâmetros

- Questões:
  1. Tipos de parâmetros são verificados?

  2. Qual é o ambiente de referenciamento correto para um subprograma enviado como parâmetro?

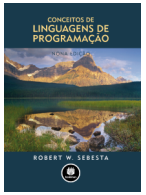


Parâmetros que são subprogramas: verificação de tipos dos parâmetros

- C e C++: as funções não podem ser passadas como parâmetros, mas ponteiros para funções podem

- FORTRAN 95 possui um mecanismo para fornecer tipos de parâmetros para subprogramas que são passados como parâmetros

- Ada não permite que subprogramas sejam passados como parâmetros; a funcionalidade é fornecida pelos recursos de tipos genéricos

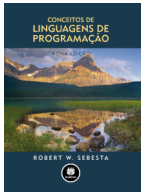


Parâmetros que são subprogramas: ambiente de referenciamento

- *Vinculação rasa (shallow binding)*:
  - O ambiente da **sentença de chamada** que **chama** o subprograma passado

- *Vinculação profunda (Deep binding)*:
  - O ambiente da **definição** do subprograma passado

- *Vinculação ad hoc (Ad hoc binding)*:
  - O ambiente da **sentença de chamada** que **passou** o subprograma como um parâmetro real

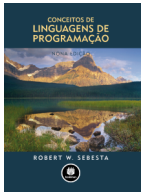


Subprogramas sobrecarregados

- Um *subprograma sobrecarregado* é um subprograma que **possui o mesmo nome de outro subprograma** no mesmo ambiente de referenciamento

- C++, Java, C# e Ada **incluem** subprogramas **sobrecarregados** pré-definidos

- Ada, Java, C++ e C# permitem aos usuários escrever múltiplas versões de subprogramas com o mesmo nome

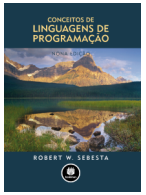


Subprogramas genéricos

- Um subprograma *genérico* ou *polimórfico* recebe parâmetros de diferentes tipos em diferentes ativações

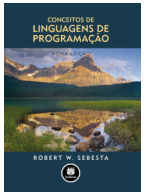
- Subprogramas sobrecarregados fornecem um tipo particular de polimorfismo chamado de *poliformismo ad hoc*

- O *polimorfismo paramétrico* é fornecido por um subprograma que recebe parâmetros genéricos que são usados em expressões de tipo que descrevem os tipos dos parâmetros do subprograma



Subprogramas genéricos (continuação)

- C++
  - Versões de um subprograma genérico são criados implicitamente quando o subprograma é nomeado em uma chamada ou quando seu operador tem o endereço &
  - Subprogramas genéricos são precedidos pela cláusula **template**, que `lista` as variáveis genéricas



Subprogramas genéricos (continuação)

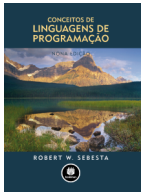
- Java 5.0
  - Diferenças entre genéricos em Java 5.0 e aqueles em C++ e Ada:

1. Parâmetros genéricos em Java 5.0 podem ser classes

2. Métodos genéricos em Java 5.0 são instanciados apenas uma vez, como métodos realmente genéricos

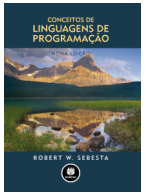
3. As restrições podem ser especificadas na faixa de classes que podem ser passados para o método genérico como parâmetros genéricos

4. Tipos coringas de parâmetros genéricos



Subprogramas genéricos (continuação)

- C# 2005
  - Similares em termos de recursos àqueles de Java 5.0
  - Uma **diferença**: os **parâmetros** de tipo reais em uma chamada podem ser **omitidos** se o compilador puder inferir o tipo não especificado

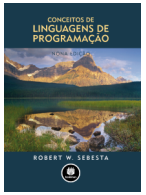


Exemplos de poliformismo de parâmetros: C++

```
template <class Type>
Type max(Type first, Type second) {
    return first > second ? first : second;
}
```

- O modelo acima pode ser instanciada para qualquer tipo em que o operador > é definido

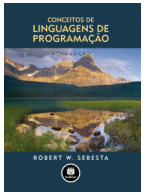
```
int max (int first, int second) {
  return first > second? first : second;
}
```



Questões de projeto para funções

- Os efeitos colaterais são permitidos?
  - Os parâmetros para funções devem ser sempre parâmetros no modo de entrada

- Que tipos de valores podem ser retornados?
  - A maioria das linguagens de programação imperativas restringe os tipos que podem ser retornados
  - C permite que quaisquer tipos, exceto matrizes e funções
  - C++ também permite tipos definidos pelo usuário
  - Em Ada, subprogramas podem retornar qualquer tipo, mas não podem ser retornadas a partir de funções
  - Métodos em Java e C# podem retornar qualquer tipo
  - Em Python e Ruby, métodos são objetos que podem ser tratados como qualquer outro
  - Lua permite funções retornaram múltiplos valores



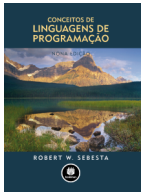
Corrotinas

- Uma *corrotina* é um subprograma especial que possui múltiplas entradas; as corrotinas chamadora e chamada estão em um relacionamento mais igualitário

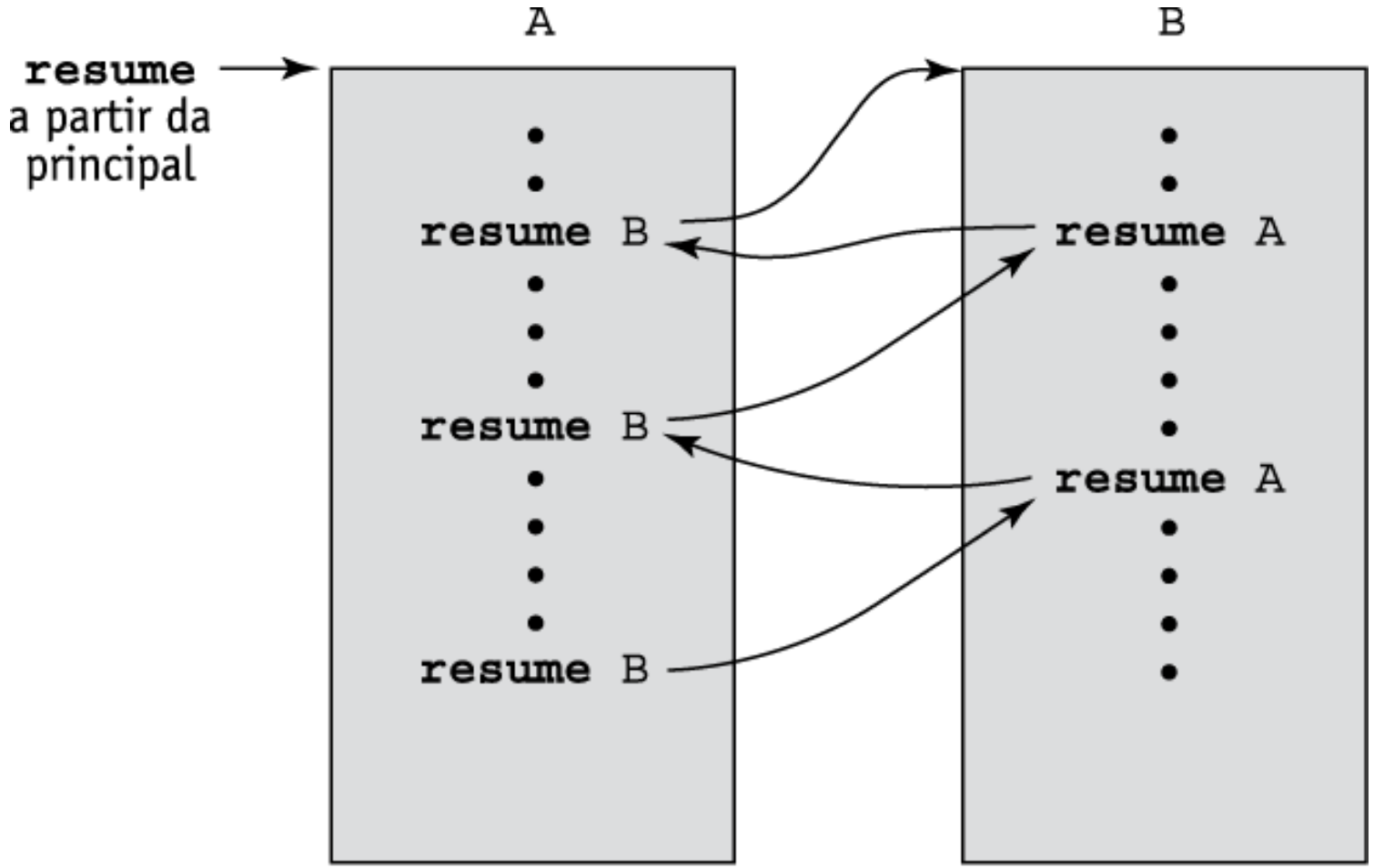
- A invocação de uma corrotina é chamada de uma continuação em vez de uma chamada

- Na verdade, o mecanismo de controle das corrotinas é frequentemente chamado de modelo de controle de unidades simétrico

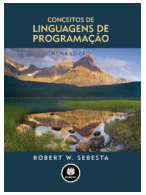
- Corrotinas fornecem execução de *quasi-concorrência* de unidades de programas



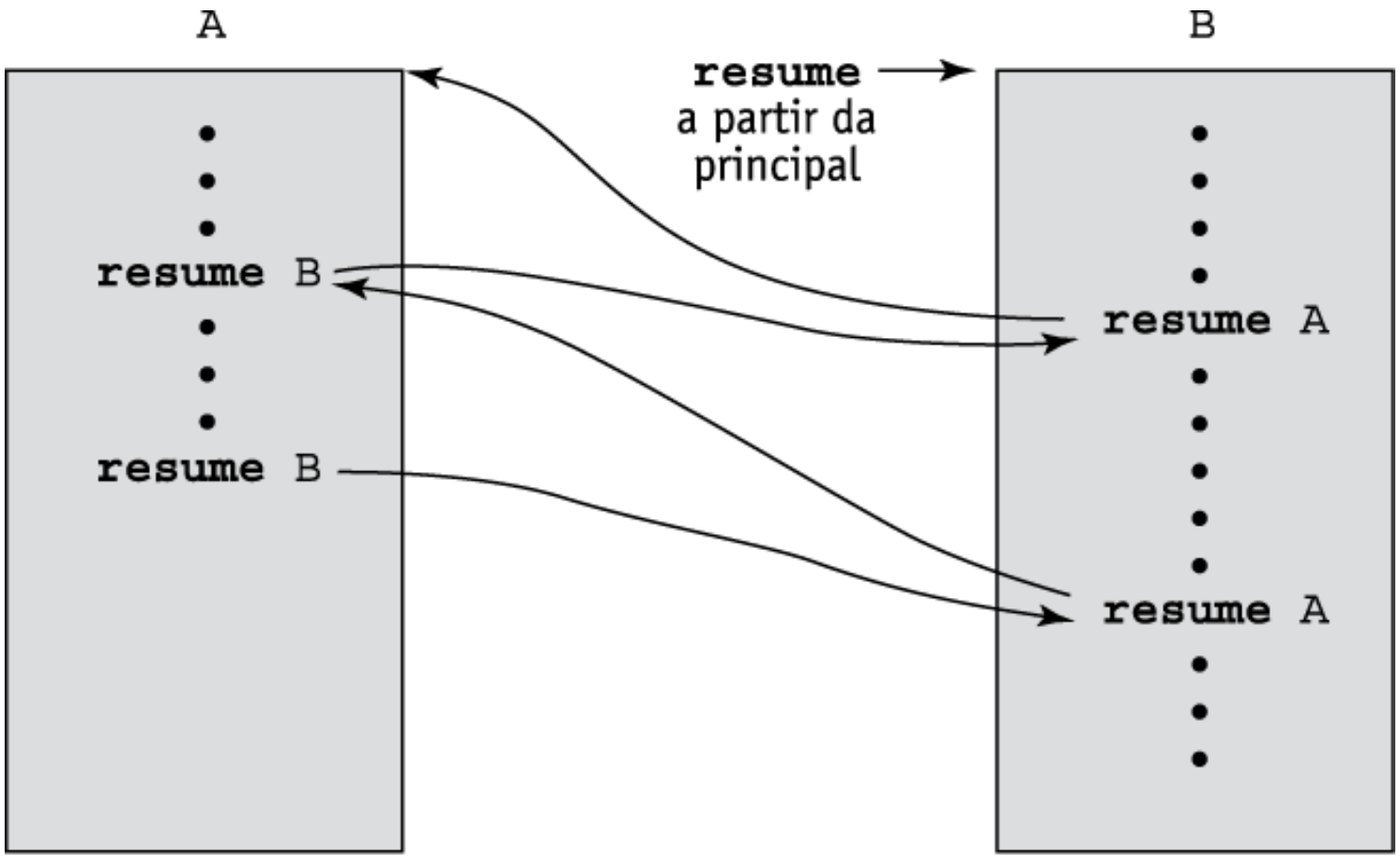
Sequências de controle de execução possíveis



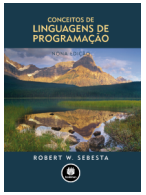
(a)



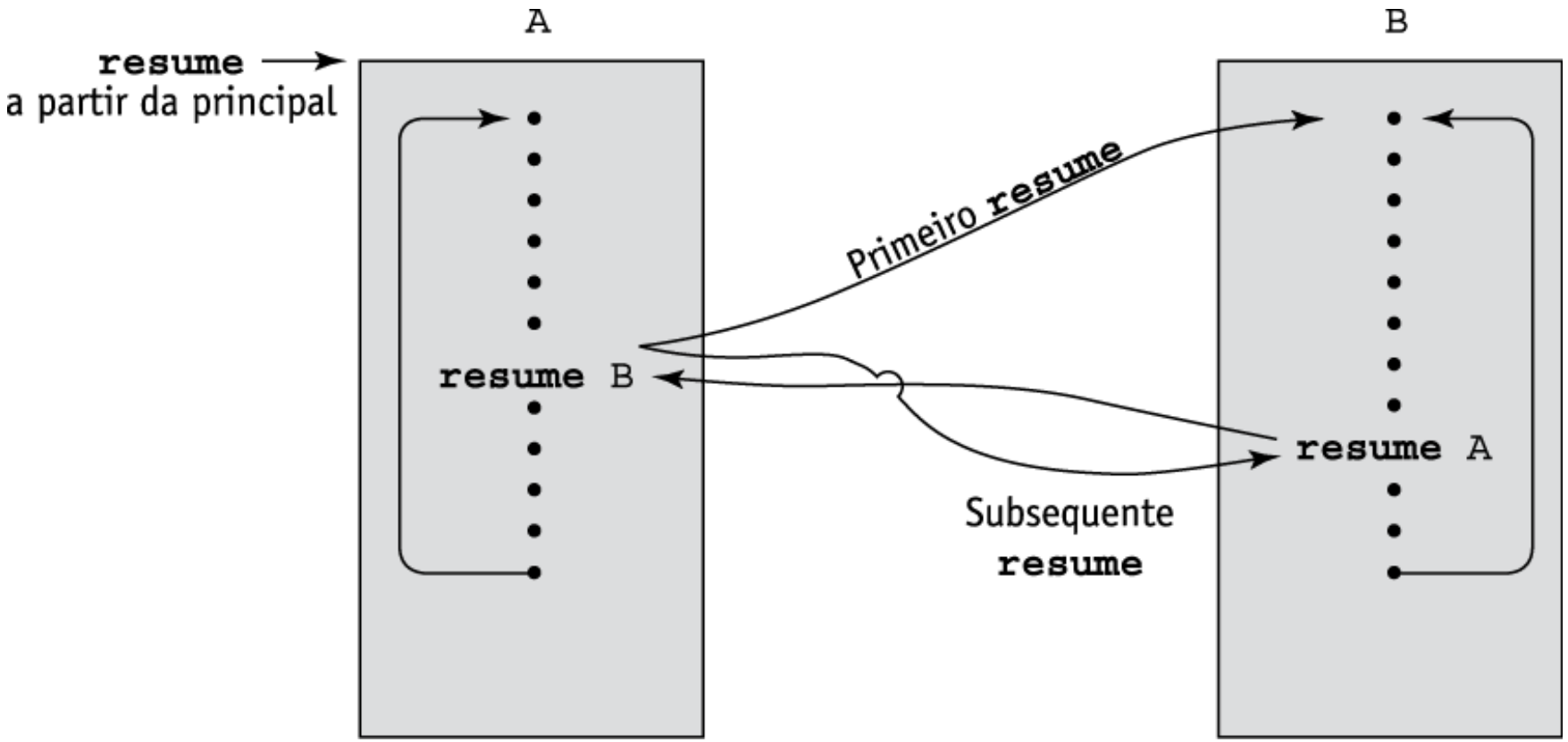
Sequências de controle de execução possíveis

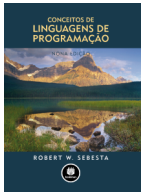


(b)



Sequência de execução de corrotinas com laços





Resumo

- Uma definição de subprograma descreve as ações representadas pelo subprograma
- Subprogramas podem ser funções ou procedimentos
- Variáveis locais em subprogramas podem ser dinâmicas da pilha ou estáticas
- Três modelos fundamentais de passagem de parâmetros: modo de entrada, modo de saída e modo de entrada e saída
- Algumas linguagens permitem sobrecarga de operatores
- Subprogramas podem ser genéricos
- Uma corrotina é um subprograma especial que tem múltiplas entradas