Paradigmas de Linguagens de Programação

Prof. Sergio D. Zorzo
Departamento de Computação - UFSCar
1º semestre / 2013
Aula 7

Material adaptado do Prof. Daniel Lucrédio

Subprogramas

- Programação estruturada
  - Subprogramas
  - Parte essencial
    - Modularização
    - Reutilização
    - Controle de fluxo
- Como são implementados?

Chamada a subprogramas

- Quando um subprograma é chamado:
  - Mecanismo do método de passagem de parâmetros
    - Cópia, referência, nome...
  - Alocação dinâmica de variáveis locais
  - Salvar o status da unidade chamadora
  - Acesso a variáveis não-locais visíveis

Chamada a subprogramas

- Quando um subprograma encerra:
  - Cópia dos parâmetros de saída (se for o caso)
  - Desalocar as variáveis locais
  - Restabelecer o status da unidade chamadora
  - Restabelecer o acesso a variáveis não-locais
  - Devolver o controle à unidade chamadora

FORTRAN 77

- Começaremos pelo mais simples
  - Variáveis não-locais são sempre globais
  - Sem recursividade
  - Variáveis estaticamente alocadas
  - Ambiente completamente estático
    - Ou seja, TUDO pode ser decidido e preconfigurado durante a compilação

FORTRAN 77

- Chamada a subprograma:
  1. Salvar o status de execução da unidade de programa atual
  2. Executar o processo de passagem de parâmetros
  3. Passar o endereço de retorno para o chamado
  4. Transferir o controle para o chamado

FORTRAN 77

- Retorno de subprograma:
  1. Se forem usados parâmetros passados por resultado (cópia), os valores são transferidos para os parâmetros reais
  2. Se o subprograma for uma função, o valor funcional é transferido para um lugar acessível ao chamador
  3. O status de execução do chamador é restabelecido
  4. O controle será devolvido para o chamador

FORTRAN 77

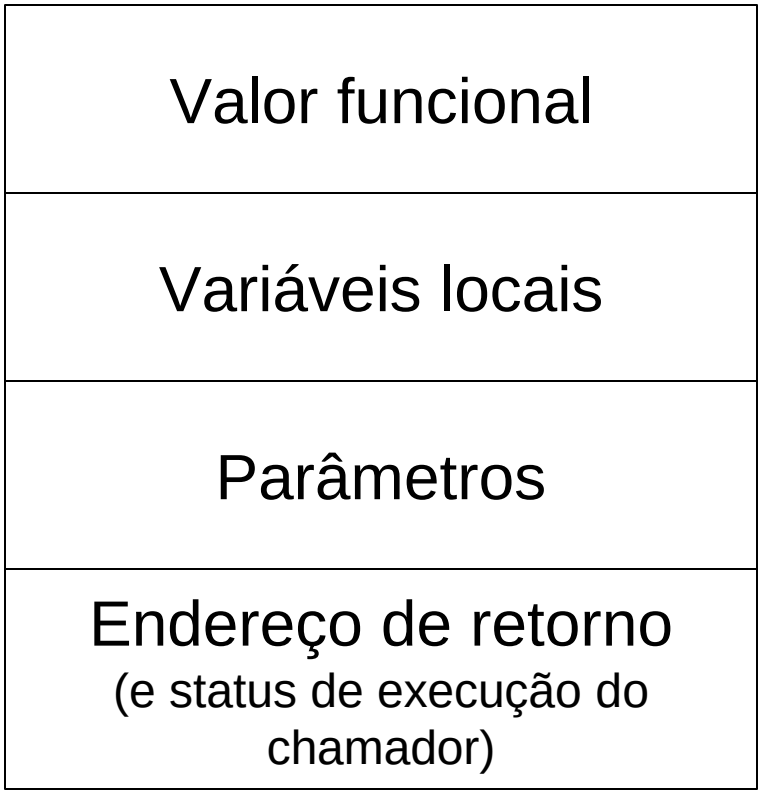
- Nesse cenário, é necessário espaço para:
  - Informações do status do chamador
  - Parâmetros
  - Endereço de retorno
  - Valor funcional para subprogramas do tipo função
  - Variáveis locais
  - Código do subprograma

FORTRAN 77

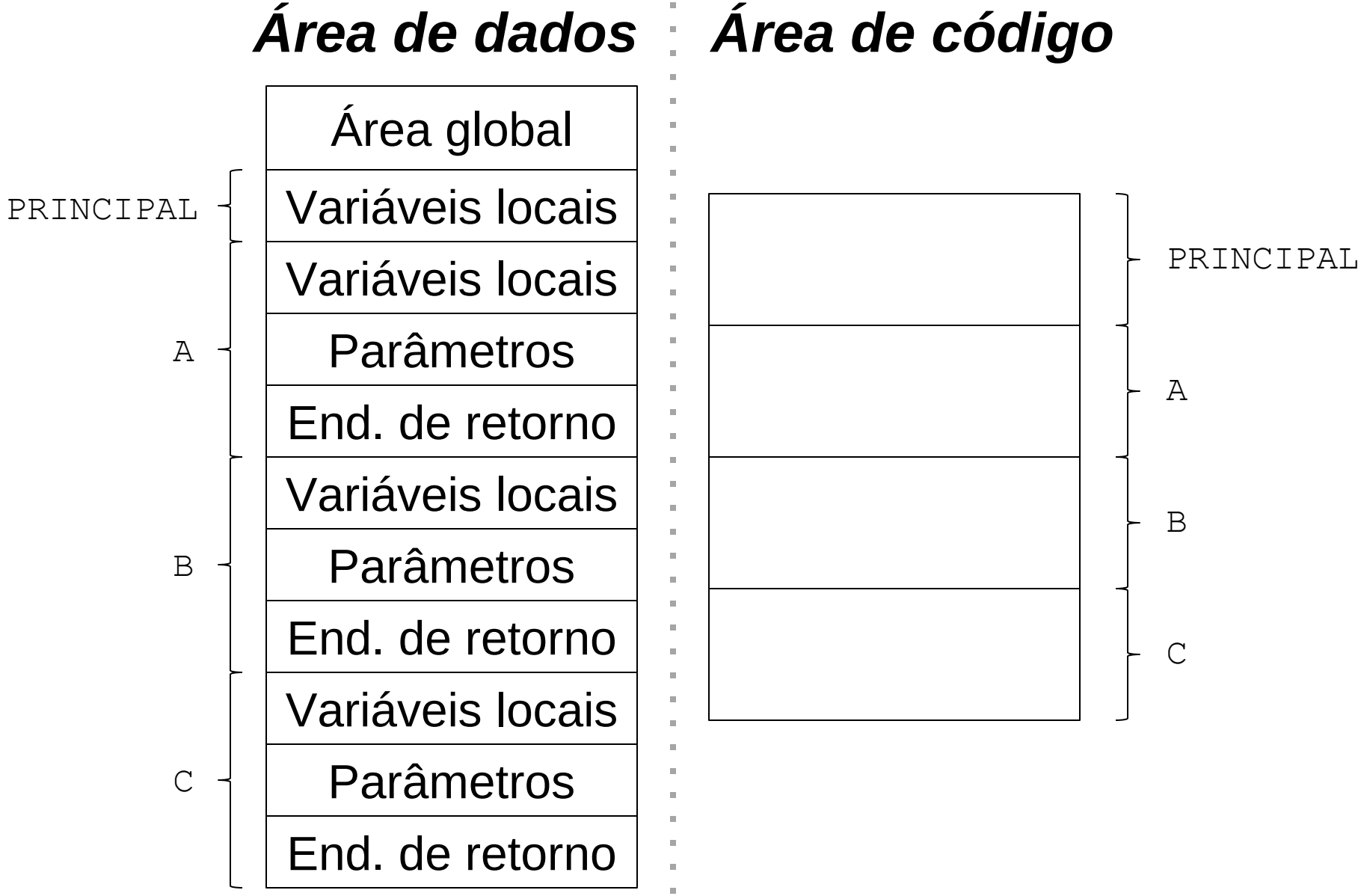
- Área de código – conteúdo e tamanho fixos
- Área de dados
  - Conteúdo dinâmico
  - Tamanho fixo (calculado pelo compilador)
  - Registro de ativação
    - Formato da área de dados
  - Instância do registro de ativação
    - Dados do registro propriamente ditos
  - Em FORTRAN, não há recursividade
    - Portanto, só pode haver uma **única instância do R.A.** para cada subprograma ativo

FORTRAN 77

- Exemplo de um registro de ativação
  - O status de execução do chamador será omitido daqui em diante



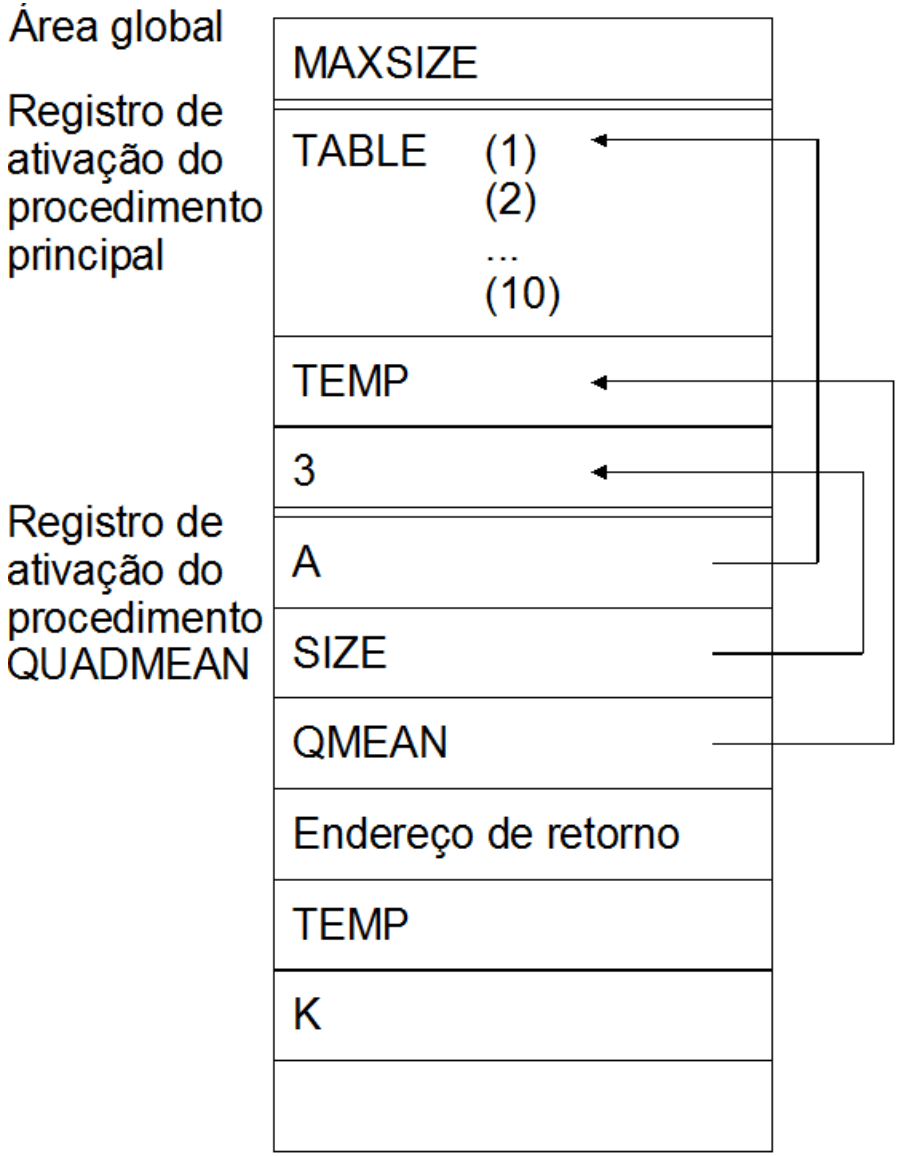
Exemplo


Área de dados
Área de código
Área global
PRINCIPAL
Variáveis locais
Variáveis locais
A
Parâmetros
End. de retorno
Variáveis locais
B
Parâmetros
End. de retorno
Variáveis locais
C
Parâmetros
End. de retorno
PRINCIPAL
A
B
C

FORTRAN 77

- Na prática, os trechos de código podem estar em outro local
  - Junto com as instâncias dos R.A.
- As unidades podem ser compiladas separadamente
  - E depois unidas pelo ligador (linker)
    - Que precisa “remendar” os endereços
  - Endereçamento relativo é também utilizado
    - Funcionalidade do sistema operacional
    - Pois o programa pode estar em qualquer local da memória

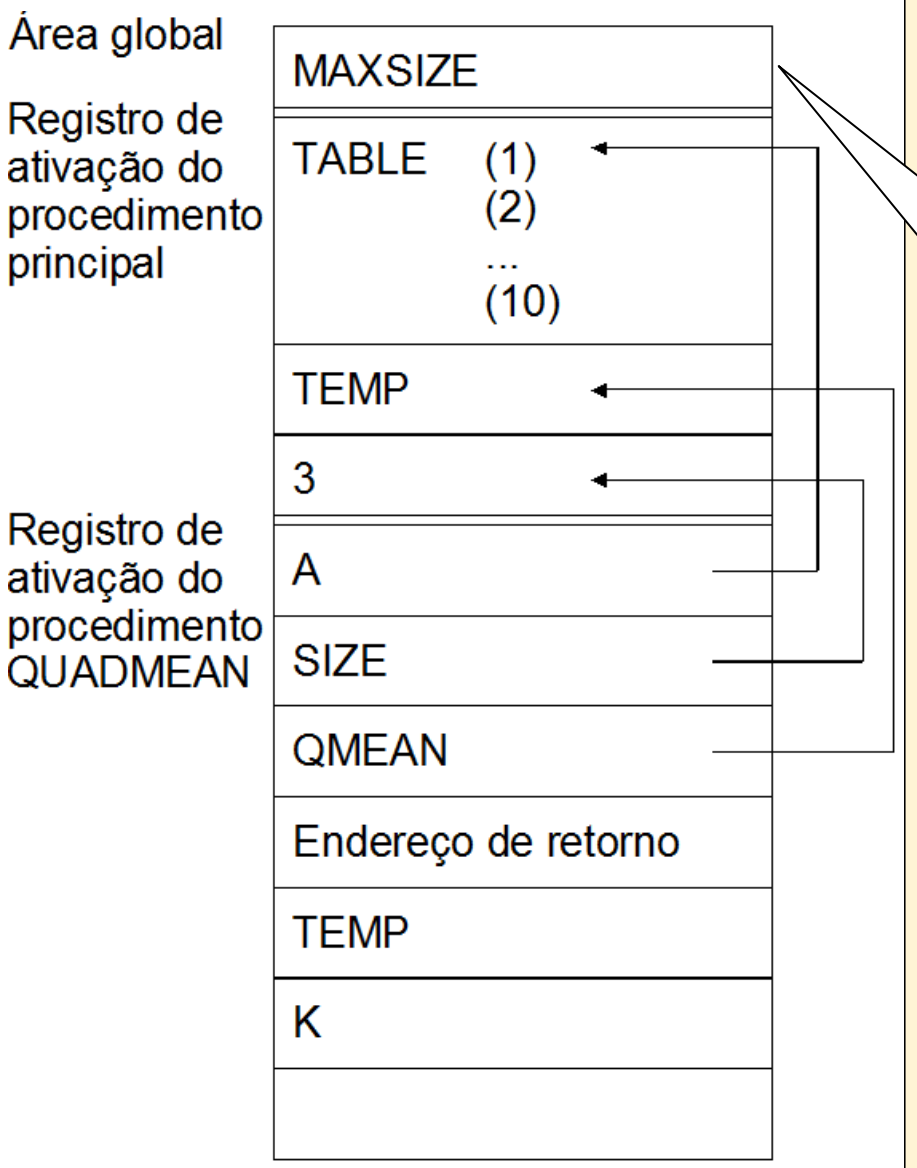
Exemplo: FORTRAN77




```
PROGRAM TEST
COMMON MAXSIZE
INTEGER MAXSIZE
REAL TABLE(10), TEMP
MAXSIZE=10
READ *, TABLE(1), TABLE(2), TABLE(3)
CALL QUADMEAN(TABLE,3,TEMP)
PRINT *, TEMP
END

SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
COMMON MAXSIZE
INTEGER MAXSIZE, SIZE
REAL A(SIZE), QMEAN, TEMP
INTEGER K
TEMP=0.0
IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO
99
DO 10 K=1, SIZE
            TEMP=TEMP+A(K)*A(K)
10 CONTINUE
99 QMEAN=SQRT(TEMP/SIZE)
RETURN
END
```

Exemplo: FORTRAN77

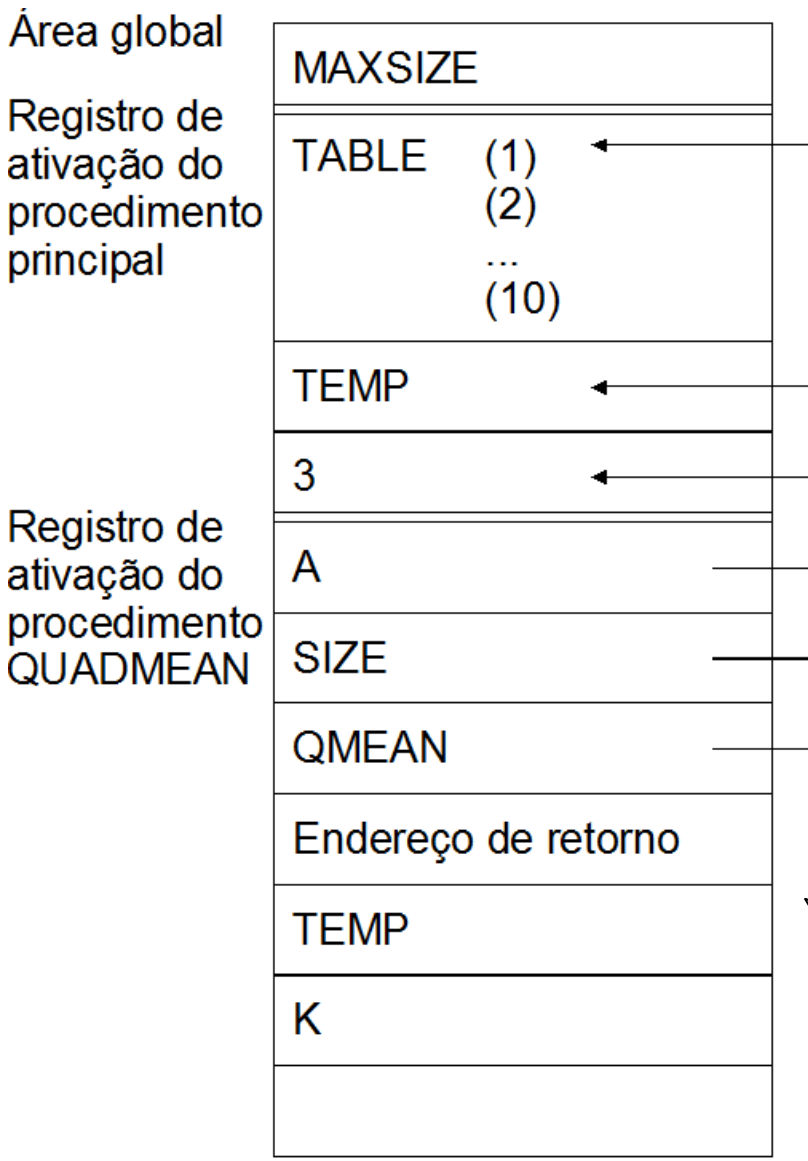



```
PROGRAM TEST
COMMON MAXSIZE
INTEGER MAXSIZE
REAL TABLE(10), TEMP
MAXSIZE=10
AD *, TABLE(1), TABLE(2), TABLE(3)
UADMEAN(TABLE,3,TEMP)
EMP
END

IZE,QMEAN)
INTEGER SIZE
REAL A(SIZE), QMEAN, TEMP
IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO
99
DO 10 K=1, SIZE
        TEMP=TEMP+A(K)*A(K)
10 CONTINUE
99 QMEAN=SQRT(TEMP/SIZE)
RETURN
END
```

A área global armazena a única variável global definida nesse trecho de código

Exemplo: FORTRAN77




```
PROGRAM TEST
COMMON MAXSIZE
INTEGER MAXSIZE
REAL TABLE(10), TEMP
MAXSIZE=10
READ *, TABLE(1), TABLE(2), TABLE(3)
CALL QUADMEAN(TABLE,3,TEMP)
PRINT *, TEMP
END

SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
COMMON MAXSIZE
INTEGER MAXSIZE, SIZE
```

Valores dos parâmetros são referências de memória.
Consequências:
- referência adicional para se acessar o valor de parâmetro
- matrizes não precisam ser copiadas, basta referenciar sua base
- constantes precisam ser armazenadas

Ambientes de execução totalmente estáticos

- Limitações
  - Estruturas criadas dinamicamente
    - Não há mecanismo para alocação de memória em tempo de execução
  - Procedimentos recursivos
    - Necessitam de mais de um registro de ativação criado no momento de sua chamada
    - A cada ativação, variáveis locais dos procedimentos recebem novas posições de memória
    - Em um dado momento, vários registros de ativação de um mesmo procedimento podem existir na memória

ALGOL e linguagens derivadas

- Projeto do ALGOL prevê algumas funcionalidades adicionais às do FORTRAN
  - Passagem por valor ou por referência
  - Alocação dinâmica de variáveis locais
  - Recursividade
    - Ativações múltiplas de um subprograma
    - Mais de uma instância do R.A. para um mesmo subprograma
    - Cada instância precisa de sua própria cópia das variáveis locais
  - Acesso a variáveis não-locais (e não-globais)
    - Escopo estático / blocos aninhados

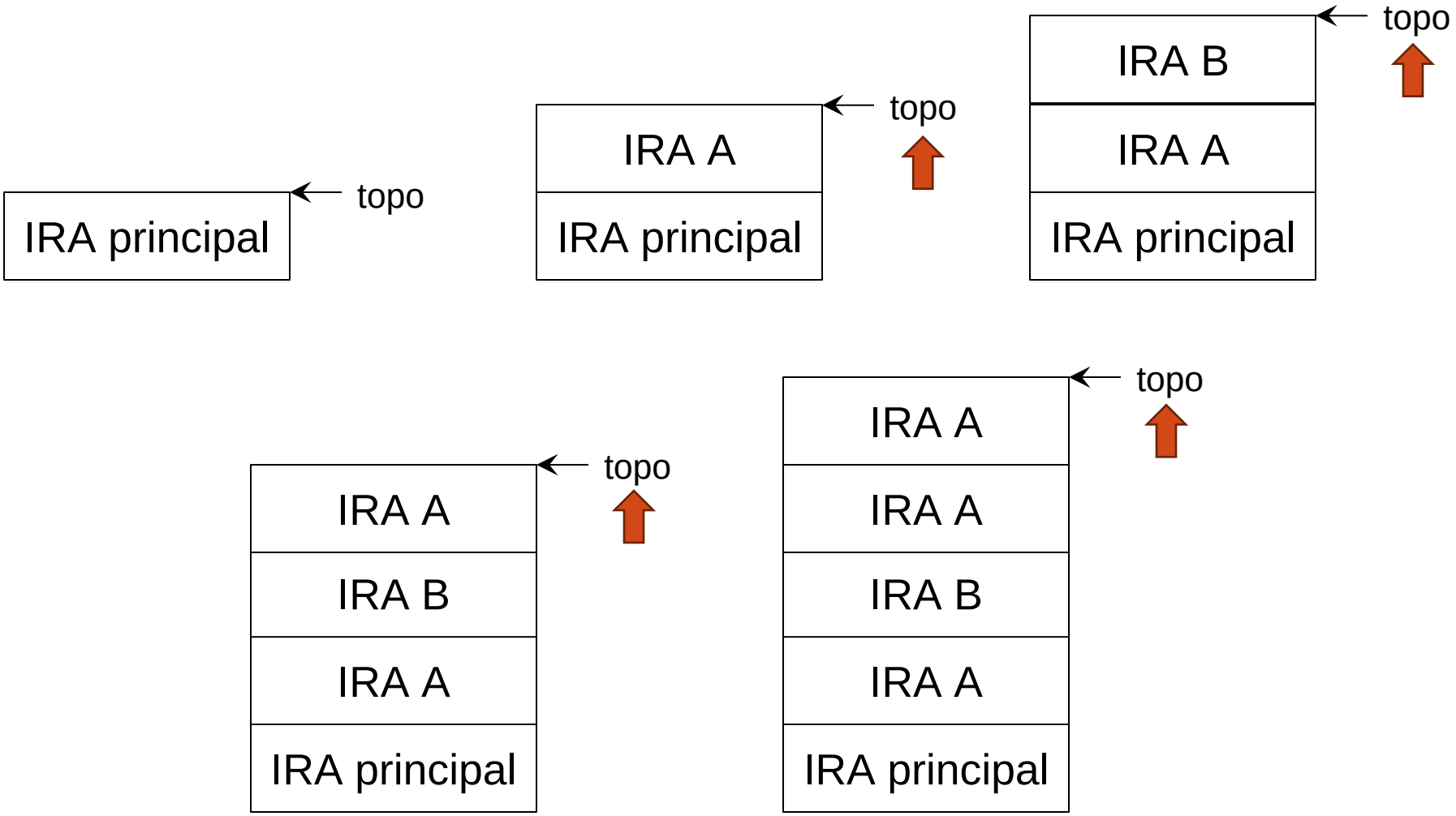
ALGOL e linguagens derivadas

- Múltiplas chamadas a um mesmo procedimento
- Modelo semântico
  - Último a ser chamado é o primeiro a terminar
  - LIFO = modelo pilha
- Ambientes dinâmicos baseados em pilha
  - Instâncias dos registros de ativação são criadas dinamicamente
  - E inseridas em uma estrutura de pilha
  - A cada chamada, uma nova I.R.A. é adicionada no topo da pilha

IRA = Instância do Registro de Ativação

ALGOL e linguagens derivadas

- Ex: Principal → A → B → A → A

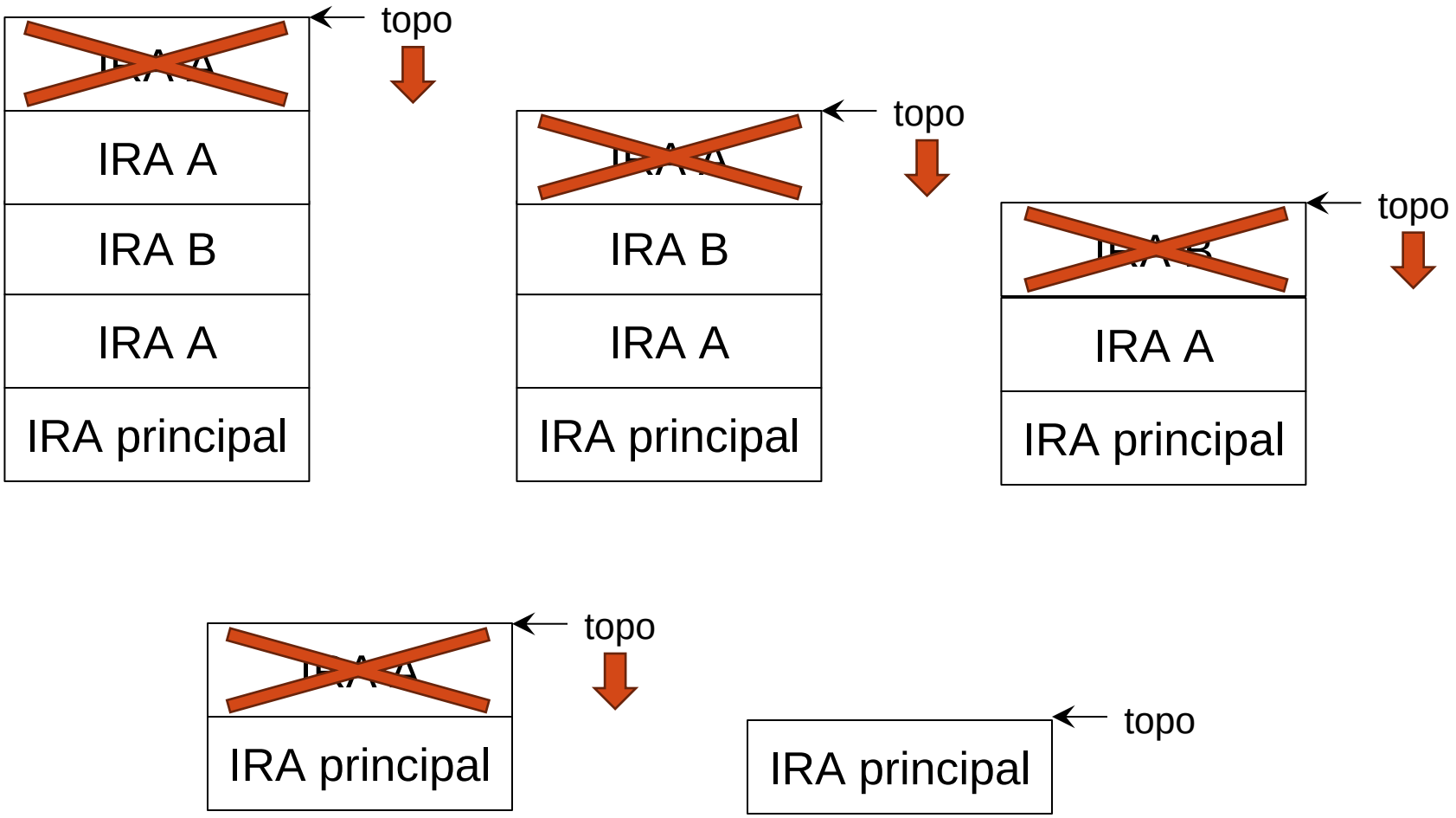


ALGOL e linguagens derivadas

- A cada retorno
  - A IRA correspondente é desalocada
  - Antes do controle ser devolvido à unidade chamadora
    - Sua IRA precisa ser restaurada
- Para implementar essa funcionalidade
  - Basta eliminar o topo da pilha
- Importante
  - Endereço de retorno aponta para **área de código** da unidade chamadora
  - A IRA abaixo do topo da pilha é a **área de dados** da unidade chamadora

ALGOL e linguagens derivadas

- Ex: Principal → A → B → A → A



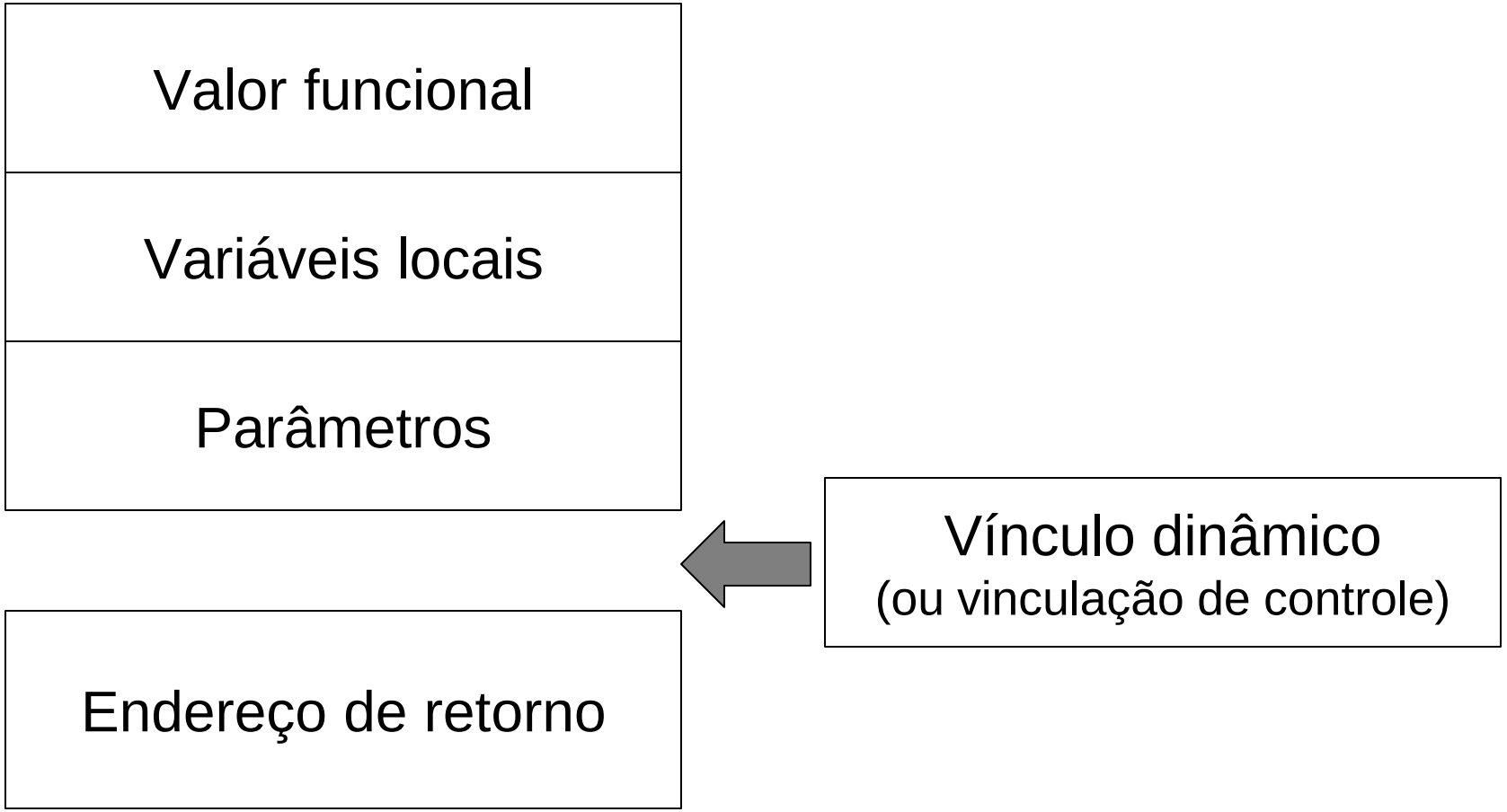
ALGOL e linguagens derivadas

- Problema 1
  - Cada chamada de subprograma causa um empilhamento de uma IRA
  - Ao encerrar um subprograma, a IRA criada é desalocada (elemento é retirado da pilha)
  - Como retornar à IRA do subprograma chamador?
    - Que pode ser o mesmo
- Solução (parcial)
  - A princípio, o tamanho dos IRA é conhecido pelo compilador
    - Ou seja, o compilador sabe o tamanho necessário para cada variável local e parâmetro de um subprograma
  - Portanto, basta subtrair o tamanho da IRA (conhecido) do ponteiro de topo da pilha

ALGOL e linguagens derivadas

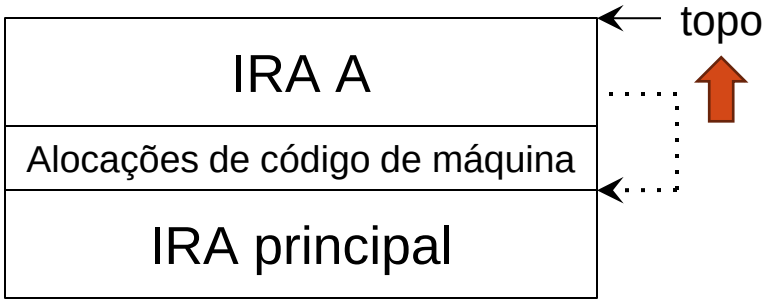
- Problema com a solução anterior
  - Podem existir outras alocações na pilha
    - Valores temporários que são inseridos pela versão em linguagem de máquina do subprograma
    - Detalhes do código gerado pelo compilador
  - Ou seja, apenas subtrair o tamanho do topo não é suficiente
- Solução
  - Utilizar um ponteiro específico para isso
    - Vínculo dinâmico
    - Vinculação de controle

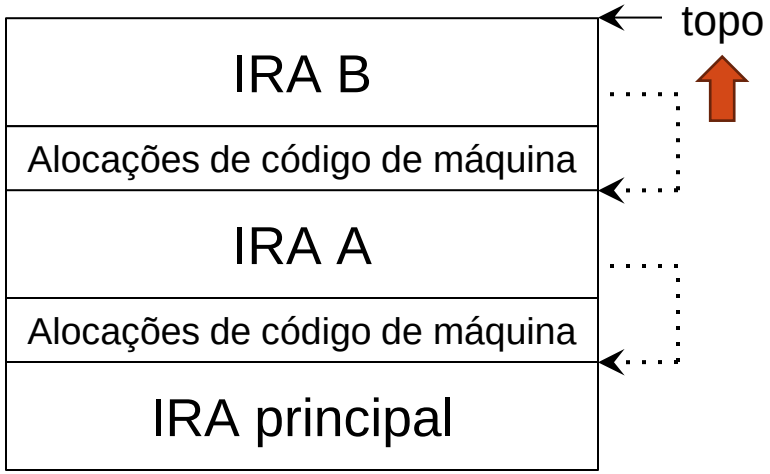
ALGOL e linguagens derivadas



ALGOL e linguagens derivadas

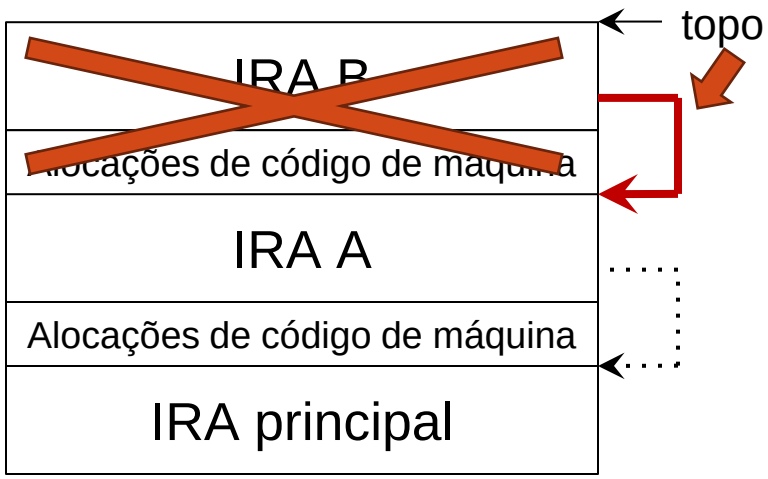
- Ex: Principal → A → B

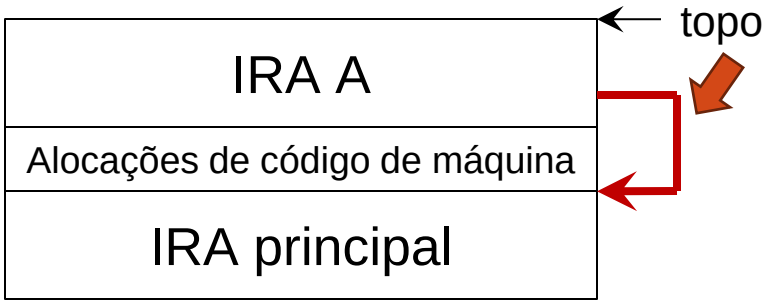




ALGOL e linguagens derivadas

- Ex: Principal → A → B

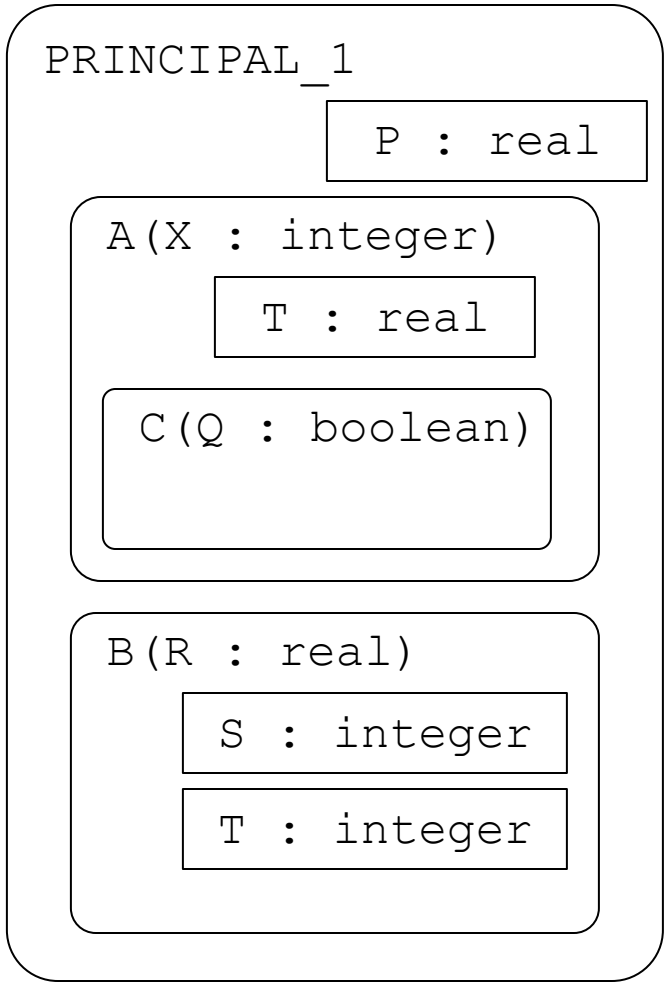




Exemplo

```
program PRINCIPAL_1;
  var P : real;
  procedure A(X : integer);
    var T : boolean;
    procedure C(Q : boolean);
      begin { C }
      ...
      end;  { C }
    begin   { A }
    ...
    C(Y);
    ...
    end;    { A }
  procedure B(R : real);
    var S, T : integer;
    begin   { B }
    ...
    A(S);
    ...
    end;    { B }
  begin     { PRINCIPAL_1 }
  ...
  B(P);
  ...
  end.      { PRINCIPAL_1 }
```

Aninhamento de subprogramas:



Relação chama-chamado:

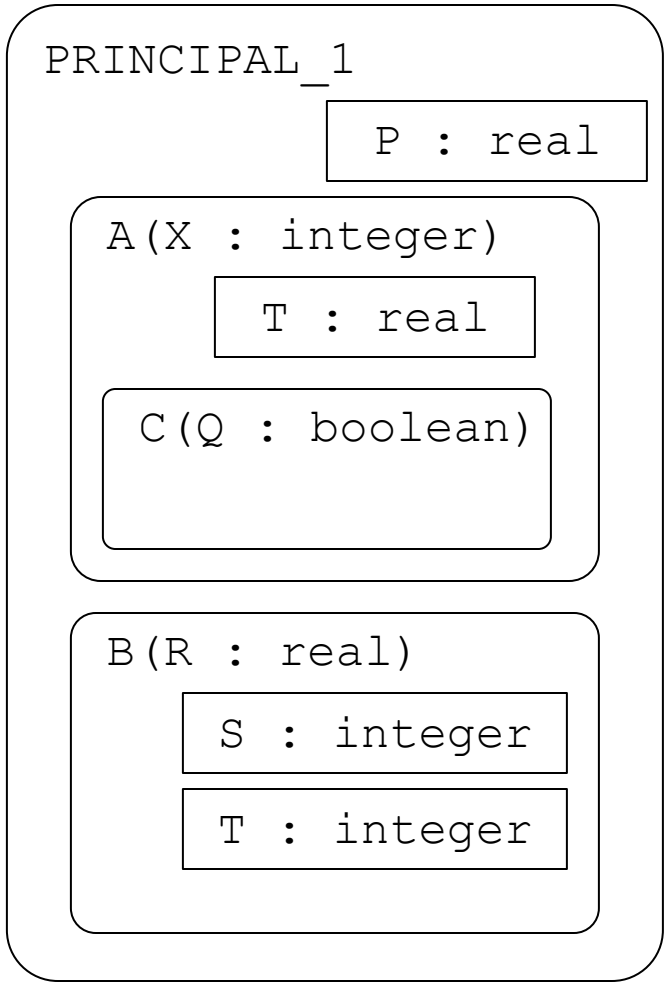
PRINCIPAL_1 → B → A → C

Exemplo

```
program PRINCIPAL_1;
  var P : real;
  procedure A(X : integer);
    var T : boolean;
    procedure C(Q : boolean);
      begin { C }
      ...
      end;  { C }
    begin   { A }
    ...
    C(Y);
    ...
    end;    { A }
  procedure B(R : real);
    var S, T : integer;
    begin   { B }
    ...
    A(S);
    ...
    end;    { B }
  begin     { PRINCIPAL_1 }
  ...
  B(P);
  ...
  end.      { PRINCIPAL_1 }
```

1

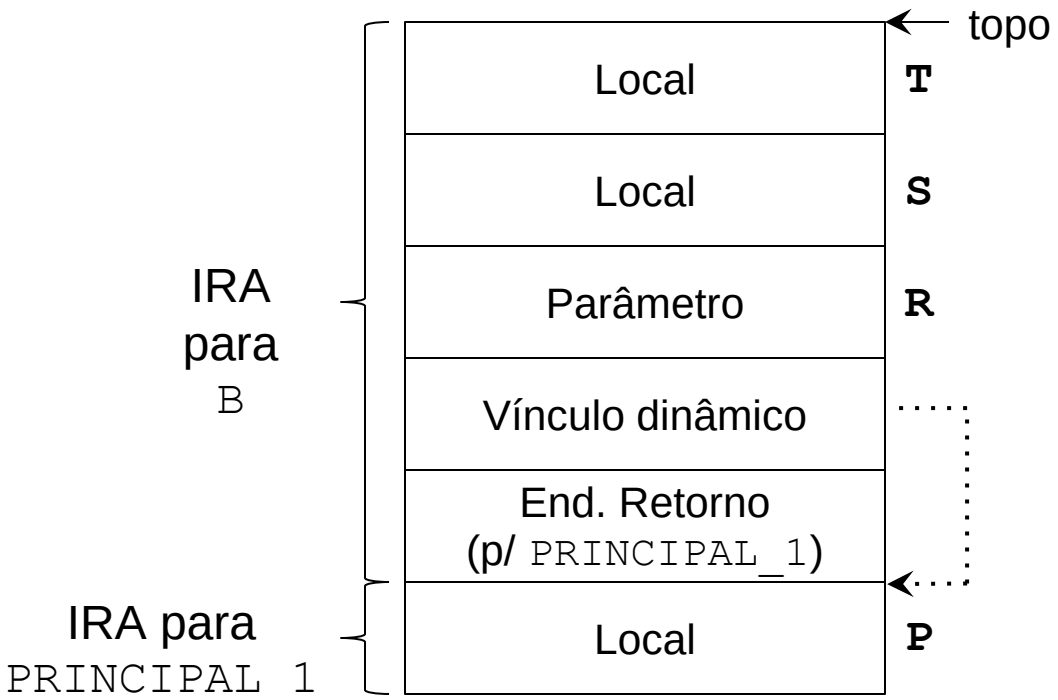
Aninhamento de subprogramas:



Relação chama-chamado:

PRINCIPAL_1 → B → A → C

Exemplo

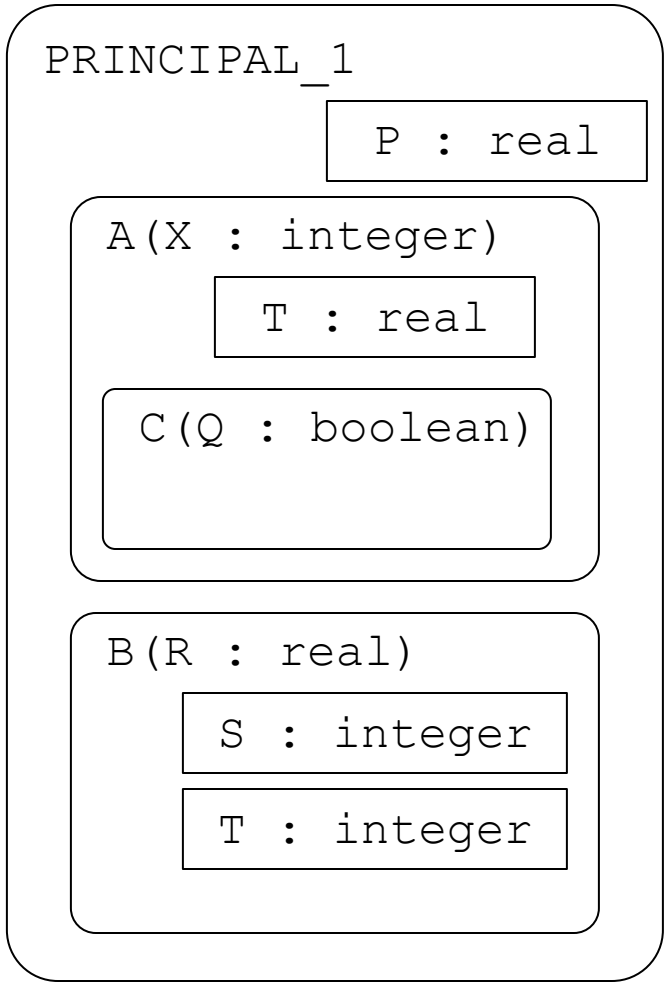

topo
Local
T
Local
S
Parâmetro
R
Vínculo dinâmico
End. Retorno
(p/ PRINCIPAL_1)
Local
P
IRA
para
B
IRA para
PRINCIPAL_1

Exemplo

```
program PRINCIPAL_1;
  var P : real;
  procedure A(X : integer);
    var T : boolean;
    procedure C(Q : boolean);
      begin { C }
      ...
      end;  { C }
    begin   { A }
    ...
    C(Y);
    ...
    end;    { A }
  procedure B(R : real);
    var S, T : integer;
    begin   { B }
    ...
    A(S);
    ...
    end;    { B }
  begin     { PRINCIPAL_1 }
  ...
  B(P);
  ...
  end.      { PRINCIPAL_1 }
```

2

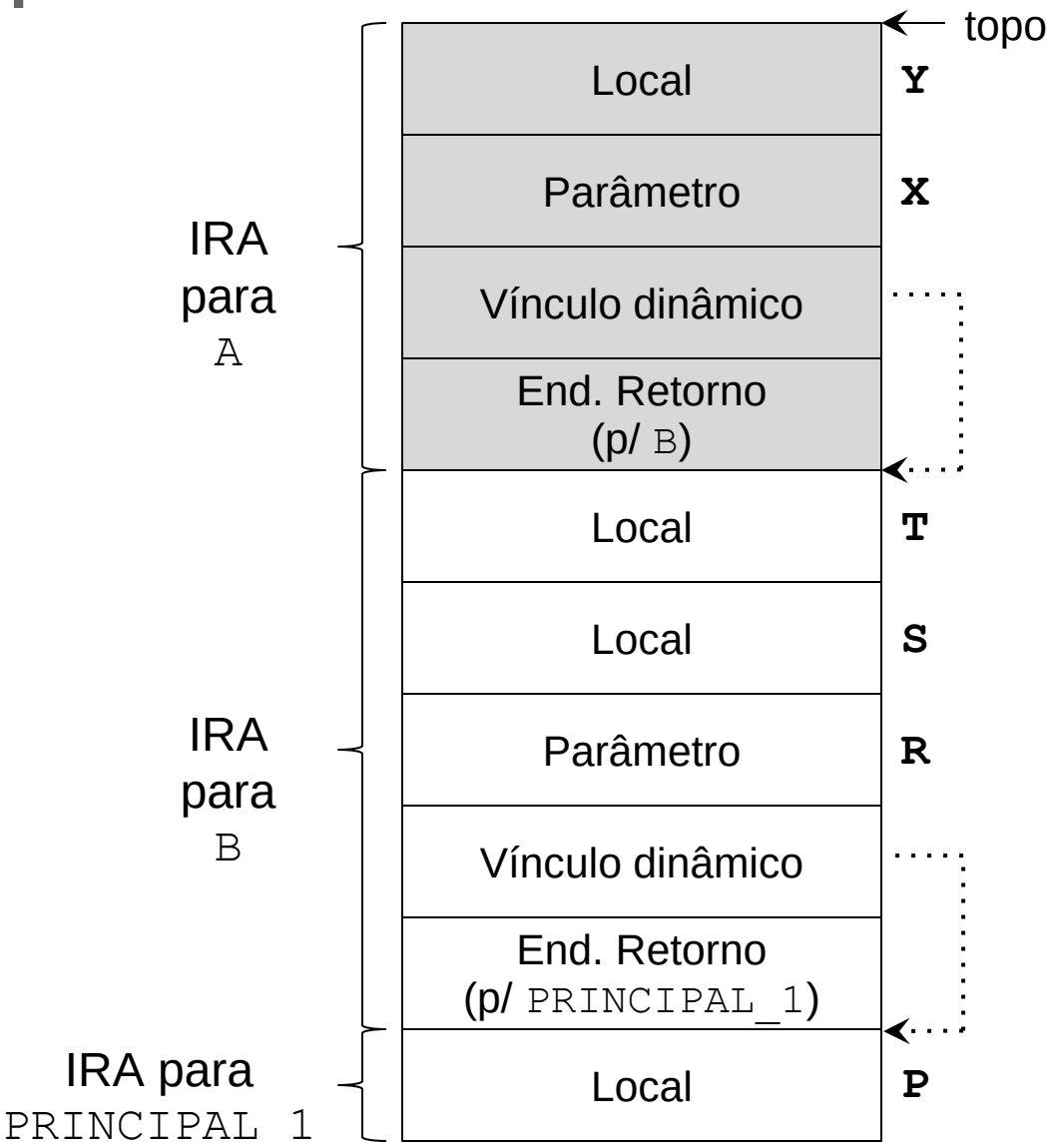
Aninhamento de subprogramas:



Relação chama-chamado:

PRINCIPAL_1 → B → A → C

Exemplo

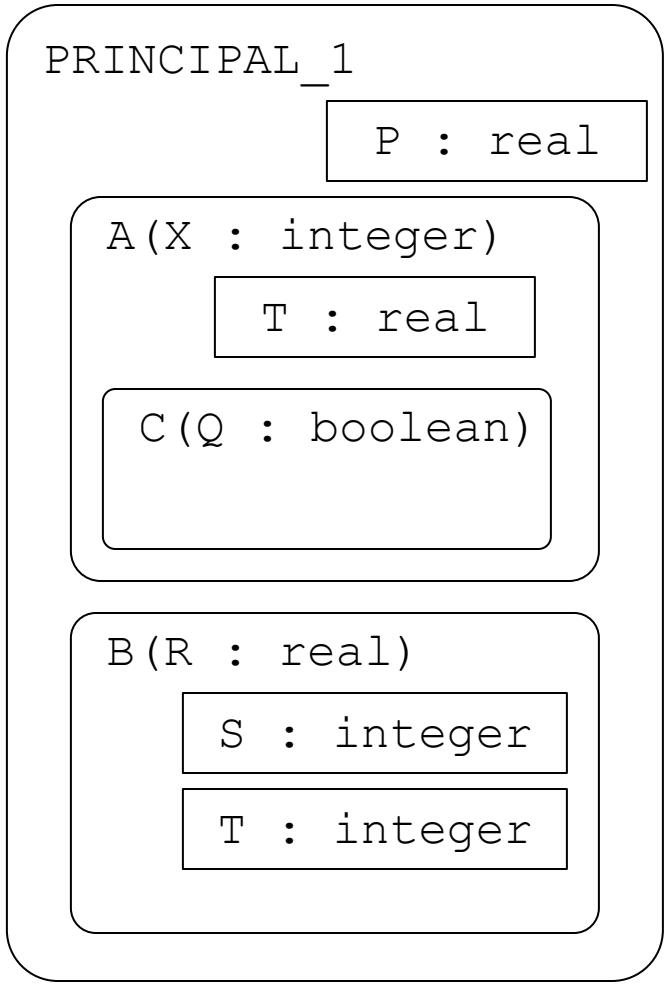

topo
Local
Y
Parâmetro
X
IRA
para
A
Vínculo dinâmico
End. Retorno
(p/ B)
Local
T
Local
S
Parâmetro
R
IRA
para
B
Vínculo dinâmico
End. Retorno
(p/ PRINCIPAL_1)
IRA para
PRINCIPAL_1
Local
P

Exemplo

```
program PRINCIPAL_1;
  var P : real;
  procedure A(X : integer);
    var T : boolean;
    procedure C(Q :
boolean);
      begin { C }
      ...
      end;  { C }
    begin   { A }
    ...
    C(Y);
    ...
    end;    { A }
  procedure B(R : real);
    var S, T : integer;
    begin   { B }
    ...
    A(S);
    ...
    end;    { B }
  begin     { PRINCIPAL_1 }
  ...
  B(P);
  ...
  end.      { PRINCIPAL_1 }
```

3

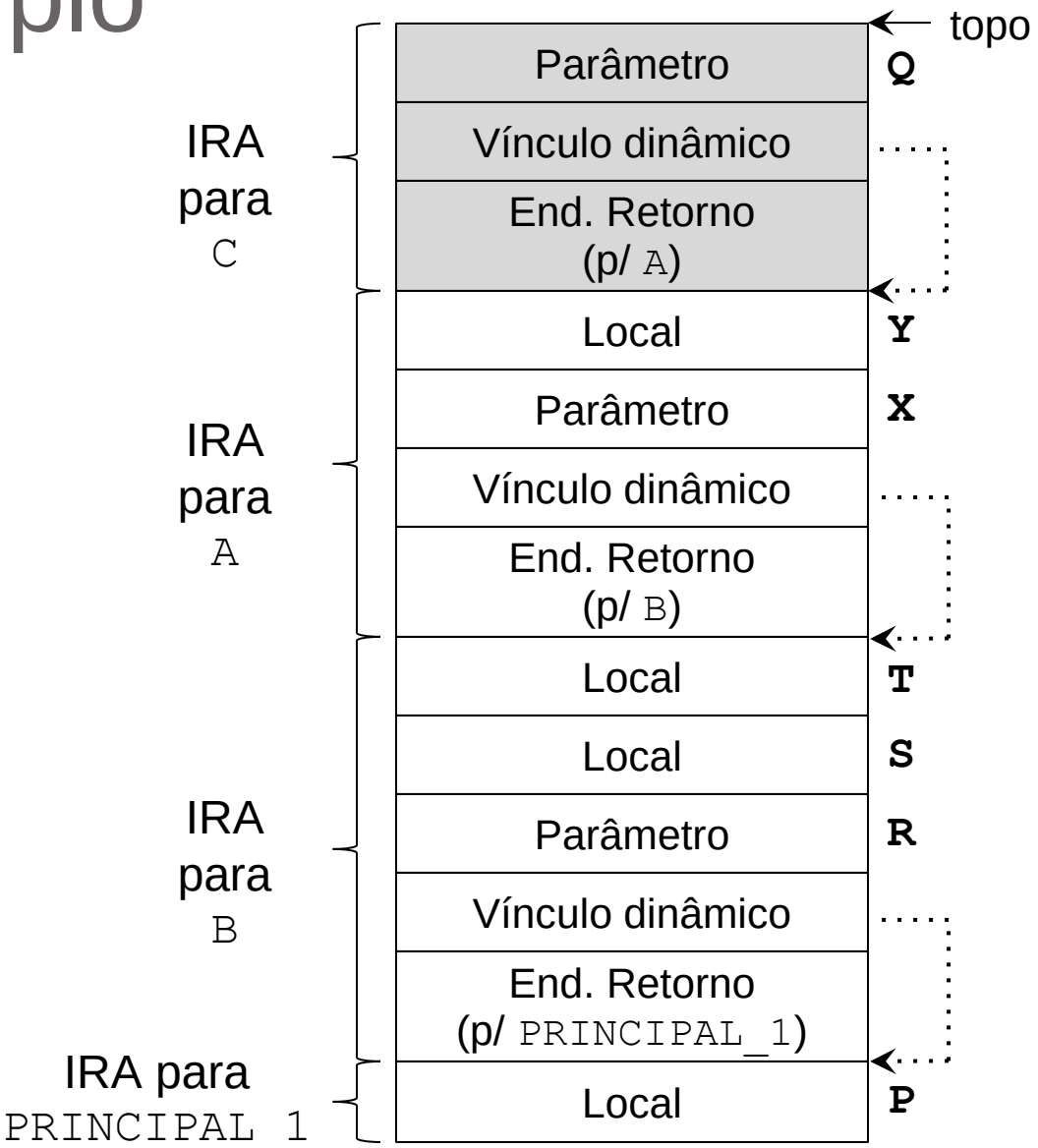
Aninhamento de subprogramas:



Relação chama-chamado:

PRINCIPAL_1 → B → A → C

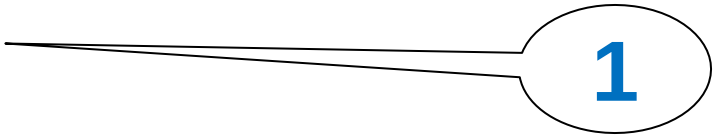
Exemplo


topo
Parâmetro
Q
Vínculo dinâmico
End. Retorno
(p/ A)
IRA
para
C
Local
Y
Parâmetro
X
Vínculo dinâmico
End. Retorno
(p/ B)
IRA
para
A
Local
T
Local
S
Parâmetro
R
Vínculo dinâmico
End. Retorno
(p/ PRINCIPAL_1)
IRA
para
B
Local
P
IRA para
PRINCIPAL_1

Exemplo

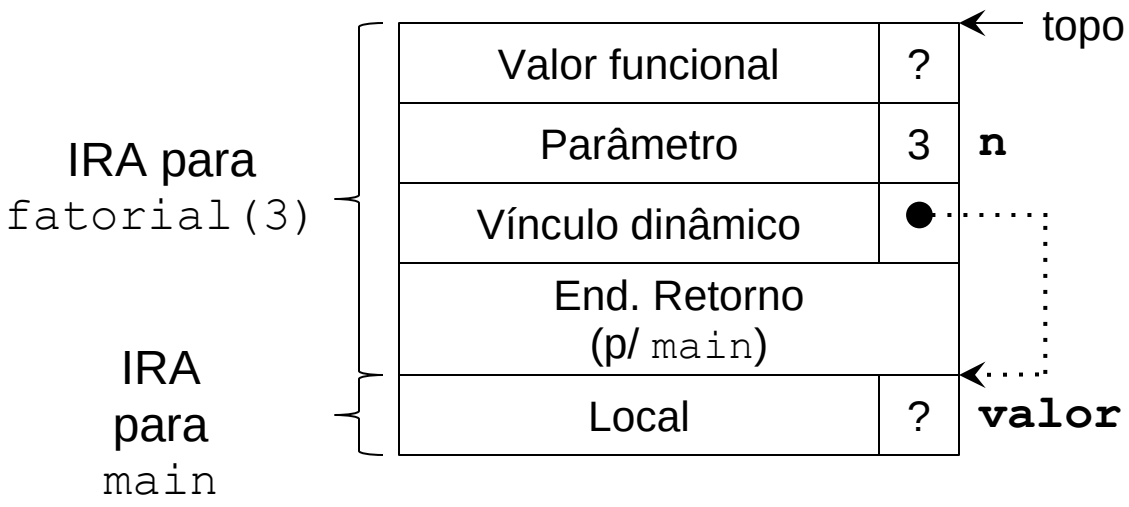
```
int fatorial(int n) {
  if (n <= 1)
    return 1;
  else return (n * fatorial(n - 1));
}

void main() {
  int valor;
  valor = fatorial(3);
}
```



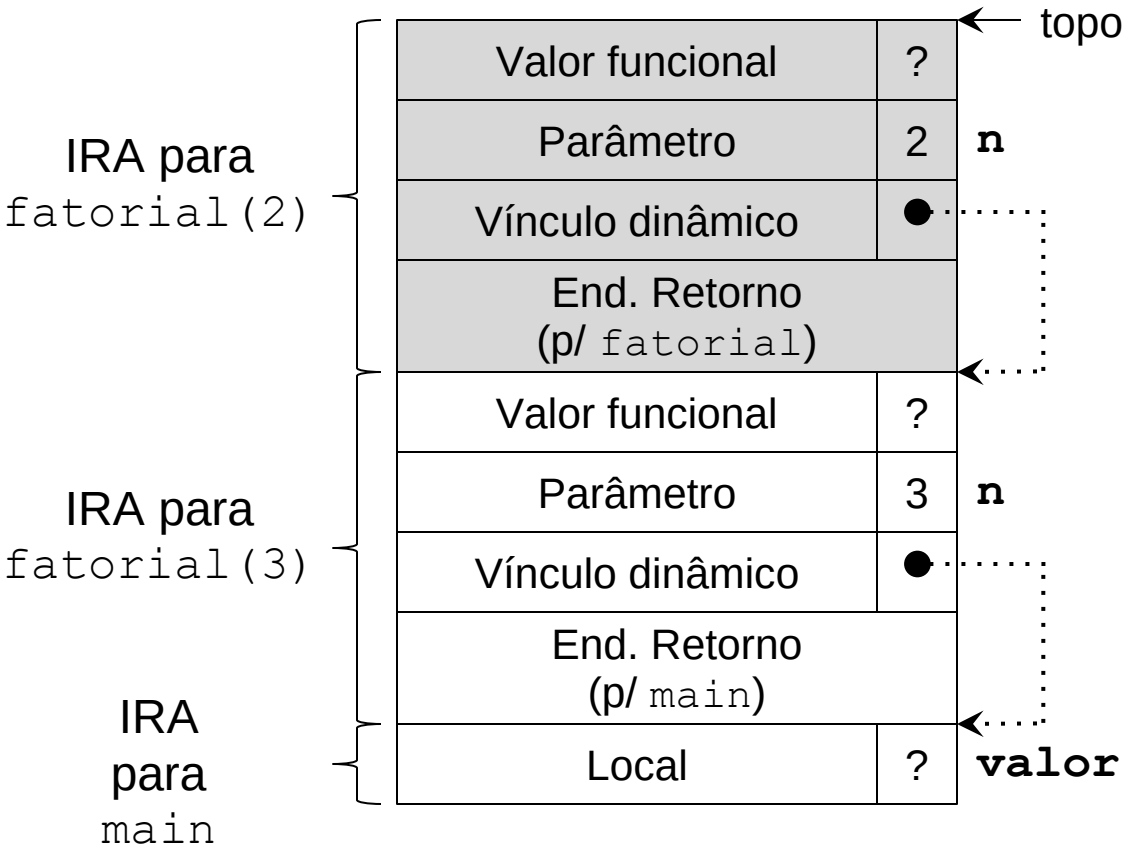
Exemplo

- Primeira chamada



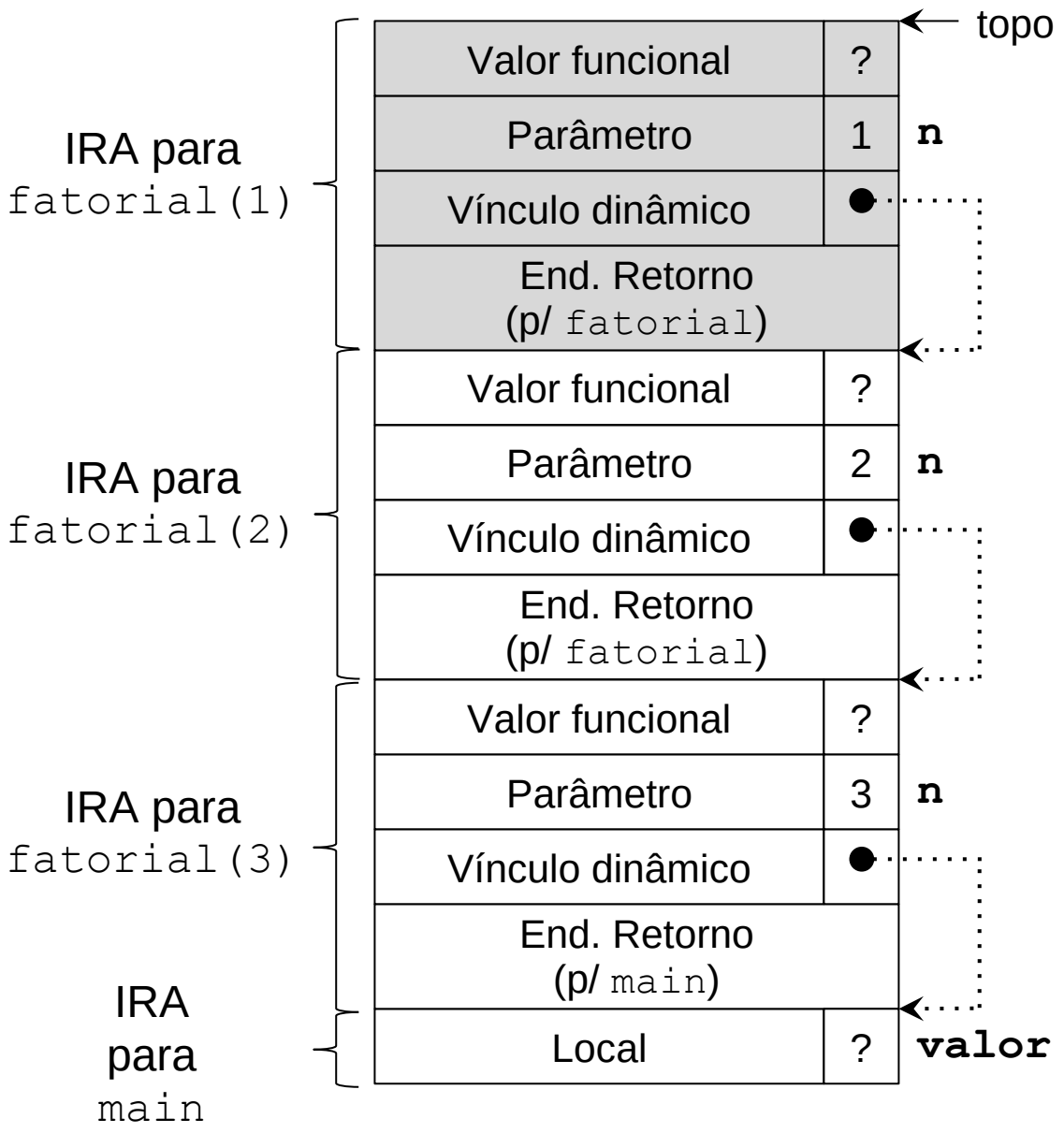
Exemplo

- Segunda chamada



Exemplo

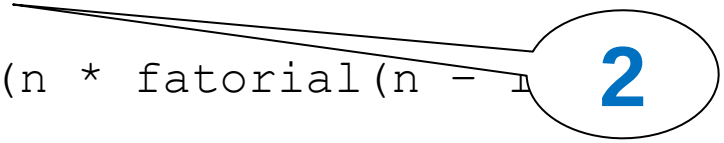
- Terceira chamada



Exemplo

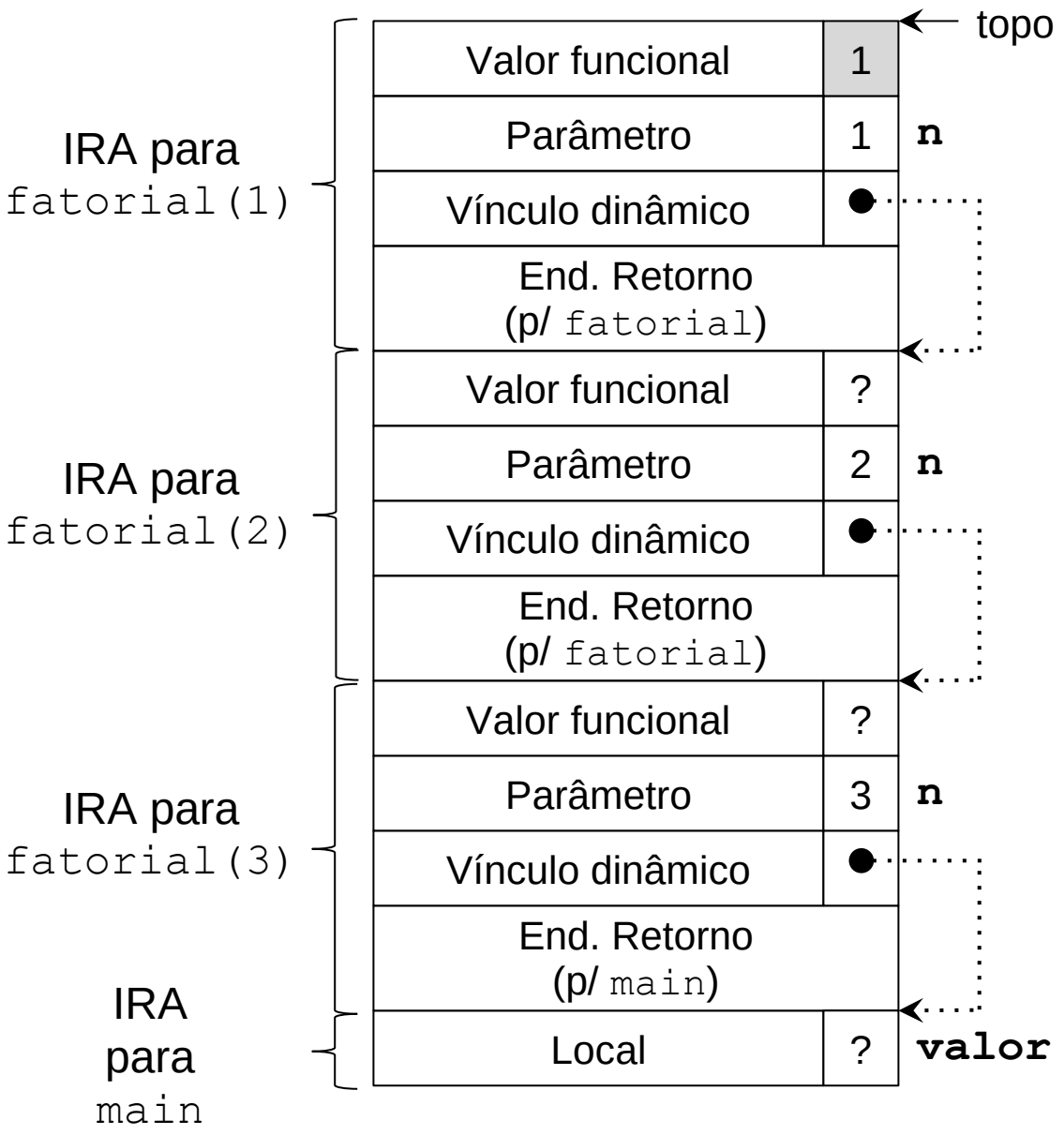
```
int fatorial(int n) {
  if (n <= 1)
    return 1;
  else return (n * fatorial(n - 1));
}

void main() {
  int valor;
  valor = fatorial(3);
}
```



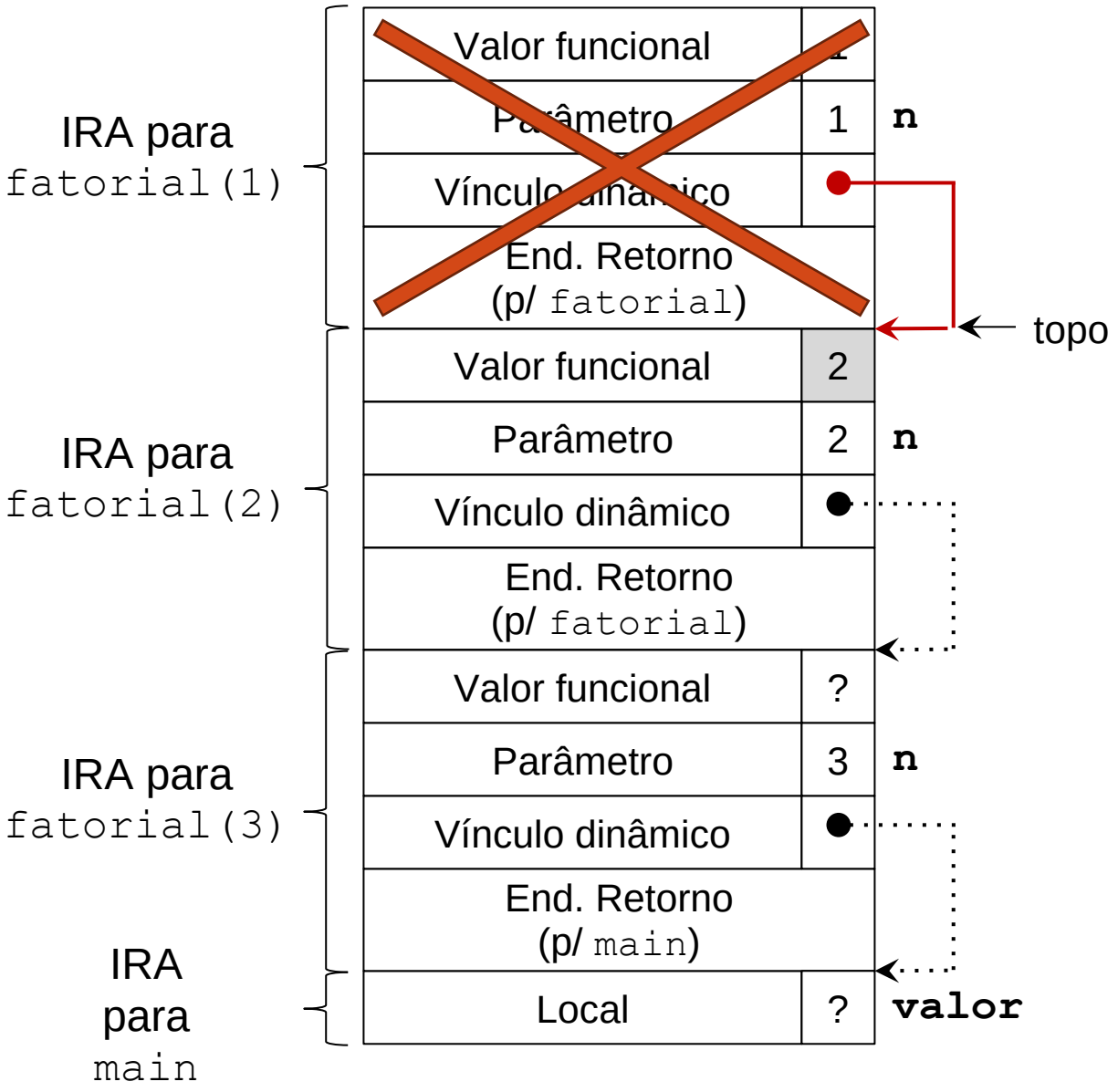
Exemplo

- Terceira chamada



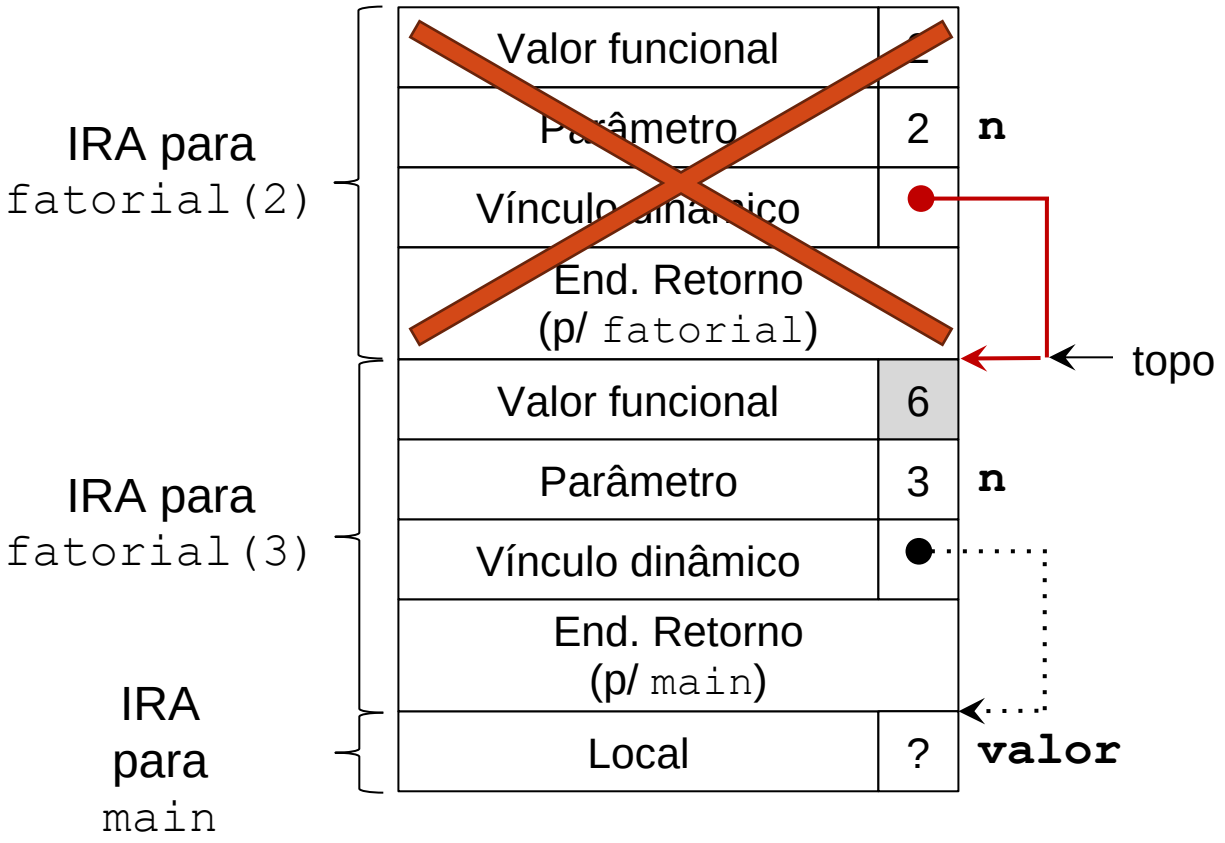
Exemplo

- Segunda chamada



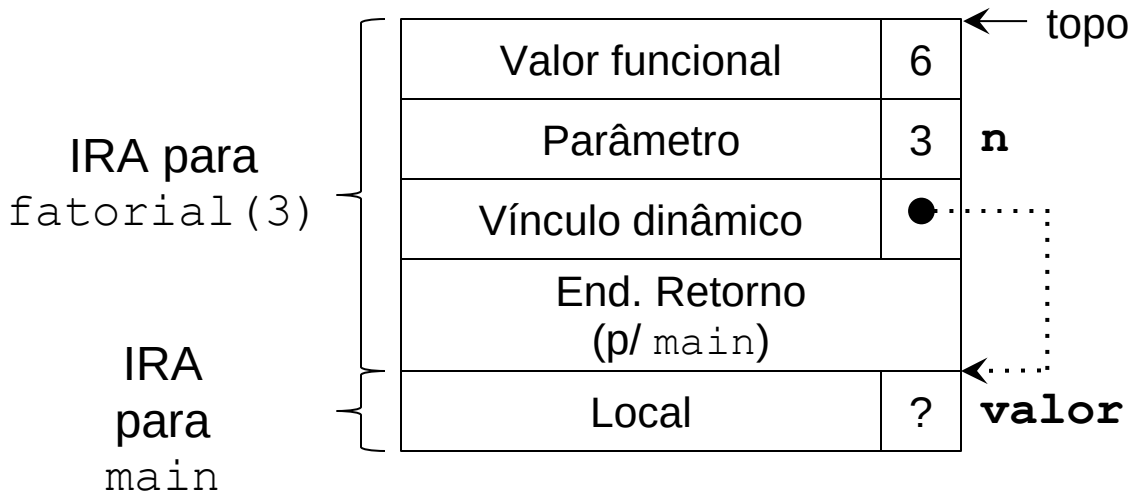
Exemplo

- Primeira chamada



Exemplo

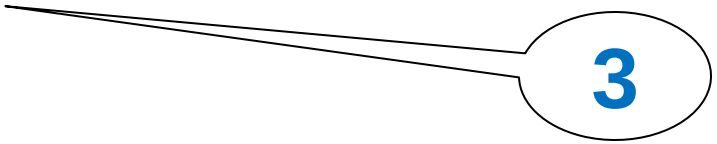
- Primeira chamada



Exemplo

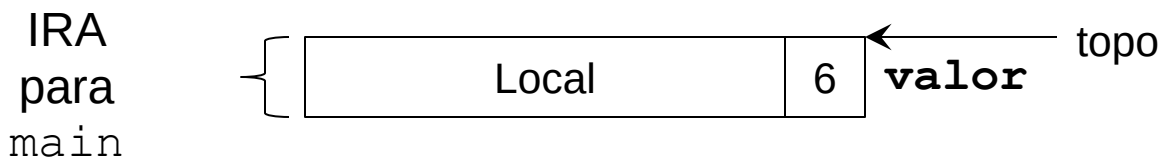
```
int fatorial(int n) {
  if (n <= 1)
    return 1;
  else return (n * fatorial(n - 1));
}

void main() {
  int valor;
  valor = fatorial(3);
}
```



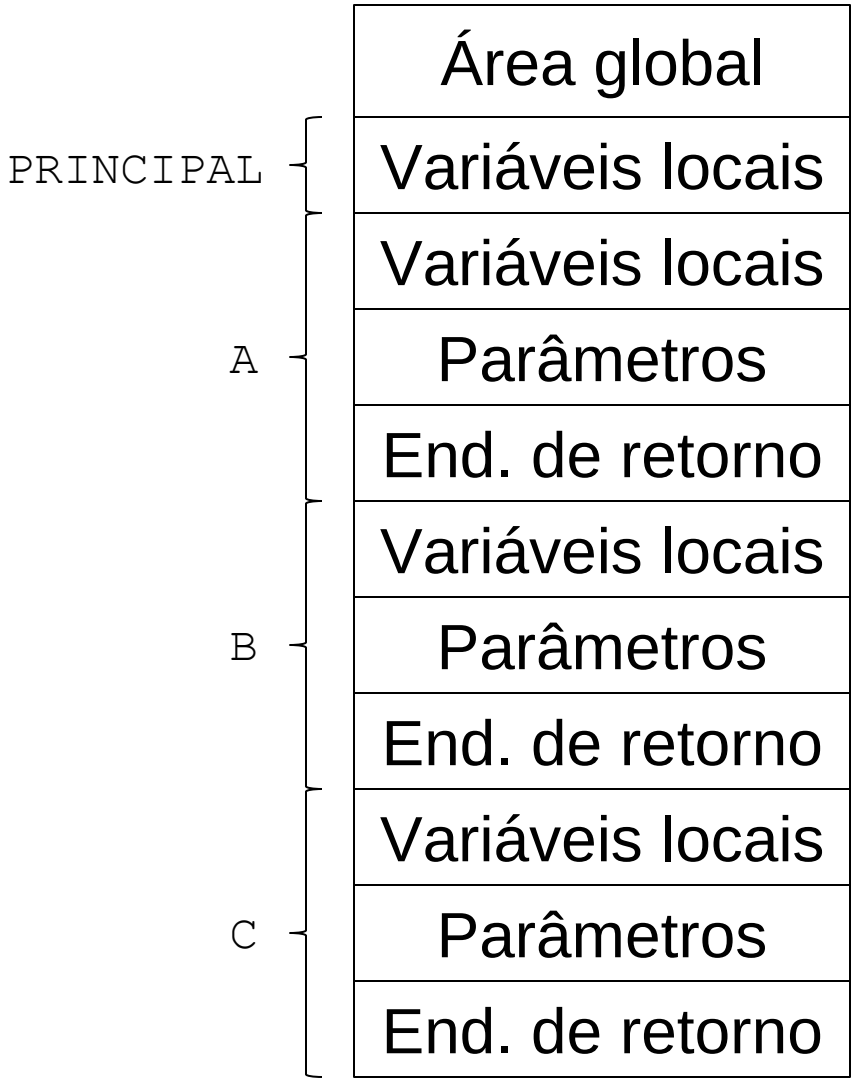
Exemplo

- Primeira chamada



ALGOL e linguagens derivadas

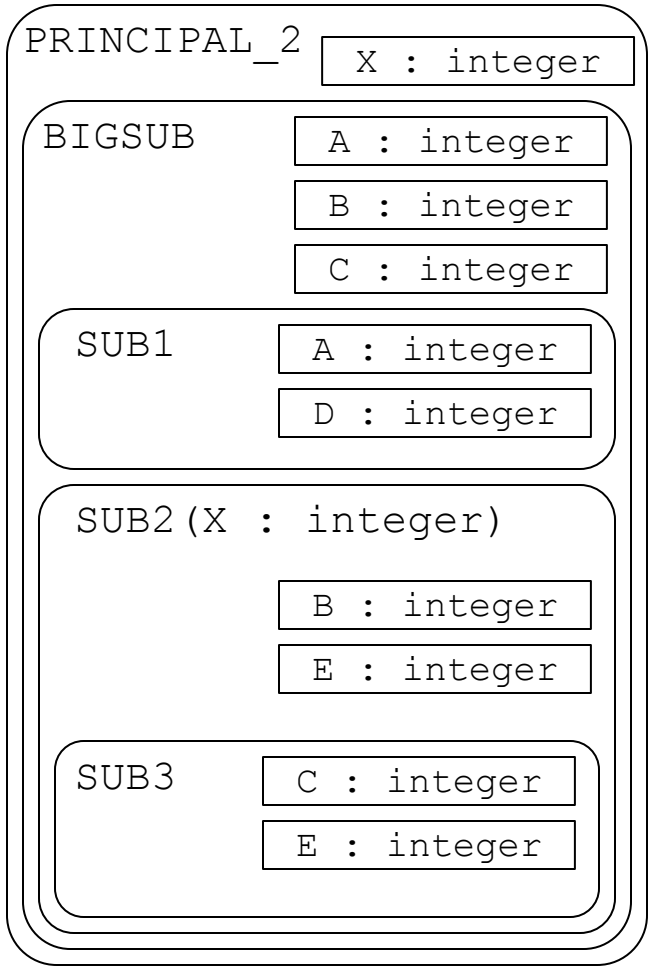
- Problema 2
  - Como implementar o acesso a variáveis não-locais?
  - Lembrando: no FORTRAN somente havia variáveis globais estaticamente alocadas



Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                          { SUB1 }
      A := B + C;
      end;                           { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin                        { SUB3 }
        SUB1;
        E := B + A;
        end;                         { SUB3 }
      begin                          { SUB2 }
      SUB3;
      A := D + E;
      end;                           { SUB2 }
    begin                            { BIGSUB }
    SUB2(7);
    end;                             { BIGSUB }
  begin                              { PRINCIPAL_2 }
  BIGSUB;
  end.                               { PRINCIPAL_2 }
```

Aninhamento de subprogramas:



Relação chama-chamado:

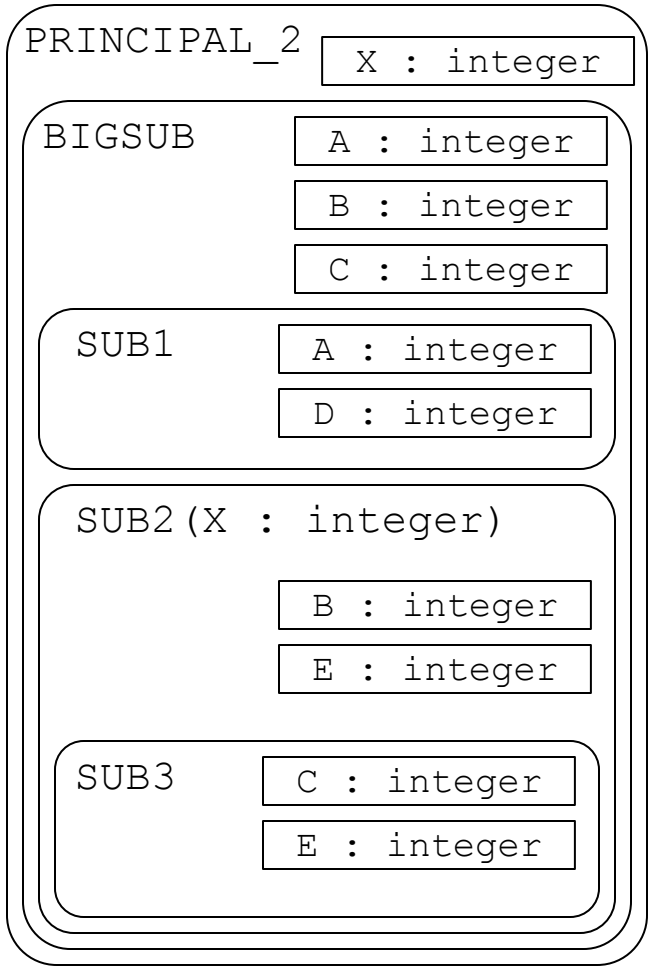
PRINCIPAL_2 → BIGSUB → SUB2(7) → SUB3 → SUB1

Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                          { SUB1 }
      A := B + C;
      end;                           { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin                        { SUB3 }
        SUB1;
        E := B + A;
        end;                         { SUB3 }
      begin                          { SUB2 }
      SUB3;
      A := D + E;
      end;                           { SUB2 }
    begin                            { BIGSUB }
    SUB2(7);
    end;                             { BIGSUB }
  begin                              {
PRINCIPAL_2 }
  BIGSUB;
  end.                               {
PRINCIPAL_2 }
```

1

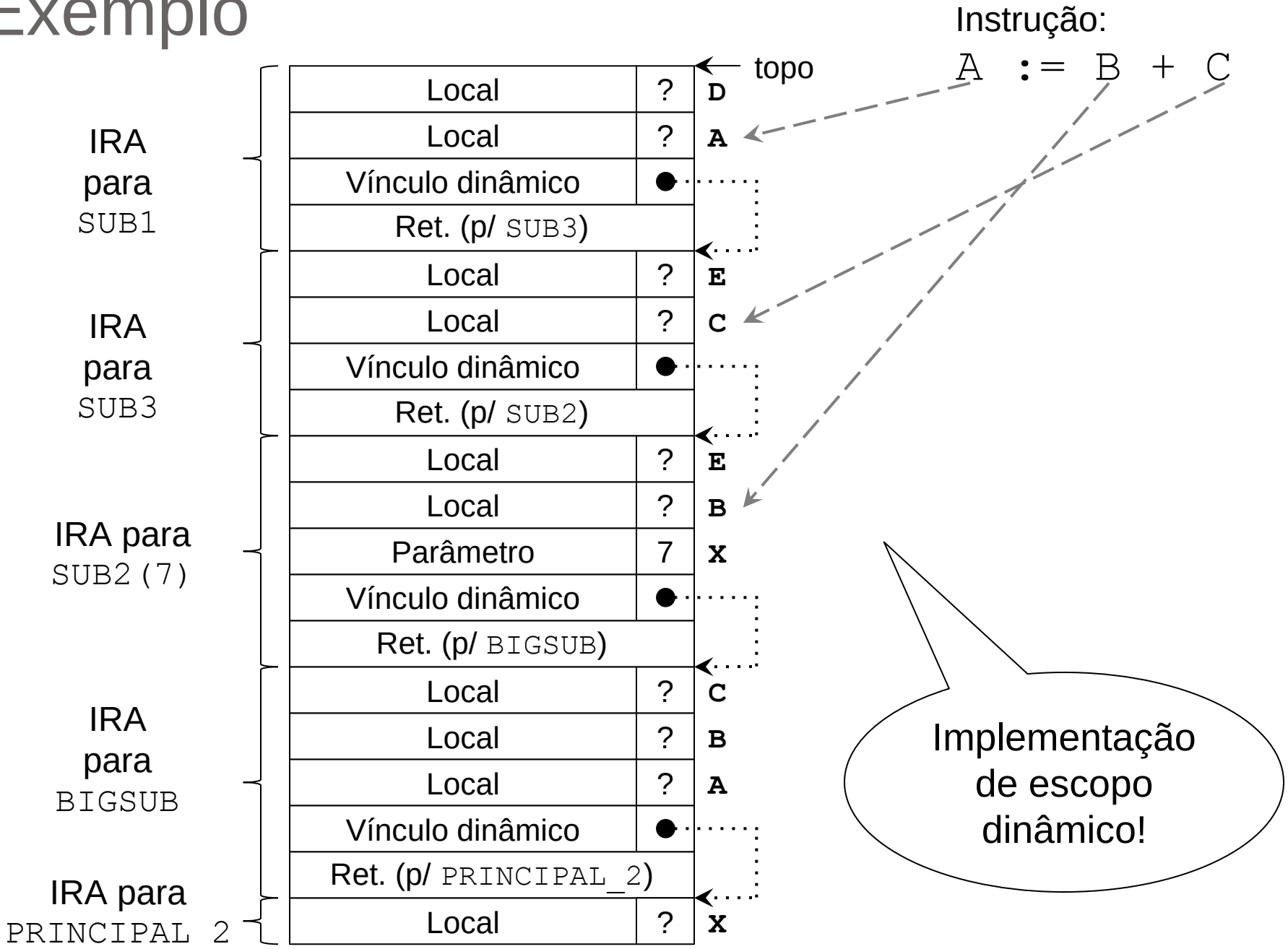
Aninhamento de subprogramas:



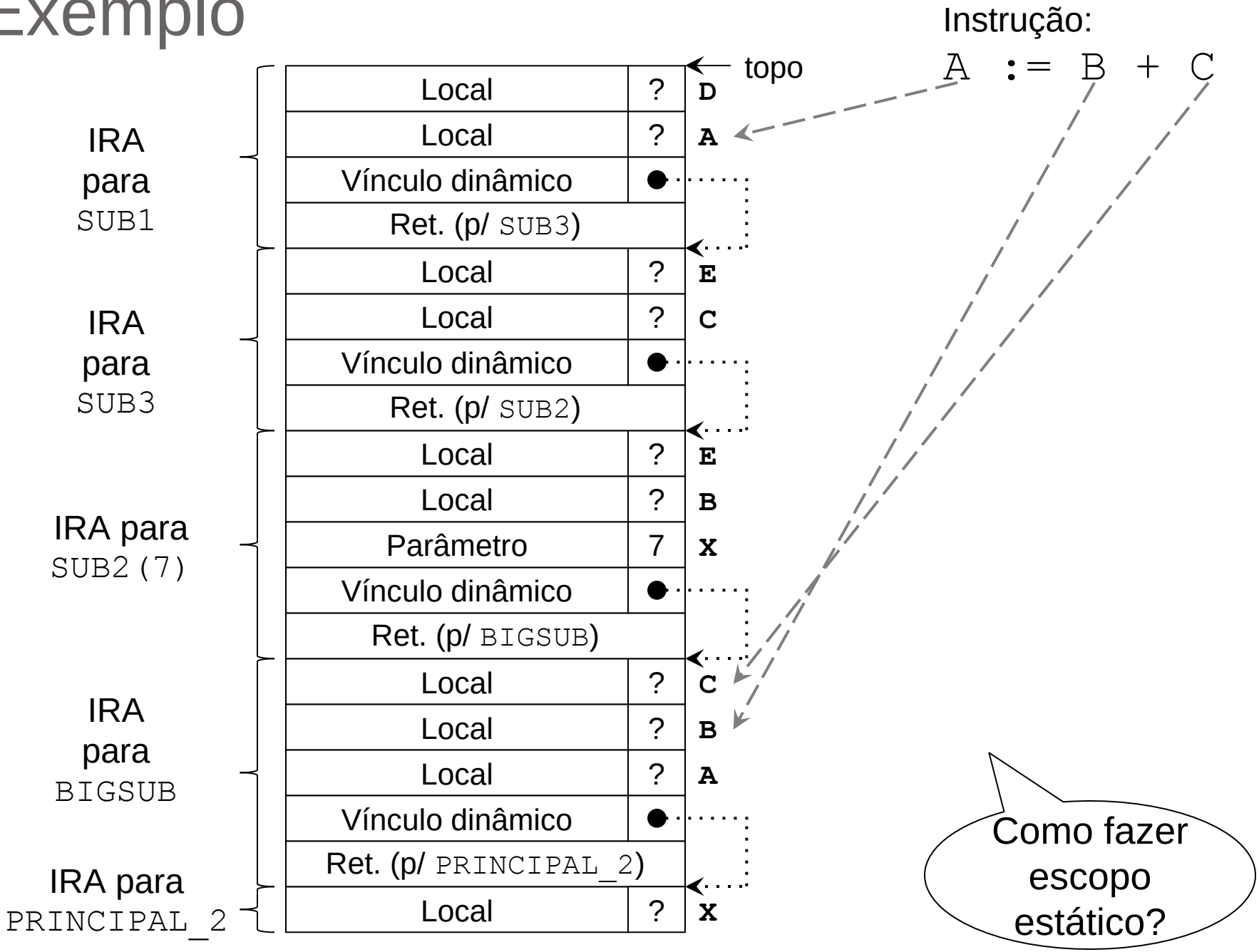
Relação chama-chamado:

PRINCIPAL_2 → BIGSUB → SUB2(7) → SUB3 → SUB1

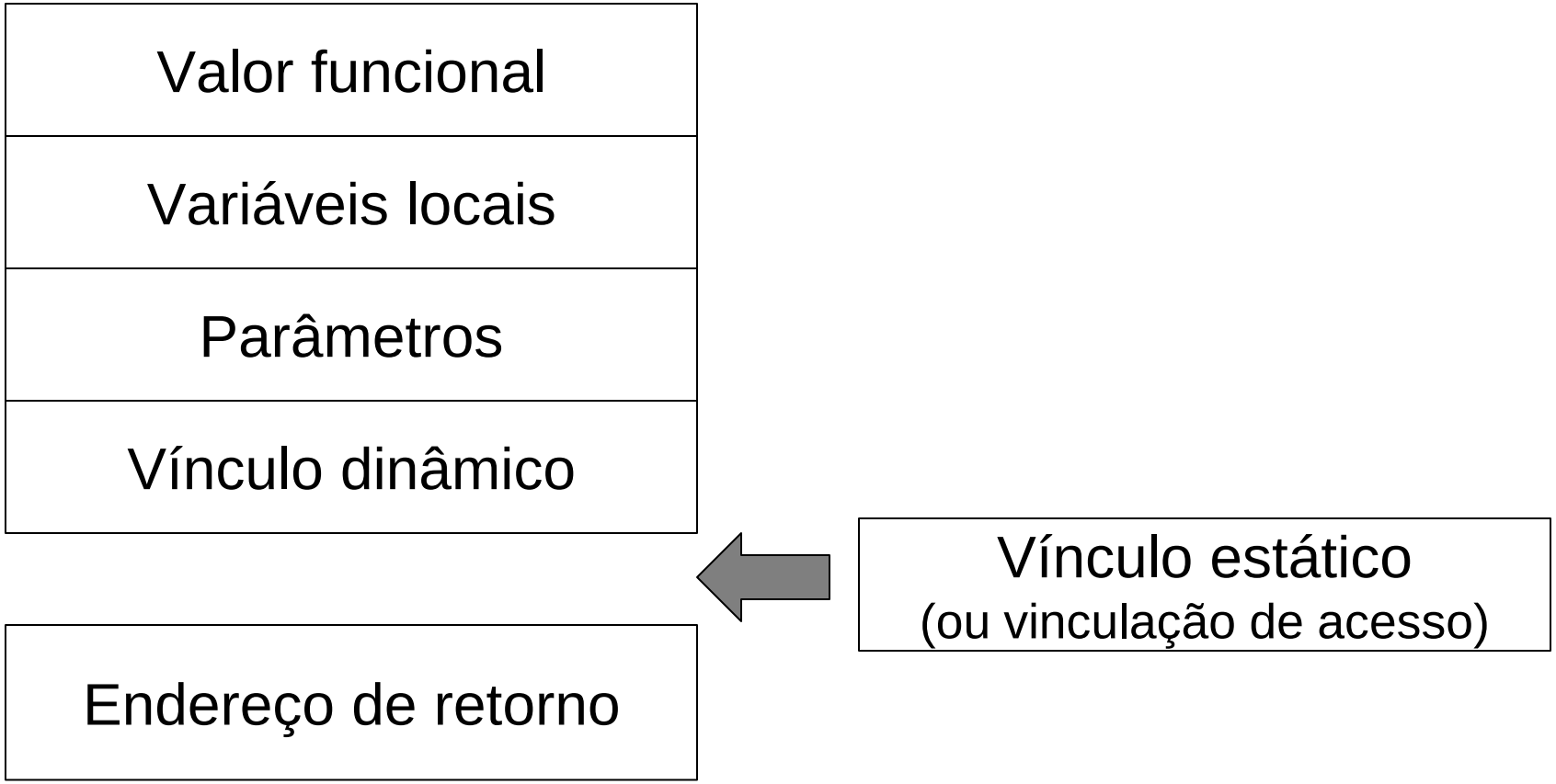
Exemplo


Instrução:
A := B + C
topo
IRA para SUB1
IRA para SUB3
IRA para SUB2(7)
IRA para BIGSUB
IRA para PRINCIPAL_2
Local ? D
Local ? A
Vínculo dinâmico
Ret. (p/ SUB3)
Local ? E
Local ? C
Vínculo dinâmico
Ret. (p/ SUB2)
Local ? E
Local ? B
Parâmetro 7 X
Vínculo dinâmico
Ret. (p/ BIGSUB)
Local ? C
Local ? B
Local ? A
Vínculo dinâmico
Ret. (p/ PRINCIPAL_2)
Local ? X
Implementação de escopo dinâmico!

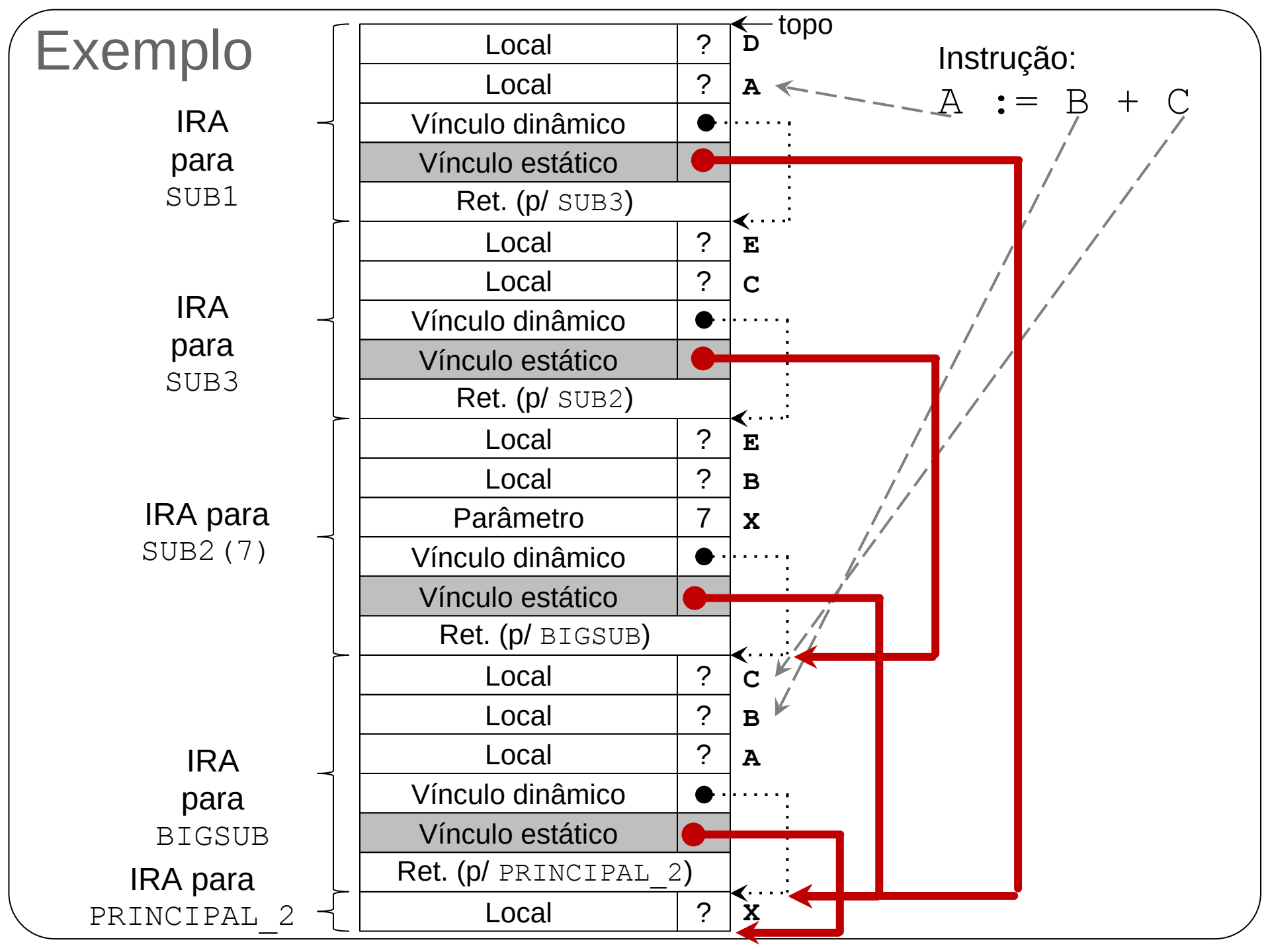
Exemplo


Instrução:
A := B + C
topo
IRA para SUB1
IRA para SUB3
IRA para SUB2(7)
IRA para BIGSUB
IRA para PRINCIPAL_2
Local ? D
Local ? A
Vínculo dinâmico
Ret. (p/ SUB3)
Local ? E
Local ? C
Vínculo dinâmico
Ret. (p/ SUB2)
Local ? E
Local ? B
Parâmetro 7 X
Vínculo dinâmico
Ret. (p/ BIGSUB)
Local ? C
Local ? B
Local ? A
Vínculo dinâmico
Ret. (p/ PRINCIPAL_2)
Local ? X
Como fazer escopo estático?

ALGOL e linguagens derivadas



Exemplo


topo
Instrução:
A := B + C
IRA para SUB1
IRA para SUB3
IRA para SUB2 (7)
IRA para BIGSUB
IRA para PRINCIPAL_2
Local ? D
Local ? A
Vínculo dinâmico
Vínculo estático
Ret. (p/ SUB3)
Local ? E
Local ? C
Vínculo dinâmico
Vínculo estático
Ret. (p/ SUB2)
Local ? E
Local ? B
Parâmetro 7 X
Vínculo dinâmico
Vínculo estático
Ret. (p/ BIGSUB)
Local ? C
Local ? B
Local ? A
Vínculo dinâmico
Vínculo estático
Ret. (p/ PRINCIPAL_2)
Local ? X

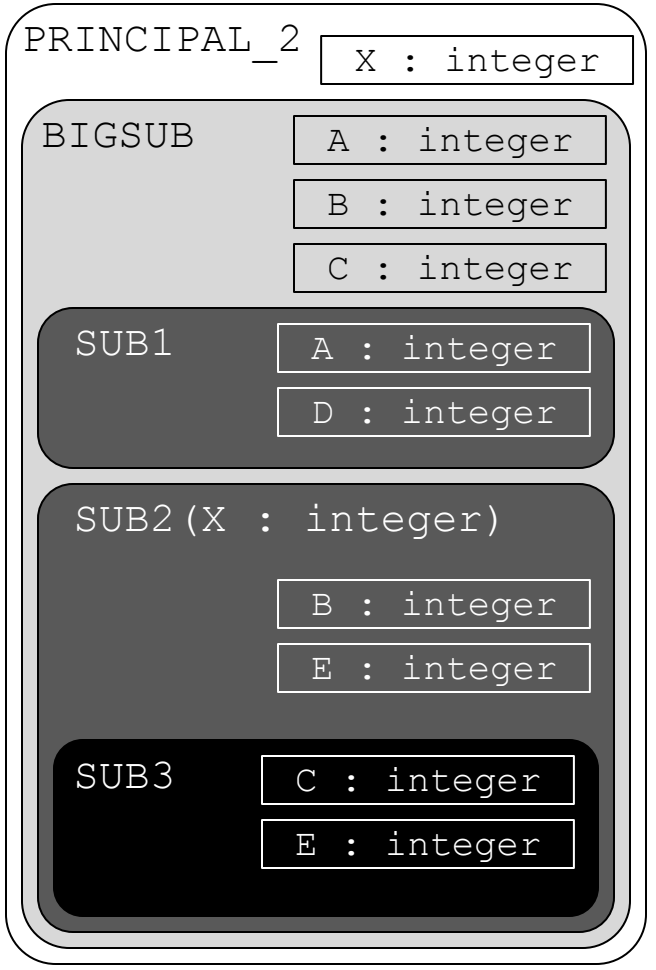
ALGOL e linguagens derivadas

- Para implementar escopo estático
  - Para cada referência a variável
  - Se for local, basta acessar a IRA corrente
  - Senão, basta buscar através dos vínculos estáticos
    - Técnica chamada de encadeamento estático
- Uma técnica melhor – deixar para o compilador o trabalho de encontrar as variáveis
  - Basta “contar” os níveis de aninhamento estático
    - O compilador consegue fazer isso
  - E a cada acesso, calcular a diferença X
    - E navegar X passos abaixo na pilha

Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                          { SUB1 }
      A := B + C;
      end;                           { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin                        { SUB3 }
        SUB1;
        E := B + A;
        end;                         { SUB3 }
      begin                          { SUB2 }
      SUB3;
      A := D + E;
      end;                           { SUB2 }
    begin                            { BIGSUB }
    SUB2(7);
    end;                             { BIGSUB }
  begin                              {
PRINCIPAL_2 }
  BIGSUB;
  end.                               {
PRINCIPAL_2 }
```

Níveis de aninhamento estático:



Nível 0 | Nível 2

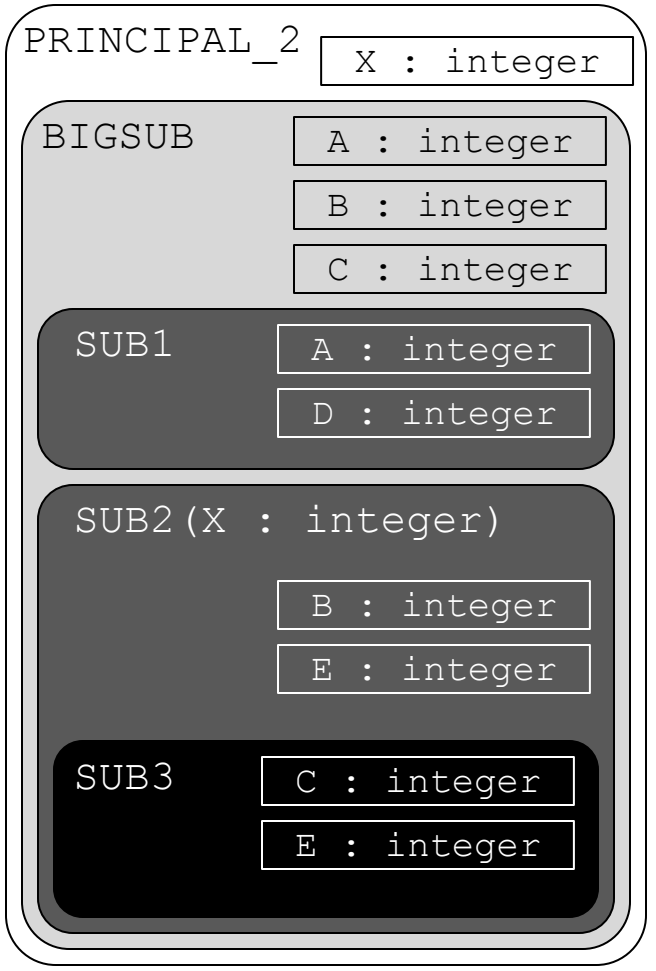
Nível 1 | Nível 3

Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                                { SUB1 }
      A := B + C;
      end;                                 { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin                              { SUB3 }
        A := B + C;
        end;                               { SUB3 }
      begin                                { SUB2 }
      SUB3;
      end;                                 { SUB2 }
    begin                                  { BIGSUB }
    SUB2(7);
    end;                                   { BIGSUB }
  begin                                    {
PRINCIPAL_2 }
  BIGSUB;
  end.                                     {
PRINCIPAL_2 }
```

Nível Acesso = 2
Nível Declaração = 2
Resultado = 2 – 2 = 0

Níveis de aninhamento estático:



Nível 0 | Nível 2 | Nível 1 | Nível 3

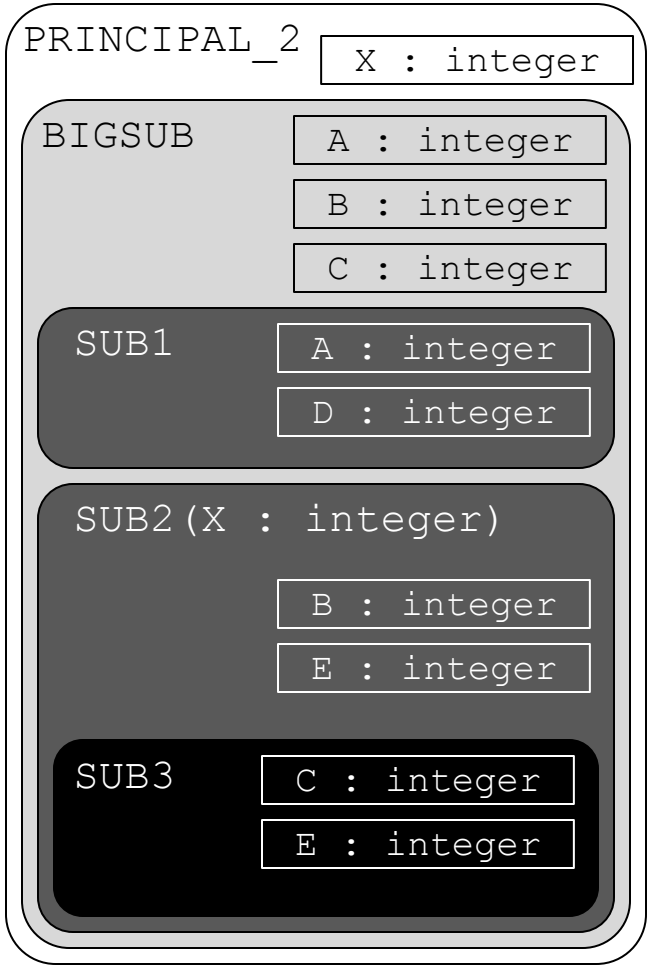
Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                                { SUB1 }
      A := B + C;
      end;                                 { SUB1 }
    procedure SUB2(X : integer);
      var B, E : integer;
      procedure SUB3;
        var C, E : integer;
        begin                              { SUB3 }
        A := B + C;
        end;                               { SUB3 }
      begin                                { SUB2 }
      SUB3;
      end;                                 { SUB2 }
    begin                                  { BIGSUB }
    SUB2(7);
    end;                                   { BIGSUB }
  begin                                    {
PRINCIPAL_2 }
  BIGSUB;
  end.                                     {
PRINCIPAL_2 }
```

```
A := B + C;
(0)
```

Nível Acesso = 2
Nível Declaração = 1
Resultado = 2 – 1 = 1

Níveis de aninhamento estático:



Nível 0 | Nível 2
Nível 1 | Nível 3

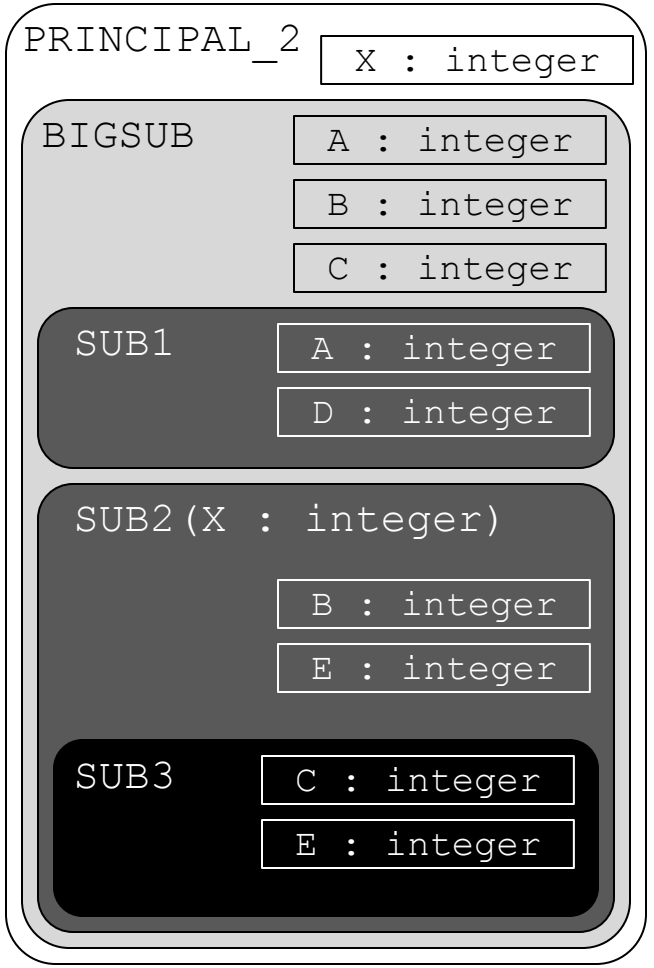
Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                                { SUB1 }
      A := B + C;
      end;                                 { SUB1 }
    begin                                  { BIGSUB }
    SUB2(7);
    end;                                   { BIGSUB }
  begin                                    {
PRINCIPAL_2 }
  BIGSUB;
  end.                                     {
PRINCIPAL_2 }
```

```
A := B + C;
(0)   (1)
```

Nível Acesso = 2
Nível Declaração = 1
Resultado = 2 – 1 = 1

Níveis de aninhamento estático:



Nível 0 | Nível 2
Nível 1 | Nível 3

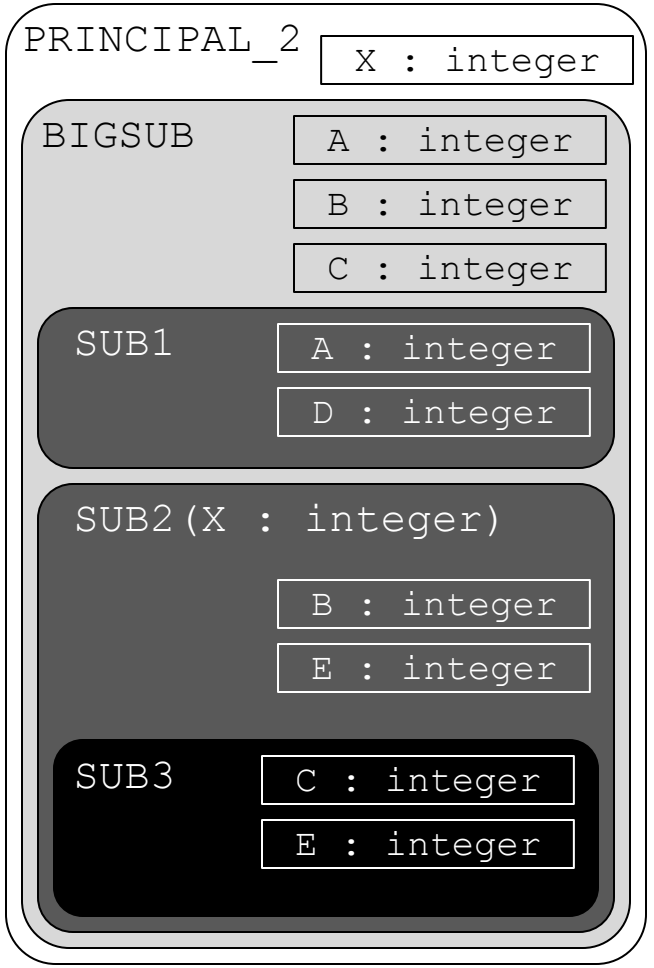
Exemplo

```
program PRINCIPAL_2;
  var X : integer;
  procedure BIGSUB;
    var A, B, C : integer;
    procedure SUB1;
      var A, D : integer;
      begin                                { SUB1 }
      A := B + C;
      end;                                 { SUB1 }

                                           { BIGSUB }
    begin
    SUB2(7);
    end;                                   { BIGSUB }
  begin                                    {
PRINCIPAL_2 }
  BIGSUB;
  end.                                     {
PRINCIPAL_2 }
```

```
A := B + C;
(0)   (1)  (1)
```

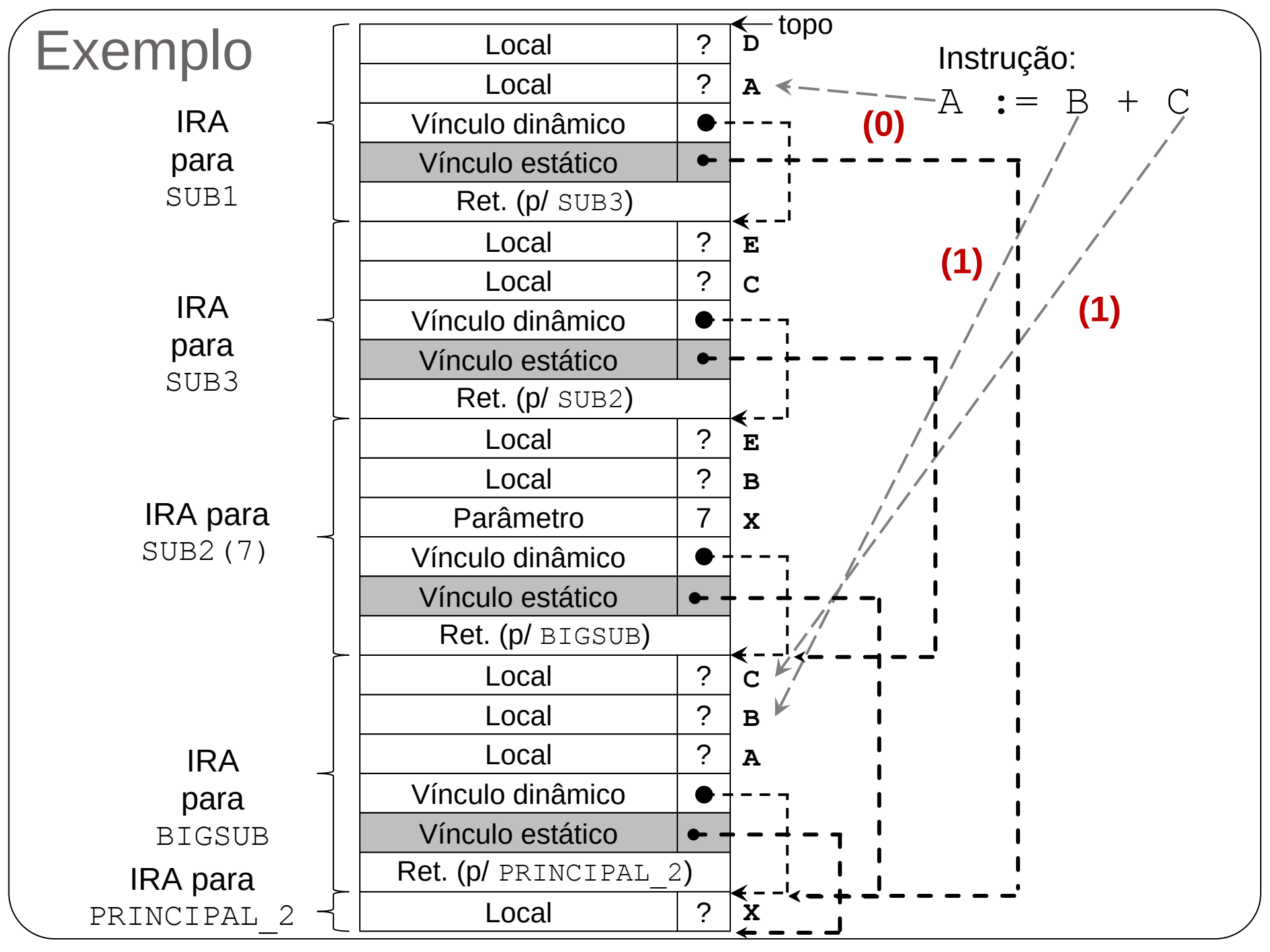
Níveis de aninhamento estático:



Nível 0 | Nível 2

Nível 1 | Nível 3

Exemplo


IRA para SUB1
IRA para SUB3
IRA para SUB2(7)
IRA para BIGSUB
IRA para PRINCIPAL_2
Local ? D
Local ? A
Vínculo dinâmico
Vínculo estático
Ret. (p/ SUB3)
Local ? E
Local ? C
Vínculo dinâmico
Vínculo estático
Ret. (p/ SUB2)
Local ? E
Local ? B
Parâmetro 7 X
Vínculo dinâmico
Vínculo estático
Ret. (p/ BIGSUB)
Local ? C
Local ? B
Local ? A
Vínculo dinâmico
Vínculo estático
Ret. (p/ PRINCIPAL_2)
Local ? X
topo
Instrução:
A := B + C
(0)
(1)
(1)

ALGOL e linguagens derivadas

- Encadeamento estático é custoso
  - Exige acesso indireto
  - Além disso, mesmo sabendo-se o número de níveis necessários
    - Pode ser difícil estimar exatamente a quantidade de acessos
      - Pois a mesma pode se alterar sempre que o código é modificado
    - Crítico em sistemas de tempo real
- Alternativa: uso de *displays*
  - Consiste em manter os vínculos estáticos em uma matriz separada, de tal forma que em no máximo 2 acessos chega-se à variável não-local
  - Não veremos mais detalhes nesse curso

ALGOL e linguagens derivadas

- Escopo estático de blocos aninhados
- Ex:

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

ALGOL e linguagens derivadas

- É possível implementar da mesma forma vista anteriormente
  - Tratando cada bloco como um subprograma sem parâmetros
  - Como se a chamada fosse feita no momento da abertura do bloco
  - As regras de vinculação estática e dinâmica funcionariam da mesma forma
- Mas existe uma forma mais simples

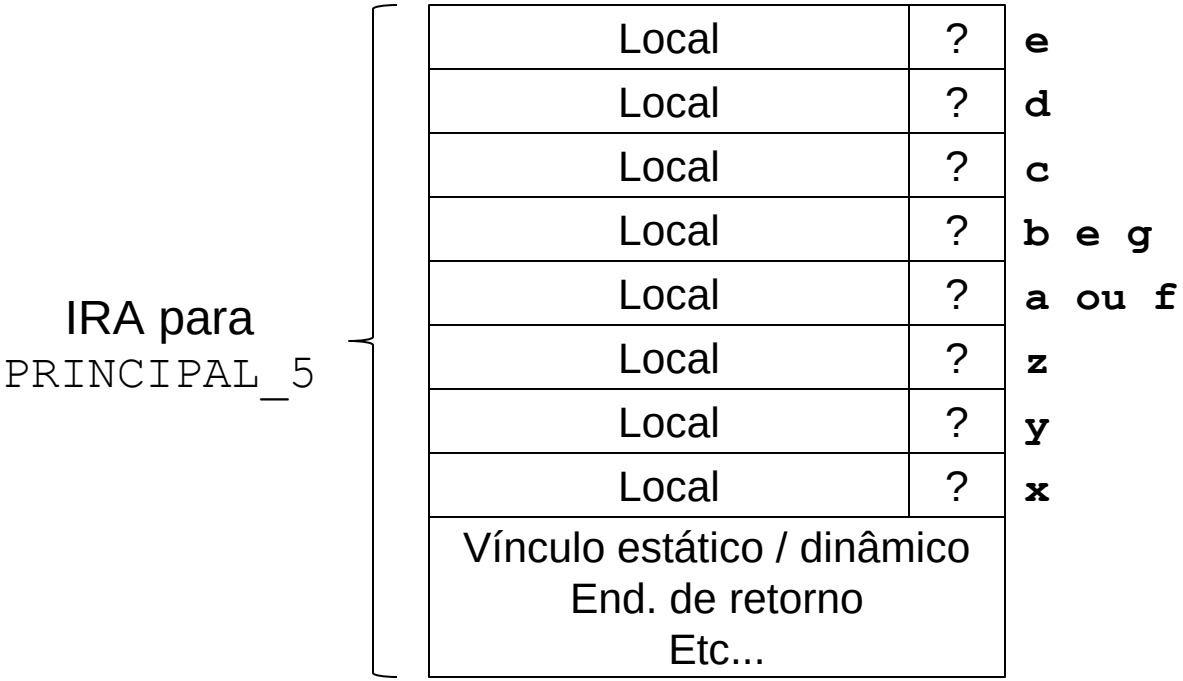
ALGOL e linguagens derivadas

- O compilador consegue determinar a maior quantidade de variáveis ativas em um determinado trecho

```
PRINCIPAL_5() {
  int x, y, z;          // 3 vars: x, y, z
  while (...) {
    int a, b, c;        // 6 vars: x, y, z, a, b, c
    ...
    while (...) {
      int d, e;         // 8 vars: x, y, z, a, b, c, d, e
      ...
    }
  }
  while (...) {
    int f, g;           // 5 vars: x, y, z, f, g
    ...
  }
  ...
}
```

ALGOL e linguagens derivadas

- Basta, portanto, alocar em uma IRA, o tamanho máximo necessário para aquele trecho

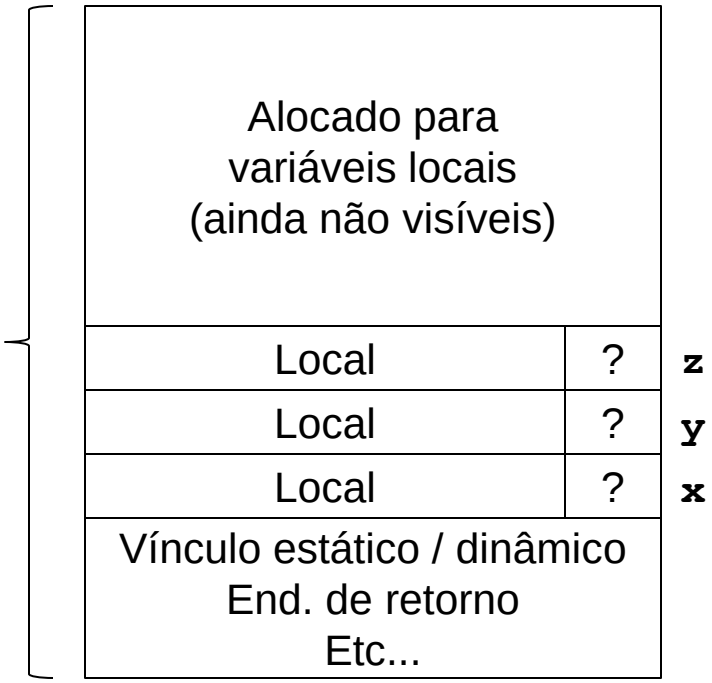


Exemplo

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

1

IRA para PRINCIPAL_5

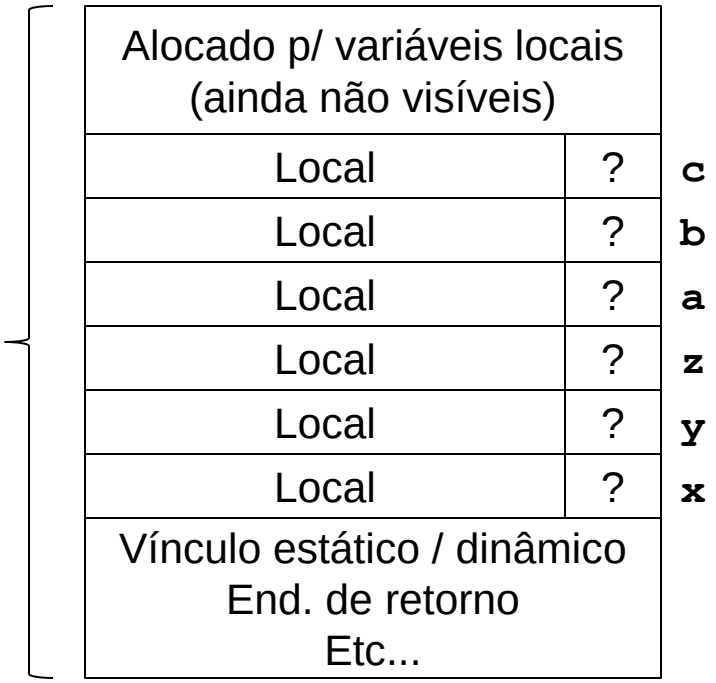


Exemplo

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

2

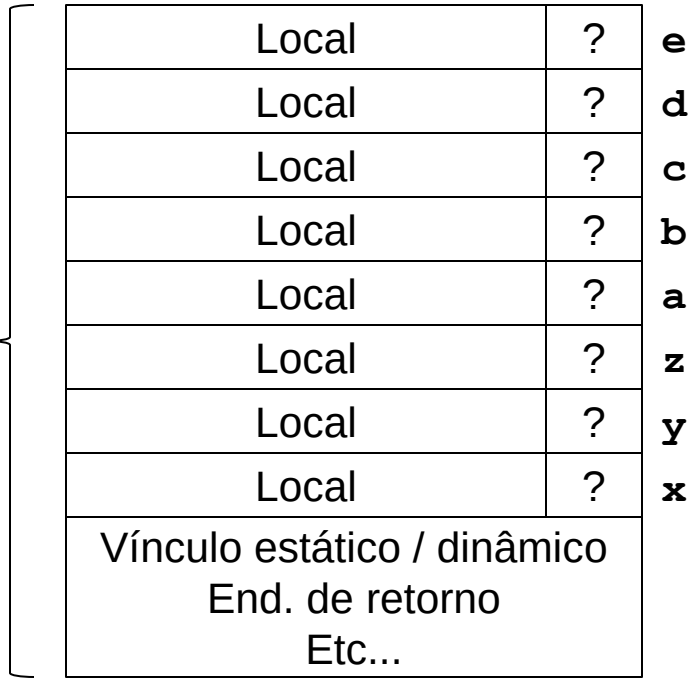
IRA para PRINCIPAL_5



Exemplo

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

3 IRA para PRINCIPAL_5

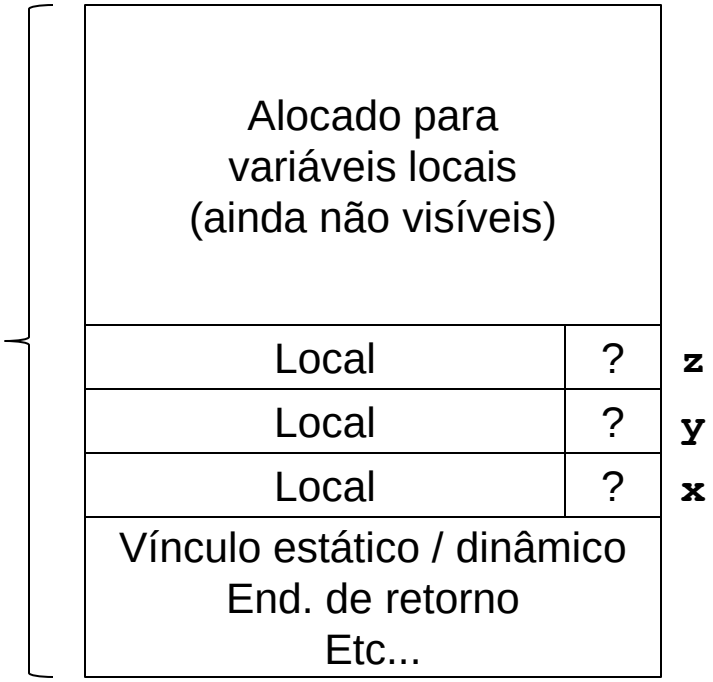


Exemplo

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

4

IRA para
PRINCIPAL_5



Exemplo

```
PRINCIPAL_5() {
  int x, y, z;
  while (...) {
    int a, b, c;
    ...
    while (...) {
      int d, e;
      ...
    }
  }
  while (...) {
    int f, g;
    ...
  }
  ...
}
```

5

IRA para PRINCIPAL_5

Alocado para variáveis locais (ainda não visíveis)		
Local	?	**g**
Local	?	**f**
Local	?	**z**
Local	?	**y**
Local	?	**x**
Vínculo estático / dinâmico End. de retorno Etc...		

ALGOL e linguagens derivadas

- Escopo dinâmico
  - Já vimos anteriormente
  - Ou seja, basta buscar na pilha através do encadeamento dinâmico
  - Essa abordagem é conhecida como acesso **profundo**
    - Pois busca na pilha em profundidade
    - Processo custoso
- É possível também utilizar acesso **raso**
  - Uma pilha para cada variável
  - Pilha mantida dinamicamente
    - Cada variável declarada, se já existir outra com o mesmo nome, vai para o topo da pilha correspondente
  - Acesso é rápido, porém, a manutenção da pilha é custosa
  - Existem alternativas, mas não veremos no curso

Procedimentos como parâmetros

- Antes de analisar o próximo caso, responda:
  - Num ambiente completamente estático
  - Ou num ambiente sem procedimentos locais
  - É possível passar procedimentos como parâmetros?
  - Se sim, como seria?

Procedimentos como parâmetros

- Num ambiente completamente estático
  - Resposta = Sim
  - Basta passar um ponteiro para o endereço do registro de ativação do procedimento
  - Uma chamada para esse procedimento consiste em iniciar a sequência de ativação normalmente
    - Não há problemas em perda de dados de vinculação
    - Uma vez que todos os endereços estão fixados na memória

Procedimentos como parâmetros

- Num ambiente baseado em pilhas sem procedimentos locais
  - Resposta = Sim
  - Nomes não locais são disponíveis a todos
  - Nomes locais não precisam ser acessados por procedimentos que não estão ativados
    - Em outras palavras, não importa a ordem de ativação

Procedimentos como parâmetros

- Num ambiente baseado em pilhas com procedimentos locais
  - Resposta = Não
  - Quando um procedimento é passado como parâmetro, a informação de vinculação de acesso simplesmente não existe
    - Porque como o procedimento é um parâmetro, ele só será conhecido em tempo de execução
    - E a informação de vinculação de acesso é conhecida somente em tempo de compilação
  - Portanto, o modelo anterior não funciona

Com procedimentos locais e que podem ser passados como parâmetros

- Solução
  - Ao invés de passar por parâmetro somente um ponteiro para o procedimento
    - Passa-se também a vinculação de acesso daquele procedimento
      - ponteiro de código = ip
      - ponteiro de vinculação estática = ep
  - Isso é possível, pois o ativador sempre conhece a vinculação de acesso do procedimento passado como parâmetro

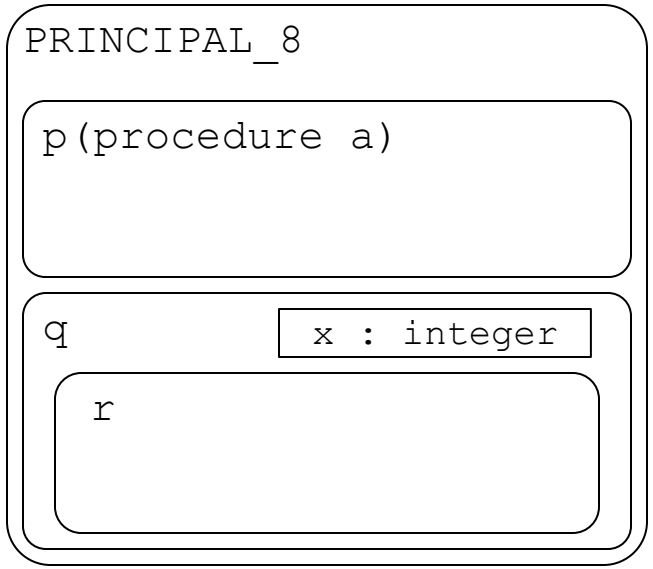
Com procedimentos locais e que podem ser passados como parâmetros

- Solução
  - Em outras palavras:
    - **p** chama **q**, passando **f** como parâmetro
    - **q** só descobre quem é **f** durante a execução
      - Ou seja, não conhece sua vinculação de acesso
      - Portanto, quando **f** for ativado por **g**, não poderá acessar dados definidos fora de **f**
    - Mas **p** sabe quem é **f**
      - Já que é **p** quem está fazendo a chamada
      - Portanto **p** conhece a vinculação de acesso de **f**
    - Portanto, **p** passa para **q** o fechamento de **f**, ou seja:
      - ip : o ponteiro para o código do procedimento **f**
      - ep : o ponteiro para a vinculação de acesso de **f**
    - Assim, quando **g** ativar **f**, pode definir a vinculação de acesso normalmente

Exemplo

```
program PRINCIPAL_8;
  procedure p(procedure a);
    begin
    a;
    end;
  procedure q;
    var x:integer;
    procedure r;
      begin
      writeln(x);
      end;
    begin
      x := 2;
      p(r);
    end;
  begin
  q;
  end.
```

Aninhamento de subprogramas:



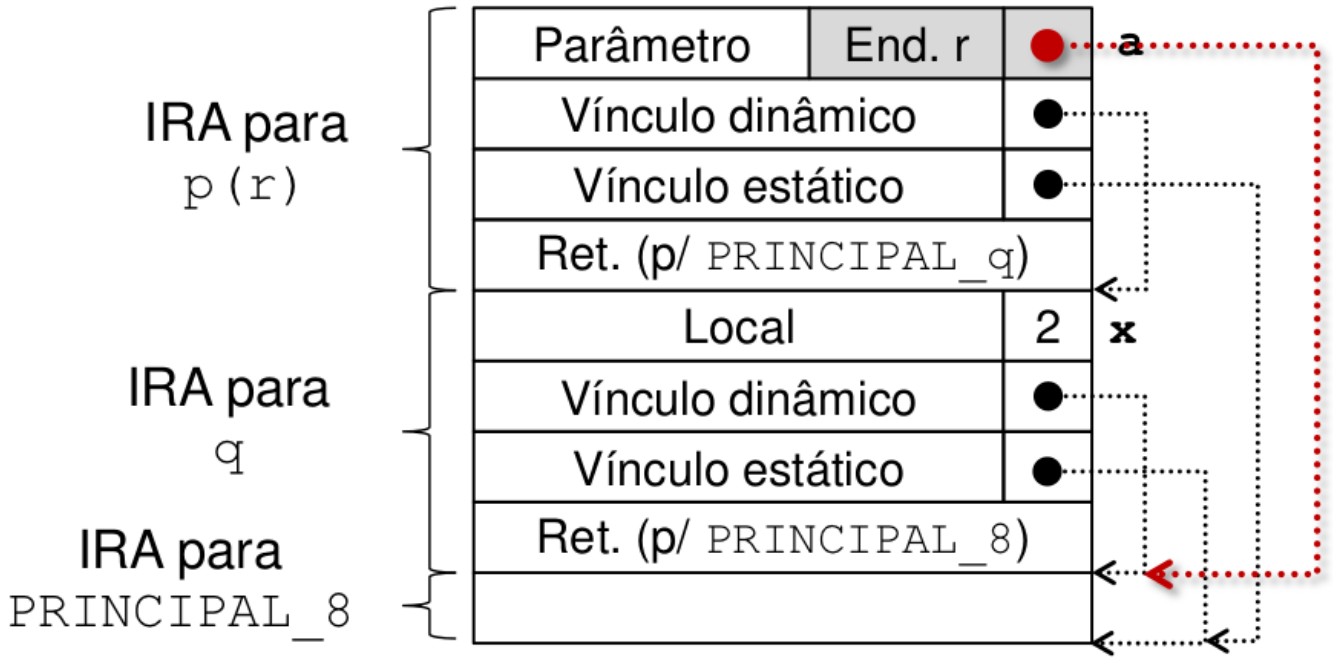
Relação chama-chamado:

PRINCIPAL_8 → q → p(r) → r [a]

Exemplo

```
program PRINCIPAL_8;
  procedure p(procedure
a);
    begin
    a;
    end;
  procedure q;
    var x:integer;
    procedu
      begin
      write
      end;
    begin
      x :=
      p(r);
    end;
  begin
  q;
  end.
```

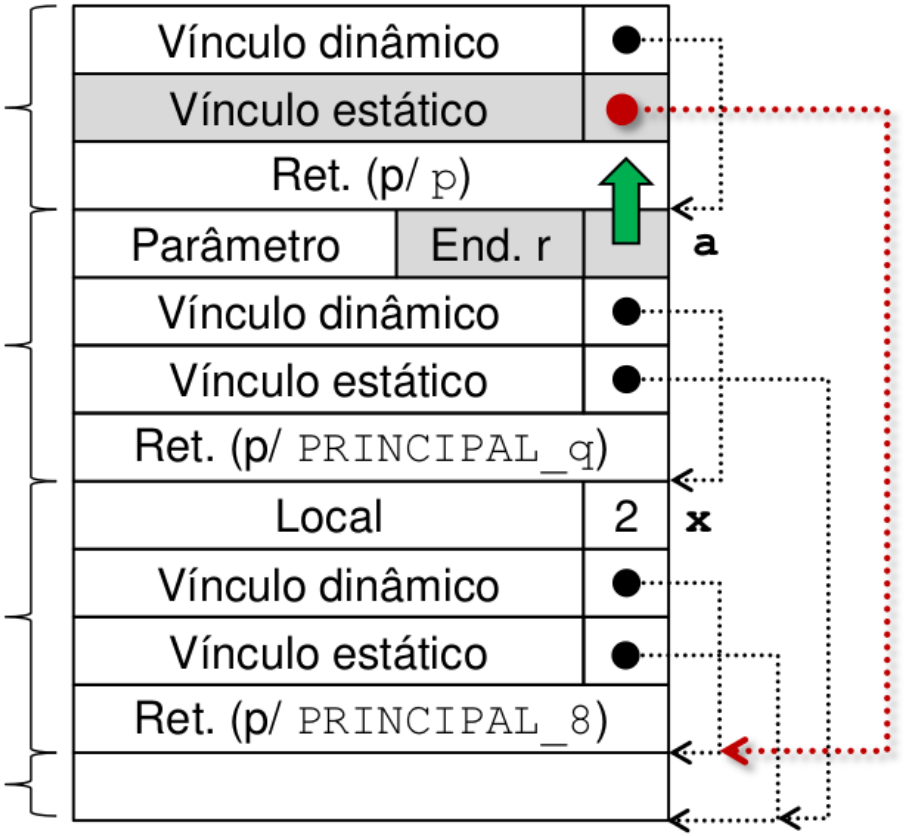
1



Exemplo

```
program PRINCIPAL_8;
  procedure p(procedure
a);
    begin
    a;
    end;
  procedure q;
    var x:integer;
    procedure r;
      begin
      writeln(x);
      end;
    begin
      x := 2;
      p(r);
    end;
  begin
  q;
  end.
```

2



Procedimentos como parâmetros

- Em alguns casos, a implementação em pilha não é suficiente, como vimos na última demonstração
- Conceito envolvido: fechamento (closure)
  - Função + ambiente de referenciamento para variáveis livres (não-locais)
  - Tempo de vida das variáveis pode se estender além do “normal” para seu escopo estático
  - Comum em linguagens funcionais
- Outros mecanismos são necessários
  - Em C#, por exemplo, quando detectado que um delegate cria a necessidade de fechamento
    - Uma classe anônima é gerada pelo compilador
    - O delegate é transformado em um método
    - As variáveis não-locais são transformadas em atributos

Resumo

Resumo

- Vimos a implementação da parte essencial da programação estruturada
  - Chamada de procedimentos
  - Recursividade
  - Acesso não-local (e não-global)
    - Escopo estático e dinâmico
  - Procedimentos como parâmetros

Fim