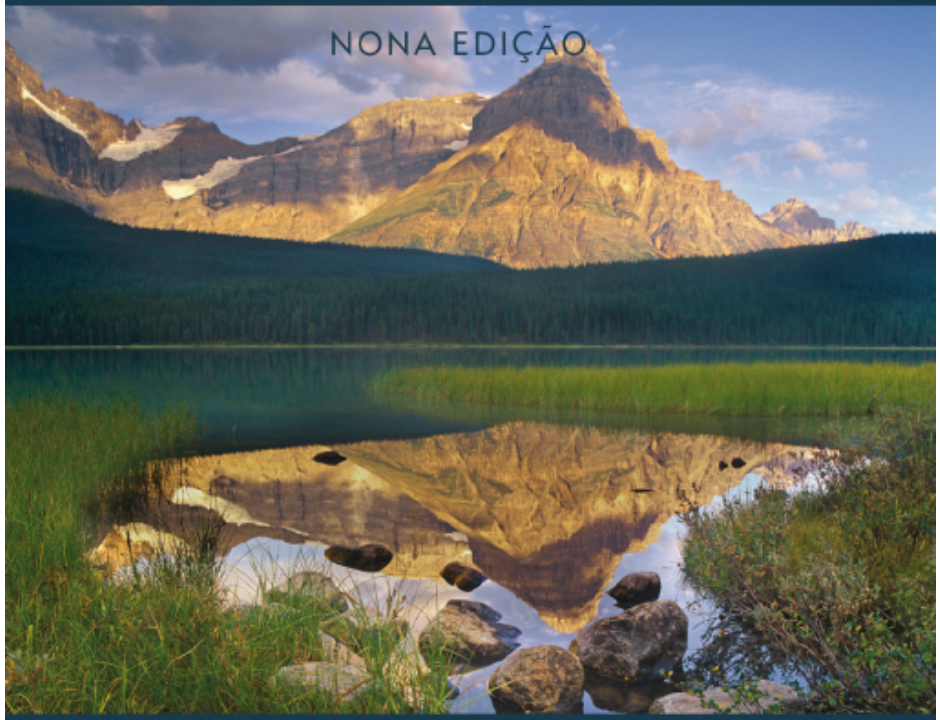
CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

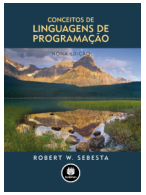


ROBERT W. SEBESTA

bookman

Capítulo 8

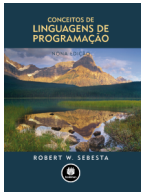
Estruturas de Controle no Nível de Sentença



Níveis de fluxo de controle

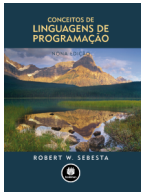
- Computações são realizadas por meio da avaliação de expressões e da atribuição dos valores a variáveis

- Para tornar a computação mais flexível e poderosa criou-se:
  - Formas de selecionar entre caminhos alternativos de fluxo de controle (execução da sentença)
  - Execução repetida de sentenças ou de sequência de sentenças chamada de controle de fluxo



Níveis de fluxo de controle

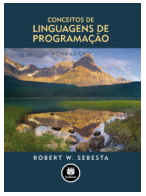
- O controle do fluxo em um programa ocorre em diversos níveis:
  - Dentro das expressões
    - Regras de associatividade
    - Regras de precedência de operadores
  - Entre unidades de programa
    - Subprogramas
  - Entre as sentenças



Sentenças de controle: Evolução

- Sentenças de controle em FORTRAN I foram baseados diretamente no hardware do IBM 704

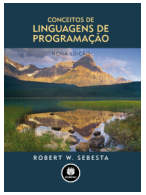
- Boa parte da pesquisa e da discussão foi devotada às sentenças de controle nos anos 1960
  - Um resultado importante: foi provado que todos os algoritmos que podem ser expressos por diagramas de fluxo podem ser codificados em uma linguagem de programação com apenas duas sentenças de controle
  - 1. Escolher dentre dois caminhos de fluxo de controle
  - 2. Controlar logicamente as iterações



Estrutura de controle

- Uma estrutura de controle é uma sentença de controle e a coleção de sentenças cuja execução ela controla

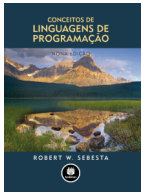
- Questão de projeto
  - A estrutura de controle deve ter múltiplas entradas?
    - Afetam a legibilidade
    - Ocorrem apenas em LP que incluem *goto* e rótulos (*labels*) de instruções



Sentenças de seleção

- Uma *sentença de seleção* fornece os meios para escolher entre dois ou mais caminhos de execução em um programa

- Duas categorias gerais:
  - Dois caminhos
  - Seleção múltipla, ou n caminhos

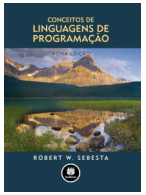


Dois caminhos

- Forma geral:

```
if expressão_de_controle
     cláusula então
     cláusula senão
```

- Questões de projeto:
  - Qual é a forma e o tipo da expressão que controla a seleção?
  - Como são especificadas as cláusulas **então** e **senão**?
  - Como o significado dos seletores aninhados deve ser especificado?

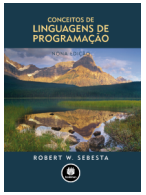


A expressão de controle

- Expressões de controle são especificadas entre parênteses se a palavra reservada **then** (ou algum outro marcador sintático) não for usada para introduzir a cláusula **então** Ex:
  - C `if (expressao) comando; else comando;`
  - Pascal `if expressao then comando else comando`

- Em C89, C99, Python e C++, a expressão de controle pode ser **aritmética**

- Em linguagens como Ada, Java, Ruby e C#, a expressão de controle deve ser **booleana**

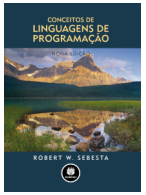


Forma da cláusula

- Em muitas linguagens contemporâneas, as cláusulas **então** e **senão** aparecem ou como sentenças **simples** ou como sentenças **compostas**

- Em Perl, todas as cláusulas **então** e **senão** devem ser sentenças **compostas**

- Em Fortran 95, Ada e Ruby, as cláusulas **então** e **senão** são sequências de sentenças



Forma da cláusula

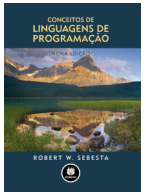
- As linguagens baseadas em C usam chaves para formar sentenças compostas

  – Simples:

```
x = 2;
```

  – Composta:

```
{
        i = 3;
        printf("%d\n", i);
        i++;
}
```

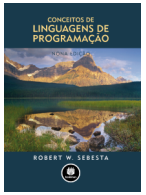


Forma da cláusula

- Python usa indentação para especificar sentenças compostas

```
if x > y :
   x = y
   print "case 1”
```

- Todas as instruções igualmente identadas pertencem à mesma sentença composta



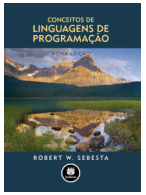
Aninhando seletores

- Exemplo em Java

```
if (sum == 0)
  if (count == 0)
      result = 0;
else result = 1;
```

- Para qual `if` a cláusula `else` está associada?

- Regra de semântica estática de Java:
  - `else` sempre casa com a `if` mais próxima



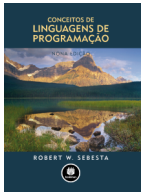
Aninhando seletores (continuação)

- Para forçar a semântica alternativa em Java, o if interno é colocado em uma sentença composta, como em:

```
if (sum == 0) {
 if (count == 0)
      result = 0;
}
else result = 1;
```

- A solução acima é usada em C, C++ e C#

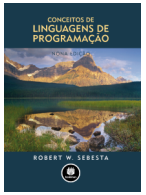
- Perl requer que todas as cláusulas **então** e **senão** sejam compostas



Aninhando seletores (continuação)

- Sequências de sentenças como cláusulas: **Ruby**

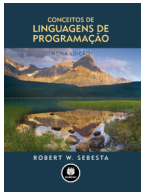
```
if sum == 0 then
    if count == 0 then
      result = 0
    end
else          //casa com o if mais externo
      result = 1
end
```



Aninhando seletores (continuação)

- Python

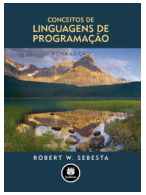
```
if sum == 0 :
   if count == 0 :
      result = 0
 else :
      result = 1
```



Construções de seleção múltipla

- Permite a seleção de uma dentre qualquer número de sentenças ou de grupos de sentenças

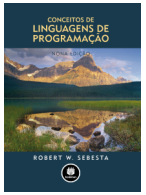
- Questões de projeto:
  1. Qual é a forma e o tipo da expressão que controla a seleção?
  2. Como são especificados os segmentos selecionáveis?
  3. O fluxo de execução por meio da estrutura pode incluir apenas um único segmento selecionável?
  4. Como os valores de cada caso são especificados?
  5. Como valores da expressão de seleção que não estão representados devem ser manipulados, se é que o devem?



Exemplos de seletores múltiplos

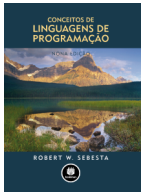
- C, C++ e Java

```
switch (index) {
        case 1:
            sentença_1;
            break;
        case 2:
            sentença_2;
            break;
        default:
            printf("Erro");
}
```



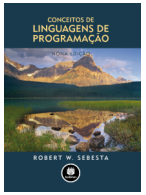
Exemplos de seletores múltiplos

- Escolhas de projeto para o switch de C
  1. A expressão de controle pode ser apenas do tipo inteiro
  2. As sentenças selecionáveis podem ser sequências de sentenças, sentenças compostas ou blocos
  3. Qualquer número de segmentos pode ser executado em uma execução da construção
  4. O segmento opcional default é usado para valores não representados (se o valor da expressão de controle não é representado e nenhum segmento padrão está presente, a construção não faz nada)



Exemplos de seletores múltiplos

- C#
  - Se difere do C ao ter uma regra de semântica estática que proíbe a execução implícita de mais de um segmento
  - Cada segmento selecionável deve terminar com uma sentença de desvio incondicional explícita (**`goto`** ou **`break`**)
  - Em C#, expressão de controle e as construções case podem ser cadeias

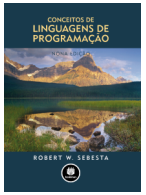


Seleção múltipla usando `if`

- Perl, Python e Lua não tem construções de seleção múltipla.

- Seletores múltiplos podem aparecer diretamente como extensões de seletores de dois caminhos, usando cláusulas else-if. Por exemplo, em Python:

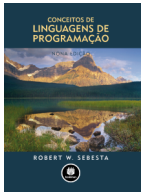
```
if count < 10 :
  bag1 = True
elif count < 100 :
  bag2 = True
elif count < 1000 :
  bag3 = True
```



Seleção múltipla usando `if`

- O exemplo de Python pode ser escrito como a sentença `case` de Ruby

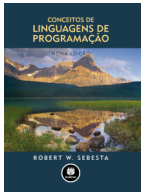
```
case
   when count < 10 then bag1 = true
   when count < 100 then bag2 = true
   when count < 1000 then bag3 = true
end
```



Sentenças de iteração

- **Sentenças** que fazem com que uma sentença ou uma coleção de sentenças seja executada zero, uma ou mais vezes. Uma construção de iteração é frequentemente **chamada de um laço**

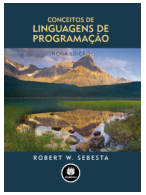
- Questões de projeto:
  1. Como a iteração é controlada?
  2. Onde o mecanismo de controle deve aparecer na construção de laço?



Laços controlados por contador

- Uma sentença de controle iterativa de contagem tem uma variável de laço, que inclui os valores inicial e final e o tamanho do passo

- Questões de projeto:
  1. Qual é o tipo e o escopo da variável de laço?
  2. Deve ser legal para a variável ou para os parâmetros de laço serem modificados nele, e, se isso for possível, essa mudança afeta o controle do laço?
  3. Os parâmetros de laço devem ser avaliados apenas uma vez ou uma vez para cada iteração?

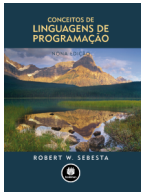


Exemplos de sentenças de iteração

- FORTRAN 95

```
 DO rótulo variável = inicial, final [, tamanho do passo]
do I = 0, 100, 5
      bloco de comandos;
```

- Tamanho do passo pode ser qualquer valor, menos zero
- Parâmetros podem ser expressões
- Questões de projeto:
  1. A variável de laço deve ser do tipo INTEGER
  2. A variável de laço não pode ser mudada no laço, mas os parâmetros podem; porque eles são avaliados apenas uma vez, isso não afeta o controle do laço
  3. Parâmetros do laço são avaliados apenas uma vez

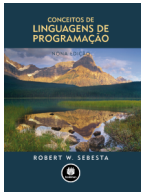


Exemplos de sentenças de iteração

- Ada

```
count : Float : 1.35;
for count in 1..10 loop
      sum := sum + count;
end loop
```

- A variável count não é afetada pelo laço for. Quando encerrado o laço, seu valor será 1.35



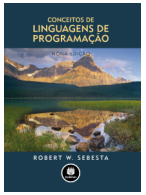
Exemplos de sentenças de iteração

- Linguagens baseadas em C

```
for ([expr_1] ; [expr_2] ; [expr_3]) statement
```

  - O corpo do laço pode ser uma única sentença, uma sentença composta ou uma sentença nula
  - O valor de uma expressão de sentenças múltiplas é o valor da última sentença na expressão
  - Se a segunda expressão está ausente, é um laço infinito

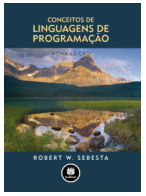
- Escolhas de projeto:
  - Tudo pode ser mudado no laço
  - A primeira expressão é avaliada uma vez, mas as outras duas são avaliadas com cada iteração



Exemplos de sentenças de iteração

- C++ se difere de C de duas maneiras:
  1. A expressão de controle pode ser booleana
  2. A primeira expressão pode incluir definições de variáveis (o escopo de uma variável definida na sentença for é a partir de sua definição até o final do corpo do laço)

- Java e C#
  – Diferem de C++ porque a expressão de controle de laço é restrita a valores booleanos

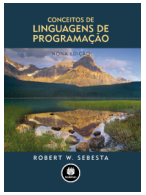


Exemplos de sentenças de iteração

- Python

```
for cont in [2, 4, 6]:
   print cont
```

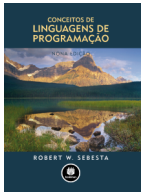
  - O objeto é frequentemente uma faixa, uma lista de valores em colchetes ([2, 4, 6]), ou uma chamada à função range:
    - range(5), que retorna 0, 1, 2, 3, 4
    - range(2, 7) retorna [2, 3, 4, 5, 6]
    - range(0, 8, 2) retorna [0, 2, 4, 6]



Sentenças de iteração: laços controlados logicamente

- Controle de repetição é baseado em uma expressão booleana

- Questões de projeto:
  - O controle deve ser de pré ou pós-teste?
  - O laço controlado logicamente deve ser uma forma especial de um laço de contagem ou uma sentença separada?



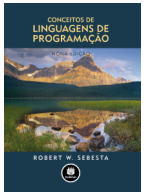
Sentenças de iteração: laços controlados logicamente

- C e C++ incluem tanto laços controlados logicamente com pré-teste quanto com pós-teste:

```
while (expressao){
     comandos;
}
```

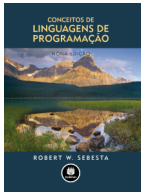
```
do{
  comandos;
}while (expressao)
```

- Java é semelhante a C e C++, exceto que a expressão de controle deve ser booleana



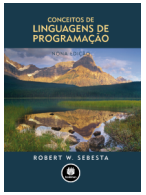
Sentenças de iteração: laços controlados logicamente

- Ada tem um laço lógico com pré-teste, mas nenhuma versão pós-teste
- FORTRAN 95 não tem um laço lógico, nem com pré-teste, nem com pós-teste
- Perl e Ruby têm dois laços lógicos com pré-teste: `while` e `until`. Perl também tem dois laços com pós-teste



Sentenças de iteração: Perl

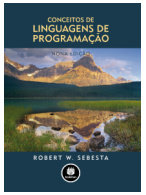
```
#!/usr/bin/perl
$i = 1;
print "primeiro a raiz quadrada de 1 até 10...\n\n";
while ($i <= 10)
{
      print "A raiz quadrada de ", $i," é", sqrt($i);
      $i++;
}
$i = 1;
print "e agora os quadrados de 1 até 10...\n\n";
unti l ($i > 10)
{
      print "O quadrado de ", $i, " é", $i * $i, "\n";
      $i++;
}
```



Mecanismos de controle de laços posicionados pelo usuário

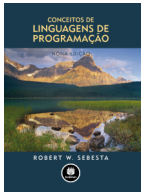
- Em algumas situações, é conveniente para um programador escolher uma posição para o controle do laço em vez do início ou o final do laço

- Questões de projeto
  1. O mecanismo condicional deve ser uma parte integral da saída?
  2. E possível sair apenas de um corpo de laço ou e possível sair também dos laços que o envolvem?



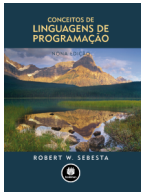
Mecanismos de controle de laços posicionados pelo usuário: `break` e `continue`

- C , C++, Python, Ruby e C# têm saídas não rotuladas incondicionais (**`break`**)
- Java e Perl têm saídas incondicionais rotuladas (**`break`** em Java, **`last`** em Perl)
- C, C++ e Python incluem uma sentença de controle não rotulada, **`continue`**, que transfere o controle para o mecanismo de controle do menor laço que o envolve
- Java e Perl têm sentenças similares ao **`continue`**



Exemplo em Java

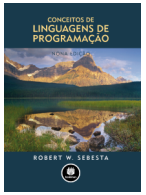
```
outerLoop:
    for (linha = 0, linha < numLinhas; linha++){
        for (col = 0; col < numCol; col++){
            soma += mat[linha][col];
            if (soma > 1000.0)
                break outerLoop;
        }
    }
```



Exemplo em C, C++

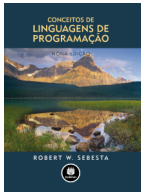
```
while (soma < 1000){
      getNext(valor);
      if (valor < 0) continue;
      soma += valor;
}

while (soma < 1000){
      getNext(valor);
      if (valor < 0) break;
      soma += valor;
}
```



Desvio incondicional

- Transfere o controle da execução para uma posição especificada no programa
- Mecanismo mais conhecido: `goto`
- Maior preocupação: legibilidade
- Algumas linguagens não têm suporte para `goto` (por exemplo, Java)
- C# oferece `goto` (pode ser usado em sentenças `switch`)
- Sentenças de saída de laços são restritas e camuflam sentenças `goto`

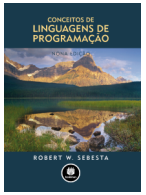


Comandos protegidos

- Sugeridos por Dijkstra
- Finalidade: fornecer sentenças de controle que suportariam uma metodologia de projeto de programas que garantisse a corretude durante o desenvolvimento

- Base para dois mecanismos linguísticos para programação concorrente (em CSP e Ada)

- Ideia básica: se a ordem de avaliação não é importante, o programa não deve especificar uma

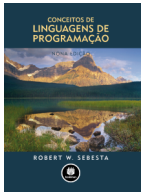


Seleção de comandos protegidos

- Forma

```
if <expressão booleana> -> <sentença>
[] <expressão booleana> -> <sentença>
 ...
[] <expressão booleana> -> <sentença>
fi
```

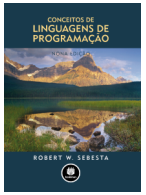
- Semântica: quando a construção é alcançada,
  - Avalia todas as expressões booleanas
  - Se mais de uma é verdadeira, escolhe uma de maneira não determinística
  - Se nenhuma é verdadeira, ocorre um erro em tempo de execução



Exemplo

```
if i = 0 → soma := soma + i
[] i > j → soma := soma + j
[] j < i → soma := soma + i
fi
```

- Se i = 0 e j > i então é escolhida, de forma não determinística, entre as sentenças de atribuições 1 e 3
- 2. Se i = j e não é zero, um erro em tempo de execução ocorre porque nenhuma das condições é igual a 0



Laço de comandos protegidos

- **Forma**

  do <expressão booleana> -> <sentença>

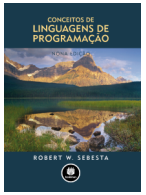
  [] < expressão booleana > -> <sentença>

  ...

  [] < expressão booleana > -> <sentença>

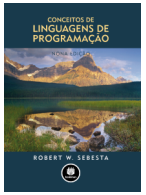
  od

- Semântica: para cada iteração
  - Avalia todas as expressões booleanas
  - Se mais de uma for verdadeira, uma é não deterministicamente escolhida para execução; depois são avaliadas novamente
  - Se nenhuma é verdadeira, o laço termina



Comandos protegidos

- A conexão entre sentenças de controle e verificação de programas é grande
- A verificação é impossível com sentenças `goto`
- A verificação é simplificada com apenas laços lógicos e seleções ou apenas comandos protegidos



Conclusões

- Variedade de estruturas no nível de sentença

- Programas escritos com apenas seleção e laços lógicos com pré-teste são geralmente menos naturais em sua estrutura, mais complexos e, dessa forma, mais difíceis de serem escritos e de serem lidos

- As estruturas de controle das linguagens de programação funcionais e programação lógica são todas bastante diferentes