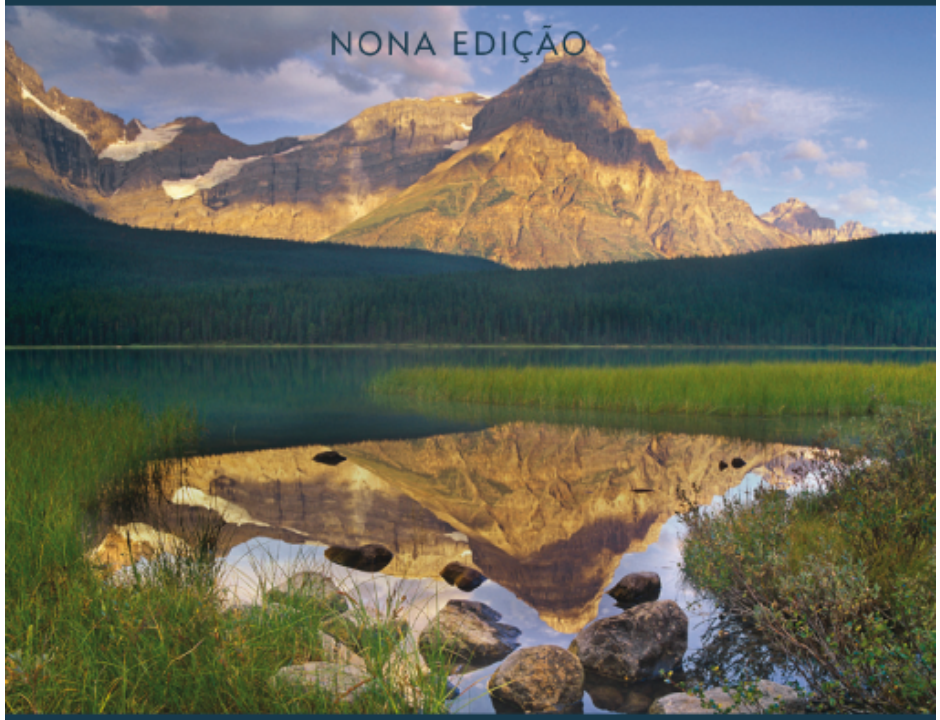
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

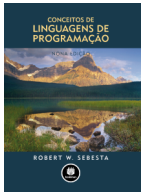


ROBERT W. SEBESTA

bookman

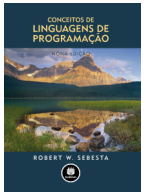
Capítulo 9

Subprogramas



Introdução

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



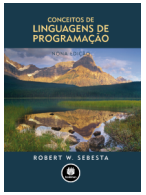
Fundamentos de subprogramas

- Um **subprograma** é um tipo de abstração de processo
  - Instruções para realizar uma tarefa são agrupadas e tratadas como uma unidade lógica
  - O conceito economiza espaço e esforço de desenvolvimento e codificação

- 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



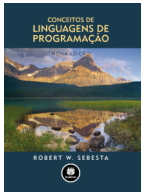
Definições básicas

- Uma *definição de subprograma* consiste na descrição de seu cabeçalho e de seu corpo
  - O cabeçalho de um subprograma inclui tipo do subprograma, nome, parâmetros formais, tipo de retorno
  - O corpo é a descrição das ações da abstração do subprograma

- Uma *chamada a subprograma* é a requisição explícita que diz que o subprograma deve ser executado

- Um parâmetro formal é uma variável que aparece no cabeçalho do subprograma e é usada no seu corpo

- Um parâmetro real representa um valor ou endereço usado na chamada do subprograma



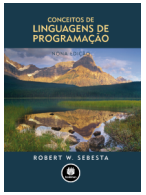
Definições básicas

•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 e, se for uma função, seu tipo de retorno

•Declarações de funções em C e C++ são chamadas de *protótipos*

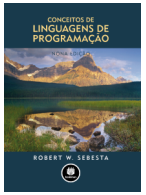
•Uma *declaração de subprograma* fornece o protocolo, mas não inclui seu corpo



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

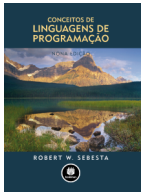


Valores padrão de parâmetros formais

- 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

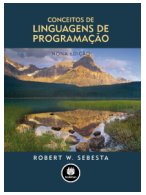
```
int calcular(int a, int b, int c = 1)
x = calcular(10,20);    //c receberá 1
```

- Número variável de parâmetros
  - 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



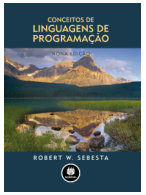
Procedimentos e funções

- Existem duas categorias de subprogramas
  - *Procedimentos* 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



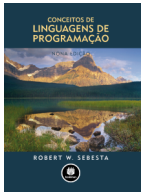
Questões de projeto para subprogramas

- 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?



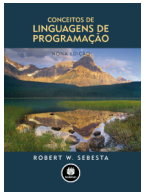
Ambientes de referenciamento local

- Variáveis locais (definidas dentro de um subprograma e, geralmente, tem o mesmo escopo do subprograma)
- Podem ser dinâmicas (o ambiente é criado de acordo com a ativação)
  - 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
  - Ex: COBOL, C (por default é dinâmica, mas pode usar o **static**

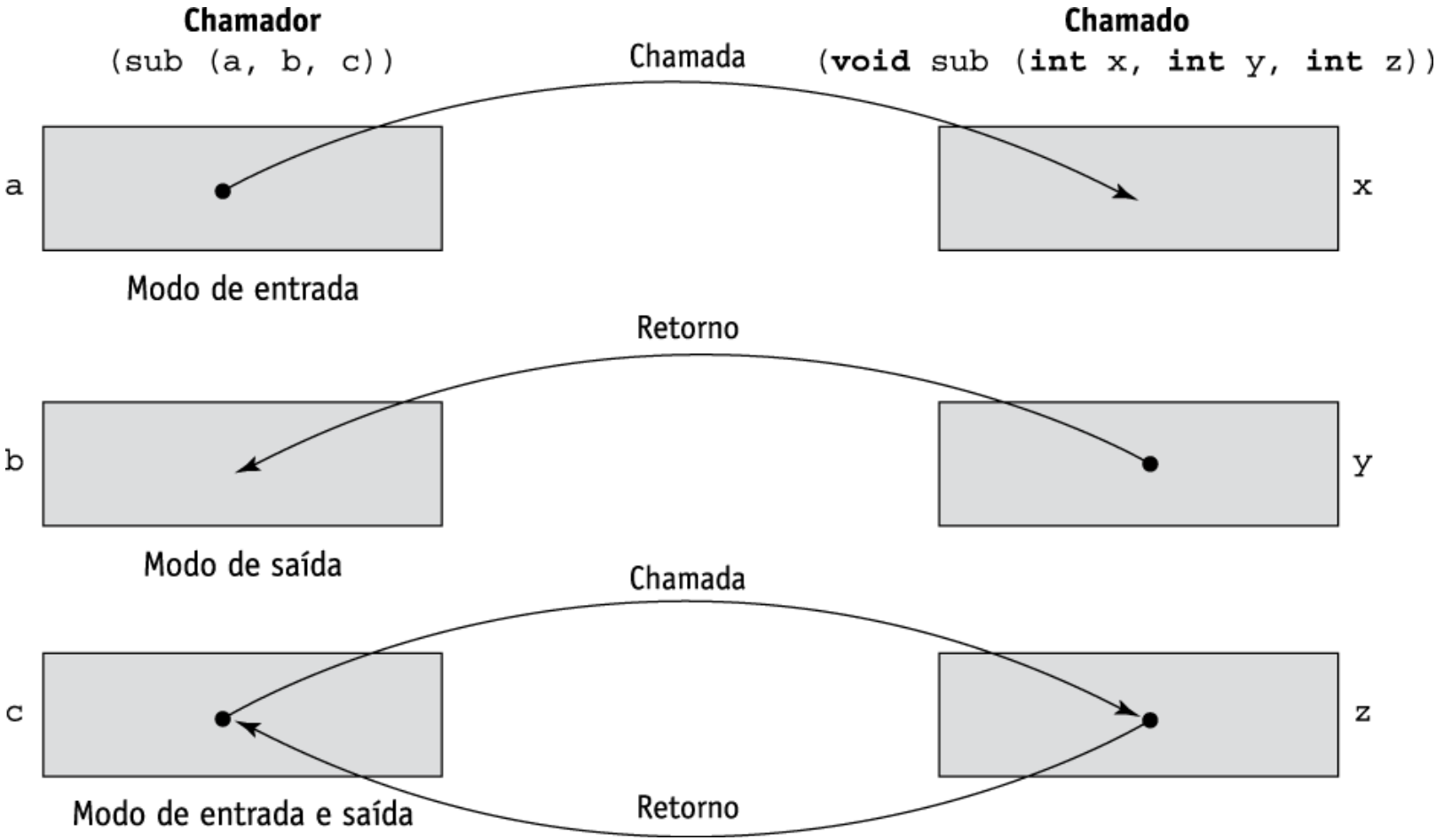


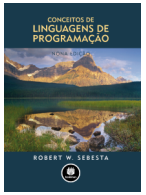
Modelos semânticos de passagem de parâmetros

- Formas de transmitir os parâmetros para os subprogramas:
- Modo de entrada: receber dados a partir do parâmetro real correspondente
- Modo de saída: transmitir dados para o parâmetro real
- Modo de entrada e saída: receber e transmitir



Modelos de passagem de parâmetros



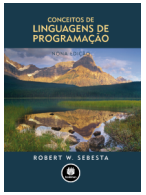


Modelos conceituais de transferência de dados

- Um valor real é copiado (movimento físico): para o chamador, para o chamado ou para ambos

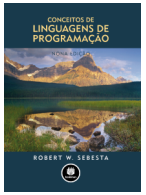
- Um caminho de acesso é transmitido (por um ponteiro ou por uma referência)

- Formas:
  - Passagem por valor (modo de entrada)
  - Passagem por resultado (modo de saída)
  - Passagem por valor/resultado (modo de entrada/saída)
  - Passagem por referência (mode de entrada/saída)
  - Passagem por nome (modo múltiplo)



Passagem por valor (modo de entrada)

- 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

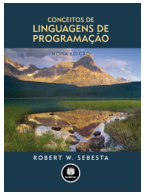


Passagem por resultado (modo de saída)

- Quando um parâmetro é passado por resultado (modo de saída), nenhum valor é transmitido para o subprograma

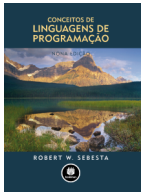
- O parâmetro formal correspondente age como uma variável local, mas logo antes de o controle ser transmitido de volta para o chamador

- Problema em potencial: `sub(p1, p1);` pode existir uma colisão entre parâmetros reais



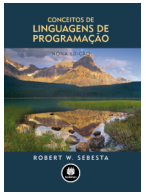
Passagem por valor-resultado(entrada/saída)

- Transferência de valores em ambas as direções
- Também conhecido como passagem por cópia
- Parâmetros formais têm armazenamento local
- Desvantagens:
  - As mesmas da passagem por resultado
  - As mesmas da passagem por valor



Passagem por referência

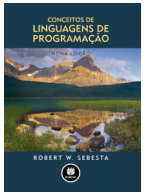
- Transmite um caminho de acesso
- Vantagem: processo de passagem é eficiente (não são necessárias cópias nem espaço duplicado) → parâmetro real é compartilhado
- Desvantagens
  - Acessos mais lentos (do que na passagem por valor)
  - Potenciais efeitos colaterais (colisões)
  - Apelidos podem ser criados



Passagem por nome (modo múltiplo)

- Parâmetro formal é substituído textualmente pelo real

- 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

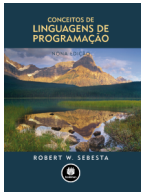


Implementando métodos de passagem de parâmetros

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

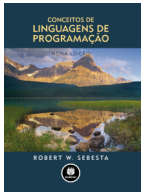
- Passagem por referência é a mais simples de implementar; apenas seu endereço deve ser colocado na pilha

- 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
- C#
  - Método padrão: passagem por valor
  - Passagem por referência é especificada precedendo um parâmetro formal e seu real correspondente com **ref**

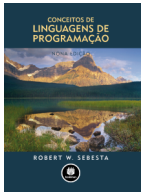


Métodos de passagem de parâmetros das principais linguagens (continuação)

- PHP: similar a C#

- Perl: parâmetros reais são implicitamente colocados em uma matriz pré-definida chamada de `@_`

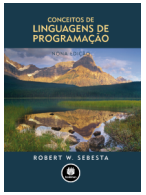
- Python e Ruby usam passagem por atribuição (todos os valores de dados são objetos)



Métodos de passagem de parâmetros (C)

- Transmissão ocorre somente por valor. Porém, transmitindo-se o endereço e o efeito é “semanticamente semelhante” à passagem por referência

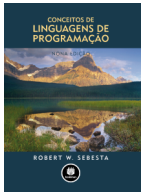
```
void swap(int *x, *y){
     int temp = *x;
     *x = *y;
     *y = temp;
}
swap(&a, &b);
```



Métodos de passagem de parâmetros (C++)

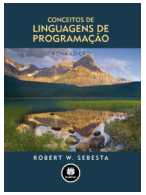
- Passagem de valor como em C, contudo, também permite passagem por referência explícita através do operador '&' no parâmetro formal

```
void swap(int &x, int &y){
      int temp = x;
      x = y;
      y = temp;
}
swap(a, b);
```



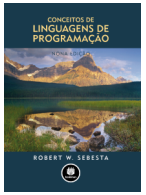
Verificação de tipos dos parâmetros

- 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

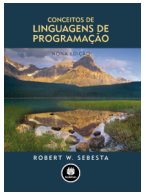
- Se uma **matriz** multidimensional é **passada** para um subprograma e o **subprograma** é **compilado** separadamente, o compilador precisa **saber** o **tamanho** declarado dessa matriz para construir a função de mapeamento de armazenamento



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

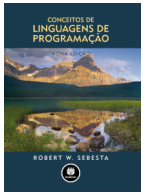
- A função pode avaliar a função de mapeamento de armazenamento escrita pelo usuário usando aritmética de ponteiros cada vez que um elemento da matriz precisar ser referenciado



Matrizes multidimensionais como parâmetros : Java e C#

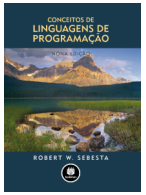
- Matrizes são objetos; elas são todas de uma única dimensão, mas os elementos podem ser matrizes

- 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



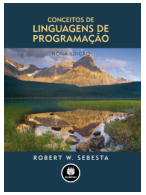
Considerações de projeto para passagens de parâmetros

- 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



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



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