



Linguagens de Programação

Aula 6

Celso Olivete Júnior

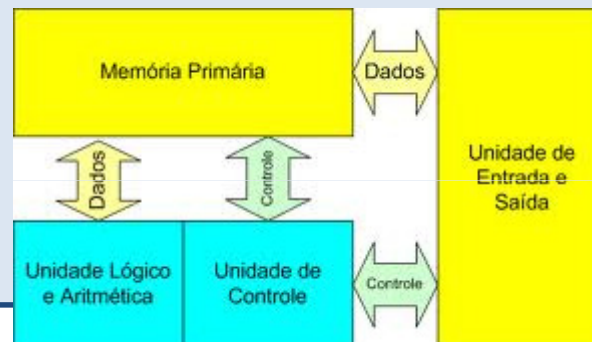
olivete@fct.unesp.br

Nomes, vinculações e escopos

Cap. 5 - Sebesta

Introdução

- ❑ LP's imperativas são abstrações da arquitetura de Von Neumann
 - Memória: armazena instruções e dados
 - Processador: fornece informações para modificar o conteúdo da memória.
 - ✓ As abstrações para as células de memória da máquina são as **variáveis**



Nomes

❑ Questões a considerar:

1. Os nomes são sensíveis à capitalização (*case sensitive*)?

Altura **ALTURA** altura

2. As palavras especiais da linguagem são palavras reservadas ou palavras-chave?

➤ Palavras especiais nomeiam ações → **while do repeat** e são separadores de unidades sintáticas: **BEGIN END**



Nomes

1. formato de nomes

- ❑ **Identificador** ou **nome** é uma cadeia de caracteres usada para identificar alguma entidade em um programa
 - ❑ Devem iniciar com **letras** (a..z) ou **_** (*underline*) e seguidos por **letra**, **_** ou **digitos**
 - ❑ Underline com o tempo foi substituído pelo formato "camelo"
 - ✓ ValorDoSalario
 - ❑ **Em algumas linguagens (como as baseadas na ling. C) são sensíveis ao contexto**
 - Ex: rosa, Rosa e ROSA são distintos
 - Sério detrimento à **legibilidade** → denotam 3 entidades distintas.
 - Dificulta a escrita



Nomes

1.1 tamanho de nomes

- ❑ FORTRAN I: máximo 6
- ❑ COBOL: máximo 30
- ❑ FORTRAN 90 e ANSI C: máximo 31
- ❑ Ada: sem limite
- ❑ C++: sem limite, mas implementadores freqüentemente estabelecem limite



Nomes

2. palavras especiais

- ❑ Palavras especiais em uma LP são usadas para tornar os programas mais legíveis ao nomearem as ações a serem realizadas.
 - ❑ Não podem ser usadas como um nome (identificador)
 - ❑ Quanto mais palavras reservadas, mais difícil inventar nomes para variáveis
- ❑ Exemplos de palavras reservadas
 - ❑ **do repeat until**



Nomes

2. palavras especiais

- ❑ **Palavra-chave:** palavra que é especial apenas em certos contextos
 - Ex.: em FORTRAN
 - INTEGER** REAL //variável chamada real e tipo integer
 - REAL** INTEGER
 - Desvantagem: legibilidade pobre, diferenciação pelo contexto

- ❑ **Palavra reservada:** palavra que não pode ser usada como um nome definido pelo usuário
 - ❑ Quanto maior a quantidade de palavra reservada, maior a dificuldade do usuário definir nomes para as variáveis



Variáveis

- ❑ **Variável** é uma abstração de uma célula de memória
 - ❑ Surgiram durante a mudança das LP de baixo para alto nível
- ❑ Pode ser caracterizada por 6 atributos:
 - Nome, endereço, tipo, valor, tempo de vida e escopo
- ❑ **Nome**: identificador
- ❑ **Endereço**: localização da memória a ela associado
- ❑ **Tipo**: intervalo de possíveis valores e operações
- ❑ **Valor**: o que está armazenado na variável num determinado momento
- ❑ **Tempo de vida**: tempo durante o qual a memória permanece alocada para a variável
- ❑ **Escopo**: partes do programa onde a variável é acessível



Variáveis

nome

❑ **Nome:** cadeia de caracteres que identifica uma entidade (variáveis, constantes, rotinas, etc.) num programa

❑ Questões a considerar

1. Comprimento máximo?
2. Nomes com maiúsculas e minúsculas?
3. Palavras especiais

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)



Variáveis endereço

- ❑ **Endereço:** endereço de memória a ela associado
 - Uma variável pode ter diferentes endereços de memória durante a execução
 - Uma variável pode ter diferentes endereços em diferentes lugares de um programa. Ex:
 - ✓ mesmo nome de variável em subprogramas distintos
 - ✓ em diferentes momentos da execução do programa. Ex.: cada ativação de um subprograma chamado recursivamente
 - ✓ Exemplo: uso em functions

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)



Variáveis endereço

- ❑ **Endereço**: endereço de memória a ela associado
 - ❑ Pode haver dois nomes para um mesmo endereço (*aliases* ou apelidos)
 - o valor da variável muda com uma atribuição a qualquer um de seus nomes
 - desvantagem: legibilidade

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)



Variáveis tipo

❑ **Tipo:** determina a faixa de valores que ela pode armazenar e o conjunto de operações definidas para valores do tipo.

➤ Ex: tipo *int* em Java

✓ Faixa: -2147483648 a 2147483647

✓ Operações: adição, subtração, multiplicação, divisão e módulo

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)

Variáveis

- ❑ **Valor:** é o conteúdo da(s) célula(s) de memória associada(s) a ela
 - ✓ As vezes chamado de **lado direito** porque é requerido quando a variável é usada no lado direito de uma sentença de atribuição.
 - ✓ Para acessar o lado direito, o lado esquerdo precisa primeiro ser determinado
 - ✓ Qual o “valor” da variável **a** em **a := a + 1**
 - ✓ **valor-r**(right-value, valor da direita): seu valor
 - ✓ **valor-l**(left-value, valor da esquerda): seu endereço



Variáveis

Tempo de vida

- ❑ **Tempo de Vida** de uma Variável:
 - Tempo entre a criação e a destruição da variável
- ❑ Tempo de Vida:
 - Variável Global: mesmo do programa
 - Variável Local: mesmo do bloco onde foi declarada.

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)



Variáveis

escopo

❑ **Escopo:** trecho do programa em que uma declaração tem efeito.

- Variável Local: bloco onde foi declarada
- Variável Global: programa

Variável (nome, endereço, tipo, valor, tempo de vida e escopo)



Variáveis

Tempo de vida e escopo

```
program P  
var m: integer;  
  procedure R (n: integer);  
  begin  
    if n > 0 then R (n-1)  
  end;  
begin  
  R(2)  
end.
```

Escopo de n

Escopo de m

início P | *início R(2)* | *início R(1)* | *início R(0)* | *fim R(0)* | *fim R(1)* | *fim R(2)* | *fim P*

tempo de vida n=0

tempo de vida n=1

tempo de vida n=2

tempo de vida m



Variáveis

Tempo de vida e escopo

❑ Variáveis estáticas (*static*) de C

- Variável local com tempo de vida de uma variável global

```
1. int a = 1;
2. void f ( )
3. {   int b = 1;           // inicializado a cada chamada de f
4.   static int c = a;    // inicializado somente uma vez
5.   cout<<"a=" << a++ << "b=" << b++ << "c=" << c++ << end;
6.   c = c + 2;
7. }
8. int main ( )
9. { while (a < 4) f ( ); }
```



O conceito de Vinculações

- ❑ Associação, amarração ou vinculação (*binding*): associação de um (nome) identificador a sua declaração no programa
- ❑ Ambiente: conjunto de associações
- ❑ Exemplo: programa em C++



Vinculações

exemplo 1

❑ Exemplo: programa em C++

```
1. const int z= 0;
2. char c;
3. int p () {
4.     const float c= 3.14;
5.     bool b;
6.     ..... // ambiente: { c: constante 3.14;  b: variável lógica;
7.           // p: função;  z: constante 0 }
8. }
9. int main () {
10.    .....// ambiente { z: constante 0; c: variável char;
11.           //           p: função }
12.}
```



Vinculações exemplo 2

Considerando a expressão **A := B + C** a vinculação seria:

1. Obter o endereço de locações para um resultado e 1 ou mais operandos
2. Obter o dado (valor) operando da(s) locação(ões) do(s) operando(s)
3. Computar o dado resultado dos dados operandos
4. Armazenar o dado resultado na locação do resultado



unesp

Vinculações funcionamento

- $A := B + C$ seria executado como:
 - 1 - Obter os endereços de A, B e C
 - 2 - Obter os dados dos endereços (memória) B e C
 - 3 - Computar o resultado de $B + C$
 - 4 - Armazenar o resultado na locação de A



Vinculações

- ❑ Apesar da abstração de endereços em **nomes**, as LPs imperativas mantém os 4 passos como uma unidade de programa padrão.
- ❑ Essa unidade de execução se tornou a unidade fundamental de LP imperativas – chamada de **comando de atribuição**.

A := B + C



Vinculações

- ❑ De fundamental importância para a performance dessa **atribuição** é o estabelecimento e uso de um número de vinculações ("bindings"):
 - passo 1: vinculação entre nomes e locações de operandos e resultados
 - passo 2: vinculação entre locação e valor para estabelecer valores de operandos
 - **passo 3: computação**
 - passo 4: vinculação entre locação do resultado e o valor computado

A := B + C



Vinculações

- ❑ No passo 3, a computação depende da interpretação dos dados e dos operadores definidos.
 - ❑ LPs **relacionam dados** e **operadores através de tipos**.
 - ❑ Veremos o papel de tipo em vinculação.
 - ❑ Além disso será discutido o Escopo das vinculações: definição e mudança de vinculações.
 - ❑ Dado objeto.



Vinculações

- ❑ Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo
- ❑ Vinculação é a atribuição de valor a um dos quatro componentes acima. Essas vinculações podem ser alteradas em certas ocasiões.



Vinculações

- ❑ Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo

- ❑ As **vinculações** podem ocorrer em:
 - ❑ tempo de **compilação**
 - ❑ tempo de **loading** (quando o programa LM gerado pelo compilador está sendo alocado à locações específicas na memória)
 - ❑ ou em tempo de **execução**.



Tipos de vinculações

- ❑ **Vinculações de locação** geralmente ocorrem em tempo de *loading*, mas também podem ocorrer em tempo de *execução* (variáveis em procedimentos e alocação dinâmica).
- ❑ **Vinculações de nome** ocorrem tipicamente em tempo de *compilação*, quando uma declaração é encontrada.

Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo



Tipos de vinculações

- ❑ **Vinculações de valor** ocorrem tipicamente em tempo de **execução**.

Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo



Tipos de vinculações

- ❑ **Vinculações de tipo** ocorrem geralmente em tempo de **compilação**, através de declarações de tipo. Veja exemplos dos efeitos da declaração de tipos em ADA nas figuras seguintes (exemplo 2).

Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo



Tipos de vinculações

- ❑ **Vinculações de tipo dinâmico** ocorre durante tempo de execução, não havendo declarações de tipo. O tipo do dado objeto é determinado pelo tipo do valor, e portanto, uma mudança de valor implica em nova vinculação. Ex: linguagem PHP

Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo



Vinculação de tipo exemplo (1)

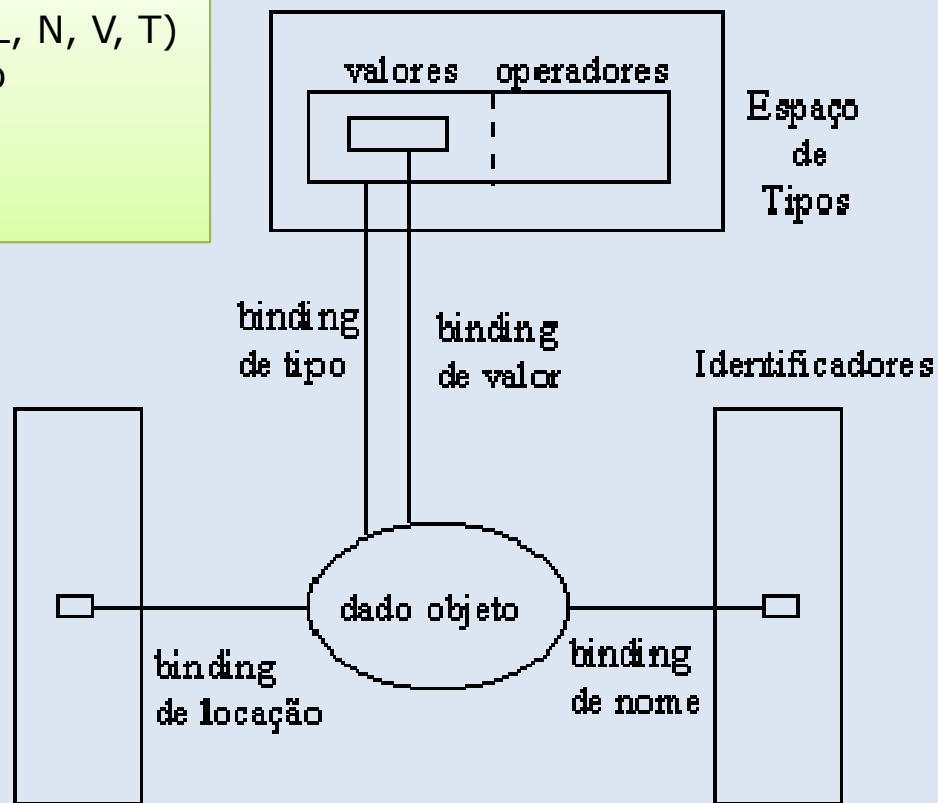
$x = x + 5$

1. O tipo de x é vinculado em tempo de compilação
2. O conjunto de possíveis valores de x é vinculado em tempo de *loading*
3. O significado de $+$ é vinculado em tempo de compilação, após a determinação dos tipos dos operandos
4. 5 é vinculado em tempo de *loading*
5. *count* é vinculado em tempo de execução

Dado objeto = (L, N, V, T) onde L - locação, N - nome, V - valor, T - tipo

Vinculação de tipo exemplo (2)

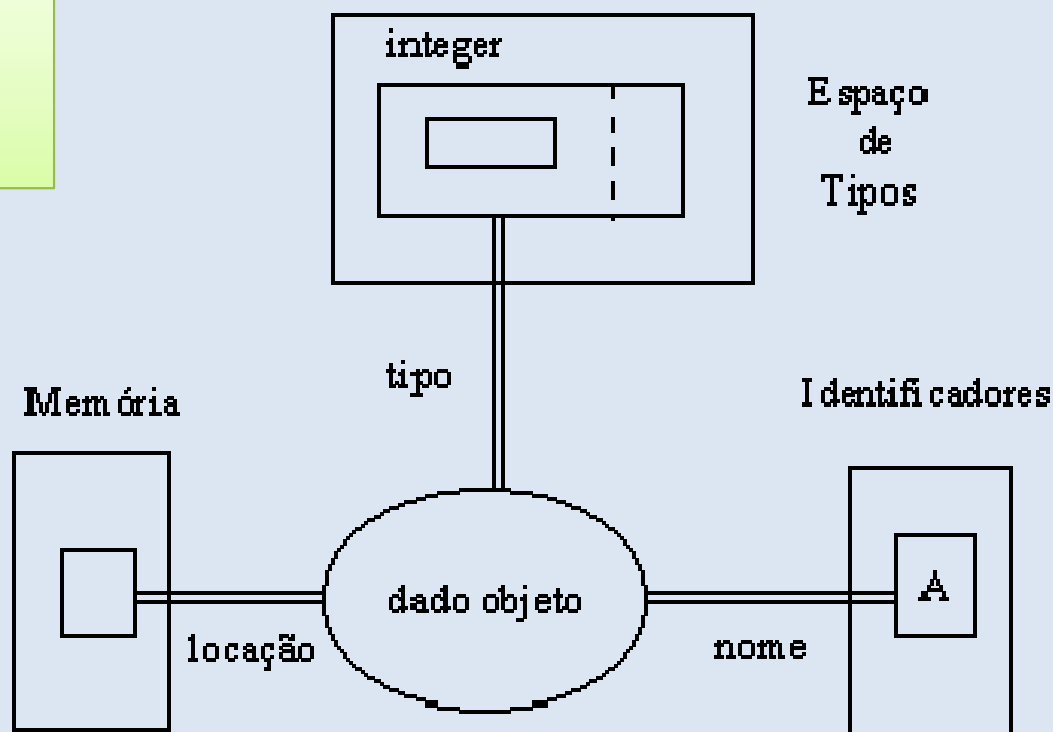
objeto = (L, N, V, T)
 L - locação
 N - nome,
 V - valor,
 T - tipo



Vinculação de tipo exemplo (2)

objeto = (L, N, V, T)
L - locação
N - nome,
V - valor,
T - tipo

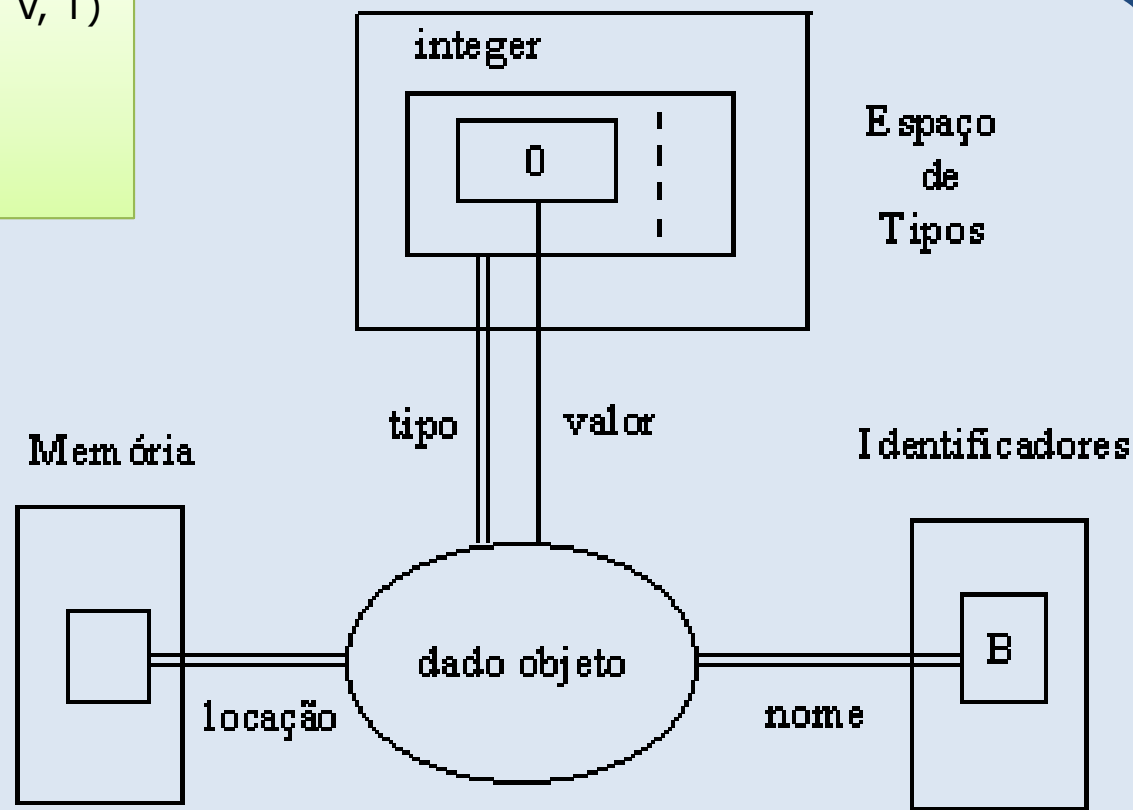
A:integer;



Vinculação de tipo exemplo (2)

objeto = (L, N, V, T)
 L - locação
 N - nome,
 V - valor,
 T - tipo

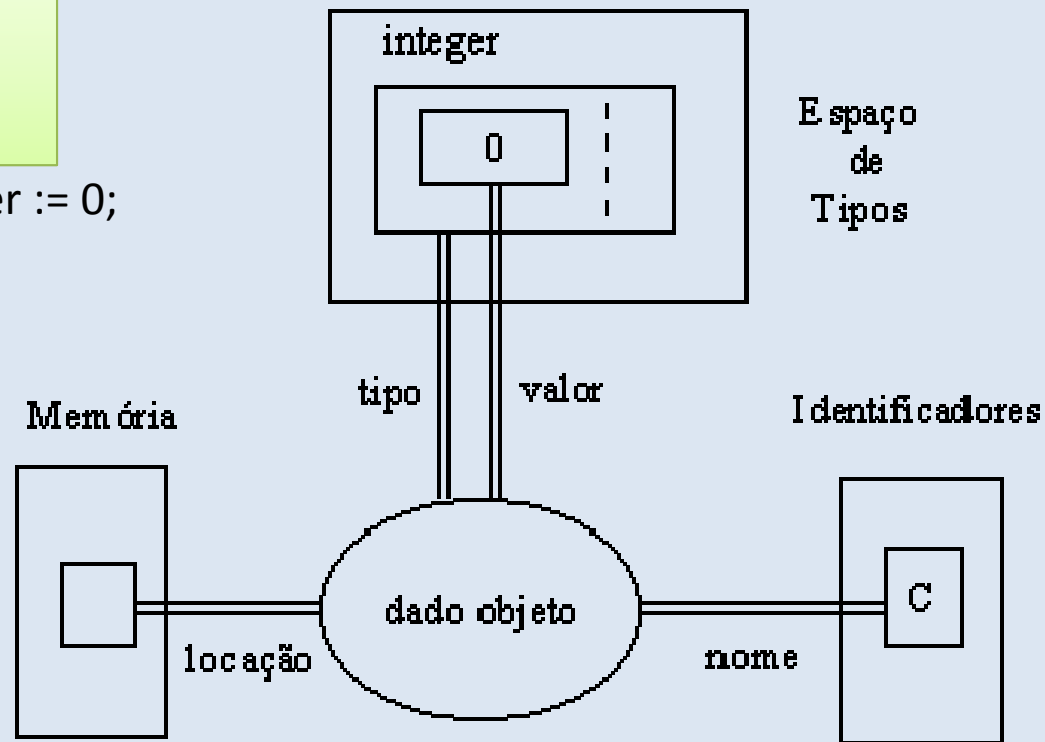
B:integer:=0;



Vinculação de tipo exemplo (2)

objeto = (L, N, V, T)
 L - locação
 N - nome,
 V - valor,
 T - tipo

C : constant integer := 0;





Escopo de vinculação e Unidades de Execução

- ❑ Divisões de um **programa**:
 - **programa** - maior divisão; unidade executável fundamental
 - **blocos**
 - **comando** - menor divisão; unidade indivisível
- ❑ O objetivo de se juntar unidades de execução, neste caso, é para identificar o **escopo de vinculações**.



Escopo de vinculação e Unidades de Execução

```
int a = 1;
void f ( ) // bloco
{   int b = 1; // comandos ...
    static int c = a;
    cout<<"a=" << a++ << "b=" << b++ << "c=" << c++ << end;
    c = c + 2;
}
int main ( ) // programa
{   while (a < 4) f ( ); }
```



Tipos de vinculações

- ❑ Uma **vinculação** é dita **estática** se ocorre antes da execução e permanece inalterado durante a execução do programa
- ❑ Uma **vinculação** é dita **dinâmica** se ocorre durante a execução ou pode ser alterado durante a execução do programa



unesp

Tipos de vinculações estática

- ❑ Como um tipo é especificado?
- ❑ Quando acontece a vinculação?
- ❑ Se **estático**, o tipo pode ser especificado ou por uma declaração explícita ou por declaração implícita
 - Uma **declaração explícita** é uma sentença declarativa para o tipo da variável
 - Uma **declaração implícita** é um mecanismo default para a especificação de tipos – acontece na primeira vez que uma variável é encontrada



Tipos de vinculações estática

- ❑ A maioria das LP's, projetadas a partir de 1960, requer a declaração explícita.

int a;

- ❑ Exemplos: FORTRAN, PL/I, BASIC e Perl têm mecanismos de declaração implícita
- ❑ Em Fortran, se uma variável não for explicitamente declarada, usa-se a convenção para declará-la implicitamente:
 - ❑ Começou com I, J, K, L, M ou N, recebe tipo integer
 - ❑ Demais casos, recebe tipo real



Tipos de vinculações dinâmico

- ❑ O tipo de uma variável não é especificado por uma sentença de declaração, nem pode ser determinado pelo nome da variável.
 - ❑ A variável é vinculada a um tipo quando é atribuído um valor a ela.
 - ❑ Ex: PHP

```
$varA = 10; //variável int
```

```
$varA = "teste"; //passa a ser uma string
```
- ❑ Vantagem: flexibilidade
- ❑ Desvantagem:
 - ✓ Alto custo (checagem de tipo e interpretação)
 - ✓ Detecção de erros de tipo pelo compilador é difícil



Vinculações de armazenamento e Tempo de Vida

- ❑ O caráter fundamental de uma LP é determinado pelo projeto das **vinculações** de armazenamento para suas variáveis.
 - ❑ Necessário saber como ocorrem essas **vinculações**.
 - ❑ Célula de memória a qual uma variável é vinculada é obtida a partir das células de memória disponíveis → **alocação**
 - ❑ Quando a variável não necessita mais desta célula de memória, torna-se necessário devolvê-la ao conjunto de células disponíveis → **liberação**



Vinculações de armazenamento e Tempo de Vida

- ❑ O **tempo de vida** de uma variável é o durante o qual ela está vinculada a uma célula de memória.
- ❑ Os tipos de variáveis são classificadas em:
 1. Variáveis estáticas
 2. Variáveis dinâmicas na pilha
 3. Variáveis dinâmicas no monte (*heap*) explícitas
 4. Variáveis dinâmicas no monte (*heap*) implícitas



Vinculações de armazenamento e Tempo de Vida

1. variáveis estáticas

- ❑ Vinculadas a células de memória antes da execução, permanecendo na mesma célula de memória durante toda a execução
 - Exemplos: variáveis em FORTRAN 77, variáveis estáticas em C (cláusula *static*)
- ❑ Vantagem:
 - Eficiência (endereçamento direto)
 - Não há sobrecarga em tempo de execução para alocação e liberação



Vinculações de armazenamento e Tempo de Vida

1. variáveis estáticas

❑ Desvantagem:

- falta de flexibilidade (sem recursão)
- Armazenamento não compartilhado entre variáveis.
 - Exemplo: **um programa com dois subprogramas que requerem grandes vetores.** Suponha que os dois subprogramas nunca estão ativos ao mesmo tempo, se os vetores são estáticos eles não podem compartilhar o mesmo armazenamento para seus vetores



Vinculações de armazenamento e Tempo de Vida

2. variáveis dinâmicas na pilha

- ❑ **Variáveis dinâmicas de pilha:** vinculações são criadas quando suas sentenças de declaração são efetuadas, mas o tipo é estaticamente vinculado.
 - Ex: em Java, as declarações de variáveis que aparecem no início de um método são elaboradas quando o método é chamado e as variáveis definidas por essas declarações são liberadas quando o método completa sua execução
 - São alocadas a partir da pilha de tempo de execução



Vinculações de armazenamento e Tempo de Vida

2. variáveis dinâmicas na pilha

❑ Vantagem:

- permitem recursão e otimizam o uso de espaço em memória

❑ Desvantagens:

- sobrecarga de alocação e liberação
- referência ineficiente (endereçamento indireto)



unesp

Vinculações de armazenamento e Tempo de Vida

3. variáveis dinâmicas monte (heap) explícitas

- ❑ **Variáveis dinâmicas do monte (*heap*) explícitas** são células de memória não nomeadas (abstratas), que são alocadas e liberadas por instruções explícitas, especificadas pelo programador, que tem efeito durante a execução



unesp

Vinculações de armazenamento e Tempo de Vida

3. variáveis dinâmicas monte explícitas

- ❑ Essas variáveis, alocadas a partir do monte e liberadas para o monte podem ser referenciadas por ponteiros
 - Exemplos: objetos dinâmicos em C e tudo em JAVA
 - Vantagem: armazenamento dinâmico
 - Desvantagem: Ineficientes e não confiável



unesp

Vinculações de armazenamento e Tempo de Vida

3. variáveis dinâmicas monte explícitas

- ❑ Exemplo de variáveis dinâmicas no heap - alocação e liberação

```
int *intnode; //cria um ponteiro  
intnode = new int; //aloca - cria variável dinâmica no heap-  
                // operador new  
...  
delete new; //libera a variável do heap
```



unesp

Vinculações de armazenamento e Tempo de Vida

4. variáveis dinâmicas monte implícitas

- ❑ Variáveis dinâmicas do monte implícitas são vinculadas ao armazenamento no monte (**heap**) apenas quando são atribuídos valores a elas.
- ❑ Exemplo: PHP
 - \$alunos = (2, 5, 6, 1); //independente do que a variável \$alunos foi usada, agora ela passa ser um vetor de inteiros com 4 valores numéricos
 - Vantagem: flexibilidade
 - Desvantagem:
 - ✓ Sobrecarga em tempo de execução
 - ✓ Sem detecção de erros



Verificação de tipos de variáveis

- ❑ *Verificação ou Checagem de Tipo* é uma atividade de garantia de que operandos e operadores são de tipos compatíveis
- ❑ Tipo compatível é um tipo que é legal para um operando, ou é permitido, segundo as regras da linguagem. Pode ser que haja a conversão automática para garantir a compatibilidade (coerção)
- ❑ Uma linguagem é fortemente tipada se erros de tipo são sempre detectados



Verificação de tipos de variáveis

- ❑ Exemplo: a linguagem C e C++ não é fortemente tipificada
 - ❑ Permitem funções cujos parâmetros não são verificados quanto ao tipo
- ❑ Linguagem java é fortemente tipificada

Escopo

- ❑ O **escopo** de uma variável é a faixa de instruções na qual a variável é visível
 - Uma variável é visível em uma instrução se puder ser referenciada nessa instrução
- ❑ As **variáveis não-locais** de uma unidade ou de um bloco de programa são as visíveis dentro deste, mas não são declaradas lá

Blocos

- ❑ Conjunto de comandos para atender a determinado fim.
 - 1 - Escopo de estrutura de controle
 - Estruturas condicionais
 - Estruturas iterativas
 - 2 - Escopo de procedimentos ou funções
 - 3 - Unidades de compilação
 - Blocos compilados separadamente e depois unidos para execução
 - 4 - Escopo de vinculações
 - Bloco de comandos sobre os quais vinculações específicas são válidas



Escopo de vinculação de Nome

- ❑ Blocos que definem um escopo de vinculação de nome contém, em geral, duas partes:
 - Uma seção de DECLARAÇÕES, que define as vinculações que valem dentro do bloco
 - Uma seção EXECUTÁVEL, que contém os comandos do bloco, onde valem as vinculações
- ❑ Sintaticamente, isso requer delimitadores.



Escopo de vinculação de Nome

Exemplo

```
BLOCK A;  
  DECLARE I;  
  BEGIN A  
  ...    {I de A} - binding válido  
  END A;
```



Dois tipos de vinculações (local e não local) em um Bloco (Exemplo)

Blocos aninhados:

```
1. program P;
2.   declare X;
3.   begin P
4.     ...      {X de P}
5.     block A;
6.       declare Y;
7.       begin A
8.         ...      {X de P; Y de A}
9.         block B;
10.          declare Z;
11.          begin B
12.            ...   {X de P; Y de A; Z de B}
13.             end B;
14.             ...   {X de P; Y de A}
15.           end A;
16.           ...     {X de P}
17.           block C;
18.             declare Z;
19.             begin C
20.               ...   {X de P; Z de C}
21.             end C;
22.             ...     {X de P}
23.           end.
```



Política do Escopo Léxico ou Estático

- ❑ Se um nome tem uma declaração num bloco, este nome é ligado ao objeto especificado na declaração.
- ❑ Se um nome não tem declaração num bloco, ele é ligado ao mesmo objeto ao qual ele foi ligado no bloco que contém o bloco atual, no texto do programa.
- ❑ Se o bloco não está contido em outro, ou se o nome não foi ligado no bloco que contém este, então o nome não está ligado a qualquer objeto no bloco atual.

Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```

Regra de Escopo

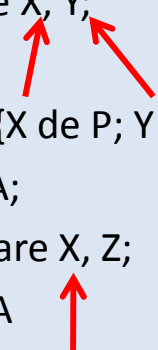
- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```

Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

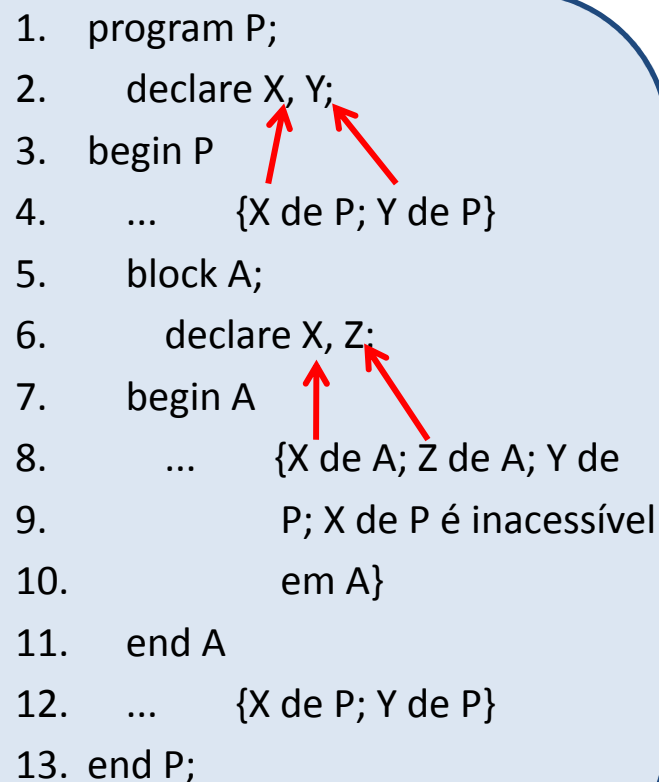
```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```



Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

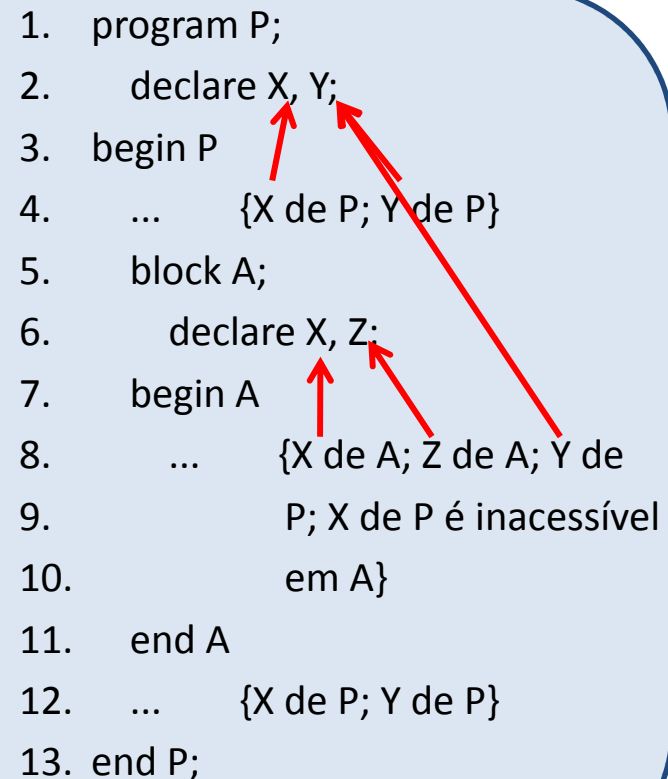
```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```



Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

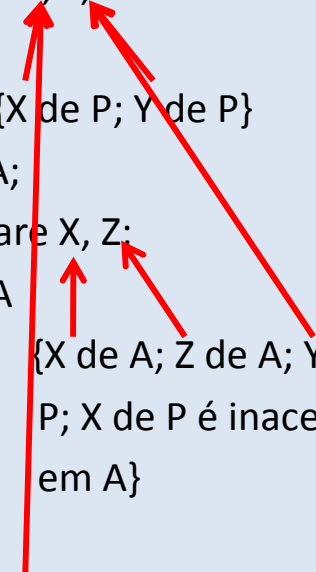
```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```



Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

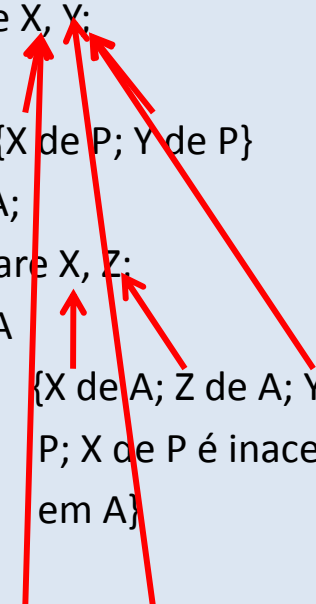
```
1. program P;  
2.   declare X, Y;  
3. begin P  
4.   ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.   begin A  
8.     ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.   end A  
12.   ... {X de P; Y de P}  
13. end P;
```



Regra de Escopo

- ❑ Quando um bloco redeclara um nome já ligado no ambiente que o contém, a declaração local se sobrepõe ao vínculo não-local, fazendo o objeto ligado não-localmente inacessível no bloco atual.

```
1. program P;  
2.   declare X, Y;  
3.   begin P  
4.     ... {X de P; Y de P}  
5.   block A;  
6.     declare X, Z;  
7.     begin A  
8.       ... {X de A; Z de A; Y de  
9.         P; X de P é inacessível  
10.        em A}  
11.    end A  
12.  ... {X de P; Y de P}  
13. end P;
```





Escopo Estático X Escopo Dinâmico

- ❑ **Escopo Dinâmico:** O ambiente "global" de uma unidade (procedure ou function) é o ambiente da unidade que a chamou.
 - ❑ Uma procedure herda como ambiente global aquele da unidade que a chamou e, portanto, este só pode ser determinado em tempo de execução.



Escopo de vinculação de Localização (dinâmico)

- ❑ Hipótese assumida: vinculação de localização feito em tempo de *loading*, permanecendo válido durante toda a execução do programa.
- ❑ Efeito colateral: ao re-executar um bloco, as variáveis locais reteriam os valores da execução anterior.
- ❑ Se uma nova vinculação de localização for feita a cada nova entrada no bloco, nenhuma suposição poderia ser feita.



Escopo de vinculação de Localção (dinâmico) - Exemplo

```
1. program P
2.   declare I;
3.   begin P
4.     for I := 1 to 10 do
5.       block A;
6.         declare J;
7.         begin A
8.           if I = 1 then    {I de P; J de A}
9.             J := 1
10.          else
11.            J := J * I {assume-se que J retém valor de execução prévia}
12.          endif
13.        end A;
14.    end P
```



Comentários

- ❑ **Desvantagem:** memória para todos os blocos do programa deve ser reservada por todo o tempo de execução do programa.
- ❑ **Alternativa:** (vinculação dinâmica) Fazer a vinculação de locação, bem como o de nome, em tempo de execução, a cada entrada de um bloco, desfazendo essas vinculações quando da saída do bloco. (Alocação dinâmica de memória).



Implementação

Escopo de vinculação de Locação (dinâmico)

- ❑ A implementação é via registros de ativação, que são registros contendo informações sobre uma unidade de execução, necessárias para restabelecer sua execução, depois de ela ter sido suspensa.



Implementação

Escopo de vinculação de Locação (dinâmico)

- ❑ Para efeito de vinculação de locação (dinâmico), o registro de ativação precisará conter apenas as locações para todos os objetos ligados localmente, mais um ponteiro para o registro de ativação do bloco que o contém.



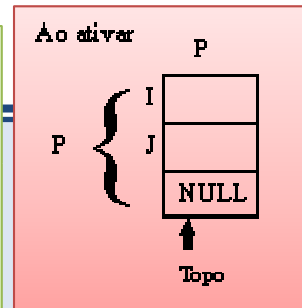
Funcionamento

- ❑ Conforme um bloco é ativado, seu registro é colocado no topo de uma pilha. No término de sua execução, seu registro é desempilhado.



Exemplo

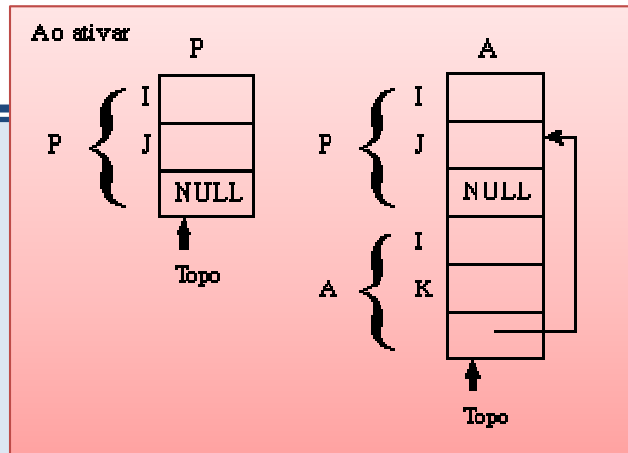
```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
      declare I, L;  
    begin B;  
      ...  
    end B;  
    ...  
  end A;  
  block C;  
    declare I, N;  
  begin C  
    ...  
  end C;  
end P;
```





Exemplo

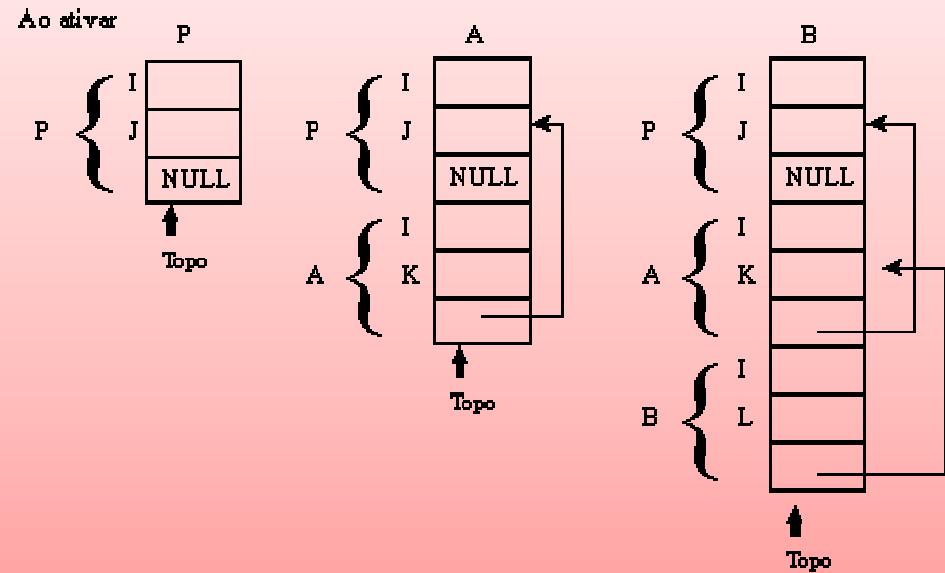
```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
      declare I, L;  
    begin B;  
      ...  
    end B;  
    ...  
  end A;  
  block C;  
    declare I, N;  
  begin C  
    ...  
  end C;  
end P;
```





Exemplo

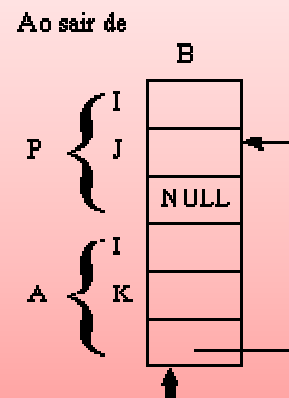
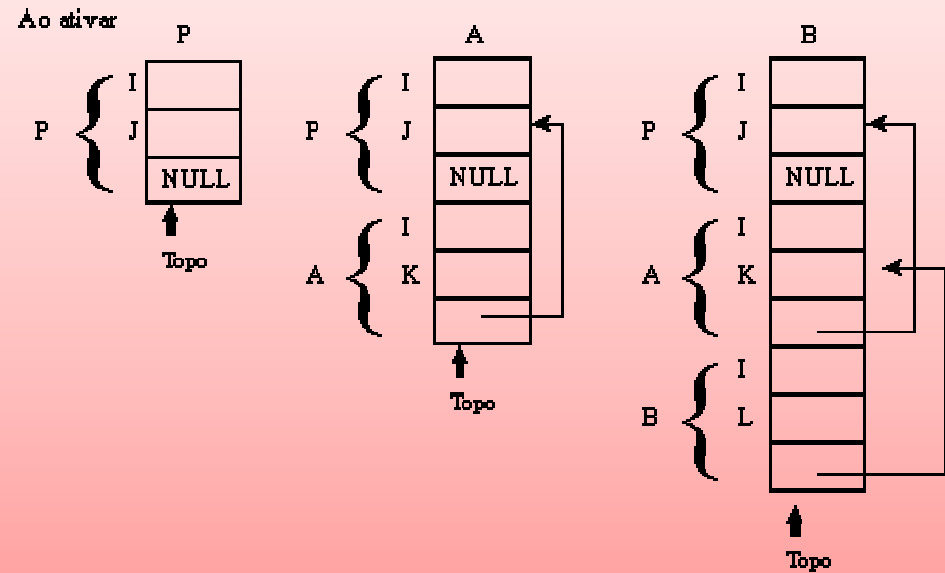
```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
    declare I, L;  
    begin B;  
    ...  
  end B;  
  ...  
end A;  
block C;  
  declare I, N;  
begin C  
  ...  
end C;  
end P;
```





Exemplo

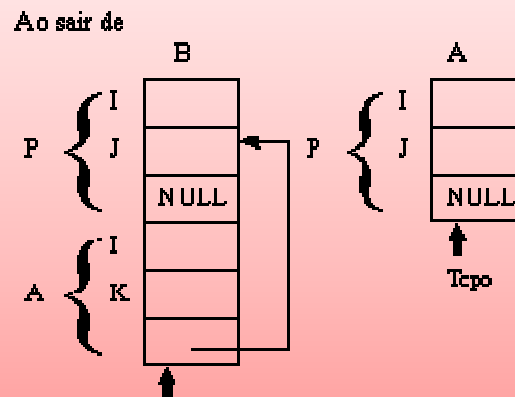
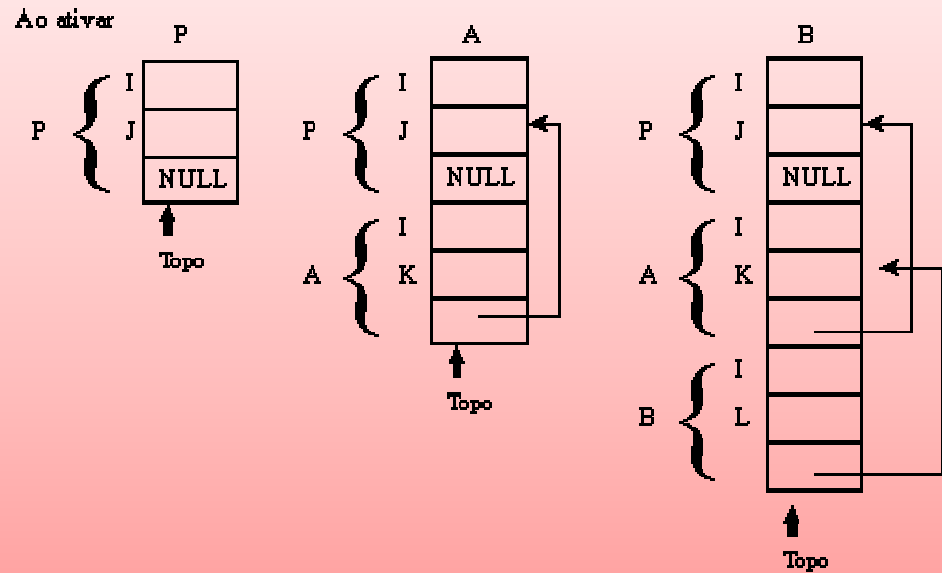
```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
      declare I, L;  
    begin B;  
      ...  
    end B;  
    ...  
  end A;  
  block C;  
    declare I, N;  
  begin C  
    ...  
  end C;  
end P;
```





Exemplo

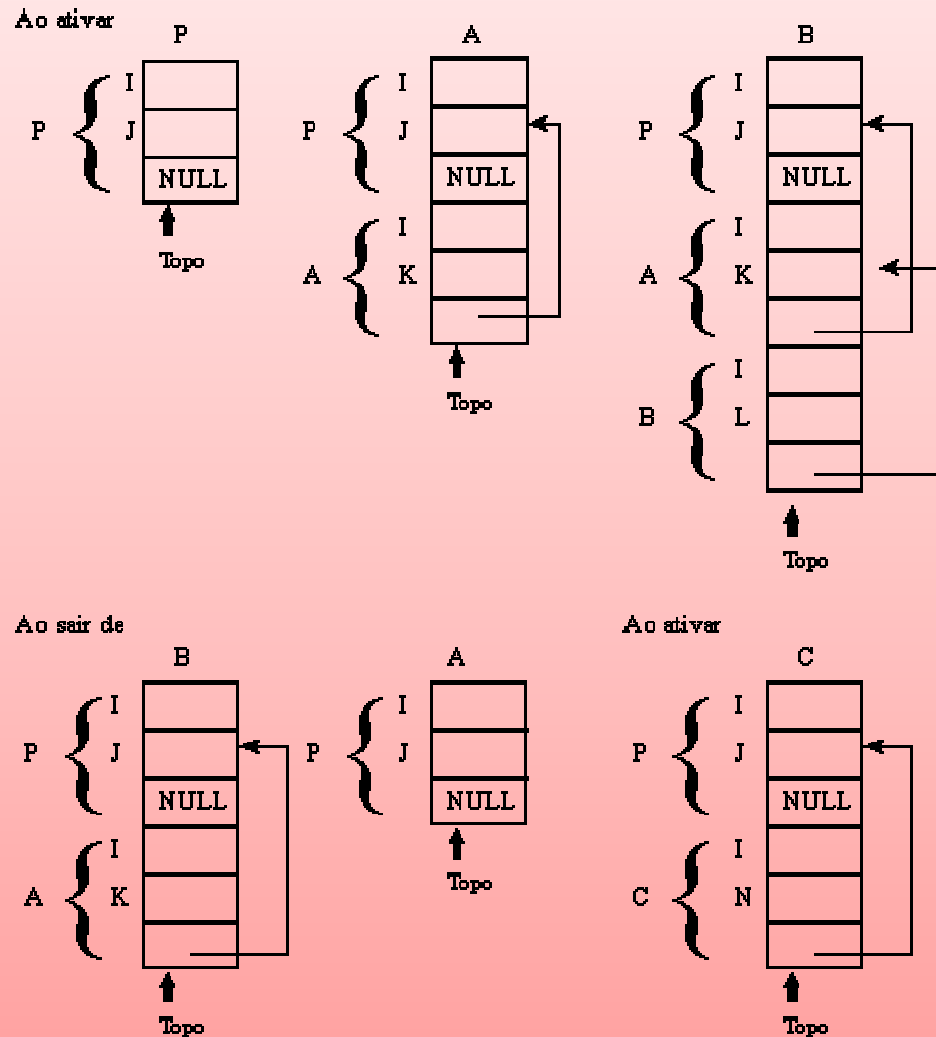
```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
      declare I, L;  
    begin B;  
      ...  
    end B;  
    ...  
  end A;  
  block C;  
    declare I, N;  
  begin C  
    ...  
  end C;  
end P;
```





Exemplo

```
program P;  
  declare I, J;  
begin P  
  block A;  
    declare I, K;  
  begin A  
    block B;  
      declare I, L;  
    begin B;  
      ...  
    end B;  
    ...  
  end A;  
  block C;  
    declare I, N;  
  begin C  
    ...  
  end C;  
end P;
```





Comentários

- ❑ Pela disposição dos blocos no texto, é possível determinar o escopo sem o uso de vinculação dinâmica. Assim, seria desvantagem a sua utilização.
- ❑ Porém, normalmente utilizamos procedures e functions e só conhecemos a execução desses blocos em tempo de execução. Dessa forma, o uso de vinculação dinâmica é utilizada.

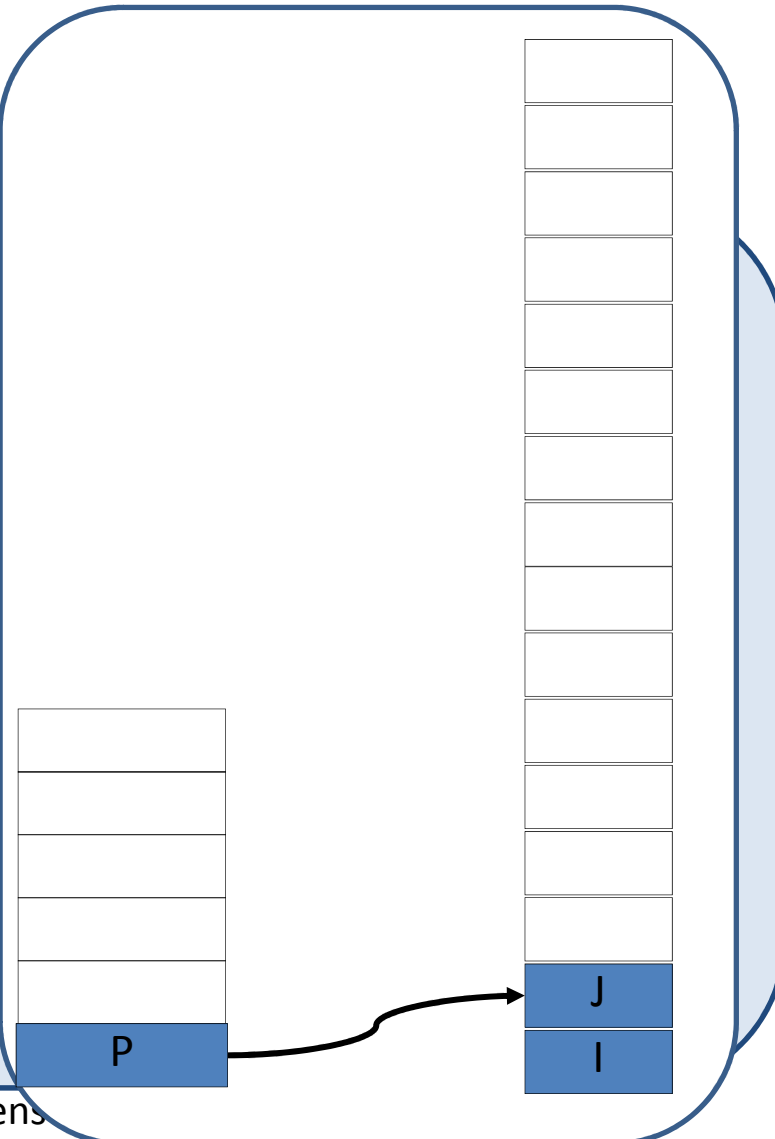


Uso da pilha para localizar objetos

1. Procura no registro do topo pelo objeto ligado ao nome.
2. Se não encontrar, procure-o no registro apontado por ele; continue nesse processo até encontrá-lo na lista, ou a lista acabar.
3. Esta busca pode ser otimizada, uma vez que se sabe, em tempo de compilação, a estrutura da pilha e de cada registro de ativação

Exemplo

```
program P;  
  declare I, J;  
  
  procedure Proc_A;  
    declare I, K;  
  begin A  
    ...  
  end A  
  
  Procedure Proc_B;  
    declare I, L;  
  begin B;  
    ... Proc_A;  
  end B;  
  
  Procedure Proc_C;  
    declare I, N;  
  begin C  
    ... Proc_B;  
  end C;  
  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

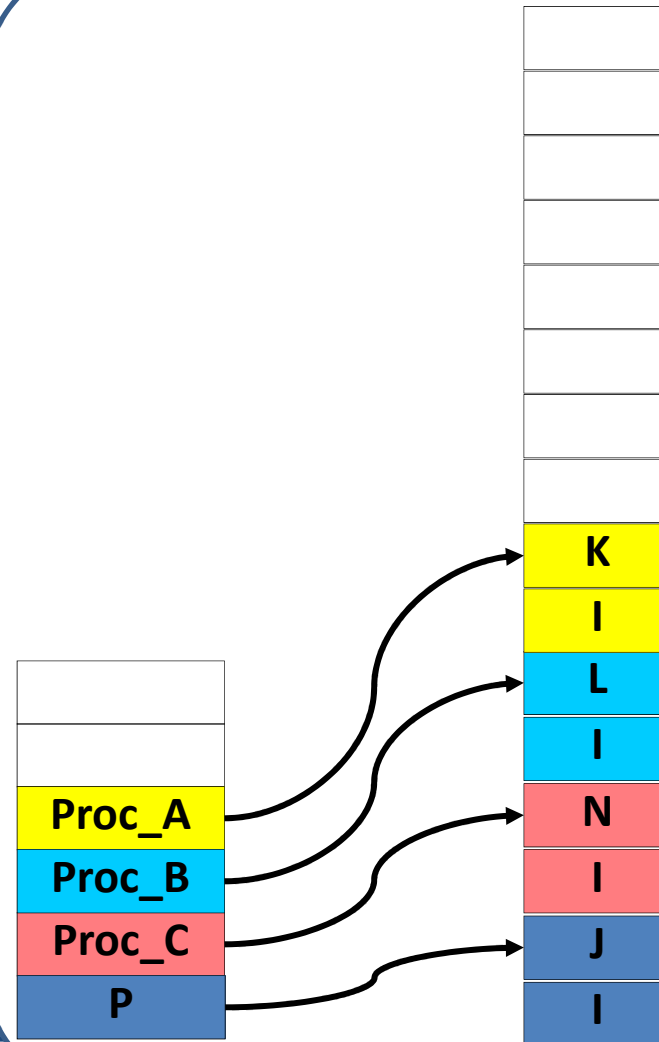


adagens

Exemplo

```
program P;  
  declare I, J;  
  procedure Proc_A;  
    declare I, K;  
  begin A  
    ...  
  end A  
  Procedure Proc_B;  
    declare I, L;  
  begin B;  
    ... Proc_A;  
  end B;  
  Procedure Proc_C;  
    declare I, N;  
  begin C  
    ... Proc_B;  
  end C;  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

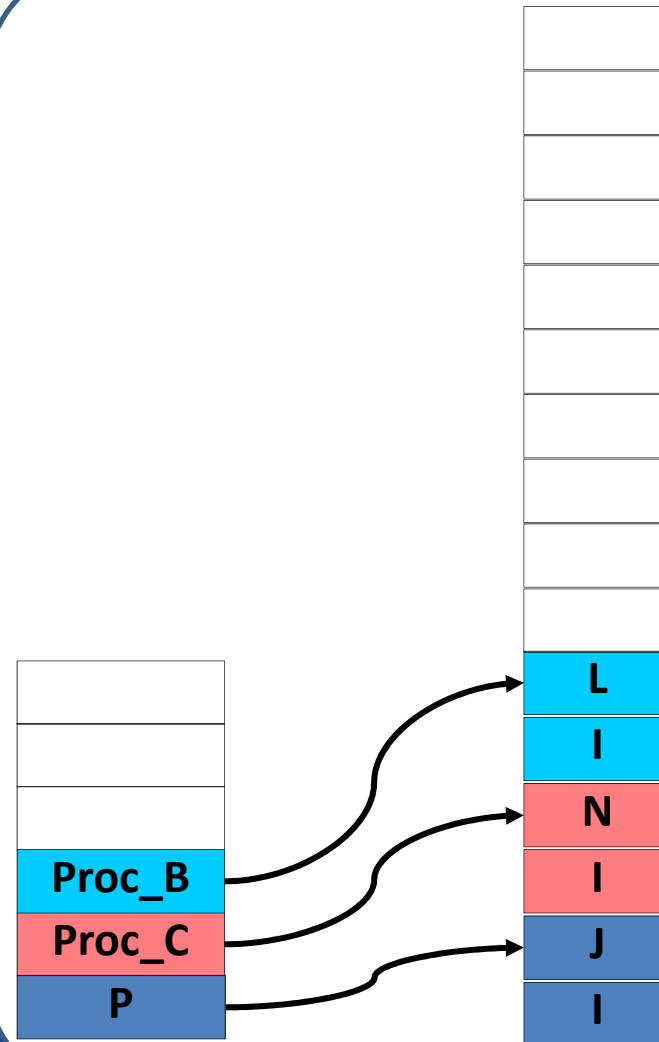
Chamando o Proc_A



Exemplo

```
program P;  
  declare I, J;  
  procedure Proc_A;  
    declare I, K;  
    begin A  
      ...  
    end A  
  Procedure Proc_B;  
    declare I, L;  
    begin B;  
      ... Proc_A;  
    end B;  
  Procedure Proc_C;  
    declare I, N;  
    begin C  
      ... Proc_B;  
    end C;  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

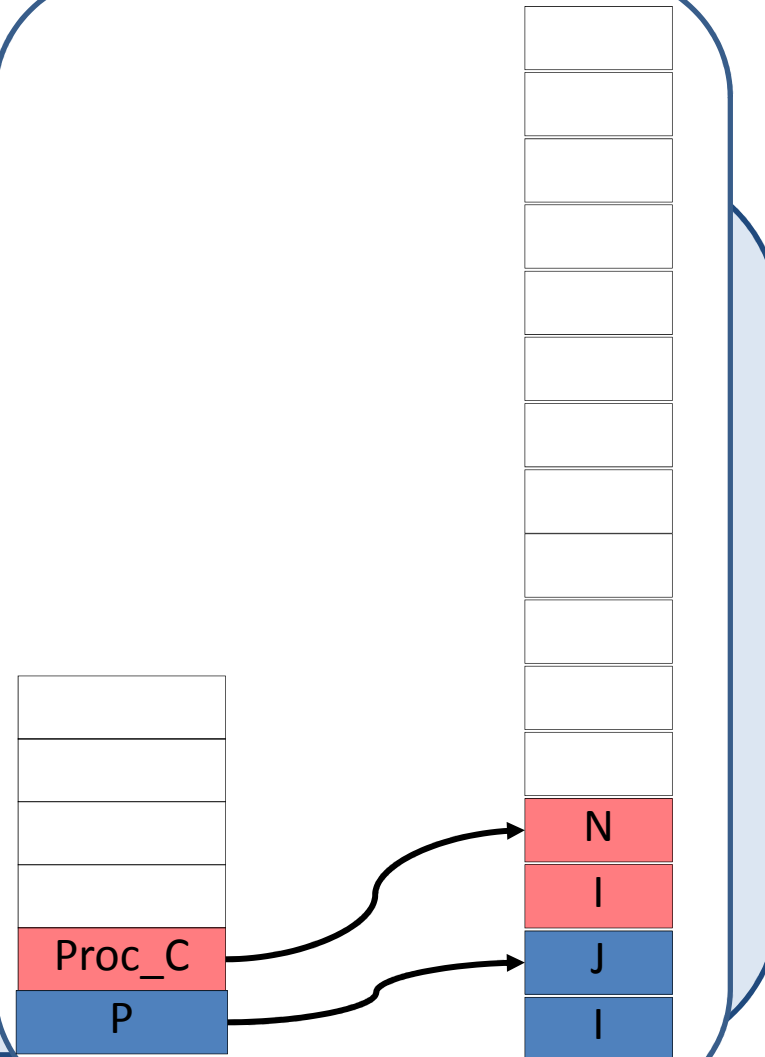
Saindo do Proc_A



Exemplo

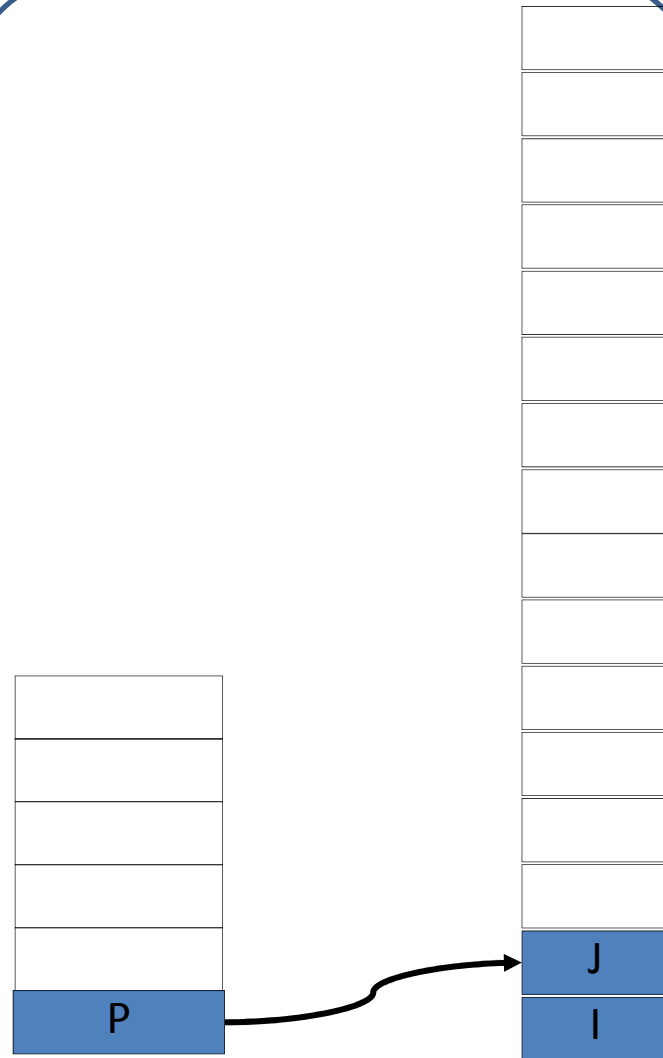
```
program P;  
  declare I, J;  
  procedure Proc_A;  
    declare I, K;  
    begin A  
      ...  
    end A  
  Procedure Proc_B;  
    declare I, L;  
    begin B;  
      ... Proc_A;  
    end B;  
  Procedure Proc_C;  
    declare I, N;  
    begin C  
      ... Proc_B;  
    end C;  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

Saindo do Proc_C



Exemplo

```
program P;  
  declare I, J;  
  procedure Proc_A;  
    declare I, K;  
    begin A  
      ...  
    end A  
  Procedure Proc_B;  
    declare I, L;  
    begin B;  
      ... Proc_A;  
    end B;  
  Procedure Proc_C;  
    declare I, N;  
    begin C  
      ... Proc_B;  
    end C;  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

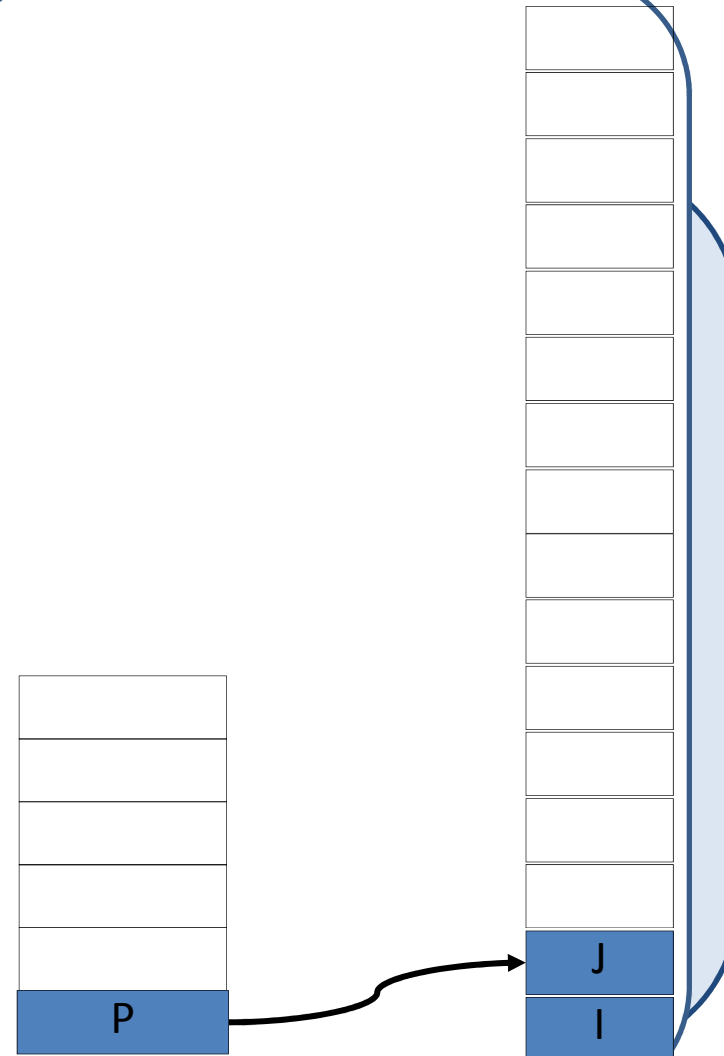


adagens

Exemplo

```
program P;  
  declare I, J;  
  procedure Proc_A;  
    declare I, K;  
    begin A  
      ...  
    end A  
  Procedure Proc_B;  
    declare I, L;  
    begin B;  
      ... Proc_A;  
    end B;  
  Procedure Proc_C;  
    declare I, N;  
    begin C  
      ... Proc_B;  
    end C;  
  begin P  
    Proc_C  
    Proc_A  
  end P;
```

Saindo do Proc_A





Resumo

- ❑ Nomes
 - Tamanho; caracteres de conexão; distinção entre maiúsculas e minúsculas; palavras especiais
- ❑ Variáveis
 - nome, endereço, valor, tipo, tempo de vida, escopo
- ❑ Vinculação (Binding) é a associação de atributos a entidades do programa
- ❑ Variáveis escalares são categorizadas como
 1. Variáveis estáticas
 2. Variáveis dinâmicas na pilha
 3. Variáveis dinâmicas no monte (*heap*) explícitas
 4. Variáveis dinâmicas no monte (*heap*) implícitas



Exercícios

- ❑ Capítulo 5 – questões de revisão
 - ❑ 1, 6, 7, 9, 12, 13, 16 e 19
- ❑ Capítulo 5 – conjunto de problemas
 - ❑ 3, 6, 8, 10 e 11