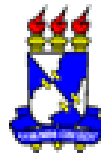


Vínculos

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



**Departamento de Computação
Universidade Federal de Sergipe**

Roteiro

- Contextos e Vínculos (Bindings)
- Escopo
- Declarações



Contextos e Vínculos

- Uma declaração produz um Vínculo
 - Associação entre um identificador e a entidade (constante, variável, função, procedimento, tipo, etc.) que ele irá representar
 - ex: `const n = 200`
 - vínculo: $n \rightarrow 200$ (n representa o inteiro 200)
- Contexto = conjunto de vínculos
 - Toda expressão ou comando é interpretada dentro de um contexto
 - Todas as variáveis na expressão devem ter vínculos no contexto



Contextos e Vínculos

Pascal

```
program exemplo;  
  const z = 0;  
  var c: char;  
  procedure q;  
    const c = 5.0e6;  
    var b : boolean;  
    begin  
      ... A  
    end;  
  begin  
    ... B  
  end.
```

- O contexto em **B**
{ z → constante 0,
 c → variável caractere,
 q → procedimento }
- O contexto em **A**
{ z → constante 0,
 c → número 5000000.0,
 b → variável booleana,
 q → procedimento }



Escopo

- Escopo de uma declaração:
 - Porção do código onde a declaração é efetiva
- Conceito de Bloco
 - Porção do programa que delimita o escopo das declarações que ele contém
- Pascal: só inclui blocos baseados em procedimentos e funções
- C++ e Java: definições de classes são blocos
- Ada: pode-se definir um bloco em qualquer lugar

ADA

declare

Temp: integer;

begin

temp:=1;

....

end;

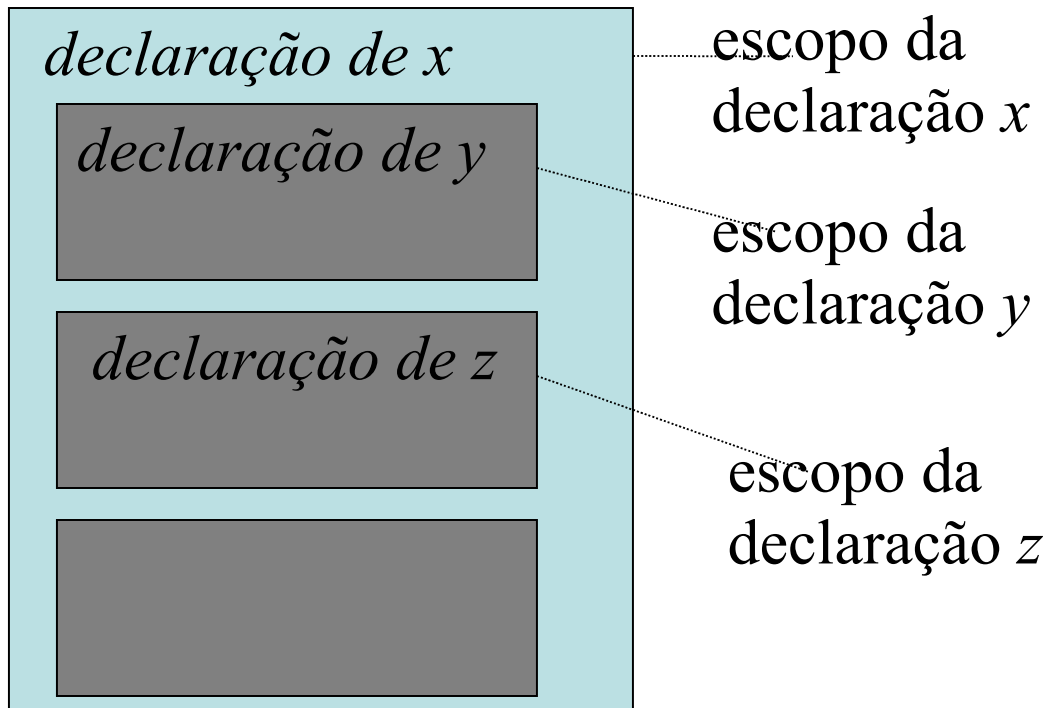


Escopo: estruturas de Bloco

Estrutura Monolítica
(ex: Cobol antigo)

declaração de x

Estrutura Plana
(ex: Fortran)



escopo da
declaração x

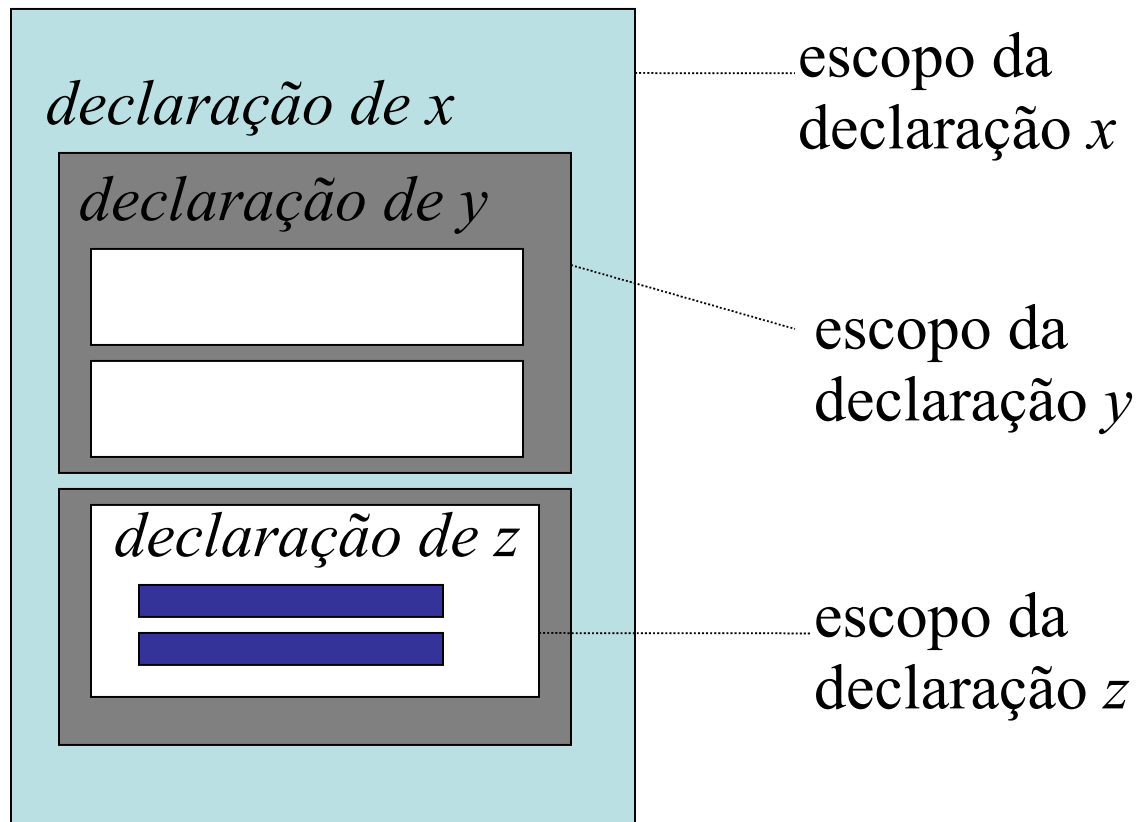
escopo da
declaração y

escopo da
declaração z



Escopo: estruturas de Bloco

Estrutura Aninhada
(ex: Algol e Pascal)



Escopo e Visibilidade

- Identificadores aparecem em dois contextos diferentes :
 - "Ocorrência de Vínculo"
 - `const x=15` {onde o vínculo é estabelecido}
 - "Ocorrência Aplicada"
 - ... `x*(x+10)` ... {onde o vínculo representa o inteiro 15}
- Um mesmo identificador pode ser declarado em diferentes blocos, representando diferentes entidades



Escopo e visibilidade

- A estrutura de blocos aninhados produz “buracos” na visibilidade de identificadores, isto é, trechos em que o identificador perde a visibilidade

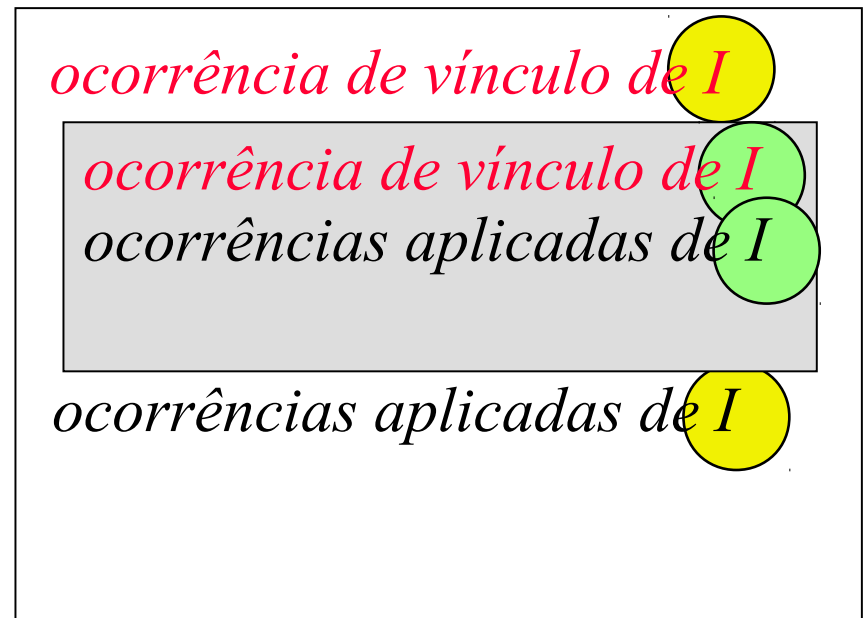
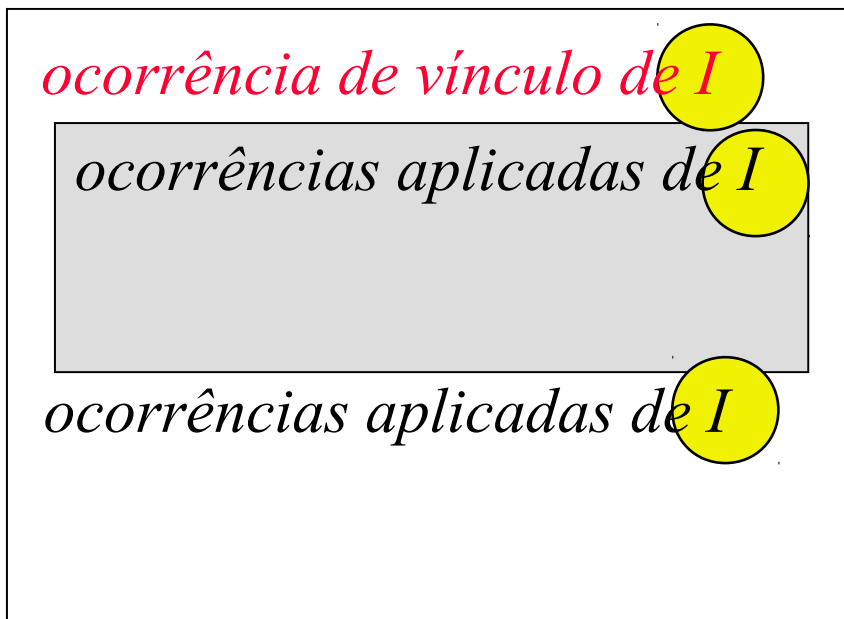
```
void sub() {  
    int count;  
    while (...) {  
        int count;  
        count++;  
        ...  
    }  
    ...  
}
```

trecho válido em C e C++, mas
não em Java



Escopo e Visibilidade

- De um modo geral que acontece se um identificador é declarado em dois blocos aninhados?



Vínculo Estático vs Dinâmico

```
...
const s = 2;
function reescala (d:Integer):Integer;
begin
    reescala := d*s
end;
procedure qualquer;
const s = 3;
begin
2 ... reescala(h)...
end
begin
1 ... reescala(h) ...
end
...
```

- Qual o resultado de reescala(3) em **1** ????
- e em **2** ????
- Depende de como a ocorrência de s é encarada no corpo da function "reescala".



Vínculo Estático vs. Dinâmico

- 1ª. Interpretação: Vínculo Estático (VE)
 - o corpo da função é avaliado no contexto da definição da função
 - "s representa 2" e então será retornado o valor inteiro 6 (independentemente de 1 ou 2)
- 2ª. interpretação: Vínculo Dinâmico (VD)
 - o corpo da função é avaliado no contexto da chamada de função
 - "s representa 2" em 1: retorna 6
 - "s representa 3" em 2: retorna 9



Vínculo Estático vs. Dinâmico

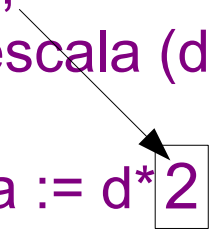
- Terminologia de VE e VD é baseada
 - Em quando podemos determinar qual "ocorrendia de vínculo" de um I corresponde a uma "ocorrendia de aplicação" de I
 - Tempo de compilação X Tempo de Execução



Vínculo Estático vs. Dinâmico

- Consideração 1
 - VE: "s" do corpo de "reescala" pode ser substituído por 2

```
...
const s = 2;
function reescala (d:Integer):Integer;
begin
    reescala := d*2
end;
procedure qualquer;
const s = 3;
begin
2    ... reescala(h)...
end
begin
1    ... reescala(h) ...
end
...
```



Vínculo Estático vs. Dinâmico

- Consideração 2
 - Vínculo Dinâmico e Checagem de Tipo Estática: não faz sentido !!! Por quê?
 - Suponha declaração global "**const** s = 0.5" ao invés de "**const** s = 2"

Checagem Estática teria que “prever” qual chamada seria feita em tempo de execução para decidir se daria erro ou não

```
...  
const s = 0.5;  
function reescala (d:Integer):Integer;  
begin  
    reescala := d*s  
end;  
procedure qualquer;  
const s = 3;  
begin  
2    ... reescala(h)...  
    end  
begin  
1    ... reescala(h) ...  
    end  
...
```

→ ã seria afetada

→ erro na checagem de tipo



Declarações

- Declaração = uma "construção" de um programa a partir da qual os vínculos são definidos
- Alguns tipos de declaração:
 - Definições
 - Declarações de Tipo
 - Declarações de Variável
 - Declarações Seqüenciais
 - Declarações Colaterais
 - Declarações Recursivas



Definições

- Único efeito é produzir vínculo
- Exemplos
 - Pascal:
 - Declarações de constantes (Ex: `const I = 10;`)
 - Definições de funções e procedimentos
 - C++ e Java permitem vinculação dinâmica de constantes ?
 - Verifiquem!



Declarações de Tipo

- Só serve para vincular um identificador a um tipo existente
 - Só em lings. com "equivalência de tipos estrutural"
 - Ex: declarações type de Haskell
- Não existe em Pascal: criação de novo tipo

Pascal

```
type Pessoa = record
```

Tipo anônimo

```
  nome : array[1..20] of Char;
```

```
  idade : Integer;
```

```
end
```

tipos incompatíveis

```
type Titulo = array [1..20] of Char;
```



Declarações de Tipos

C

```
struct data {  
    int d, m, a;  
};
```

```
union angulo {  
    int graus;  
    float rad;  
};
```

```
enum dia_util {  
    seg, ter, qua,  
    qui, sex  
};
```

Definições de tipos

C

```
struct data;  
typedef union angulo curvatura;  
typedef struct data aniversario;
```

Declarações de tipos



Declarações de variáveis

- Declaração de Variável: cria uma variável e produz um vínculo
- Definição de Variável (aliasing): vincula um identificador a uma variável JÁ EXISTENTE
 - Ex: C++ (*reference*), Ada (*renames*), Pascal (*absolute*)

ADA var a:integer; {declaração}
 b:integer renames a; {definição}

Pascal var a:integer; {declaração}
 b:integer absolute a; {definição}



Declarações de Variáveis

- Declarações de Variáveis em C

```
int k;  
union angulo ang;  
struct data d;  
int *p, i, j, k, v[10];
```

C

- Declarações com Inicialização

```
int i = 0;  
char virgula = ',';  
float f, g = 3.59;  
int j, k, l = 0, m=23;
```

C



Declarações de Variáveis

- Declarações com Inicialização Dinâmica

```
void f(int x) {  
    int i;  
    int j = 3;  
    i = x + 2;  
    int k = i * j * x;  
}
```

C

- Declarações com Inicialização em Variáveis Compostas

```
int v[3] = { 1, 2, 3 };
```

C



Declarações Seqüenciais

- Declarações seqüenciais na forma *D1; D2; etc...*
 - Vínculos criados em *D1* possam ser usados em *D2*

```
var c:integer;  
procedure ad1; Pascal  
begin  
    c:=c+1;  
end
```



Declarações Colaterais

- Devem ser independentes entre si
- Em ML:
 - A forma $\langle D1 \text{ and } D2 \rangle$ declara D1 e D2 independentemente e depois combina os vínculos produzidos

val pi = 3.14.159;

ML and sin = fn (x:real) =>
and cos = fn (x:real) =>

seria ilegal adicionar:

and tan = fn (x:real) => sin(x) / cons(x)



Declarações Recursivas

- Usa todos os vínculos que ela mesmo produz
- Ex Pascal:
 - Definição de constantes e declaração de variáveis não são recursivos
 - Seqüência de definição de procedimentos e funções é recursiva
 - Procedimentos e funções mutualmente recursivos com **forward**

Pascal

```
procedure a; forward;  
procedure b;  
    begin a; end;  
procedure a;  
    begin ... end;
```



Declarações recursivas

C

```
float potencia (float x, int n) {  
    if (n == 0) {  
        return 1.0;  
    } else if (n < 0) {  
        return 1.0/ potencia (x, - n);  
    } else {  
        return x * potencia (x, n - 1);  
    }  
}
```



Sugestões de Leitura

- Concepts of Programming Languages (Robert Sebesta)
 - Seções 5.4 e 5.8
- Programming Language Concepts and Paradigms (David Watt)
 - Seções 4.1 até 4.4
- Linguagens de Programação (Flávio Varejão)
 - Capítulo 2

