unesp

Linguagens de Programação

Aula 10

Celso Olivete Júnior

olivete@fct.unesp.br

unesp

Na aula passada

- As sentenças de controle das LP imperativas ocorrem em diversas categorias: seleção, seleção múltipla, iteração e desvio incondicional

unesp

Na aula de hoje

Estruturas de controle no nível de unidades

➔ **Subprogramas**

Roteiro

- Introdução
- Fundamentos de subprogramas
- Questões de projeto para subprogramas
- Ambientes de referenciamento local
- Métodos de passagem de parâmetros
- Parâmetros que são subprogramas
- Subprogramas sobrecarregados
- Questões de projeto para funções
- Corrotinas

Introdução

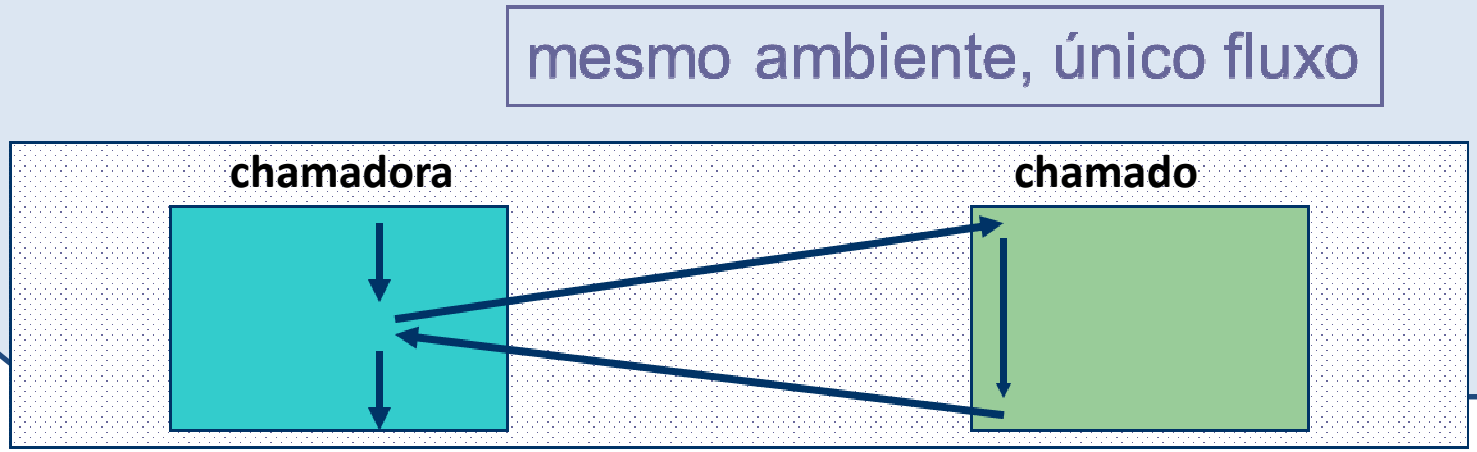
- Dois mecanismos fundamentais de abstração em LPs
  - Abstração de processos
    - ✓ Desde o início da história das linguagens de programação
  - Abstração de dados
    - ✓ Desde o início dos anos 1980

Fundamentos de subprogramas

- Um subprograma é um tipo de abstração de processo
  - Instruções para realizar uma tarefa são agrupadas e tratadas como uma unidade lógica
  - O conceito economiza espaço e esforço de desenvolvimento e codificação

Fundamentos de subprogramas

- Cada subprograma tem um único ponto de entrada
- A unidade de programa chamadora é suspensa durante a execução do subprograma chamado
- O controle sempre retorna para o chamador quando a execução do subprograma termina



Subprogramas: Definições básicas

- Uma definição de subprograma consiste na descrição de seu cabeçalho e de seu corpo
  - O cabeçalho de um subprograma inclui tipo do subprograma, nome, parâmetros formais, tipo de retorno
  - O corpo de um subprograma é a descrição das ações da abstração do subprograma
- Uma chamada a um subprograma é uma requisição explícita que diz que o subprograma deve ser executado

Definições básicas

- Um parâmetro formal é uma variável que aparece no cabeçalho do subprograma e é usada no corpo do subprograma

- Um parâmetro real representa um valor ou endereço usado na chamada do subprograma

Definições básicas

- O perfil de parâmetros (ou assinatura) de um subprograma contém o número, a ordem e os tipos de seus parâmetros formais

- O protocolo é o perfil de parâmetros mais, se for uma função, seu tipo de retorno

Definições básicas

- Declarações de funções em C e C++ são chamadas de protótipos
- Uma declaração de subprograma fornece o protocolo, mas não inclui seu corpo
- Um parâmetro formal é uma variável listada no cabeçalho do subprograma e usado nele
- Um parâmetro real representa um valor ou endereço usado na sentença de chamada do subprograma

Correspondência entre os parâmetros reais e formais

- Posicional
  - A vinculação dos parâmetros reais a parâmetros formais é por posição:
    - ✓ o primeiro real é vinculado ao primeiro formal e assim por diante
  - Seguro e efetivo

- Palavra-chave
  - O nome do parâmetro formal a que um parâmetro real deve ser vinculado é especificado com o parâmetro real
  - Ex: comando ***printf***
  - Vantagem: Parâmetros podem aparecer em qualquer ordem, evitando erros de correspondência
  - Desvantagem: O usuário deve saber os nomes dos parâmetros formais

Valores padrão de parâmetros formais

- Em certas linguagens (como C++, Python, Ruby, Ada e PHP), parâmetros formais podem ter valores padrão (se nenhum parâmetro real é passado)
  - Em C++, parâmetros padrão devem aparecer por último, já que os parâmetros são posicionalmente associados
  - Ex: protótipo **int** calcular (int a, int b, int c = 1)
  - Chamada/invocação: x = calcular (10, 20); //c receberá 1

Procedimentos e funções

- Existem duas categorias de subprogramas
  - Procedimento são coleções de instruções parametrizadas que definem uma determinada abstração
  - Funções parecem estruturalmente com os procedimentos, mas são semanticamente modeladas como funções matemáticas
    - ✓ Se uma função é um modelo fiel, ela não produz efeitos colaterais
    - ✓ Na prática, muitas funções em programas têm efeitos colaterais

Questões de projeto para subprogramas

1. As variáveis locais são alocadas estaticamente ou dinamicamente?
2. As definições de subprogramas podem aparecer em outras definições de subprogramas?
3. Que método ou métodos de passagem de parâmetros são usados?
4. Os tipos dos parâmetros reais são verificados?
5. Se os subprogramas puderem ser passados como parâmetros e puderem ser aninhados, qual é o ambiente de referenciamento de um subprograma passado como parâmetro?
6. Os subprogramas podem ser sobrecarregados?

Ambientes de referenciamento local

- Variáveis locais: são variáveis que são definidas dentro de um subprograma, e geralmente têm o mesmo escopo do subprograma
- Variáveis locais podem ser implementadas como dinâmicas da pilha
  - O ambiente é criado de acordo com a ativação. **Vantagens**:
    - ✓ Suporte para recursão
    - ✓ Armazenamento para variáveis locais é compartilhado entre alguns subprogramas
  - **Desvantagens:**
    - ✓ Custo para alocação, liberação, tempo de inicialização
    - ✓ Endereçamento indireto
    - ✓ Subprogramas não podem ser sensíveis ao histórico

Ambientes de referenciamento local

- Variáveis locais: são variáveis que são definidas dentro de um subprograma, e geralmente têm o mesmo escopo do subprograma
- Variáveis locais podem ser estáticas
  - Desvantagem: não permite recursividade
  - Vantagens:
    - não há endereçamento indireto (mais eficiente);
    - não existe reserva e libertação de memória

Ambientes de referenciamento local

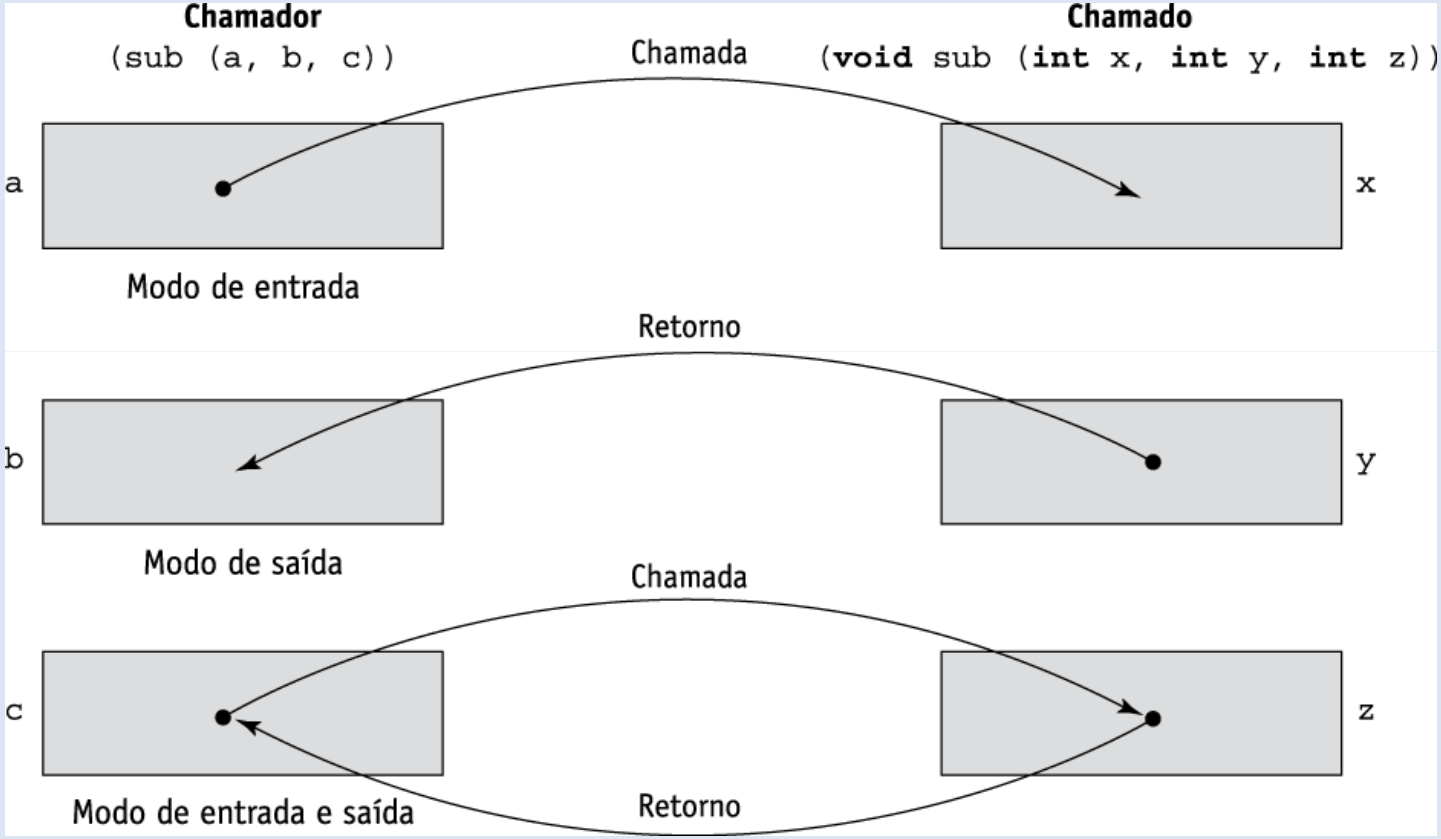
- Linguagens que permitem ambiente estático:
  - COBOL, muitas versões de FORTRAN e Algol.
  - FORTRAN 77 e 90 – Quase sempre são alocações estáticas (pode haver alocação dinâmica Pilha).
  - C - Por omissão é dinâmico de pilha mas variáveis podem ter atributo **static**;

- Linguagens que permitem ambiente de Pilha dinâmica:
  - C/C++, Pascal, Ada, Lisp, APL, SNOBOL, etc.

Modelos semânticos de passagem de parâmetros

- Formas de transmitir os parâmetros para os subprogramas
  - Modo de entrada: receber dados a partir do parâmetro real correspondente
  - Modo de saída: transmitir dados para o parâmetro real
  - Modo de entrada e saída: fazer ambos

Modelos de passagem de parâmetros



Modelos conceituais de transferência de dados

- Um valor real é copiado (movimento físico): para o chamador, para o chamado ou para ambos
- Um caminho de acesso é transmitido (por um ponteiro ou por uma referência)
- Formas:
  1. Passagem por valor (modo de entrada)
  2. Passagem por resultado (modo de saída)
  3. Passagem por valor/resultado (modo de entrada/saída)
  4. Passagem por Referência (modo entrada/saída)
  5. Passagem por nome (modo múltiplo)

1. Passagem por valor
modo de entrada

- O valor do parâmetro real é usado para inicializar o parâmetro formal correspondente
  - Normalmente implementada por cópia
  - Poderia ser implementada transmitindo um caminho de acesso para o valor do parâmetro real no chamador, mas isso requereria que o valor estivesse em uma célula com proteção contra escrita (uma que pudesse ser apenas lida)
  - Desvantagens (se cópias são usadas): é necessário armazenamento adicional para o parâmetro formal e o movimento pode ser custoso

2. Passagem por resultado
modo de saída

- Quando um parâmetro (modo de saída) é passado por resultado, nenhum valor é transmitido para o subprograma.
- O parâmetro formal correspondente age como uma variável local, mas logo antes de o controle ser transmitido de volta para o chamador, seu valor é transmitido de volta para o parâmetro real deste.
- Problema em potencial: sub(p1, p1); pode existir uma colisão entre parâmetros reais

3. Passagem por valor-resultado
modo entrada/saída

- Transferência de valores em ambas as direções
- Também conhecido como passagem por cópia
- Desvantagens:
  - As mesmas da passagem por resultado
  - As mesmas da passagem por valor

4. Passagem por referência
modo entrada/saída

- Transmite um caminho de acesso (endereço)
- Vantagem: processo de passagem é eficiente (não são necessárias cópias nem espaço duplicado) → parâmetro real é compartilhado
- Desvantagens
  - Acessos mais lentos (do que na passagem por valor)
  - Potenciais efeitos colaterais (colisões)
  - Apelidos podem ser criados

5. Passagem por nome
modo múltiplo

- Parâmetro formal é substituído textualmente pelo real

- Quando os parâmetros são passados por nome, o parâmetro real é, na prática, textualmente substituído pelo parâmetro formal correspondente em todas as suas ocorrências no subprograma

Implementando métodos de passagem de parâmetros

- Na maioria das linguagens contemporâneas, a comunicação via parâmetros ocorre por meio da pilha de tempo de execução
- Passagem por referência é a mais simples de implementar; apenas seu endereço deve ser colocado na pilha
- Um erro sutil, mas fatal, pode ocorrer com parâmetros com passagem por referência e passagem por valor-resultado: um formal correspondente a uma constante pode ser trocado erroneamente

Métodos de passagem de parâmetros das principais linguagens

- C
  - Passagem por valor
  - A passagem por referência é atingida por meio do uso de ponteiros como parâmetros
- C++
  - Inclui o tipo referência para passagem por referência
- Java
  - Todos os parâmetros têm passagem por valor
  - Parâmetros objetos têm passagem por referência

Métodos de passagem de parâmetros das principais linguagens (cont.)

- Fortran 95
  - Parâmetros podem ser declarados para serem dos modos de entrada, de saída ou de entrada e saída
- C#
  - Método padrão: passagem por valor
  - Passagem por referência é especificada precedendo um parâmetro formal e seu real correspondente com ref
- PHP: similar a C#

Métodos de passagem de parâmetros
Exemplo I

- Exemplo em Ling. C:
- Transmissão ocorre somente por valor. Porém transmitindo-se o endereço e o efeito é "semanticamente semelhante" à passagem por referência.

```
void swap(int *x, *y)
{ int temp = *x;
*x = *y;
*y = temp;
}
swap(&a,&b);
```

Métodos de passagem de parâmetros
Exemplo II

- Exemplo em Ling. C++:
- Passagem de valor como em C , contudo, também permite passagem por referência explicita através do operador '&' no parâmetro formal.

```
void swap(int & x, int & y)
{ int temp = x;
x = y;
y = temp;
}
```

- Invocação:

```
swap(a,b); // para o valor de 2 variáveis
```

Verificação de tipos dos parâmetros

- Considerado importante para confiabilidade
  - FORTRAN 77 e versão original do C: não tinham
  - Pascal, FORTRAN 90, Java e Ada: sempre requerem
  - ANSI C e C++: escolha é feita pelo usuário
  - Linguagens relativamente novas como Perl, JavaScript e PHP não requerem verificação de tipos

Matrizes multidimensionais como parâmetros

- Se uma matriz multidimensional é passada para um subprograma e o subprograma é compilado separadamente, o compilador precisa saber o tamanho declarado dessa matriz para construir a função de mapeamento de armazenamento

Matrizes multidimensionais como parâmetros: C e C++

- Matriz pode ser passada como um ponteiro, e as dimensões reais da matriz podem ser incluídas como parâmetros

- A função pode avaliar a função de mapeamento de armazenamento escrita pelo usuário usando aritmética de ponteiros cada vez que um elemento da matriz precisar ser referenciado

unesp

Considerações de projeto para passagens de parâmetros

- Duas considerações importantes
  - Eficiência
  - Transferências de dados de uma via ou de duas vias
- Mas as considerações acima estão em conflito
  - Boa programação sugere acesso limitado a variáveis, o que significa uma via, sempre que possível
  - Mas passagem por referência é mais eficiente para passar estruturas de tamanho significativo

Parâmetros que são subprogramas

- Às vezes, é conveniente enviar os nomes de subprogramas como parâmetros
- Questões:
  1. Tipos de parâmetros são verificados?
     - ➔ Pascal (inicial) e FORTRAN 77 não
     - ➔ C e C++ sim
  2. Qual é o ambiente de referenciamento correto para um subprograma enviado como parâmetro?

Parâmetros que são subprogramas: verificação de tipos dos parâmetros

- C e C++: as funções não podem ser passadas como parâmetros, mas ponteiros para funções podem

- FORTRAN 95 possui um mecanismo para fornecer tipos de parâmetros para subprogramas que são passados como parâmetros

unesp

Parâmetros que são subprogramas: ambiente de referenciamento

- Vinculação rasa (shallow binding): O ambiente do subprograma onde o subprograma atua como parâmetro atual
- Vinculação profunda (Deep binding): O ambiente da definição do subprograma passado
- Vinculação ad hoc (Ad hoc binding): O ambiente da sentença de chamada que passou o subprograma como um parâmetro real

Parâmetros que são subprogramas: ambiente de referenciamento

- Vinculação rasa (shallow binding): é utilizado em linguagens de escopo dinâmico (Pascal e SNOBOL), onde o ambiente do subprograma invocado é mais natural.

- Vinculação profunda (Deep binding): é utilizado em linguagens de escopo estático (C e C++), porque o ambiente de definição é o mais natural.

Parâmetros que são subprogramas: ambiente de referenciamento

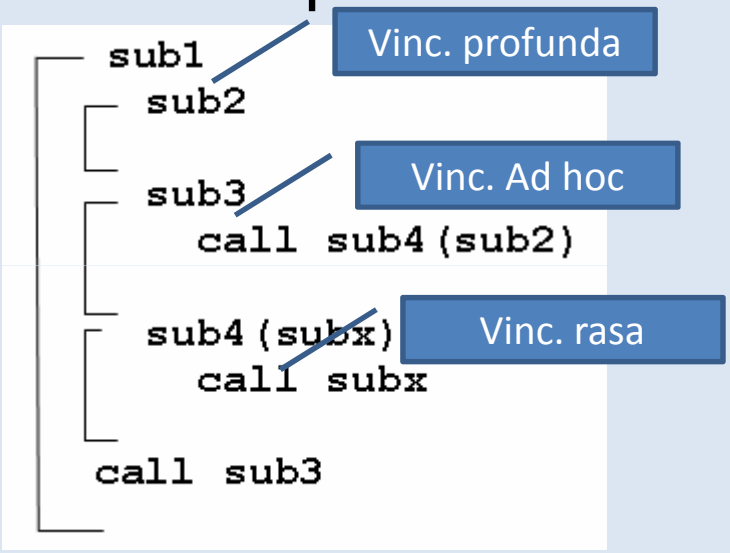
- Exemplo

```
sub1
  sub2

  sub3
    call sub4(sub2)

  sub4(subx)
    call subx

call sub3
```

Qual é o ambiente de referência de sub2 quando for chamado em sub4? (sub2 é parâmetro de sub4).

Parâmetros que são subprogramas: ambiente de referenciamento

- Exemplo



Qual é o ambiente de referência de sub2 quando for chamado em sub4? (sub2 é parâmetro de sub4).

Subprogramas sobrecarregados

- Um subprograma sobrecarregado é um subprograma que possui o mesmo nome de outro subprograma no mesmo ambiente de referenciamento
- C++, Java, C# e Ada incluem subprogramas sobrecarregados pré-definidos
- Ada, Java, C++ e C# permitem aos usuários escrever múltiplas versões de subprogramas com o mesmo nome

Subprogramas sobrecarregados
Exemplo em C++

- Um subprograma sobrecarregado é um subprograma que possui o mesmo nome de outro subprograma no mesmo ambiente de referenciamento

```
int maximo(int x, int y)
{ return x>y ? x : y; }
double maximo(double x, double y)
{ return x>y ? x : y; }
int maximo(int x, int y, int w, int z)
{ return
maximo(maximo(x,y),maximo(y,z));}
```

Questões de projeto para funções

- Os efeitos colaterais são permitidos?
  - Os parâmetros para funções devem ser sempre parâmetros no modo de entrada
- Que tipos de valores podem ser retornados?
  - A maioria das linguagens de programação imperativas restringe os tipos que podem ser retornados
  - C permite que quaisquer tipos, exceto matrizes e funções
  - C++ também permite tipos definidos pelo usuário
  - Métodos em Java e C# podem retornar qualquer tipo

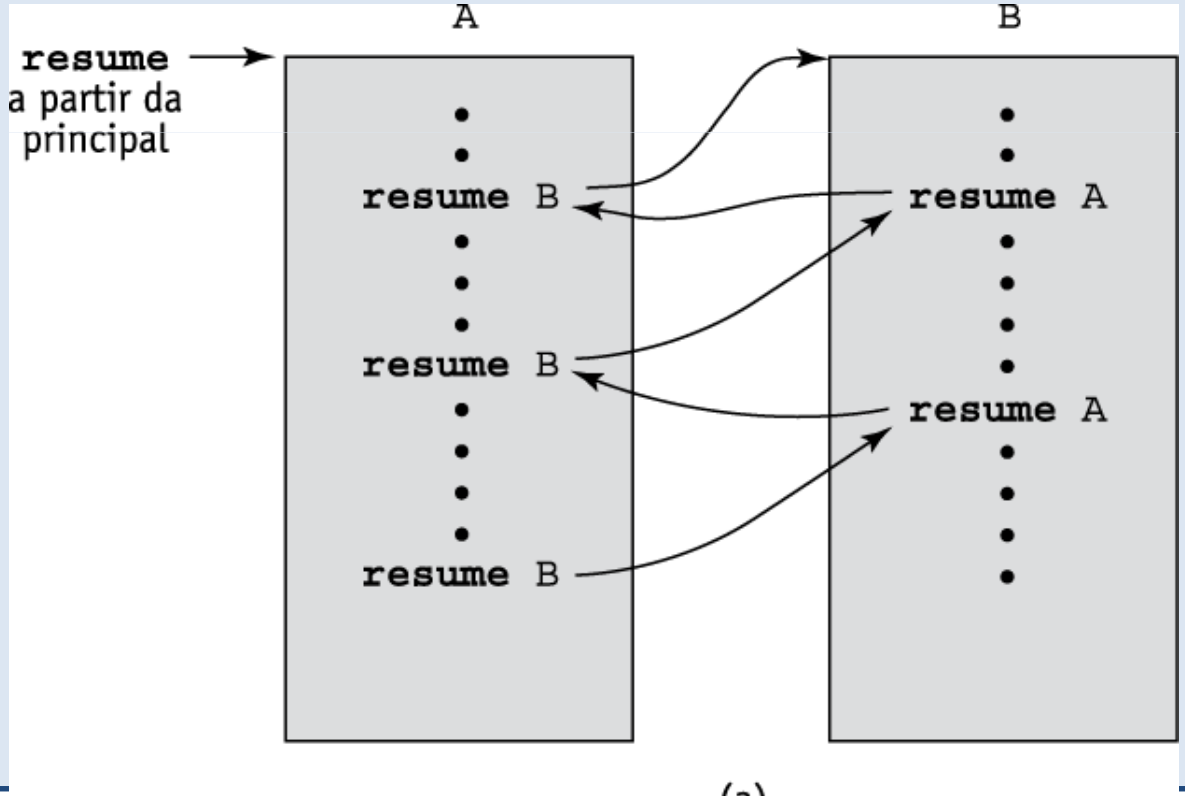
unesp

Corrotinas

- Uma corrotina é um subprograma especial que possui múltiplas entradas; as corrotinas chamadora e chamada estão em um relacionamento mais igualitário
- A invocação de uma corrotina é a chamada de uma continuação (***resume)*** em vez de uma chamada
- O mecanismo de controle das corrotinas é frequentemente chamado de modelo de controle de unidades simétrico

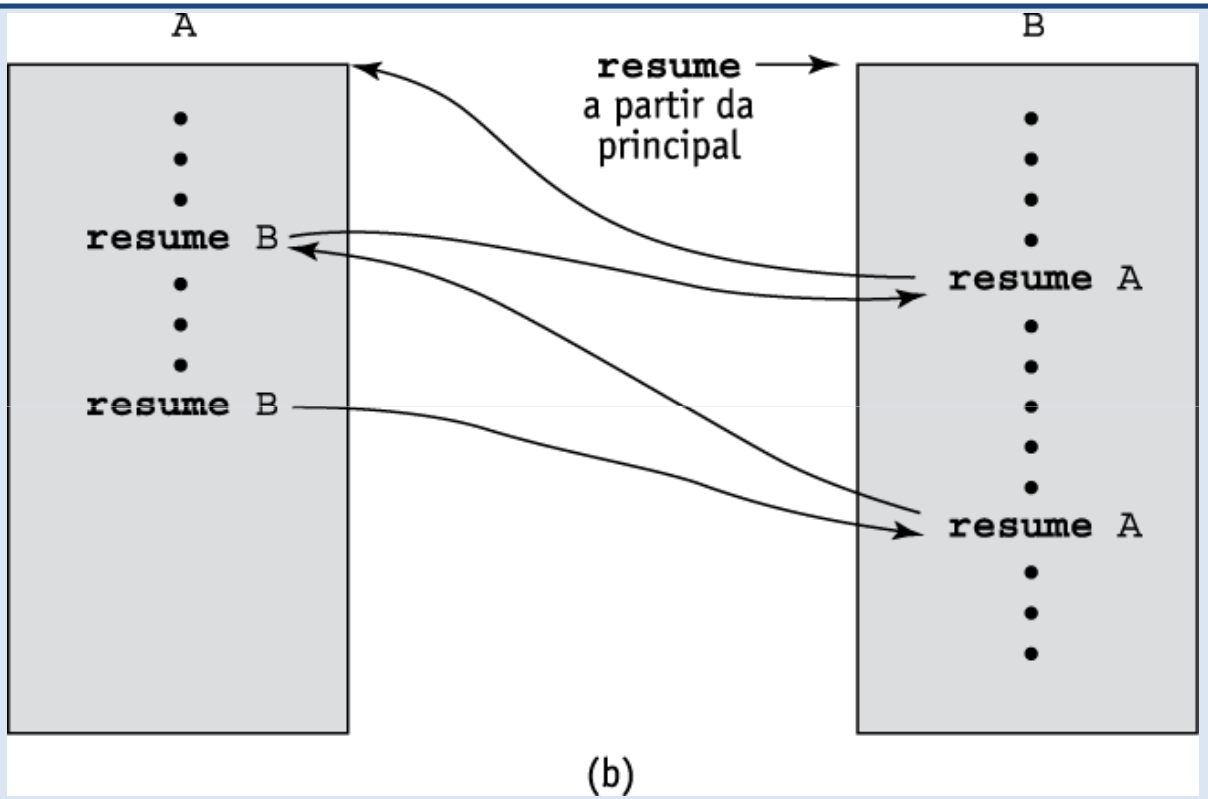
Sequências de controle de execução possíveis

Tipicamente, corrotinas repetidamente retomam a execução entre si, possivelmente em laço infinito.

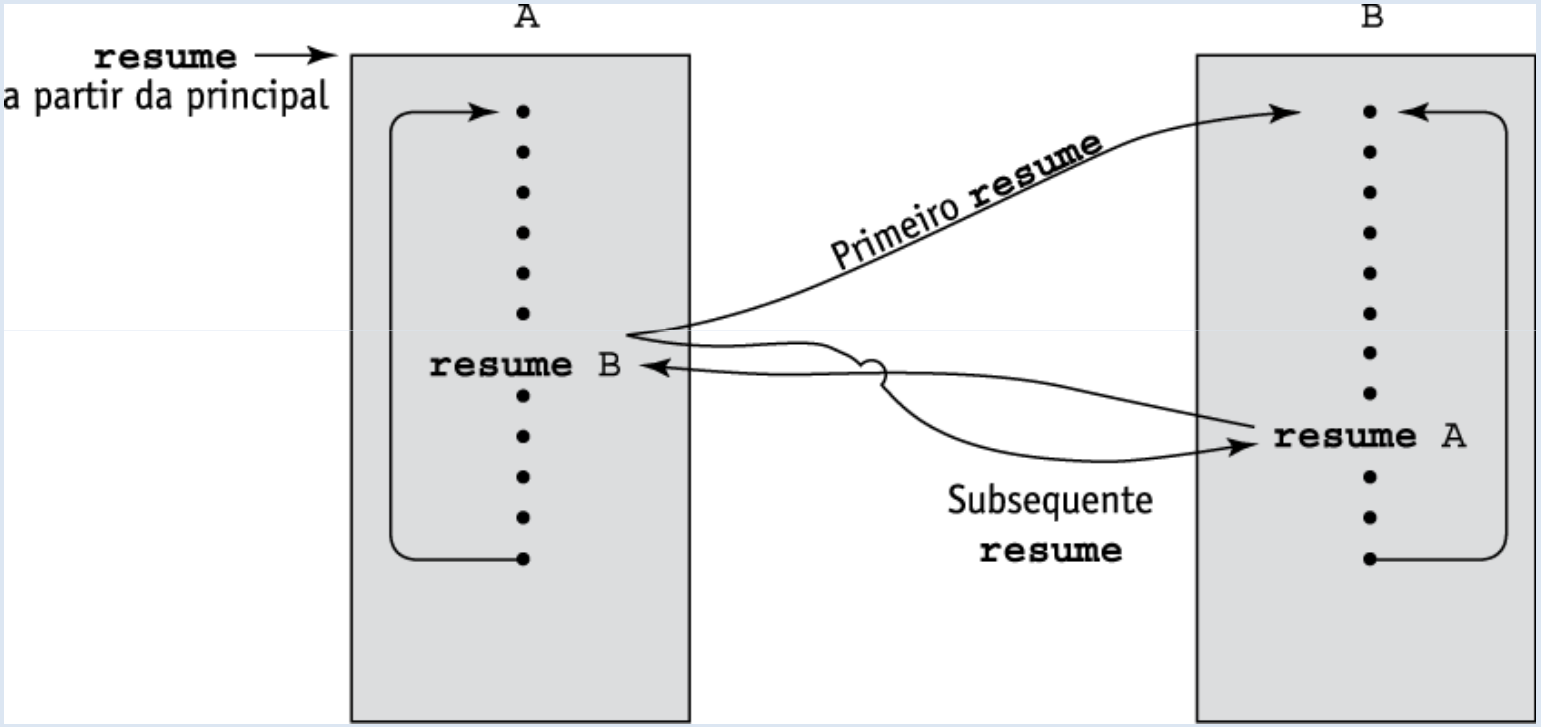


(a)

Sequências de controle de execução possíveis



Sequência de execução de corrotinas com laços



- Co-rotinas fornecem um mecanismo de execução de unidade de programas quase concorrentes
- Dentre as LPs contemporâneas, apenas Lua suporta corrotinas

unesp

Resumo

- Uma definição de subprograma descreve as ações representadas pelo subprograma
- Subprogramas podem ser funções ou procedimentos
- Variáveis locais em subprogramas podem ser dinâmicas da pilha ou estáticas
- Três modelos fundamentais de passagem de parâmetros: modo de entrada, modo de saída e modo de entrada e saída
- Algumas linguagens permitem sobrecarga de operatores
- Uma corrotina é um subprograma especial que tem múltiplas entradas

Exercícios

- Questões de revisão
  - 1, 2, 3, 5, 8, 11, 16
- Conjunto de problemas
  - 5, 7
- Exemplo de corrotina