Linguagens de Programação

Parâmetros

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

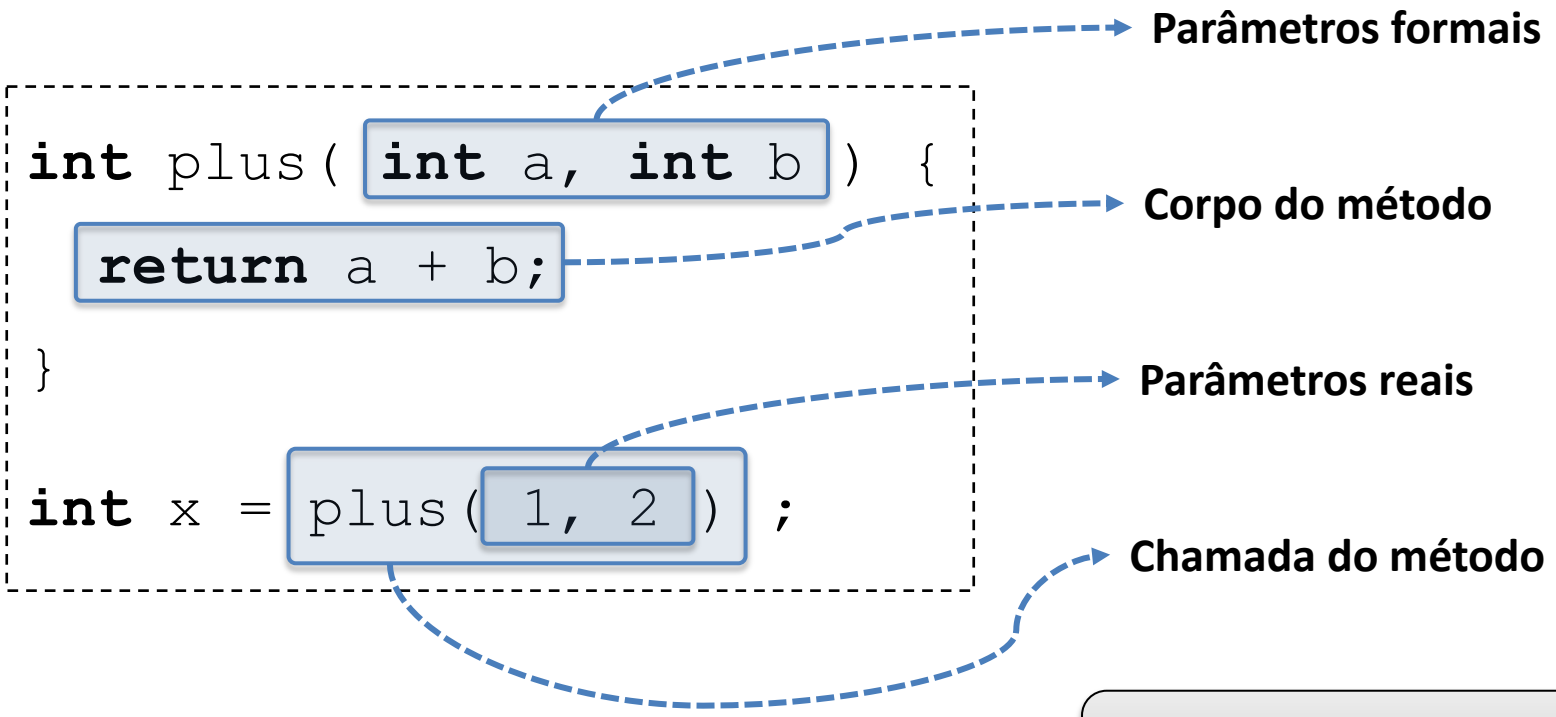
- Introdução
- Correspondência
- Passagem de parâmetros
  - Direção da passagem
  - Mecanismos de passagem
    - Passagem por valor, por resultado, por valor-resultado, por referência, por expansão de macros, por nome e por necessidade
  - Momento da passagem

INTRODUÇÃO

Linguagens de Programação

Introdução

- Terminologia básica em uma linguagem hipotética com sintaxe similar a Java

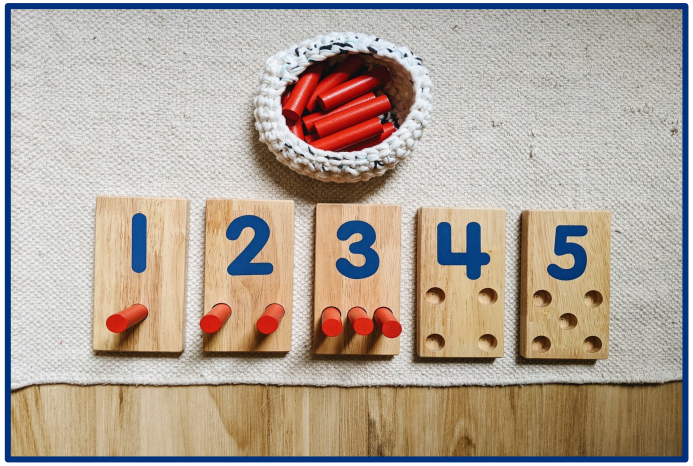


Como os parâmetros são passados para os métodos?

Introdução

- A ausência de parâmetros reduz
  - **Redigibilidade**: necessário incluir operações para atribuir os valores desejados às variáveis globais
  - **Legibilidade**: na chamada não há menção aos nomes usados nos parâmetros
  - **Confiabilidade**: não há exigência que sejam atribuídos valores a todas as variáveis globais

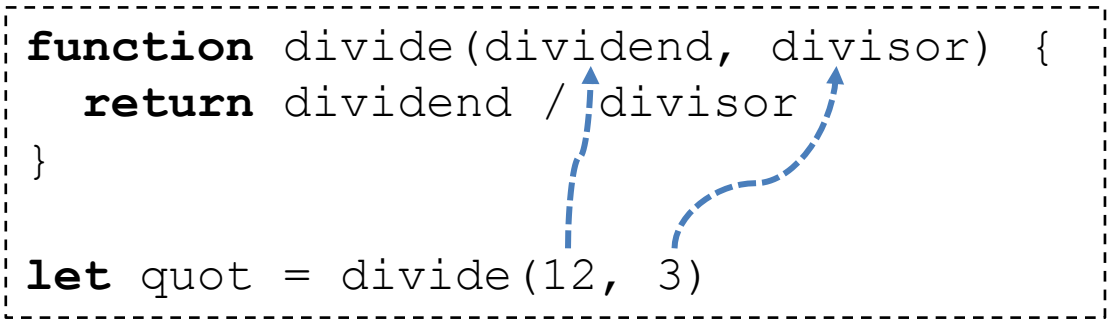
CORRESPONDÊNCIA



Correspondência Posicional

- Correspondência entre **parâmetros formais** e **parâmetros reais** (posicional)
  - A vinculação é feita pela ordem dos parâmetros formais
  - Primeiro parâmetro formal ligado ao primeiro parâmetro real e assim por diante
  - Vantagens: seguro e efetivo

- Exemplo em JavaScript



Correspondência por Palavra-Chave

- Correspondência entre **parâmetros atuais** e **parâmetros formais** (palavra-chave)
  - A vinculação é feita na chamada, especificando o nome do parâmetro formal
  - Vantagem: ordem dos parâmetros irrelevante
  - Desvantagem: o nome do parâmetro deve ser conhecido na chamada

- Exemplo em Python

```
def divide(dividend, divisor):
   return dividend // divisor

quot = divide(divisor=3, dividend=12)
```

Valores Padrões em Parâmetros Formais

- Valores definidos nos parâmetros formais podem ser utilizados na falta destes valores nos parâmetros reais
  - Existem em: ADA, C++, FORTRAN, Visual Basic
  - Devem aparecer no final em LPs com parâmetros posicionais

Como isso é implementado?

- Exemplo em C++

```
int max(int x, int y = 0, int z = 0) { ... }

max(3, 4, 5); // Retorna 5
max(3, 4);    // Retorna 4
max(3);       // Retorna 3
```

Java não suporta tal funcionalidade, como resolver?

Lista Variável em Parâmetros Formais

- Algumas linguagens suportam uma lista de parâmetros reais de qualquer tamanho
  - Oferece maior flexibilidade à LP
  - Reduz a confiabilidade, pois não é possível verificar os tipos dos parâmetros em tempo de compilação

- Exemplo em C

```
int printf(char* formatador, ...);

printf("Nome: %s", nome);
printf("%d * %d = %d", x, y, x*y);
```

Como `printf` obtém os parâmetros reais?

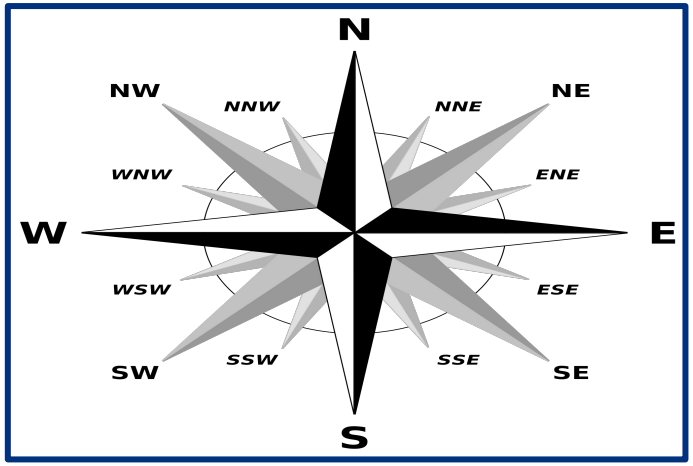
PASSAGEM DE PARÂMETROS

Linguagens de Programação

Passagem de Parâmetros

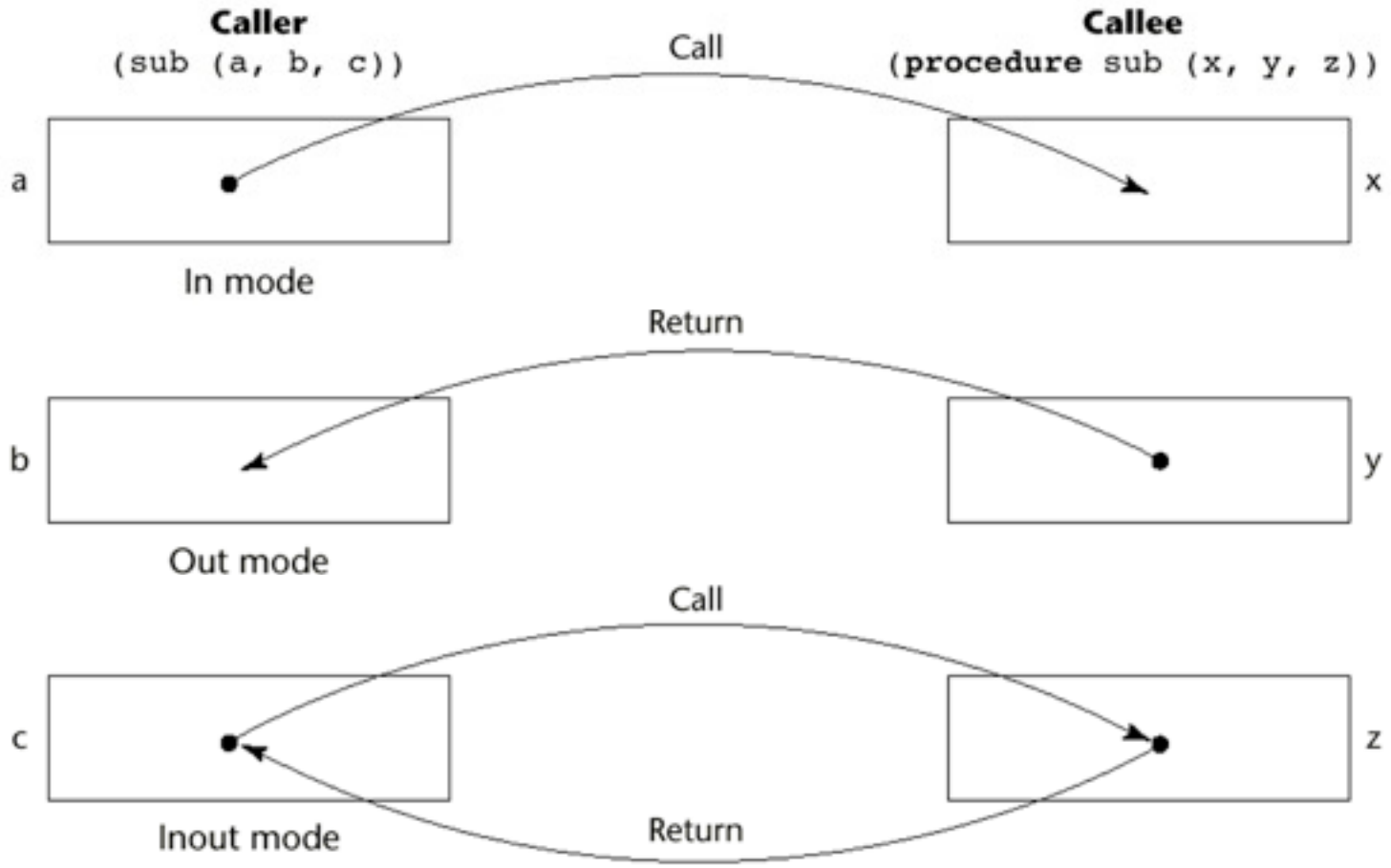
- Formas de transmitir parâmetros de e/ou para métodos chamados
- Faz parte do processo de passagem de parâmetros a eventual atualização de valores dos parâmetros reais durante a execução do método
- Três aspectos importantes
  - Direção da passagem
  - Mecanismo de implementação
  - Momento no qual a passagem é realizada

PASSAGEM DE PARÂMETROS > DIREÇÃO



Direção da Passagem

- As possíveis direções de passagem de parâmetros



Direção da Passagem

- As possíveis direções de passagem de parâmetros

Direção da passagem	Forma do parâmetro real (R)	Atribuição do parâmetro formal (F)	Fluxo
Entrada Variável	Variável, Constante ou Expressão	Sim	R → F
Entrada Constante	Variável, Constante ou Expressão	Não	R → F
Saída	Variável	Sim	R ← F
Entrada e Saída	Variável	Sim	R ↔ F

Direção da Passagem

- Passagem unidirecional
  - Entrada variável (em C)

```
void funct(int param1, char param2);
void funct(int* param1, char* param2);
```

  - Entrada constante (C++)

```
void funct(const int param1);
```

  - Saída (C#)

```
void funct(out int retval);
```

- Passagem bidirecional
  - Referência (C++)

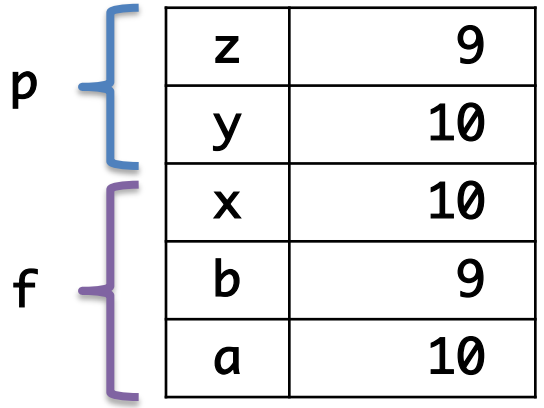
```
void funct(int& param1, char& param2);
```

PASSAGEM DE PARÂMETROS > MECANISMOS

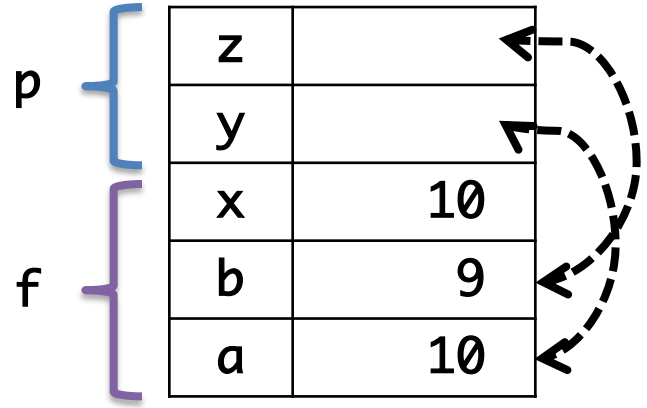
Linguagens de Programação

Mecanismos de Passagem

- Por **cópia**
  - Viabiliza a passagem unidirecional de entrada variável
  - Facilita a recuperação do estado do programa em interrupções inesperadas



- Por **caminho de acesso** (**referência**)
  - Proporciona semântica uniforme e simples na passagem de todos os tipos
  - Mais eficiente por não envolver cópia de dados
  - Pode ser ineficiente em implementação distribuída



Mecanismos de Passagem

- Mecanismos de passagem de parâmetros
  - Passagem **por valor** (*in mode*)
  - Passagem **por resultado** (*out mode*)
  - Passagem **por valor-resultado** (*in/out mode*)

  } Normalmente por cópia

  - Passagem **por referência** } Caminho de acesso
  - Passagem **por expansão de macros**
  - Passagem **por nome**
  - Passagem **por necessidade**

●●● > MECANISMOS > PASSAGEM POR VALOR

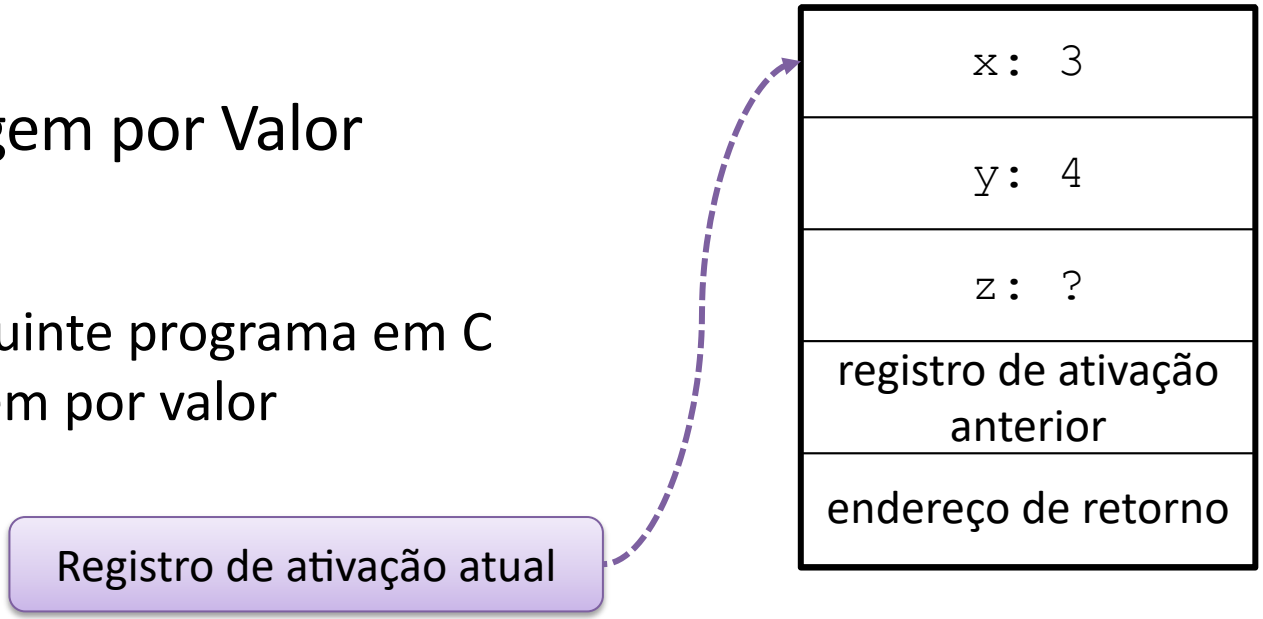
Passagem por Valor

- Na passagem de parâmetros por valor, o parâmetro formal é como uma variável local no registro de ativação do método chamado; ele é inicializado usando o valor do parâmetro real correspondente antes de começar a execução do método chamado

- É o método mais simples de passagem de parâmetros e mais amplamente utilizado
  - É o único mecanismo de passagem em Java

Passagem por Valor

- Considere o seguinte programa em C que usa passagem por valor

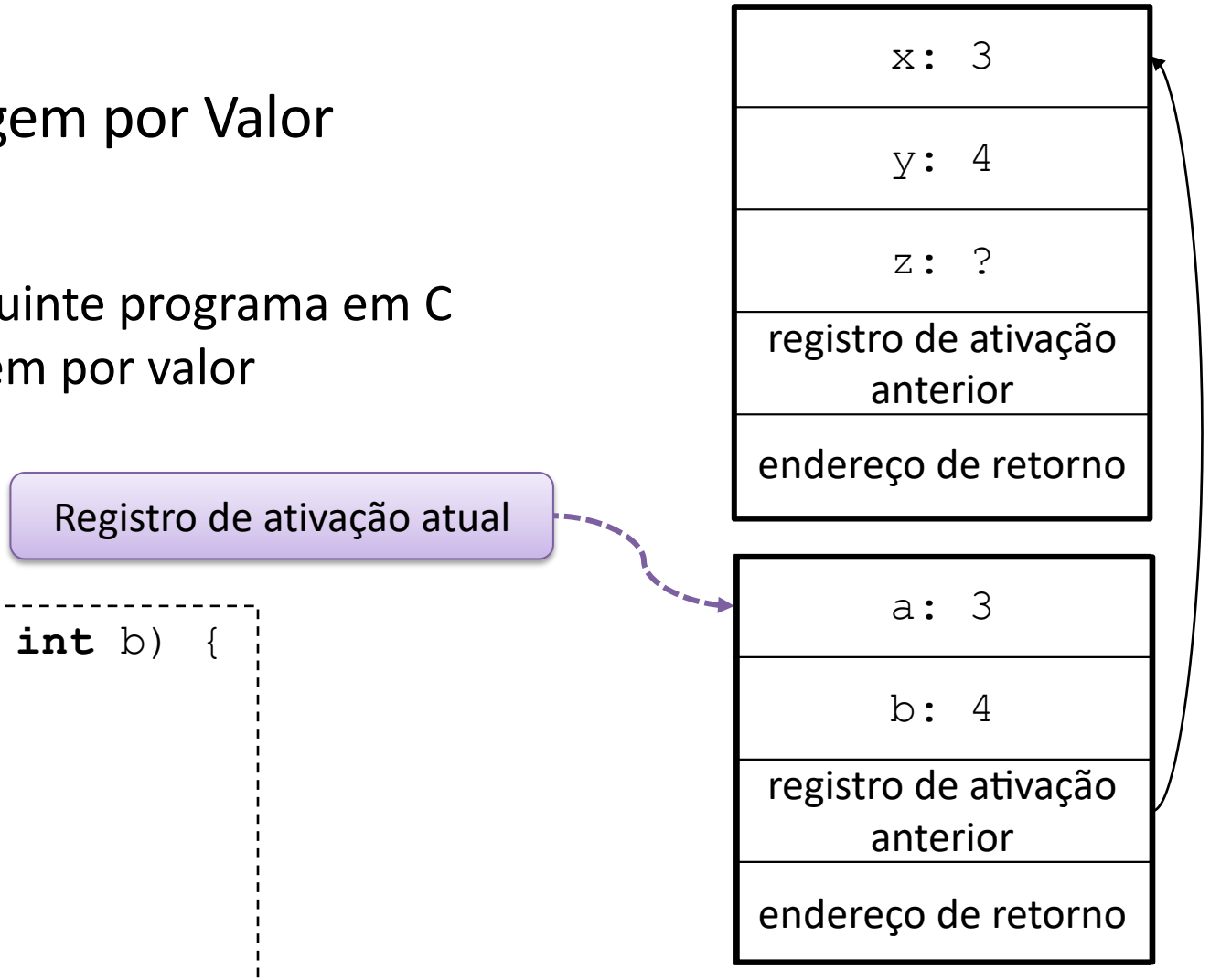



```
int plus(int a, int b) {
   a += b;
   return a;
}

void f() {
   int x = 3;
   int y = 4;
⇒  int z = plus(x, y);
   // ...
}
```

Passagem por Valor

- Considere o seguinte programa em C que usa passagem por valor

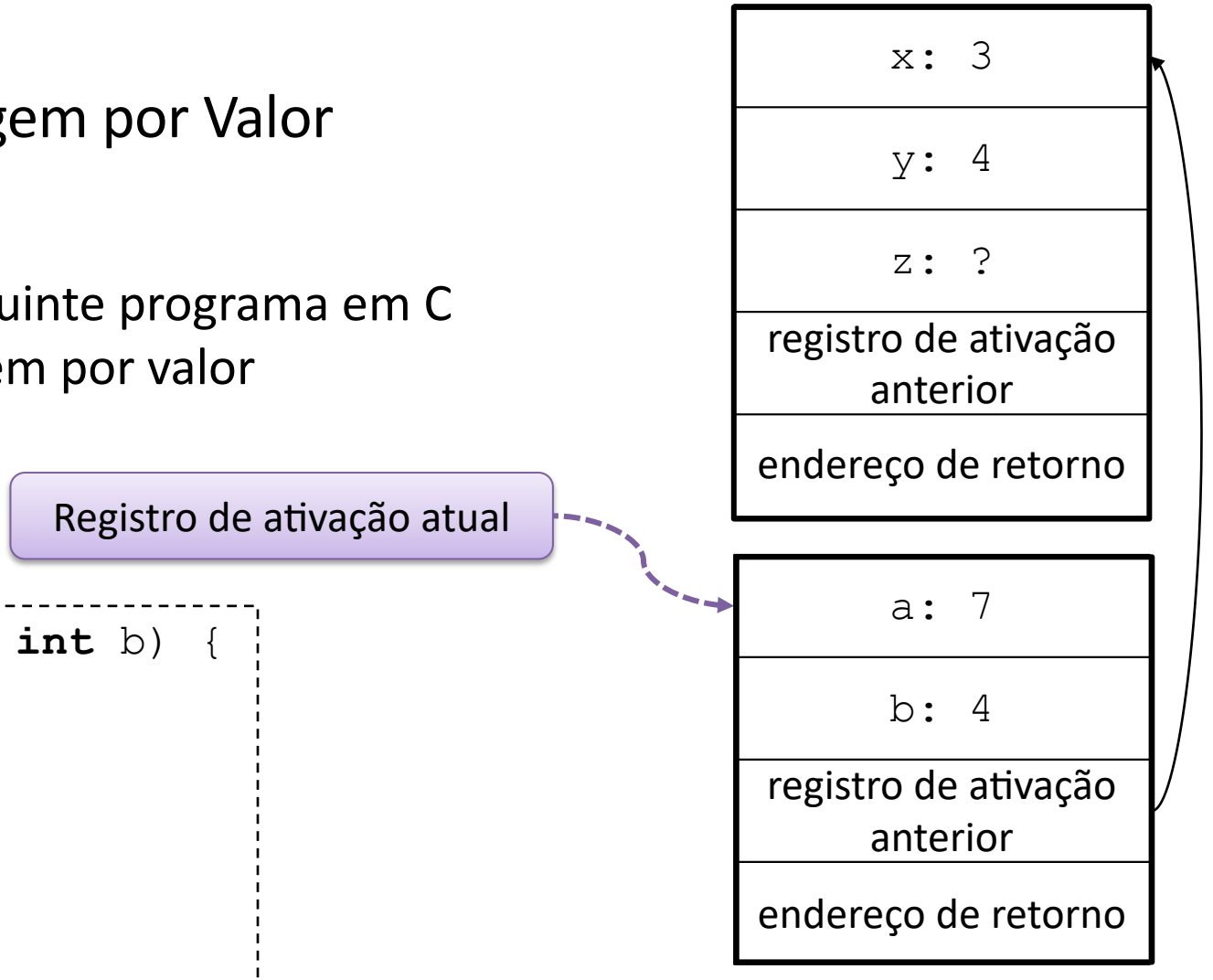
```
int plus(int a, int b) {
⇒  a += b;
   return a;
}

void f() {
   int x = 3;
   int y = 4;
   int z = plus(x, y);
   // ...
}
```



Passagem por Valor

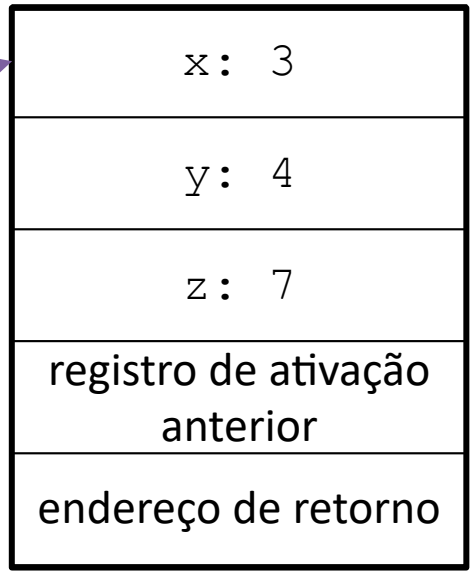
- Considere o seguinte programa em C que usa passagem por valor

```
int plus(int a, int b) {
   a += b;
⇒  return a;
}

void f() {
   int x = 3;
   int y = 4;
   int z = plus(x, y);
   // ...
}
```



Passagem por Valor

- Considere o seguinte programa em C que usa passagem por valor




```
int plus(int a, int b) {
   a += b;
   return a;
}

void f() {
   int x = 3;
   int y = 4;
   int z = plus(x, y);
⇒  // ...
}
```

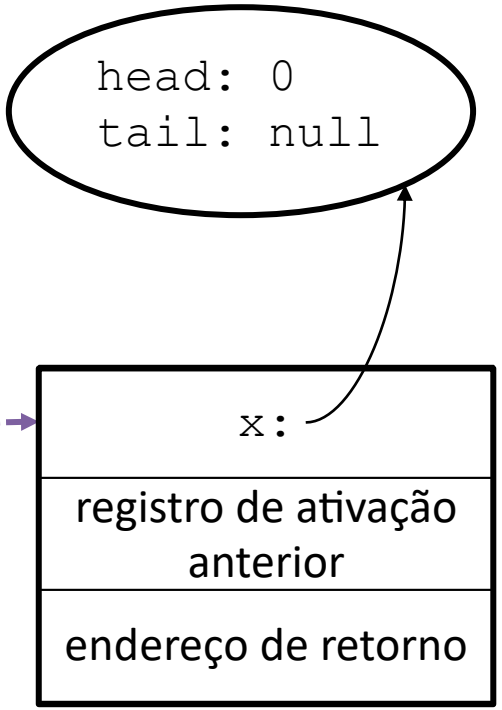
Passagem por Valor

- Quando parâmetros são passados por valor, alterações ao parâmetro formal não afetam o parâmetro real; mas ainda é possível que o método chamado faça alterações que são visíveis ao chamador

- O valor do parâmetro poderia ser um ponteiro, no caso da linguagem C, ou uma referência, no caso da linguagem Java
  - O parâmetro real não pode ser alterado, mas o objeto ou ponteiro referenciados pelo atual poderia ser

Passagem por Valor

- Considere o seguinte programa em Java que usa passagem por valor



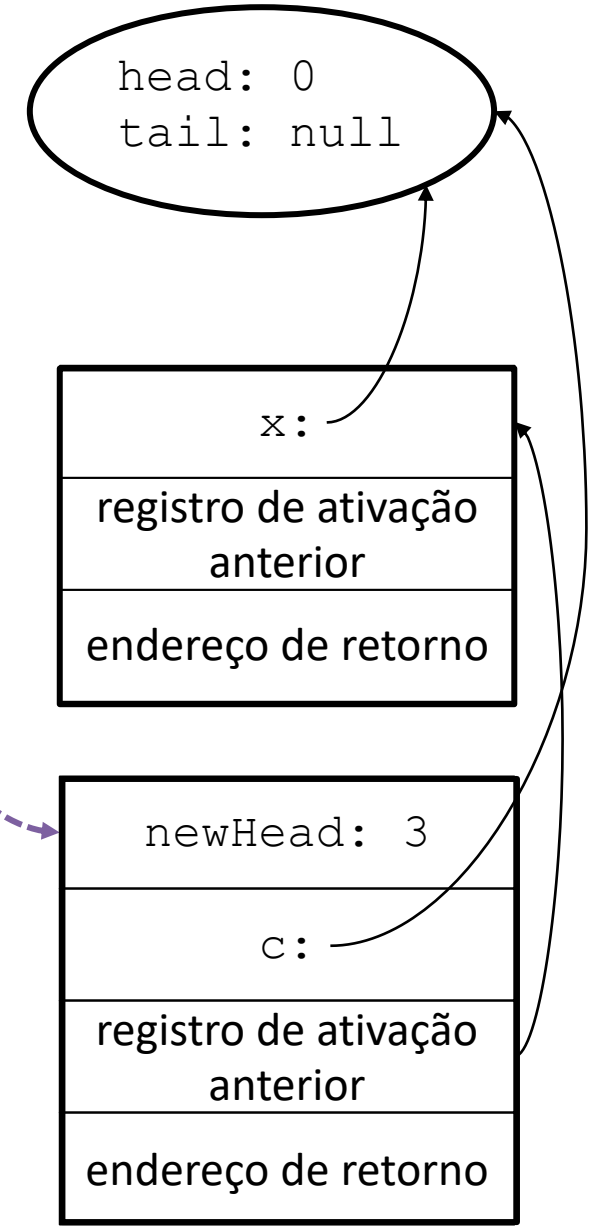

```
void f() {
  ConsCell x = new ConsCell(0, null);
⇒ alter(3, x);
  // ...
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
  c = null;
  return;
}
```

Passagem por Valor

- Considere o seguinte programa em Java que usa passagem por valor

Registro de ativação atual

```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
  // ...
}

void alter(int newHead, ConsCell c) {
⇒ c.setHead(newHead);
  c = null;
  return;
}
```

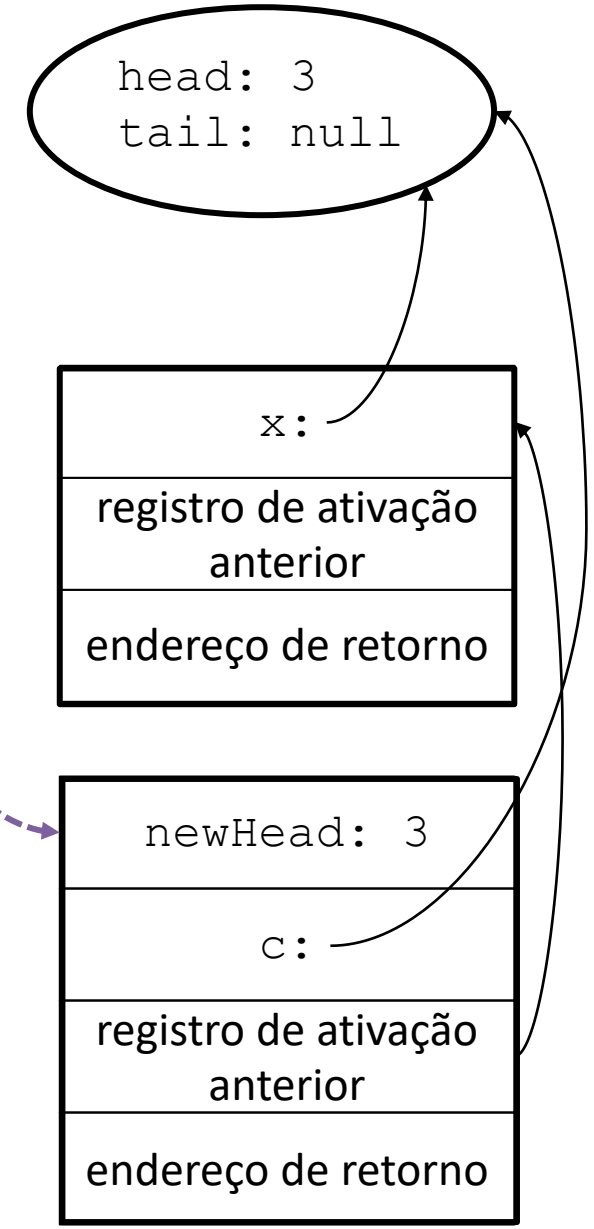


Passagem por Valor

- Considere o seguinte programa em Java que usa passagem por valor

Registro de ativação atual

```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
  // ...
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
⇒ c = null;
  return;
}
```

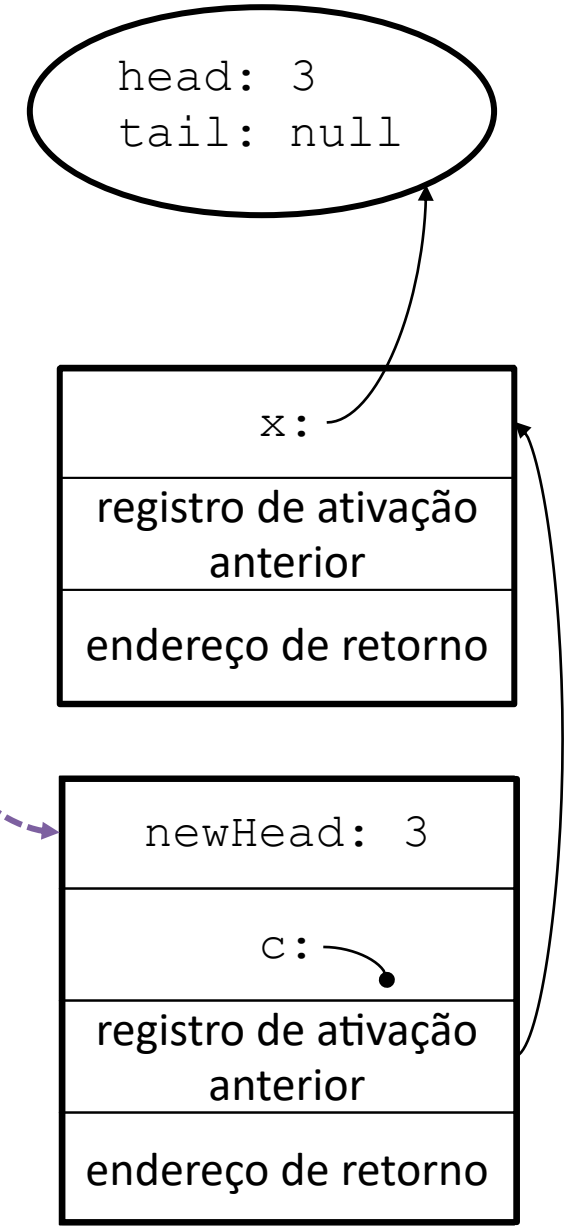


Passagem por Valor

- Considere o seguinte programa em Java que usa passagem por valor

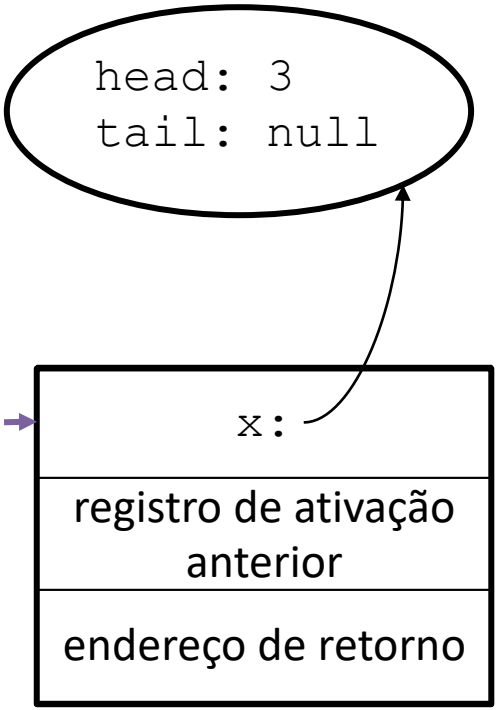
Registro de ativação atual

```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
  // ...
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
  c = null;
⇒ return;
}
```



Passagem por Valor

- Considere o seguinte programa em Java que usa passagem por valor




```
void f() {
  ConsCell x = new ConsCell(0, null);
  alter(3, x);
⇒ // ...
}

void alter(int newHead, ConsCell c) {
  c.setHead(newHead);
  c = null;
  return;
}
```

●●● > MECANISMOS > PASSAGEM POR RESULTADO

Passagem por Resultado

- Na passagem de parâmetros por resultado, o parâmetro formal é como uma variável local no registro de ativação do método chamado (ela não é inicializada); após o término da execução do método chamado, o valor final desta variável é atribuído ao parâmetro real correspondente

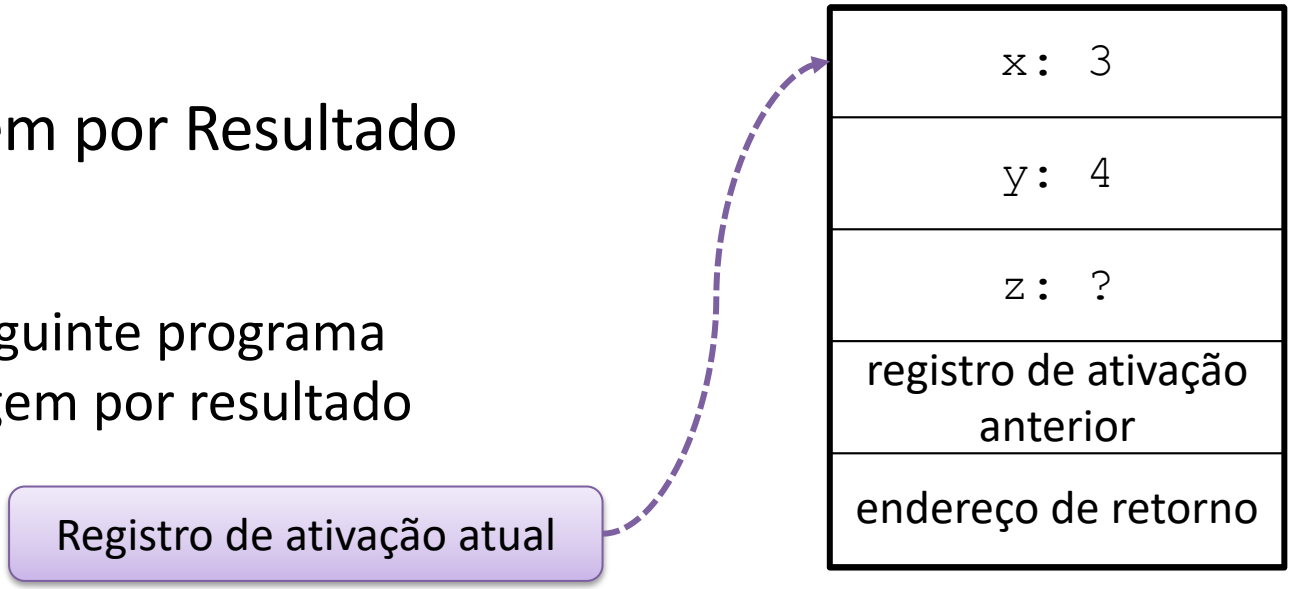
- Também conhecido como *copy-out*, ou seja, faz uma cópia na volta (após o término da execução do método)

- O parâmetro real deve ser um *lvalue*, ou seja, deve ser atribuível

Introduzido em Algol 68; em C# utiliza-se a palavra-reservada `out`

Passagem por Resultado

- Considere o seguinte programa que usa passagem por resultado



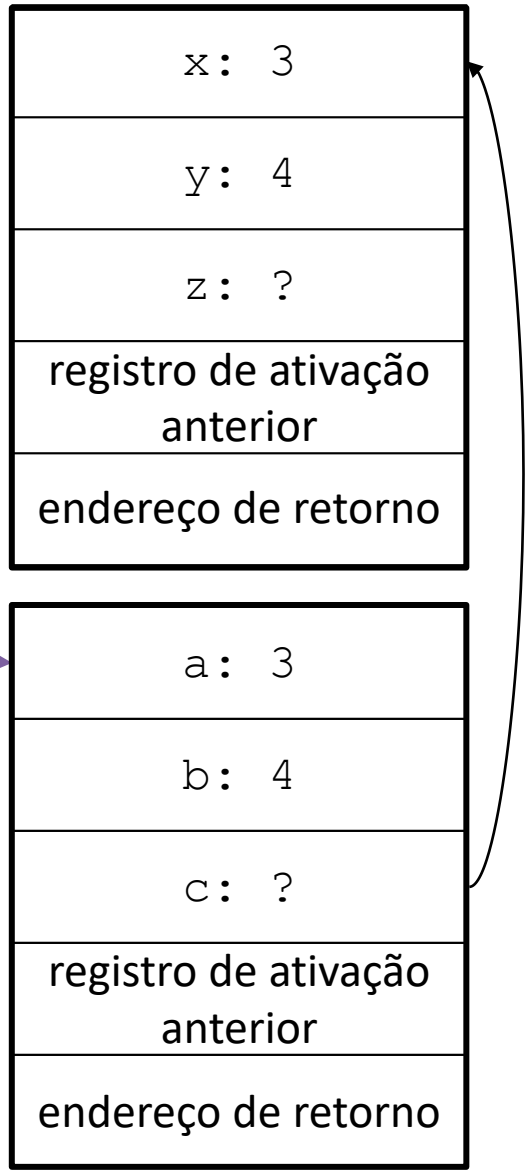

```
void plus(int a, int b, by-result int c) {
   c = a + b;
   return;
}

void f() {
   int x = 3;
   int y = 4;
   int z;
⇒  plus(x, y, z);
   // ...
}
```

Passagem por Resultado

- Considere o seguinte programa que usa passagem por resultado

Registro de ativação atual

```
void plus(int a, int b, by-result int c) {
⇒ c = a + b;
  return;
}

void f() {
  int x = 3;
  int y = 4;
  int z;
  plus(x, y, z);
  // ...
}
```

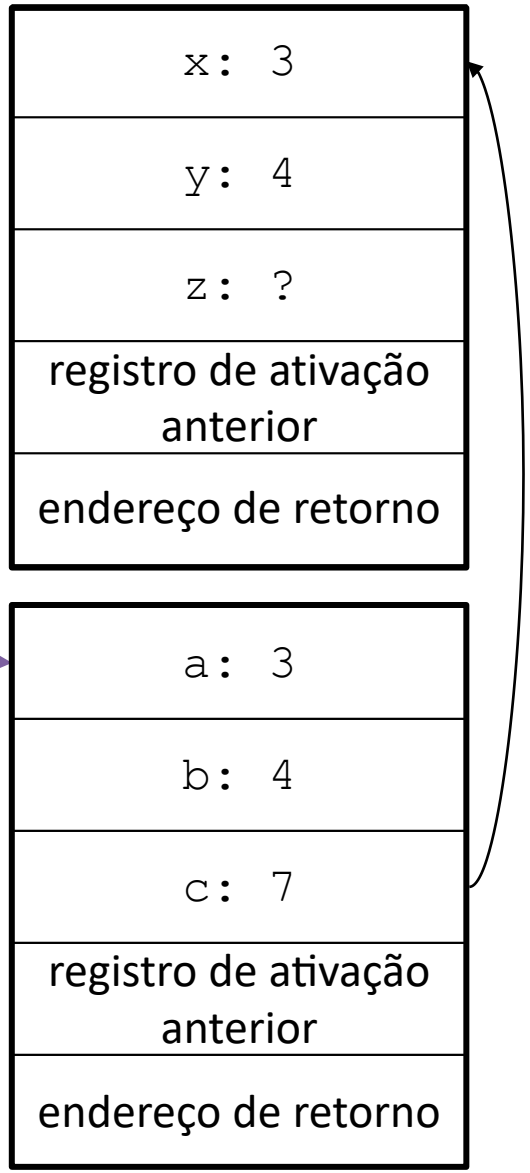


Passagem por Resultado

- Considere o seguinte programa que usa passagem por resultado

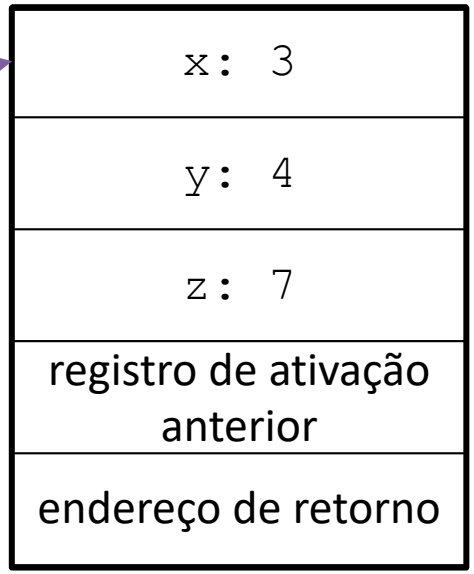
Registro de ativação atual

```
void plus(int a, int b, by-result int c) {
   c = a + b;
⇒  return;
}

void f() {
   int x = 3;
   int y = 4;
   int z;
   plus(x, y, z);
   // ...
}
```



Passagem por Resultado

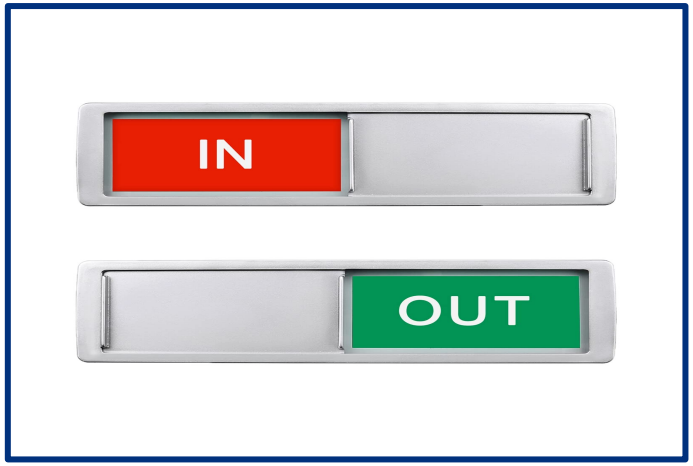
- Considere o seguinte programa que usa passagem por resultado



Registro de ativação atual

```
void plus(int a, int b, by-result int c) {
   c = a + b;
   return;
}

void f() {
   int x = 3;
   int y = 4;
   int z;
   plus(x, y, z);
⇒  // ...
}
```

●●● > MECANISMOS > PASSAGEM POR VALOR-RESULTADO



Passagem por Valor-Resultado

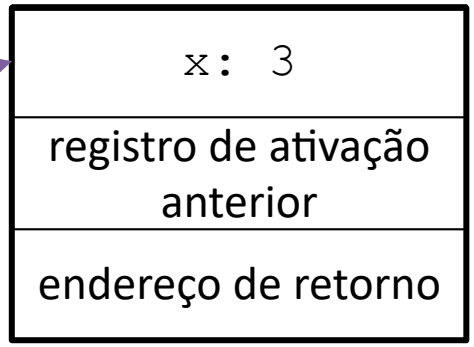
- Na passagem de parâmetros por valor-resultado, o parâmetro formal é como uma variável local no registro de ativação do método chamado; ela é inicializada usando o valor do parâmetro real correspondente antes do método começar sua execução; após o término da execução do método, o valor final do parâmetro formal é atribuído ao parâmetro real

- Também conhecido como *copy-in/copy-out*, ou seja, faz uma cópia na ida (antes da chamada do método) e na volta (após o término da execução do método)

- O parâmetro real deve ser um *lvalue*, ou seja, deve ser atribuível

Passagem por Valor-Resultado

- Considere o seguinte programa que usa passagem por valor-resultado

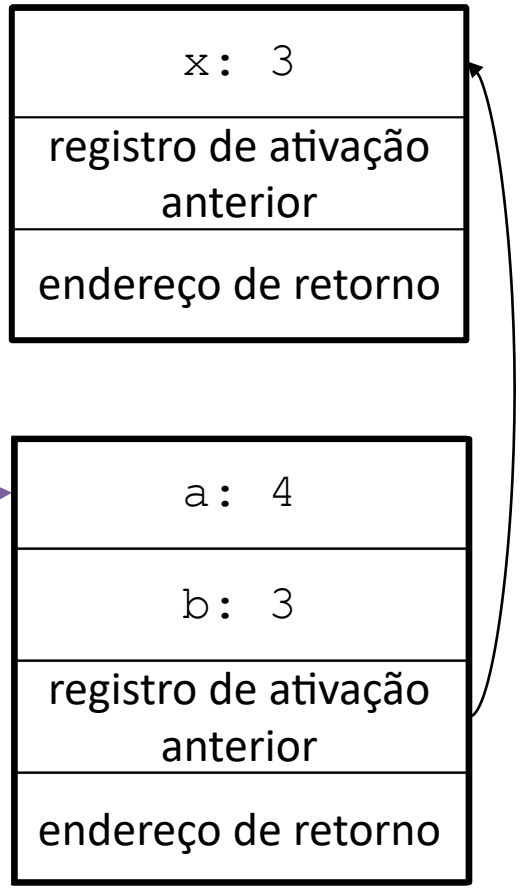


Registro de ativação atual

```
void plus(int a, by-result-value int b) {
   b += a;
   return;
}

void f() {
   int x = 3;
⇒  plus(4, x);
   // ...
}
```

Passagem por Valor-Resultado

- Considere o seguinte programa que usa passagem por valor-resultado

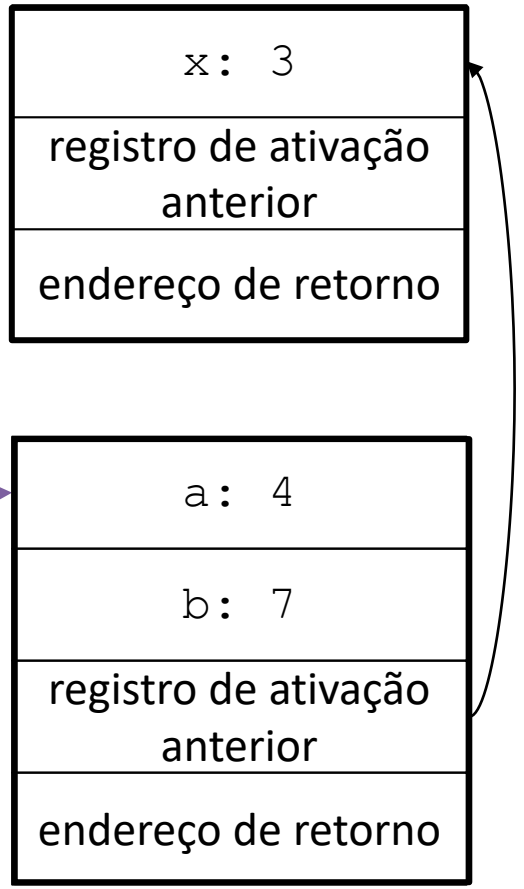


Registro de ativação atual

```
void plus(int a, by-result-value int b) {
⇒ b += a;
  return;
}

void f() {
  int x = 3;
  plus(4, x);
  // ...
}
```

Passagem por Valor-Resultado

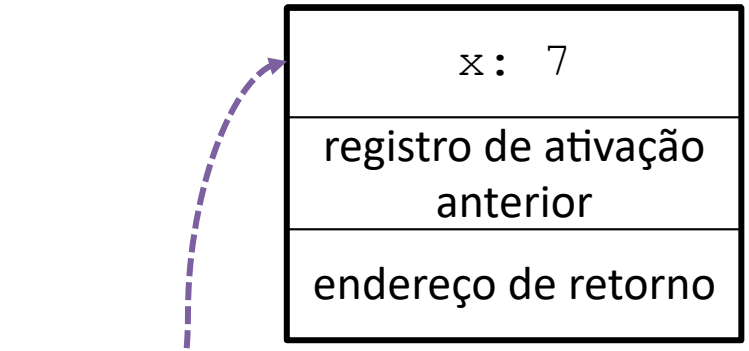
- Considere o seguinte programa que usa passagem por valor-resultado




```
void plus(int a, by-result-value int b) {
   b += a;
⇒  return;
}

void f() {
   int x = 3;
   plus(4, x);
   // ...
}
```

Passagem por Valor-Resultado

- Considere o seguinte programa que usa passagem por valor-resultado



Registro de ativação atual

```
void plus(int a, by-result-value int b) {
   b += a;
   return;
}

void f() {
   int x = 3;
   plus(4, x);
⇒  // ...
}
```

●●● > MECANISMOS > PASSAGEM POR REFERÊNCIA

Passagem por Referência

- Na passagem de parâmetros por referência, o *lvalue* do parâmetro real é computado antes que o método chamado execute; no método chamado, o *lvalue* é usado como *lvalue* do parâmetro formal correspondente; na prática, o parâmetro formal é um sinônimo (alias) do parâmetro real – um outro nome para uma mesma localização de memória

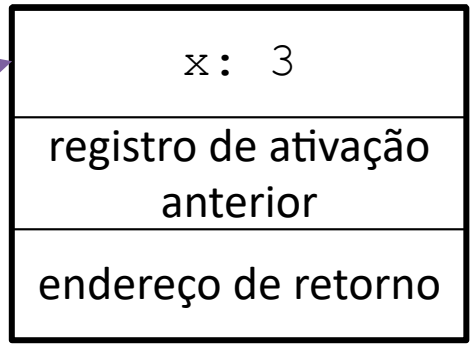
- Foi um dos primeiros métodos utilizados (em Fortran)

- Muito útil para valores grandes, como arranjos, strings, registros e objetos; tem a vantagem de não engordar os registros de ativação

- O parâmetro real deve ser um *lvalue*, ou seja, deve ser atribuível

Passagem por Referência

- Considere o seguinte programa que usa passagem por referência

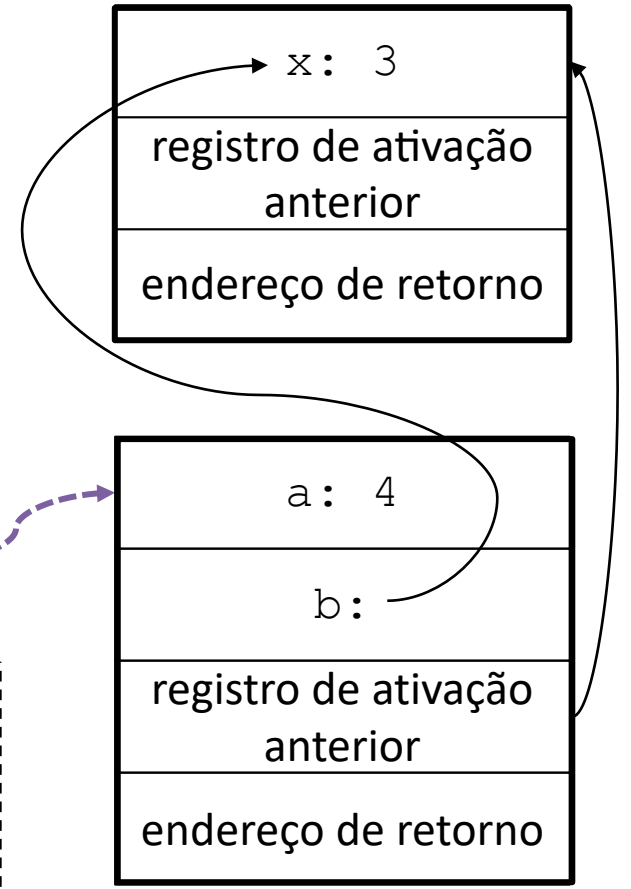


Registro de ativação atual

```
void plus(int a, by-reference int b) {
   b += a;
   return;
}

void f() {
   int x = 3;
⇒  plus(4, x);
   // ...
}
```

Passagem por Referência

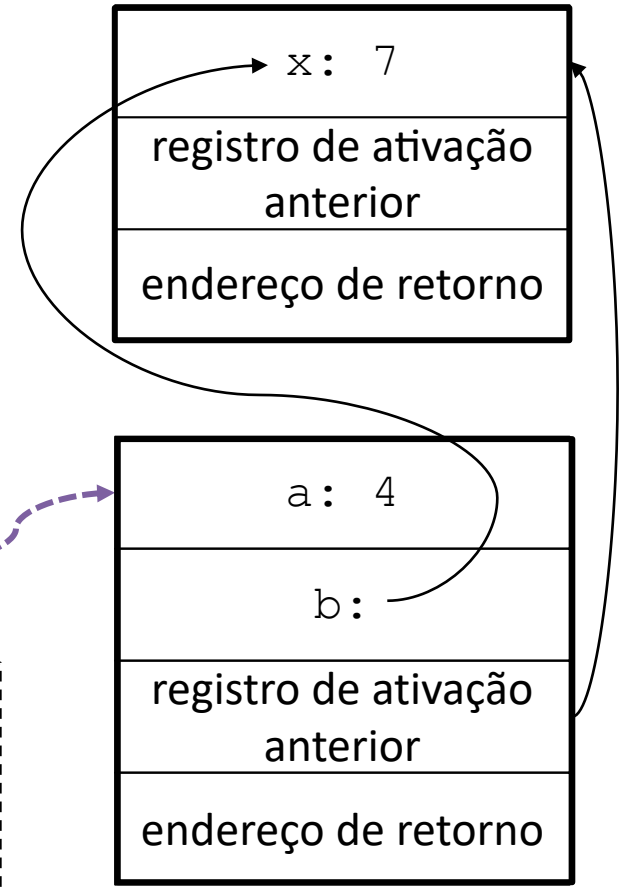
- Considere o seguinte programa que usa passagem por referência




```
void plus(int a, by-reference int b) {
⇒ b += a;
  return;
}

void f() {
  int x = 3;
  plus(4, x);
  // ...
}
```

Passagem por Referência

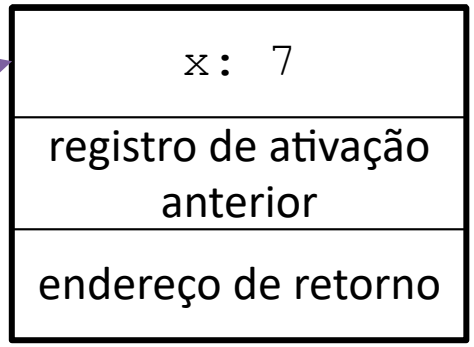
- Considere o seguinte programa que usa passagem por referência




```
void plus(int a, by-reference int b) {
   b += a;
⇒  return;
}

void f() {
   int x = 3;
   plus(4, x);
   // ...
}
```

Passagem por Referência

- Considere o seguinte programa que usa passagem por referência



Registro de ativação atual

```
void plus(int a, by-reference int b) {
   b += a;
   return;
}

void f() {
   int x = 3;
   plus(4, x);
⇒  // ...
}
```

Passagem por Cópia (via Ponteiros) vs Referência

- Comparação das estratégias usadas por C, com passagem por cópia de ponteiros, e por C++, com passagem por referência

```c
void troca(int* x, int* y) {
  int aux = *x;
  *x = *y;
  *y = aux;
}

void f() {
  int a = 3;
  int b = 4;
  troca(&a, &b);
  printf("%d, %d\n", a, b);
}
```

Em C


```c
void troca(int& x, int& y) {
  int aux = x;
  x = y;
  y = aux;
}

void f() {
  int a = 3;
  int b = 4;
  troca(a, b);
  cout << a << "," << b << endl;
}
```

Em C++

Sinônimos (*Aliasing*)

- Embora passagem por referência possa ser implementada eficientemente, ela pode ser perigosa para usar
  - Já que duas expressões diferentes que possuem o mesmo *lvalue* são sinônimos (*alias*) uma da outra

- *Alias* tornam os programas difíceis de entender
  - É de se esperar que uma expressão como `x=y+z` altere somente o valor de `x`, porém se `x` e `y` forem alias esta suposição não é válida

- Alias aparecem em diversas situações

```
ConsCell x =
    new ConsCell(0, null);
ConsCell y = x;
```

```
A[i] = A[j] + A[k];
```

Sinônimos (*Aliasing*)

- *Aliasing* que podem ocorrer na passagem de parâmetros por referência podem ser bem traiçoeiros

```
void sigsum(by-reference int n, by-reference int ans) {
   ans = 0;
   int i = 1;
   while (i <= n)
      ans += i++;
}
```

```
int f() {
   int x, y;
   x = 10;
   sigsum(x, y);
   return y;
}
```

Qual o resultado da execução de `f`?

```
int g() {
   int x;
   x = 10;
   sigsum(x, x);
   return x;
}
```

E de `g`?

●●● > MECANISMOS > PASSAGEM POR EXPANSÃO DE MACROS

Passagem por Expansão de Macros

- Na passagem de parâmetros por expansão de macros, o corpo da macro é avaliada no contexto do chamador; cada parâmetro real é avaliado em todo uso do parâmetro formal correspondente, no contexto de ocorrência desse parâmetro formal (que si próprio está no contexto do chamador)

- Exemplo de macros em C

- Implementação natural: substituição de texto antes da compilação

Passagem por Expansão de Macros

- Embora macros pareçam similares a métodos, eles não são métodos; considere a macro a seguir

```
#define MIN(X,Y) ((X)<(Y)?(X):(Y))
```

- Antes de compilar este programa, um passo de processamento é feito para substituir cada uso de macro por uma cópia completa do corpo da macro com os parâmetros formais substituídos pelos reais

```
a = MIN(b,c);
```

```
a = ((b)<(c)?(b):(c));
```

E se usasse
`MIN(b++, c++)`?

Captura

- Em um fragmento de código, qualquer ocorrência de uma variável que não é estaticamente ligada é livre

- Quando um fragmento de código é movido para um contexto diferente, suas variáveis livres podem ser ligadas

- Esse fenômeno é chamado de **captura**
  - Variáveis livres nos parâmetros formais podem ser capturadas por definições no corpo de uma macro
  - Variáveis livres no corpo de uma macro podem ser capturadas por definições no chamador

Captura

- Exemplo de problema de captura

```
#define intswap(X,Y) {int temp=X; X=Y; Y=temp;}
void main() {
  int temp=1, b=2;
  intswap(temp,b);
  printf("%d, %d\n", temp, b);
}
```

Esse código faz a troca dos valores?

```
void main() {
  int temp=1, b=2;
  {int temp=temp; temp=b; b=temp;}
  printf("%d, %d\n", temp, b);
}
```

●●● > MECANISMOS > PASSAGEM POR NOME

Passagem por Nome

- Na passagem de parâmetros por nome, cada parâmetro real é avaliado no contexto do chamador, em todos os usos do parâmetro formal correspondente
- São como expansões de macros, mas sem o problema da captura
- Como expansões de macros, parâmetros passados por nome são reavaliados sempre que são usados
  - Isso pode ser útil, mas na maioria das vezes é simplesmente desperdício de recursos computacionais
- Suportado por Algol 60 e outras linguagens, mas não é muito popular atualmente

Implementação Passagem por Nome

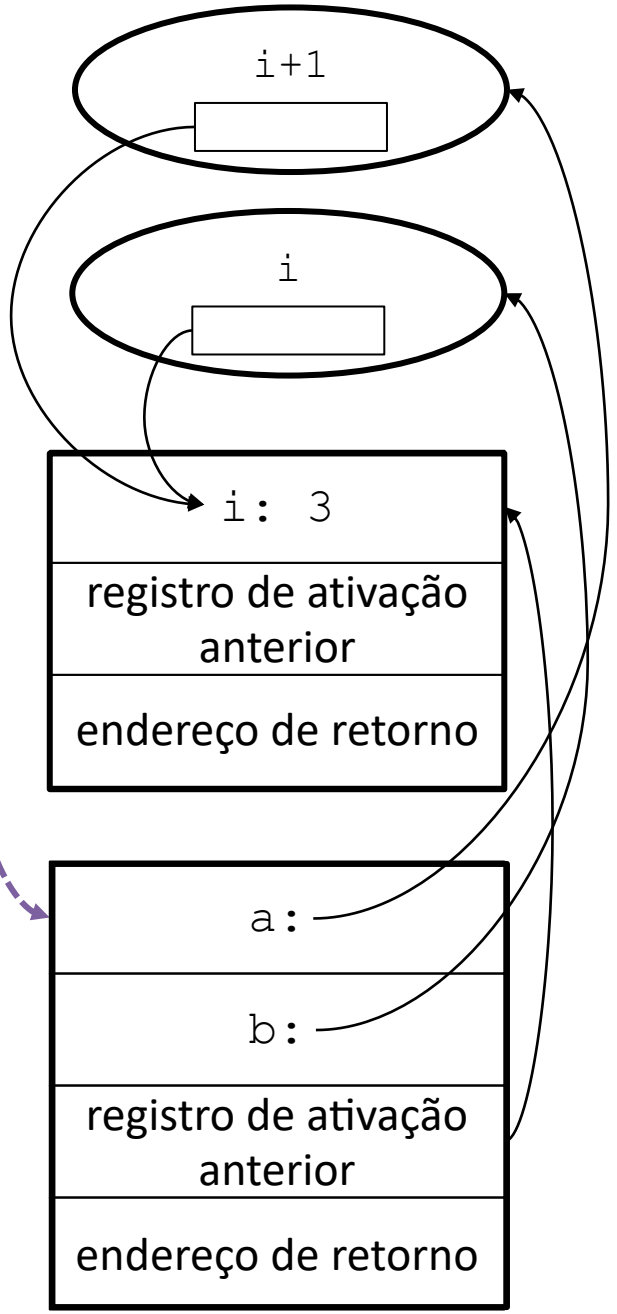
- O parâmetro real é tratado como uma pequena função anônima
  - Sempre que o método chamado precisa acessar o valor do parâmetro formal, seja como *lvalue* ou *rvalue*, ele chama essa função anônima para obtê-lo
  - A função deve ser passada com seu link de aninhamento para que possa ser avaliado no contexto do chamador

Passagem por Nome

- Considere o seguinte programa que usa passagem por nome

Registro de ativação atual

```
void f(by-name int a, by-name int b) {
⇒ b = 5;
  b = a;
  return;
}

void g() {
  int i = 3;
  f(i+1,i);
  // ...
}
```

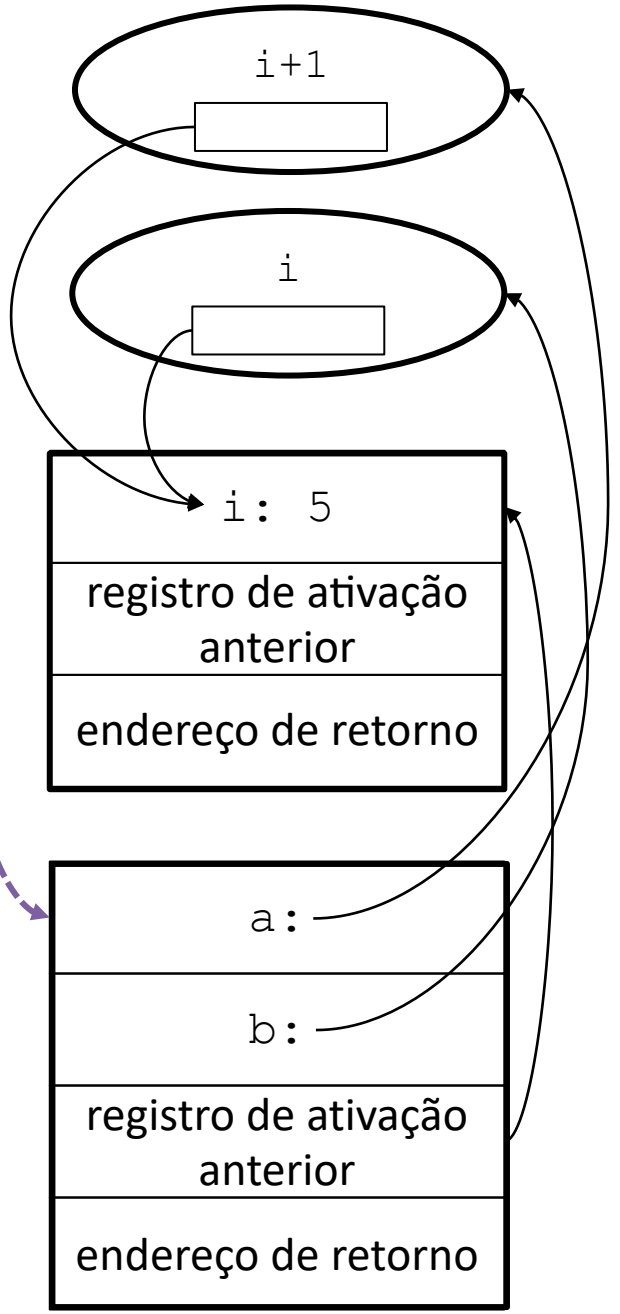


Passagem por Nome

- Considere o seguinte programa que usa passagem por nome

Registro de ativação atual

```
void f(by-name int a, by-name int b) {
   b = 5;
⇒  b = a;
   return;
}

void g() {
   int i = 3;
   f(i+1,i);
   // ...
}
```

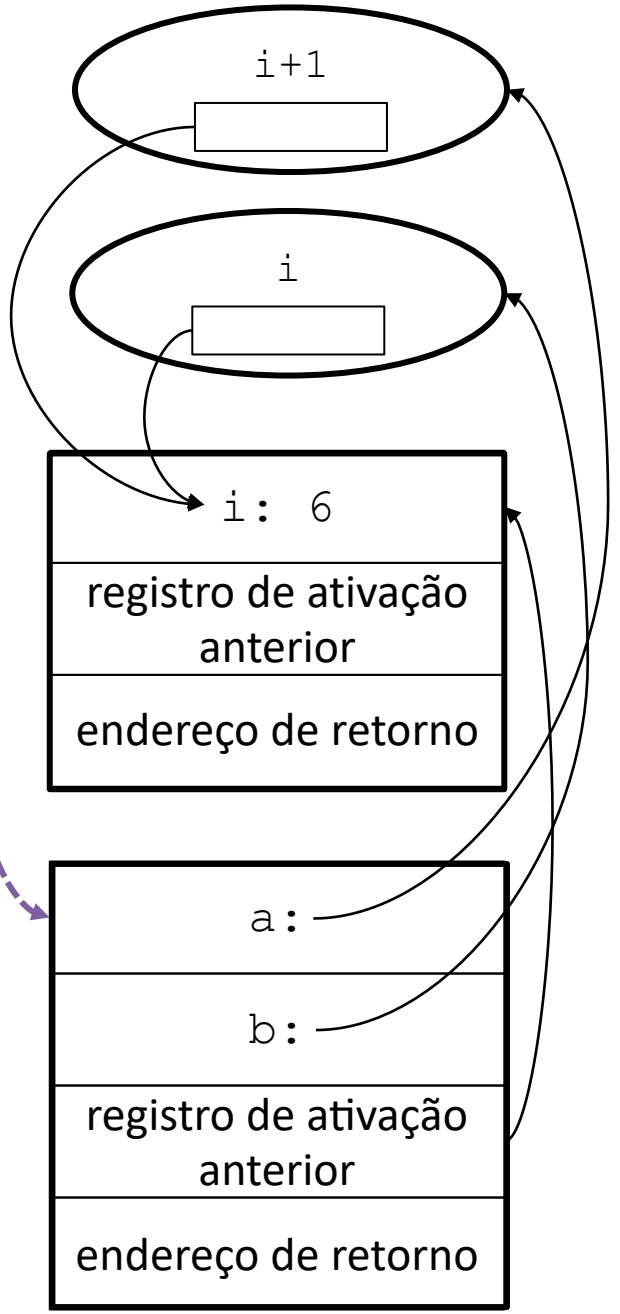


Passagem por Nome

- Considere o seguinte programa que usa passagem por nome

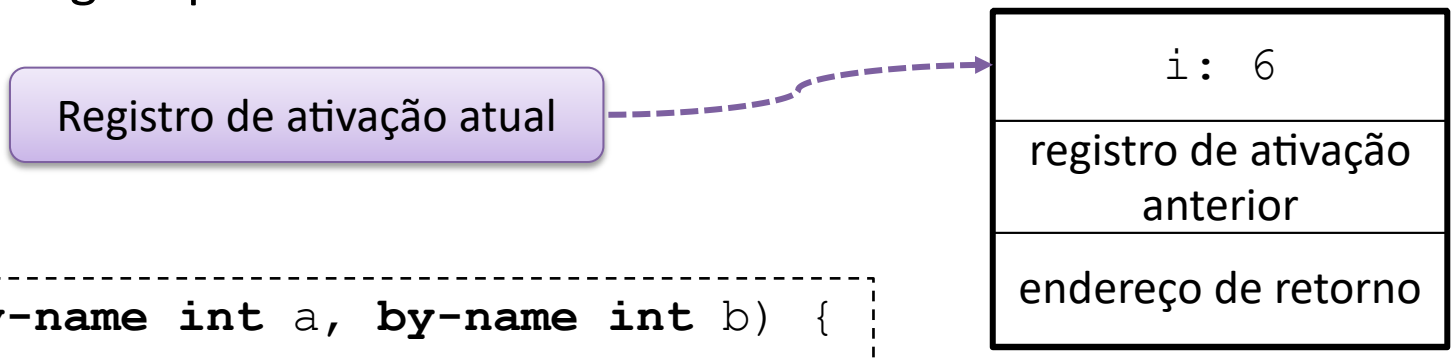
Registro de ativação atual

```
void f(by-name int a, by-name int b) {
  b = 5;
  b = a;
⇒ return;
}

void g() {
  int i = 3;
  f(i+1,i);
  // ...
}
```



Passagem por Nome

- Considere o seguinte programa que usa passagem por nome




```
void f(by-name int a, by-name int b) {
  b = 5;
  b = a;
  return;
}

void g() {
  int i = 3;
  f(i+1,i);
⇒ // ...
}
```

●●● > MECANISMOS > PASSAGEM POR NECESSIDADE

Passagem por Necessidade

- Na passagem de parâmetros por necessidade, cada parâmetro real é avaliado no contexto do chamador, somente no primeiro uso do parâmetro formal correspondente; o valor do parâmetro real é armazenado (*cached*) para que usos subsequentes do parâmetro formal correspondente não cause uma reavaliação

- Usado em linguagens com avaliação preguiçosa, como Haskell

- Evita desperdício de recomputações como na passagem por nome

Passagem por Necessidade

- Considere o seguinte programa que usa passagem por necessidade

```
void f(by-need int a, by-need int b) {
   b = a;
   b = a;
   return;
}

void g() {
   int i = 3;
   f(i+1, i);
   // ...
}
```

Qual o valor de `i` após a chamada de `f`? E se fosse usada a passagem por nome?

Avaliação Preguiçosa

- A passagem de parâmetros por necessidade (assim como expansão de macros e por nome) possuem a propriedade de não avaliar o parâmetro formal se ele não for usado

```
boolean andand(by-need boolean a,
               by-need boolean b) {
  if (!a) return false;
  else return b;
}

boolean g() {
  while (true) {}
  return true;
}

void f() {
  andand(false, g());
}
```

Esse código possui curto-circuito, assim como os operador `&&` de Java e `andalso` de ML

PASSAGEM DE PARÂMETROS > MOMENTO

Linguagens de Programação

Momento da Passagem

- Em que momento o parâmetro de chamada/real deve ser avaliado?
  - **Normal** (*eager*): avaliação na chamada do subprograma
  - **Por nome** (*by name*): avaliação quando parâmetro formal é usado
  - **Por necessidade/preguiçosa** (*lazy*): avaliação quando parâmetro formal é usado pela primeira vez

Maioria das LPs, tais como C, Pascal, Java e ADA, adotam o modo normal

Momento da Passagem

- Exemplo

```
int caso(int x, int w,
         int y, int z) {
    if (x < 0) return w;
    if (x > 0) return y;
    return z;
}
```

```
caso(p(), q(), r(), s());
```

- **Avaliação normal**
  - Avaliação desnecessária de funções na chamada caso
  - Pode reduzir eficiência e flexibilidade
- **Avaliação por nome**
  - Somente uma de `q`, `r` ou `s` seria avaliada
  - Problemas
    - `p` poderia ser avaliada duas vezes
    - `p` poderia produzir efeitos colaterais
- **Avaliação por necessidade.preguiçosa (*lazy*)**
  - Única execução de `p` e somente uma de `q`, `r` ou `s`

ISSO É TUDO, PESSOAL!