

# Nomes, vinculações e escopos

---

Marco A L Barbosa  
malbarbo.pro.br

Departamento de Informática  
Universidade Estadual de Maringá



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-CompartilhaIgual 4.0 Internacional.  
<http://github.com/malbarbo/na-lp-copl>

# Conteúdo

Introdução

Nomes

Variáveis

O conceito de vinculação

Escopo

Escopo e tempo de vida

Ambientes de referenciamento

Constantes nomeadas

# Introdução

- Linguagens imperativas são baseadas na arquitetura de von Neumann
- As variáveis são uma abstração das células de memória
  - Inteiros
  - Arranjo com 3 dimensões
- As variáveis tem um conjunto de propriedades: tipo, valor, memória, escopo, etc.
- Vamos estudar questões sobre a semântica das variáveis

**Nomes**

- **Nomes** ou identificadores são associados a rótulos, subprogramas, variáveis, etc
- Questões de projeto para os nomes
  - Os nomes são sensíveis a diferenciação entre minúsculas e maiúsculas?
  - As palavras especiais são palavras reservadas ou palavras chave?

- Tamanho
  - Limite pequeno para o tamanho do nome dificulta a leitura
  - Fortran 95: 31
  - C99: nomes internos, sem limites, mas apenas 63 são significativos
  - C99: nomes externos, 31
  - Ada: pelo menos 200
  - C++, Java, C#: sem limites

- Caracteres especiais
  - PHP: nomes de variáveis precisam iniciar com \$
  - Perl: inicia com um caractere especial que indica o tipo: \$ escalar, @ arranjo e % hash
  - Ruby: variáveis de instância iniciam com @ e variáveis de classe com @@
  - Avaliação?



- Sensível a diferenciação entre minúsculas e maiúsculas
  - Nomes nas linguagens baseadas em C são sensíveis ao caso
  - Algumas outras linguagens não são
  - Nomes pré-definidos (bibliotecas) em Java e C# incluem maiúsculas e minúsculas
  - Avaliação?
    - A diferenciação pode ser ruim pois nomes que parecem iguais são diferentes
    - A não diferenciação restringe o espaço de nomes (Em Java é comum construções do tipo  
`var pessoa = new Pessoa(...)`)

- Palavras especiais
  - Ajudam na legibilidade, são usadas para separar cláusulas

- Uma **palavra chave** é uma palavra que tem um significado especial em determinados contextos
  - `Integer apple` ; significado especial, tipo da variável
  - `Integer = 4`
  - Dificulta a legibilidade

- Uma **palavra reservada** é uma palavra que não pode ser usada como um nome
  - Muitas palavras reservadas podem restringir as opções de nomes do programador
  - Ex: Cobol tem 300 palavras reservadas

# Variáveis

- Abstração de célula(s) de memória
- Caracterizada como uma sêxtupla
  - Nome
  - Tipo
  - Endereço (*l-value*)
  - Valor (*r-value*)
  - Tempo de vida
  - Escopo

- Nome
  - Maioria das variáveis são nomeadas
  - Existem células de memória (variáveis) que não são nomeadas
- Endereço (*l-value*)
  - É o endereço da memória que a variável esta associada
  - Uma variável pode ter diferente endereços ao longo da execução do programa
  - Diversas variáveis podem ter o mesmo endereço (apelidos)

- Tipo
  - Determina o intervalo de valores e as operações
- Valor (*r-value*)
  - É o conteúdo da célula (abstrata) de memória associada com a variável
  - Célula física vs célula abstrata
- Tempo de vida e escopo são discutidos posteriormente



## **O conceito de vinculação**

# O conceito de vinculação

- Associação entre um atributo e uma entidade
- Momento da vinculação (quando a vinculação ocorre)
  - Projeto da linguagem
  - Implementação da linguagem
  - Tempo de compilação
  - Tempo de ligação
  - Tempo de carregamento
  - Tempo de execução

# O conceito de vinculação

Exemplo

```
int cont;  
...  
cont = cont + 1;  
printf("contagem: %d\n", cont);
```

Compreender quando as vinculações ocorrem é pré requisito para entender a semântica de uma linguagem de programação

- Vinculação estática
  - Ocorre antes da execução do programa e permanece inalterada
- Vinculação dinâmica
  - Ocorre ou é altera durante a execução do programa

# Vinculação estática de tipo

- Como o tipo é especificado?
  - Declaração explícita
    - `int x`
  - Declaração implícita (através de convenção)
    - Fortran: variáveis (que não foram explicitamente declaradas) cujo o nome começa com I, J, K, L, M, ou N são inteiras, caso contrário, reais
    - Perl: nome inicia com \$, @ ou %
    - Facilita a escrita, mas prejudica a confiabilidade (nem tanto no Perl)

## Vinculação dinâmica de tipo

- O tipo da variável não é especificada por uma declaração e nem é possível determinar o tipo pelo nome
- O tipo é vinculado quando um valor é atribuído a variável

```
# mudar o tipo de uma variável em um mesmo contexto  
# não é uma boa prática  
list = [10.2, 3.5]  
list = 47
```

# Vinculação dinâmica de tipo

- Vantagens

- Flexibilidade
- Expressividade
- Facilidade de escrita

```
def max(x, y):  
    return x if x >= y else y
```

- Desvantagens

- Menos confiável (a detecção de erros de tipo pelo compilador é reduzida)
- Custo (checagem dinâmica de tipo e interpretação)
- Dificuldade em criar um bom suporte nos editores

## Vinculação estática de tipo

- Inferência de tipo: os tipos são determinados pelo compilador a partir do contexto
- Exemplo ML:

```
fun vezes10(x) = 10 * x;
```

```
fun vezes10(x) = 10.0 * x;
```

```
fun quadrado(x) = x * x;
```

```
fun quadrado(x): int = x * x;
```

```
fun quadrado(x): real = x * x;
```



## Vinculação de memória e tempo de vida

- **Alocação:** tomar uma célula de memória de um conjunto de memória disponível
- **Desalocação:** devolver uma célula de memória desvinculada ao conjunto de memória disponível
- **Tempo de vida:** é o tempo durante o qual a variável está vinculada a uma célula de memória específica

- Classificação segundo tempo de vida de variáveis escalares
  - Estática
  - Dinâmica na pilha
  - Dinâmica explícita no heap
  - Dinâmica implícita no heap

# Vinculação de memória e tempo de vida

- Estática: vinculada a célula de memória antes da execução do programa e permanece vinculada a mesma célula durante a execução
  - Exemplos: Variáveis globais, variável `static` dentro de uma função em C
  - Vantagens: eficiência, sensibilidade a história
  - Desvantagens: a mesma memória não pode ser compartilhada por variáveis diferentes, redução da flexibilidade

- Dinâmica na pilha: vinculada a célula de memória quando a sua declaração é elaborada
  - Exemplos: Variáveis locais em Java
  - Vantagens: permite recursão e compartilhamento de memória
  - Desvantagens: Custo da alocação e desalocação, subprogramas não podem ser sensível ao contexto, acesso indireto

- Dinâmica explícita no heap: alocada explicitamente pelo programador, a alocação acontece em tempo de execução
  - Exemplos: `new` em C++, `malloc` em C
  - Vantagens: permite criação de estruturas de dados dinâmicas
  - Desvantagens: ineficiência e difícil utilização de ponteiros e referências

- Dinâmica implícita no heap: é vinculada a memória do heap apenas quando é atribuída
  - Exemplos: JavaScript, Python, etc
  - Vantagens: flexibilidade
  - Desvantagens: ineficiência, todos os atributos são dinâmicos

## Vinculação de memória e tempo de vida

```
#include<stdlib.h>

// estática
int max = 100;
int main() {
    // dinâmica na pilha
    int i = 0;
    while (i < max) {
        // dinâmica no heap
        int *x = malloc(sizeof(int));
        ...
    }
}
```

**Escopo**



- **Escopo** é o conjunto de expressões em que uma variável é visível
- Uma variável **não local** a um unidade de programa é uma variável não declarada nesta unidade mas acessível nela
- As regras de escopo de uma linguagens de programação determinam como uma ocorrência particular de um nome é associado a uma variável
  - Escopo estático
  - Escopo dinâmico

# Escopo estático

- Introduzido pelo Algol 60 como um método para associar nomes a variáveis não locais
- Também chamado de escopo léxico (baseado no texto do programa)
- Pode ser determinado em tempo de compilação
  - Um escopo estático que aninha outro escopo é chamado de **ancestral estático**
  - O ancestral estático mais perto é chamado de **pai estático**
  - Método de busca: busca a declaração, primeiro local, depois no pai estático, até que o nome seja encontrado
- Usado pela maioria das linguagens

## Escopo estático - Exemplo de Ada

```
procedure Big is
  X : Integer;
  procedure Sub1 is
    X : Integer;
  begin
    ... X ...
  end;
  procedure Sub2 is
  begin
    ... X ...
  end;
begin
  ...
end;
```

- A referência a variável X no corpo de Sub2 se refere a que variável? X de Big.
- A referência a variável X no corpo de Sub1 se refere a que variável? X de Sub1.
- Como a variável X de Big pode ser referenciada em Sub1? Big.X

- Variáveis podem ser ocultas em um escopo se houver uma declaração “mais” perto com o mesmo nome
- Algumas linguagens fornecem um mecanismo para acessar estas variáveis ocultas. Exemplo anterior do Ada

## Escopo estático

- Bloco é uma forma de criar escopo estático dentro de uma unidade de programa.
- Exemplo em C (não permitido em Java e C# por usar o mesmo nome count)

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

- Escopo global (variáveis declaradas fora de funções)
  - PHP: através da variável `$GLOBALS` ou declarado o nome `global`
  - Python: variável global pode ser referenciada, mas para receber uma atribuição o nome tem que ser declarado `global`

- Avaliação
  - Funciona bem em muitas situações
  - Permite mais acesso do que é necessário
  - Induz a criação de variáveis globais

- Baseia-se na sequência de chamadas a subprogramas e não na estrutura textual (temporal vs espacial)
- Determinado em tempo de execução
- As referências as variáveis são conectadas as declarações fazendo uma busca através da sequência de chamadas de subprogramas
- Exemplos de linguagens: APL, SNOBOL4, primeiras versões de Lisp, Perl, Common Lisp



## Escopo dinâmico - Exemplo

```
procedure Big is
  X : Integer;
  procedure Sub1 is
    X : Integer;
  begin
    ... X ...
  end;
  procedure Sub2 is
    begin
      ... X ...
    end;
begin
  ...
end;
```

- A referência a variável X no corpo de Sub2 se refere a que variável? Depende da sequência de chamadas de funções.
- Big chama Sub1 que chama Sub2? X de Sub1
- Big chama Sub2? X de Big.

- Avaliação
  - Quando um subprograma está em execução, suas variáveis são visíveis para todos os subprogramas chamados por ele
  - Dificuldade de leitura
  - Custo de acesso
  - Impossível fazer checagem do tipo das variáveis

**Escopo e tempo de vida**

## Escopo e tempo de vida

- Escopo e tempo de vida estão relacionados, mas são conceitos diferentes
  - Escopo (estático) é espacial
  - Tempo de vida é temporal
- Qual é o tempo de vida e o escopo de um variável local static em C/C++?

## Escopo e tempo de vida

- No exemplo a seguir, qual é o tempo de vida e o escopo da variável soma?

```
def imprime_cabecalho():
```

```
    ...
```

```
def calcula(a, b):
```

```
    soma = a + b
```

```
    imprime_cabecalho()
```

```
    ...
```

# **Ambientes de referenciamento**

## Ambientes de referenciamento

- É o conjunto de variáveis visíveis em uma determinada sentença
- Nas linguagens com escopo estático, são as variáveis locais mais todas as variáveis visíveis de todos os escopos externos
- Nas linguagens com escopo dinâmico, são as variáveis locais mais todas as variáveis visíveis em todos os subprogramas ativos

# Ambientes de referenciamento

```
procedure Exemplo is
  A, B : Integer;
  procedure Sub1 is
    X, Y : Integer;
  begin
    ... (1)
  end
  procedure Sub2 is
    X : Integer;
    procedure Sub3 is
      X : Integer;
    begin
      ... (2)
    end
  begin
    ... (3)
  end
begin
  ... (4)
end
```

- Considerando escopo estático, quais os ambientes de referenciamento nos pontos 1, 2 e 3?
- 1: X e Y de Sub1, A e B de Exemplo
- 2: X de Sub3, A e B de Exemplo
- 3: X de Sub2, A e B de Exemplo



# Ambientes de referenciamento

```
void sub1() {  
    int a, b;  
    ... (1)  
}  
void sub2() {  
    int b, c;  
    ... (2)  
    sub1();  
}  
void main() {  
    int c, d;  
    ... (3)  
    sub2();  
}
```

- Considerando escopo dinâmico, e a sequência de chamada main, sub2, sub1, quais os ambientes de referenciamento nos pontos 1, 2 e 3?
- 1: a e b de sub1, c de sub2, d de main
- 2: b e c de sub2, d de main
- 3: c e d de main

## **Constantes nomeadas**

# Constantes nomeadas

- Uma **constante nomeada** é uma variável que o valor é vinculado apenas uma vez
- Usado para parametrizar programas
- A vinculação dos valores a constantes nomeadas pode ser estáticas ou dinâmicas
- Vantagens: legibilidade e facilidade de modificação

- Robert Sebesta, Concepts of programming languages, 9<sup>a</sup> edição. Capítulo 5.