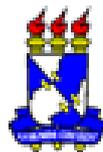


Parâmetros

Prof. Alberto Costa Neto
alberto@ufs.br

Linguagens de Programação



Departamento de Computação
Universidade Federal de Sergipe

Retrospectiva

- **Abstração de função** é uma abstração sobre uma expressão
 - Uma chamada de função é uma expressão que retorna um valor
- **Abstração de procedimento** é uma abstração sobre um comando
 - Uma chamada de procedimento é um comando que atualiza variáveis



Parâmetros

- Uma abstração pode ser parametrizada
 - Evita-se a repetição do código para a mesma computação
- **Argumento:** é um valor que pode ser passado para uma abstração parametrizada
- **Parâmetro formal:** identificador usado na abstração para denotar um argumento
- **Parâmetro real:** uma expressão passada como argumento e é fornecida no momento da invocação da abstração (chamada da abstração)



Parâmetros

- Argumentos são Valores de uma linguagem
 - Em Pascal:
 - Valores primitivos
 - Valores compostos (exceto arquivos)
 - Apontadores
 - Referências a variáveis
 - Funções e procedimentos não são valores em Pascal



Parâmetros

```
float area (float r) {  
    return 3.1416 * r * r;  
}  
main() {  
    float diametro, resultado;  
    diametro = 2.8;  
    resultado = area (diametro/2);  
}
```

Parâmetro formal

Parâmetro real

Qual o Argumento?

valor 1.4



Parâmetros: Correspondência

- Correspondência entre parâmetros reais e formais
 - Por posição
 - A sequência na qual os parâmetros são escritos define a correspondência
 - Ex: Pascal, C++, Java
 - Por palavras chave
 - Vantagem: a ordem é irrelevante
 - Desvantagem: o usuário precisa saber os nomes dos parâmetros formais
 - Valores Default



Parâmetros: Correspondência

- Por palavras chave

```
procedure exemplo is
  x: integer := 2;
  y: integer := 3;
  z: integer := 5;
  res: integer;
  function soma(a1, a2, a3: integer) return integer is
  begin
    return a1 + a2 + a3;
  end soma;
begin
  res := soma(a3=>y, a1=>z, a2=>x);
end exemplo;
```

em ADA



Parâmetros: Correspondência

- Valores Default

```
int total(int a[],int inicio = 0,int fim = 12,int incr = 1){
    int total = 0;
    for (int i = inicio; i < fim; i += incr)
        total+=a[i];
    return total;
}
main() {
    int[] vendas = {90,40,85,91,55,63,21,35,38,73,50,60};
    int vtotal, vtrim1, vtrim2, vAgoOutDez;
    vtotal = total(vendas);
    vtrim1 = total(vendas, 0, 3);
    vtrim2 = total(vendas, 3, 6);
    vAgoOutDez = total(vendas, 7, 12, 2);
}
```

em C++

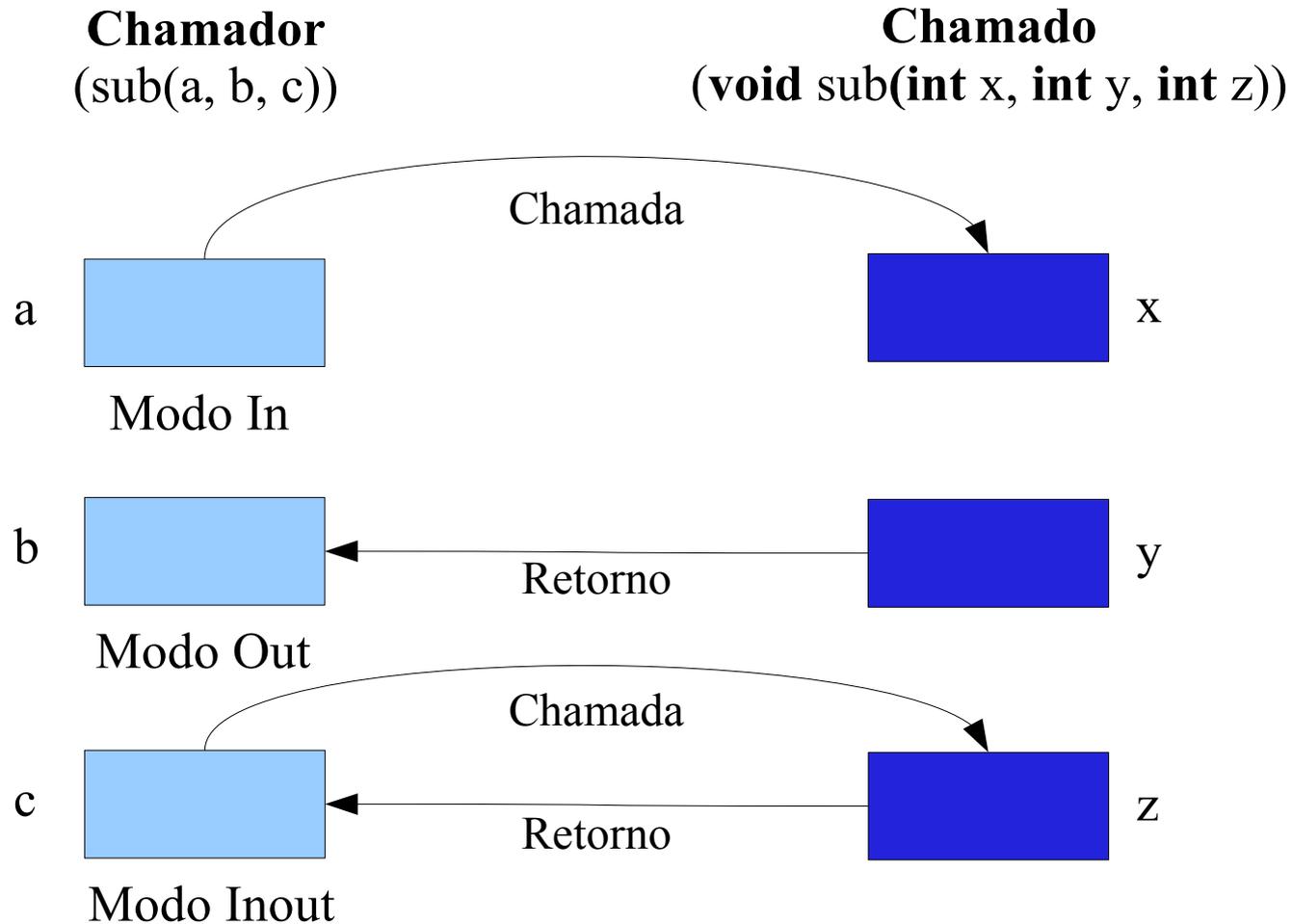


Passagem de Parâmetros

- Processo no qual os parâmetros formais assumem seus respectivos valores durante a execução de um subprograma
- Três aspectos importantes:
 - Modelo Semântico da passagem (in, out e inout)
 - Mecanismo de passagem (cópia, referência e nome)
 - Momento no qual a passagem é realizada (Eager ou Lazy evaluation)



Modelos semânticos



Mecanismo de Passagem

- Quando uma abstração é chamada, cada parâmetro formal é vinculado ao argumento correspondente
- Há três tipos de mecanismos para associar argumentos a parâmetros formais:
 - Por cópia
 - Por referência
 - Por nome
 - Substituição do parâmetro formal textualmente pelo parâmetro real (pouco usada)



Passagem por Cópia

- O parâmetro real é avaliado resultando no argumento
- O parâmetro formal denota uma variável local à abstração
 - Criada na entrada da abstração
 - Destruída na saída da abstração
- Variações e semântica correspondente:
 - Por Valor (**in**)
 - Por Resultado (**out**)
 - Por Valor-Resultado (**inout**)



Passagem por Valor

- Na entrada da abstração, o valor do argumento é atribuído ao parâmetro formal X que se comporta como uma variável local
 - Valor de X pode ser inspecionado e atualizado
 - Atualização de X não tem efeito em variáveis não-locais



Passagem por Resultado

- O argumento de entrada é uma variável (com valor indefinido)
- Na entrada da abstração, o parâmetro formal X tem um valor indefinido e se comporta como uma variável local
 - Valor pode ser atualizado
 - Não deveria ser inspecionado (dificulta implementação)
 - No final, valor de X é atribuído à variável argumento recebida



Passagem por Valor-Resultado

- Combina as passagens Por Valor e Por Resultado
- O argumento de entrada é uma variável (valor definido)
- Na entrada da abstração, o valor do argumento é atribuído ao parâmetro formal X que se comporta como uma variável local
 - Valor pode ser inspecionado e atualizado
 - No final, valor de X é atribuído de volta à variável argumento recebida



Passagem por Cópia

Ada

```
type Vector is array(1..n) of Float;  
procedure add(v, w: in Vector; sum: out Vector) is  
begin (1)  
  for i in 1 .. n loop  
    sum[i] := v[i]+ w[i];  
  end loop;  
end; (2)
```

```
var a,b,c:Vector;  
...add(a,b,c);...
```

***v* e *w* são alocadas e inicializadas com os valores de *a* e *b*;
sum é alocada, mas não é inicializada com o valor de *c*.**

***v* e *w* são desalocadas
valores de *a* e *b* permanecem inalterados
conteúdo de *sum* é copiado para *c* e *sum* é desalocada**



Passagem por Cópia

u* é alocada e inicializada com o conteúdo de *c

Ada

```
procedure normalize (u: in out Vector) is
  s: Float := 0.0;
begin (3)
  for i in 1 .. n loop
    s := s + u(i)**2;
  end loop;
  s := sqrt(s);
  for i in 1 .. n loop
    u(i) := u(i)/s;
  end loop;
end; (4)
```

`...normalize(c);`

o conteúdo de *u* é copiado para *c* e *u* é desalocada



Passagem por Referência

- Permite que o parâmetro formal seja associado diretamente com o argumento
 - Não há alocação interna (variável local)
- Tipo do parâmetro real:
 - Constante
 - Referência



Passagem por Referência

- Declarações `var` e `const` do Pascal/Ada
- Parâmetro formal é associado ao argumento, que é uma referência para uma variável (`var`) ou valor (`const`)
- Qualquer leitura ou atualização feita sobre o parâmetro formal é uma leitura ou atualização da variável



Passagem por Referência

em Pascal

```
type Vector = array[1..n] of Real;  
procedure add(const v, w: Vector; var sum:Vector);  
var i:1..n; (1)  
begin  
  for i:=1 to n do  
    sum[i]:=v[i]+w[i];  
end; (2)
```

```
var a,b,c:Vector;  
...add(a,b,c);...
```

**v e w são associadas aos valores das constantes a e b;
sum é associada com a posição de memória onde a
variável c está alocada**

nada acontece

```
for i:= 1 to n do u[i]:=u[i]/s;  
end; (4)
```



Passagem por Referência

u é associado com a posição de memória onde a variável c está alocada

em Pascal

```
begin
  for i:=1 to n do sum[i]:=v[i]+w[i];
end; (2)
procedure normalize(var u:Vector);
var i: 1..n; (3)
    s: Real;
begin
  s = 0.0;
  for i:= 1 to n do
    s := s+sqr(u[i]);
  s = sqrt(s);
  for i:= 1 to n do
    u[i]:=u[i]/s;
end; (4)
```

...normalize(c);

nada acontece



Passagem por Referência

- Vantagens:
 - Semântica uniforme e simples; válido para todos os tipos de valores
 - Eficiente (exceto para sistemas distribuídos)
- Desvantagens:
 - *Aliasing*: dois identificadores associados a uma mesma variável
 - Dificulta entendimento do código



Alias e passagem por Referência

```
procedure calcule(var a, b: Integer);  
begin  
    b := 1;  
    b := a + b;  
end;  
  
...  
i := 4;  
... calcule(i, i); ...
```

em Pascal

i resultará em que valor? 2

a e b são *aliases*!



Parâmetros nas principais LP

- ADA (suportas as 3 semânticas usando in, out e inout)
- C e C++
 - Só suportam por valor
 - Podem usar apontadores para simular referência
- Java (só suporta passagem por valor)
- Pascal
 - Suporta por valor e referência (var e const)
- C# (padrão por valor – **in**)
 - **inout** com passagem por referência (ref)
 - **out** implementada via passagem por referência



Sugestões de Leitura

- Concepts of Programming Languages (Robert Sebesta)
 - Seções 9.1 até 9.5
- Programming Language Concepts and Paradigms (David Watt)
 - Seção 5.2
- Linguagens de Programação (Flávio Varejão)
 - Seções 6.2.1.2 até 6.2.1.5

