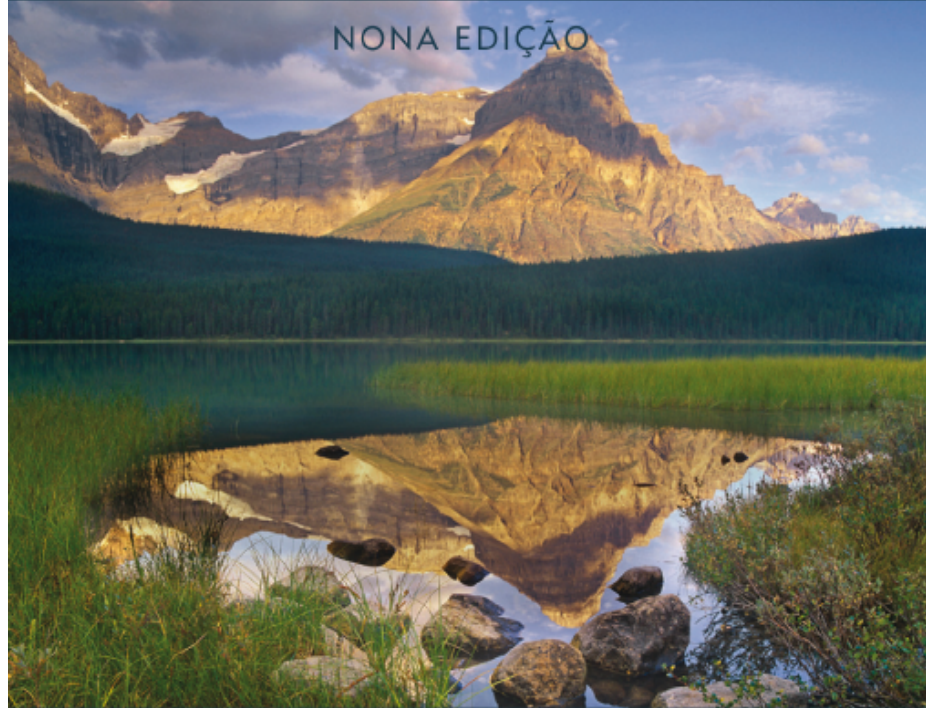


CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO

NONA EDIÇÃO

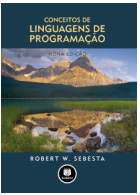


ROBERT W. SEBESTA



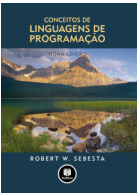
Capítulo 5

Nomes, Vinculações e Escopos



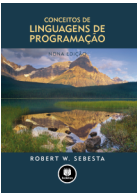
Tópicos do Capítulo 5

- Introdução
- Nomes
- Variáveis
- O conceito de vinculação
- Escopo
- Escopo e tempo de vida
- Ambientes de referenciamento
- Constantes nomeadas



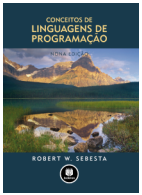
Introdução

- As linguagens de programação imperativas são abstrações da arquitetura de computadores subjacente de von Neumann
 - Memória
 - Processador
- Variáveis caracterizadas por atributos
 - Para projetar um tipo, deve-se considerar escopo, tempo de vida das variáveis, inicialização e compatibilidade



Nomes

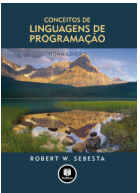
- Questões de projeto primárias para nomes:
 - Os nomes são sensíveis a capitalização?
 - As palavras especiais da linguagem são palavras reservadas ou palavras-chave?



Nomes

- **Formato**

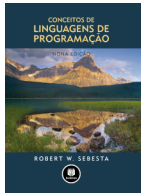
- Se for muito pequeno, não pode ser conotativo
- Exemplos:
 - FORTRAN 95: máximo de 31
 - C99: sem limitação, mas apenas os 63 são significativos; nomes externos são restritos a 31 caracteres
 - C#, Ada e Java: sem limite e todos os caracteres são significativos
 - C++: sem limite, mas os implementadores às vezes o impõem



Nomes

- Caracteres especiais

- PHP: todos os nomes de variáveis devem começar com cifrão (\$)
- Perl: todos os nomes de variáveis começam com caracteres especiais, que especificam o seu tipo
- Ruby: nomes de variáveis que começam com @ são variáveis de instância; as que começam com @@ são variáveis de classe



Nomes

- Sensibilidade à capitalização

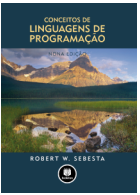
- Desvantagem: legibilidade (nomes que são parecidos, mas são diferentes)
 - Nomes em linguagens baseadas com C são sensíveis à capitalização
 - Nomes em outras não são
 - Em C++, Java e C#, o problema não pode ser evitado porque muitos dos nomes pré-definidos incluem tanto letras maiúsculas quanto minúsculas (por exemplo, `IndexOutOfBoundsException`)



Nomes

• Palavras especiais

- São usadas para tornar os programas mais legíveis ao nomearem as ações a serem realizadas
 - Uma *palavra-chave* é uma palavra especial apenas em alguns contextos – por exemplo, em Fortran
 - `Real VarName` (*Real é um tipo de dado acompanhado de um nome, portanto Real é uma palavra-chave*)
 - `Real = 3.4` (*Real é uma variável*)
- Uma palavra reservada é uma palavra especial que não pode ser usada como um nome
- Problema em potencial com as palavras reservadas: se houver muitas, o usuário tem dificuldade para inventar nomes (por exemplo, COBOL tem 300 palavras reservadas!)



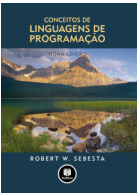
Variáveis

- Uma **variável** é uma abstração de uma célula de memória
- Uma variável pode ser caracterizada como um conjunto de seis atributos:
 - Nome
 - Endereço
 - Valor
 - Tipo
 - Tempo de vida
 - Escopo



Atributos de variáveis

- **Nome** – nem todas as variáveis têm
- **Endereço** – é o endereço de memória de máquina ao qual ela está associada
 - Chamado de **valor esquerdo** (*l-value*)
 - Uma variável pode ter diferentes endereços em diferentes momentos durante a execução
 - Uma variável pode ter diferentes endereços em diferentes lugares em um programa
 - Quando dois nomes de variáveis podem ser usados para acessar a mesma posição de memória, elas são chamadas de **apelidos** (*aliases*)
 - Apelidos são criados por ponteiros, variáveis de referência, tipos de união C e C++
 - Apelidos são um problema para a legibilidade (um leitor do programa deve sempre se lembrar de todos eles)



Atributos de variáveis

- **Tipo** - determina a **faixa de valores** que a variável pode armazenar e o **conjunto de operações** definidas para valores do tipo
- **Valor** - é o conteúdo da(s) célula(s) de memória associada(s) a ela
 - O lado esquerdo (l-value) de uma variável é seu endereço
 - O lado direito (r-value) de variável é seu valor
- **Célula abstrata de memória** – célula física ou coleção de células associadas a uma variável



O conceito de vinculação

- Uma *vinculação* é uma associação:
 - entre um atributo e uma entidade
 - entre uma operação e um símbolo

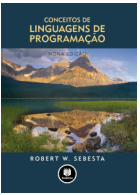
- *Tempo de vinculação* é o momento no qual uma vinculação ocorre

- Conceitos proeminentes na semântica das linguagens



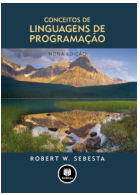
Possíveis tempos de vinculação

- **Tempo de projeto de linguagem** - vincula símbolos de operadores a operações. Ex: o * é ligado à operação de multiplicação
- **Tempo de implementação de linguagem** – vincula tipo de dados a uma faixa de valores
- **Tempo de compilação** – vincula uma variável a um tipo em C ou Java
- **Tempo de carga** – vincula uma variável a uma célula de memória
- **Tempo de ligação** – vincula variáveis de bibliotecas ou de subprogramas ao programa



Vinculação de atributos a variáveis

- Uma vinculação é **estática** se ocorre pela primeira vez antes do tempo de execução e permanece intocada ao longo da execução do programa
- Uma vinculação é **dinâmica** se ocorre pela primeira vez durante o tempo de execução e pode ser mudada ao longo do curso da execução do programa



Vinculação de tipos

- Como o tipo é especificado?
 - Declarações

- Quando a vinculação ocorre?
 - Se for estático, o tipo pode ser especificado por alguma forma de declaração explícita ou implícita



Declaração explícita/implícita

- Uma *declaração explícita* é uma sentença em um programa que lista nomes de variáveis e especifica que elas são de um certo tipo
- Uma *declaração implícita* é uma forma de associar variáveis a tipos por meio de convenções padronizadas, em vez de por sentenças de declaração (primeira aparição de um nome de variável em um programa)
- FORTRAN, BASIC e Perl têm declarações implícitas (Fortran tem explícita e implícita)
 - Vantagem: facilidade de escrita
 - Desvantagem: confiabilidade (menos problemas com Perl)



Vinculação de tipos dinâmica

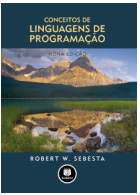
- Vinculação de tipos dinâmica (JavaScript e PHP)
- Especificada por meio de uma sentença de atribuição

por exemplo, JavaScript

```
list = [2, 4.33, 6, 8];
```

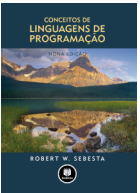
```
list = 17.3;
```

- Vantagem: flexibilidade (unidades de programa genéricas)
- Desvantagens:
 - Custo elevado
 - Detecção de erros de tipo pelo compilador é difícil



Atributos de variáveis

- **Inferência de tipos** (ML, Miranda e Haskell)
 - Em vez de por sentenças de atribuição, tipos são determinados (pelo compilador) a partir do contexto da referência (operações)
- **Vinculações de armazenamento e tempo de vida**
 - **Alocação** - obter uma célula de um conjunto de células disponíveis
 - **Liberação** – colocar a célula de volta no conjunto
- O **tempo de vida** de uma variável é o tempo durante o qual ela está vinculada a uma posição específica da memória



Categorias de variáveis por tempo de vida

- **Estáticas** - vinculadas a células de memória antes do início da execução de um programa e permanecem vinculadas a essas mesmas células de memória até que a execução do programa termine
 - **Vantagens**: eficiência (endereçamento direto), subprogramas sensíveis ao histórico (variáveis locais estáticas)
 - **Desvantagens**: redução da flexibilidade (sem recursão), sem compartilhamento entre variáveis



Categorias de variáveis por tempo de vida

- ***Dinâmicas da pilha*** - Vinculações de armazenamento são criadas quando **suas sentenças de declaração são elaboradas**, mas cujos tipos são estaticamente vinculados
(A declaração é elaborada quando a execução alcança o código com o qual a declaração está anexada)
- Se escalar, todos os atributos exceto o endereço são vinculados estaticamente
 - Variáveis locais em subprogramas C e métodos Java
- Vantagem: permite recursão; conserva o armazenamento
- Desvantagens:
 - Sobrecarga da alocação e liberação
 - Subprogramas não são sensíveis ao histórico
 - Referências ineficientes (endereçamento indireto)



Categorias de variáveis por tempo de vida

- ***Dinâmicas do monte explícitas*** - Alocadas e liberadas por instruções explícitas em tempo de execução
- Referenciadas apenas por ponteiros ou variáveis de referência, por exemplo, objetos dinâmicos em C++ (via `new` e `delete`), todos os objetos em Java
- **Vantagem:** prevê o gerenciamento de armazenamento dinâmico
- **Desvantagem:** ineficientes e não confiáveis

```
int *intnode;           //cria um ponteiro
intnode = new int;     //cria a variável dinâmica do monte
....
delete intnode;       //libera a variável para qual
                      intnode aponta
```



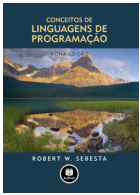
Categorias de variáveis por tempo de vida

- ***Dinâmicas do monte implícitas*** – Alocadas e liberadas por sentenças de atribuição
 - Todas as variáveis em APL; todas cadeias e matrizes em Perl, JavaScript e PHP
- **Vantagem:** flexibilidade (código genérico)
- **Desvantagens:**
 - Ineficiente, pois todos os atributos são dinâmicos
 - Sobrecarga em tempo de execução
 - Perda de detecção de erro



Atributos de variáveis: escopo

- O **escopo** de uma variável é a faixa de sentenças nas quais a variável é visível.
- As variáveis *não locais* de uma unidade ou de um bloco de programa são aquelas que estão visíveis, mas não são declaradas nessa unidade ou bloco
- As regras de escopo de uma linguagem determinam como uma ocorrência em particular de um nome é associada com uma variável



Blocos

- Um método de criar escopos estáticos em unidades de programa - do ALGOL 60
- Exemplo em C:

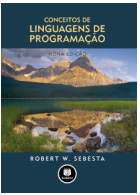
```
void sub() {  
    int count;  
    while (...) {  
        int count;  
        count++;  
        ...  
    }  
    ...  
}
```

– Note que o código é legal em C and C++, mas ilegal em Java e C#



Ordem de declaração

- C99, C++, Java e C# permitem que as declarações de variáveis apareçam em qualquer lugar que uma sentença poderia aparecer em uma unidade de programa
 - Em C99, C++ e Java, o escopo de todas as variáveis locais é de suas declarações até o final dos blocos
 - Em C#, o escopo de quaisquer variáveis declaradas em um bloco é o bloco inteiro, independentemente da posição da declaração
 - C# ainda requer que todas as variáveis sejam declaradas antes de serem usadas
- Em C++, Java e C#, variáveis podem ser declaradas em sentenças `for`
 - O escopo dessas variáveis é restrito à construção `for`



Escopo global

- C, C++, PHP e Python permitem uma estrutura de programa que é uma sequência de definição de funções, nas quais as definições de variáveis podem aparecer fora das funções
- C e C++ têm declarações e definições
 - Uma declaração de uma variável fora das definições de funções especifica que a variável é definida em um arquivo diferente



Escopo global (continuação)

- PHP

- Programas são embutidos em documentos XHTML
- O escopo de uma variável (implicitamente) declarada em uma função é local à função
- O escopo das variáveis globais se estende de suas declarações até o fim do programa, mas pulam sobre quaisquer definições de funções subsequentes

- Variáveis globais podem ser acessadas em uma função por meio do vetor

`$GLOBALS`

```

$dia = "Segunda";
$mes = "Janeiro";
function calendario(){
    $dia = "Domingo";
    global $mes;
    print "dia local é $dia <br />";
    $gdia = $GLOBALS['dia'];
    print "dia global é $gdia <br />";
    print "mês global é $mes <br />";
}
    
```



Escopo global (continuação)

- Python

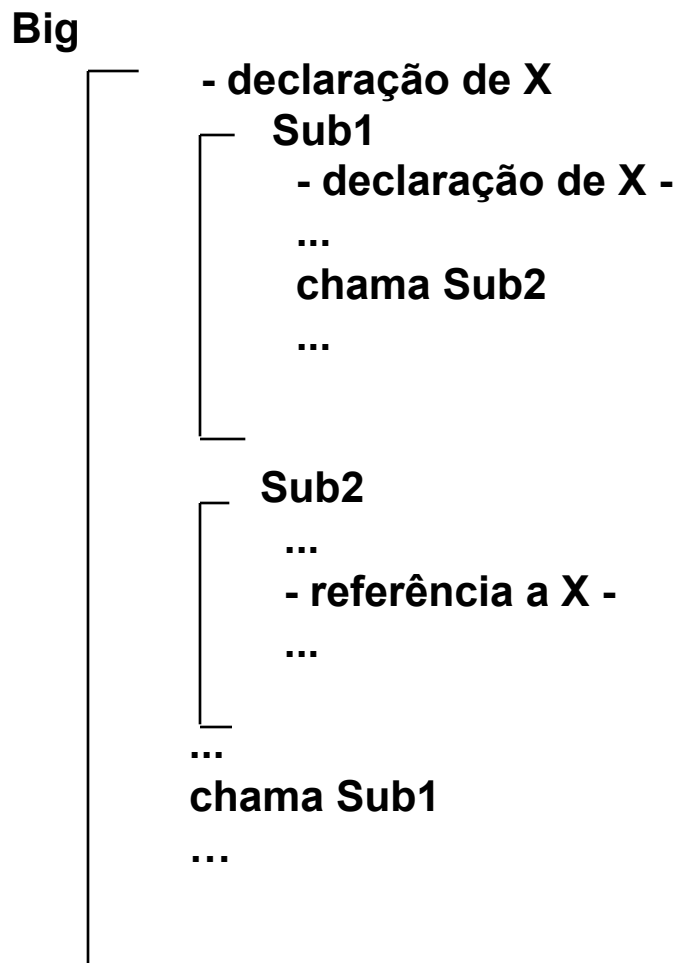
- Uma variável global pode ser referenciada em uma função, mas uma variável global pode ter valores atribuídos a ela apenas se ela tiver sido declarada como global na função

```
dia = "Segunda"
```

```
def teste():  
    print "O dia global é: ", dia  
    dia = "Domingo" ERRO  
    print "O novo valor de dia é: ", dia
```



Exemplo de escopo

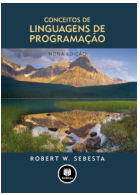


- **Big chama Sub1**
- **Sub1 chama Sub2**
- **Sub2 usa X**



Exemplo de escopo

- Escopo estático
 - Referência a X é ao X declarado em Big
- Escopo dinâmico
 - Referência a X é ao X declarado em Sub
- Avaliação do escopo dinâmico:
 - *Vantagem*: conveniência
 - *Desvantagens*:
 1. Enquanto um subprograma é executado, suas variáveis são visíveis aos subprograma que ele chama
 2. Impossibilidade de verificar os tipos das referências a não locais estaticamente
 3. Programas são mais difíceis de ler

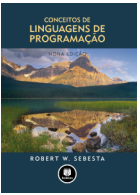


Escopo e tempo de vida

- Escopo e tempo de vida às vezes parecem ser relacionados, mas são conceitos **diferentes**
- Considere uma variável `static` em uma função C ou C++

```
void cabecalho(){
    ...
}

void processa(){
    int soma;
    ...
    cabecalho();
}
```

Ambientes de referenciamento

- O *ambiente de referenciamento* de uma sentença é a coleção de todas as variáveis visíveis na sentença
- Em uma linguagem de escopo estático, é composto pelas variáveis declaradas em seu escopo local mais a coleção de todas as variáveis de seus escopos ancestrais visíveis
- Um subprograma está **ativo** se sua execução começou, mas ainda não terminou
- Em uma linguagem de escopo dinâmico, é composto pelas variáveis declaradas localmente, mais as variáveis de todos os outros subprogramas que estão atualmente ativos



Ambientes de referenciamento - exemplo

```
void sub1(){
    int A, B;
    ... //ponto 1
}
```

Ponto 1 Ambiente de Referenciamento
A e B de sub1, C de sub2, D de main
(C de main e B de sub2 – ocultas)

```
void sub2(){
    int B, C;
    ... //ponto 2
    sub1();
}
```

2 B e C de sub2, D de main
(C de main – oculta)

```
void main(){
    int C, D;
    ... //ponto 3
    sub2();
}
```

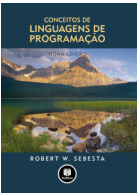
3 C e D de main



Constantes nomeadas

- Uma *constante nomeada* é uma variável que está vinculada a um valor apenas uma vez (Usadas para parametrizar programas)

```
void exemplo(){
    final int tam = 100;
    int[] intList = new int[tam];
    String[] strList = new String[tam];
    ...
    for (i = 0; i < tam; i++){
        ...
    }
    for (i = 0; i < tam; i++){
        ...
    }
    ...
    media = soma / tam;
    ...
}
```



Constantes nomeadas

- **Vantagens:** legibilidade e confiabilidade
- A vinculação de valores a constantes nomeadas podem ser estáticas (chamadas de *constantes de manifesto*) ou dinâmicas
- Linguagens:
 - FORTRAN 95: expressões constantes
 - Ada, C++ e Java: expressões de qualquer tipo
 - C# tem dois tipos, `readonly` e `const`
 - Os valores de `const` são vinculados em tempo de compilação
 - Os valores de `readonly` são dinamicamente vinculados