Subprogramas (referência: Cap. 9 do livro de Sebesta)

Fundamentos dos subprogramas

- Subprogramas – são unidades de programação para construção de programas.
- Cada subprograma possui um único ponto de entrada
- O programa que chama o subprograma é suspenso durante a execução desse subprograma
- O controle sempre retorna ao “chamador” quando a execução do subprograma é finalizada

Definições básicas

- Uma definição de subprograma descreve a interface e as ações que o subprograma implementa.
- Uma chamada (invocação) de subprograma é uma requisição explicita para que o subprograma seja executado.
- O cabeçalho do subprograma é a primeira linha de sua definição, incluindo o nome, o tipo do subprograma e os parâmetros formais.
- O perfil dos parâmetros de um subprograma é a lista de parâmetros formais, incluindo o número, ordem e seus tipos.
- O protocolo de um subprograma é o seu perfil de parâmetros, e no caso de função, conjuntamente com o tipo de retorno.
- Uma declaração de subprograma (protótipo) fornece o protocolo mas não o corpo do subprograma.
- Um parâmetro formal é uma variável fictícia, definida no cabeçalho de um subprograma. O seu escopo é geralmente igual ao do subprograma.
- Um parâmetro real (ou atual) representa o valor (ou endereço) das variáveis ou constantes, utilizadas no ponto de invocação do subprograma.

Correspondência entre parâmetros Reais e Formais

- Posicional
  - A vinculação entre os parâmetros formais e os reais é realizado pela posição:
    - O primeiro parâmetro real é vinculado ao primeiro parâmetro formal e assim por diante
  - Seguro e eficiente

- Palavra-chave
  - O nome do parâmetro formal ao qual deseja-se vincular o parâmetro real é especificado junto com o parâmetro real
  - Exemplo em Visual Basic:
    - ShowMesg(Mesg:="Hello World", MyArg1:=7)
  - Vantagens:
    - Parâmetros podem aparecer em qualquer ordem
    - Desvantagem: programador tem de conhecer os nomes dos identificadores dos parâmetros formais.
- Valores por omissão – são valores definidos nos parâmetros formais, de forma a serem utilizados na falta destes valores nos parâmetros reais.
  - Valores por omissão existem em: ADA, C++, FORTRAN 90 e Visual Basic.
    - Ex. em C++:
    - // calculo do valor máximo de 1, 2 ou 3 nº positivos
    - Protótipo:
      - int maximo(int x,int y=0,int z=0);
    - Invocação:
      - cout << maximo(5) << maximo(5, 7);
      - cout << maximo(4, 8, 9);
    - Nota: Um valor por omissão é utilizado caso não seja especificado o corresponde parâmetro atual.

Tipos de subprogramas

- Existem duas categorias de subprogramas
  - Procedimentos são coleções de instruções que definem computações parametrizadas
  - Funções lembram estruturalmente os procedimentos mas são semanticamente modeladas em funções matemáticas
    - Na prática, funções produzem efeitos colaterais
    - Se não produzirem efeitos colaterais, o valor devolvido é o seu único efeito

Questões de projeto Subprogramas

- Variáveis locais são estáticas ou dinâmicas?
- Que métodos de passagens de parâmetros existem?
- Os tipos dos parâmetros formais são verificados com os tipos dos parâmetros atuais?

- Parâmetros formais podem ser do tipo subprograma?
- Pode-se ter aninhamento de definição de subprogramas?
- Se um subprograma é transmitido como parâmetro, os seus parâmetros são verificados em relação ao tipo?
- Os subprogramas podem ser sobrecarregados?
- Subprogramas podem ser genéricos (em relação ao tipo dos seus parâmetros formais)?
- É possível a compilação separada ou independente?

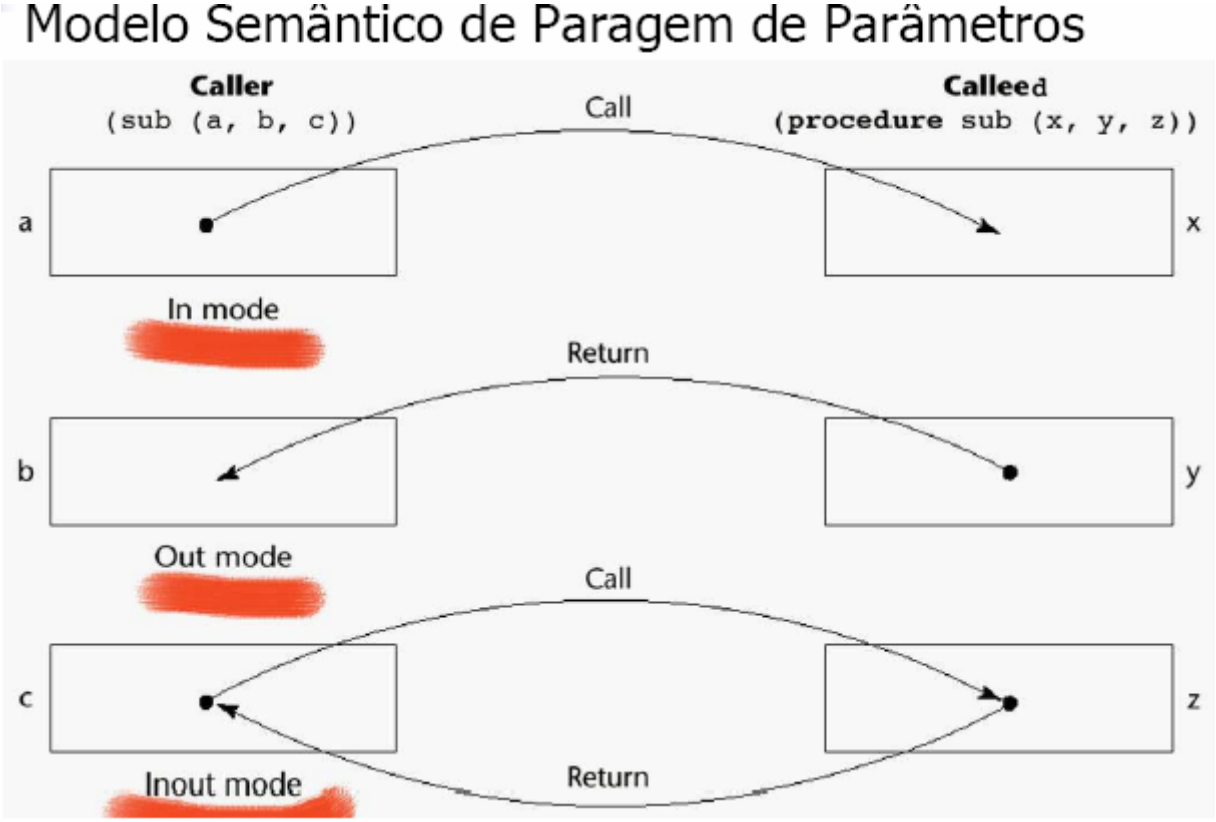
Ambientes de referência locais

- Variáveis locais podem ser dinâmicas na pilha
  - Relembrando...
    - Variáveis Dinâmicas na pilha: a associação à células de memória é efetuada em tempo de execução, na instrução de declaração, e permanece inalterável até o fim do programa.
  - Vantagens
    - Suporte à recursão
    - Locais de armazenamento podem ser compartilhados entre subprogramas
  - Desvantagens
    - Alocação/desalocação, tempo de inicialização
    - Endereçamento indireto
- Variáveis locais podem ser estáticas
  - Relembrando...
    - Variáveis Estáticas: são vinculadas à células de memória antes do início de execução do programa, e permanecem associadas às mesmas células até o programa terminar.
  - Mais eficiente
  - Sem overhead em tempo de execução
  - Não existe liberação nem alocação de memória
  - Não suportam recursão

Métodos de passagem de parâmetros

- Maneiras pelas quais se transmitem parâmetros para subprogramas
- Modelo Conceitual na Transmissão de Parâmetros:
  - Transferir fisicamente valores (copiar valores);
  - Transferir caminho de acesso (copiar endereço).

- Modelos de Implementação de Passagem de Parâmetros:
  - Passagem por valor (modo de entrada)
  - Passagem por resultado (modo de saída)
  - Passagem por valor-resultado (modo de entrada/saída)
  - Passagem por referência (modo de entrada/saída)
  - Passagem por nome (modo múltiplo)



Passagem por valor (In Mode)

- Cópia física - o parâmetro atual é avaliado e o seu valor é copiado para o parâmetro formal.
- Pode ser implementada transmitindo-se um caminho de acesso para o valor do parâmetro real (garantir proteção de escrita não é uma questão fácil)
- Desvantagens passagem por valor:
  - Desperdício de memória (ex. duplicação de um array);
  - Custo da transferência (ex. tempo de cópia do array).

Passagem por resultado (Out Mode)

- Quando um parâmetro é passado por resultado, nenhum valor é transmitido para o subprograma

- Valores do subprograma são transmitidos de volta para o invocador;
- Geralmente é utilizado Passagem por Valor;
- Deve ser assegurado que o valor do parâmetro formal não é utilizado no subprograma invocado;
- Desvantagens:
    - Tempo e espaço de memória. (Passagem por Valor);
    - Colisão de parâmetros atuais. Ex.: sub(p1, p1).
        - Qualquer um dos dois que seja atribuído por último ao seu parâmetro real correspondente irá se tornar o valor de p1

Passagem por valor-resultado (Inout Mode)

- Transferência de valores em ambas as direções;
- Também conhecido por transmissão por cópia;
- Desvantagens:
    - Mesmas que transmissão por resultado;
    - Mesmas que transmissão por valor.

Passagem por Referência (Inout Mode)

- Um caminho de acesso é transmitido
- Processo de passagem é eficiente
    - Sem cópia e sem armazenamento duplicado
- Desvantagens
    - Acesso mais lento aos parâmetros formais (comparado com passagem por valor)
    - Potenciais e não-desejados efeitos colaterais e apelidos (aliasing)

- Ex. de problemas de *aliasing*:
  - I. Colisão de parâmetros actuais (em C++):

```
void fun(int * x,int * y)
{ x = 0;
  y = 1;
}
```

Invocação:

```
int total;
fun(& total, & total);
```

Qual o resultado final da variável `total`?

Passagem por nome (Inout Mode)

- método de transmissão em modo entrada/saída. Quando parâmetros são passados por nome, o parâmetro real é, com efeito, textualmente substituído para o parâmetro formal correspondente em todas as suas ocorrências no subprograma.
  - O vinculo ocorre no momento da chamada do subprograma.
  - Mas a vinculação real a um valor ou um endereço é retardada até que o parâmetro formal seja atribuído ou referenciado.
  - Objetivo Flexibilidade
  - Desvantagem: Lentidão, dificuldade de implementação

Exemplo:

```
procedure swap(a, b: integer);
    var temp: integer;
    begin
       temp := a;
       a := b;
       b := temp;
    end;
    program main;
    var i, j: integer
       m: array[1 .. 100] of integer;
    begin
       ...
       swap(i, j);
```

```
    ...
end.
```

Se a passagem for por nome:

```
temp := j;
j := i;
i := temp;
```

Efeito: igual a parâmetro IN/OUT por referência.

Problemas:

```
swap(i, m[i]);
```

Após a substituição textual:

```
temp := i;
i := m[i];
m[i] := temp; (neste caso o "i" está modificado)
```

Métodos de passagem de parâmetros das principais linguagens

- Fortran
  - Sempre usou o modelo semântico de modo entrada/saída (inout model)
  - Antes do Fortran 77: passagem por referência
  - Fortran 77 e depois: variáveis escalares são freqüentemente passadas por valor-resultado
- C
  - Passagem por valor
  - Passagem por referência quando os parâmetros são ponteiros
- C++
  - Um tipo especial de ponteiro chamado tipo de referência – passagem por referência
- Java
  - Todos os parâmetros são passados por valor
  - Parâmetros de objeto são passados por referência
- Ada
  - Três modos semânticos: in, out, in out; in é o modo padrão
  - Parâmetros formais
    - declarados out podem ser alterados mas não referenciados
    - declarados in podem ser referenciados mas não alterados
    - in out podem ser referenciados e alterados

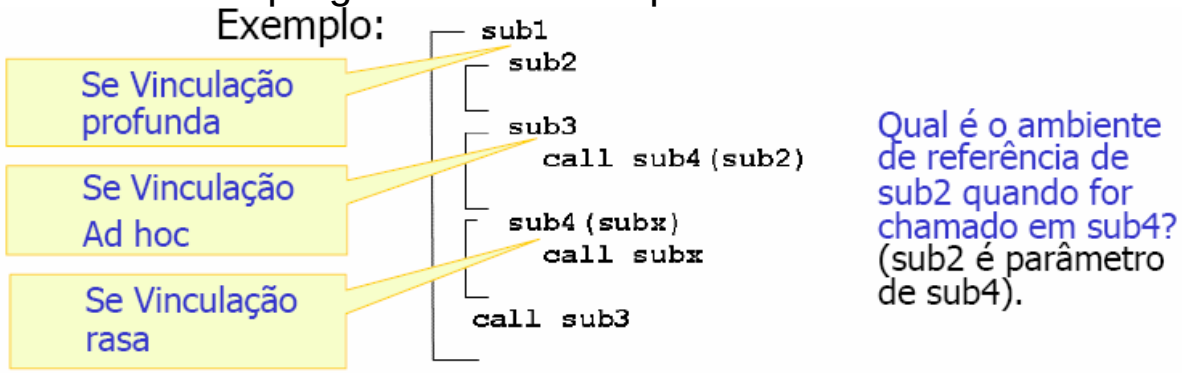
- C#
  - Método padrão: passagem por valor
  - Passagem por referência é especificado
- PHP: muito similar a C#

Parâmetros que são nomes de subprogramas

- Em alguns casos é mais conveniente manipular nomes de subprogramas como parâmetros
  - Um subprograma para avaliar alguma função matemática
  - Um subprograma para avaliar a área sobre a curva de uma função
- C e C++
  - funções não podem ser passadas como parâmetros, mas ponteiros para funções podem ser passados
- Versões antigas de Pascal e de Ada não permitem subprogramas como parâmetros

Parâmetros que são nomes de subprogramas: ambientes de referenciamento

- Qual é ambiente correto de referenciamento dos subprogramas que são enviados como parâmetro?
- Possibilidades:
  - Vinculação rasa
    - Ambiente do subprograma onde o subprograma atua como parâmetro atual;
  - Vinculação profunda
    - O ambiente da definição do subprograma passado
  - Vinculação ad hoc
    - O ambiente da instrução de chamada que passou o subprograma como um parâmetro real



```
procedure SUB1;
  var x : integer
  procedure SUB2;
    begin
    write('x = ', x)
    end {SUB2}
  procedure SUB3;
    var x : integer
    begin
    x := 3;
    SUB4(SUB2)
    end {SUB3}
  procedure SUB4(SUBX);
    var x : integer
    begin
    x := 4;
    SUBX
    end {SUB4}
  begin {SUB1}
    x := 1;
    SUB3
  end {SUB1}
```

- Considere a execução de SUB2 quando chamada de SUB4
  - Vinculação rasa
    - O ambiente de referenciamento dessa execução é SUB4
    - O resultado é x = 4
  - Vinculação profunda
    - O ambiente de referenciamento dessa execução é SUB1
    - O resultado é x = 1
  - Vinculação ad hoc
    - Vinculação é a local x em SUB3
    - O resultado é x = 3

Subprogramas Sobrecarregados

- Sobrecarga de subprograma – é um subprograma que possui o mesmo nome de outro subprograma num mesmo ambiente de referência (escopo), contudo têm de ter protocolos diferentes.
- C++ e Ada possuem sobrecarga de subprogramas predefinidos e programadores podem implementar novos subprogramas sobrecarregados.
- Ex. em C++:

```
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));}
```

Subprogramas genéricos

- Um subprograma genérico ou polimórfico é aquele que recebe parâmetros de tipos diferentes em diferentes ativações.
- Sobrecarga de subprograma é um mecanismo de polimorfismo ad hoc(para um fim especifico).

    - Por exemplo, não é preciso criar diferentes subprogramas que implementem o mesmo algoritmo de ordenação em diferentes tipos de dados
- Polimorfismo paramétrico – subprogramas utilizados em expressões, os quais recebem parâmetros de diferentes tipos especificados pelos operandos da expressão.
    - Ex. ADA e C++ utilizam polimorfismo paramétrico.

Compilação Separada e Independente

- Compilação Independente – Unidades de compilação que podem ser compiladas separadamente, sem qualquer informação sobre as outras unidades (interdependências). A coerência de tipos das interfaces não é verificada entre as unidades compiladas.
- Compilação Separada – Unidades de compilação que podem ser compiladas separadamente, utilizando informações de interface para verificar a interdependência entre as partes.
- Exemplos de linguagens:
    - C, FORTRAN 77: Compilação independente;
    - FORTRAN 90, Ada, Modula-2, C++: Compilação separada;
    - Pascal: Programa tem que ser todo compilado de uma única vez.

Questões de projeto referentes a funções

- Efeitos colaterais são permitidos?
    - Parâmetros devem ser sempre in-mode para reduzir efeitos colaterais (como em Ada)
- Quais tipos de valores podem ser retornados?
    - A maioria das linguagens imperativas restringem o tipo de retorno
    - C permite qualquer tipo, exceto arrays e funções
    - C++ é similar a C, mas permite tipos definidos pelo usuário
    - Ada permite qualquer tipo
    - Java e C# não possuem funções, mas métodos permitem qualquer tipo

Acesso a Ambientes não Locais

- Variáveis não locais de um subprograma são aquelas que são visíveis (podem ser utilizadas) mas não estão declaradas no subprograma.

- Variáveis Globais são aquelas que podem ser visíveis em todos os subprogramas.
- Ex. em FORTRAN (Blocos COMMON):
  - Antes do FORTRAN 90, os blocos COMMON eram a única forma de aceder a variáveis não locais.
  - Estes blocos podem ser utilizados para partilhar dados ou partilhar memória.

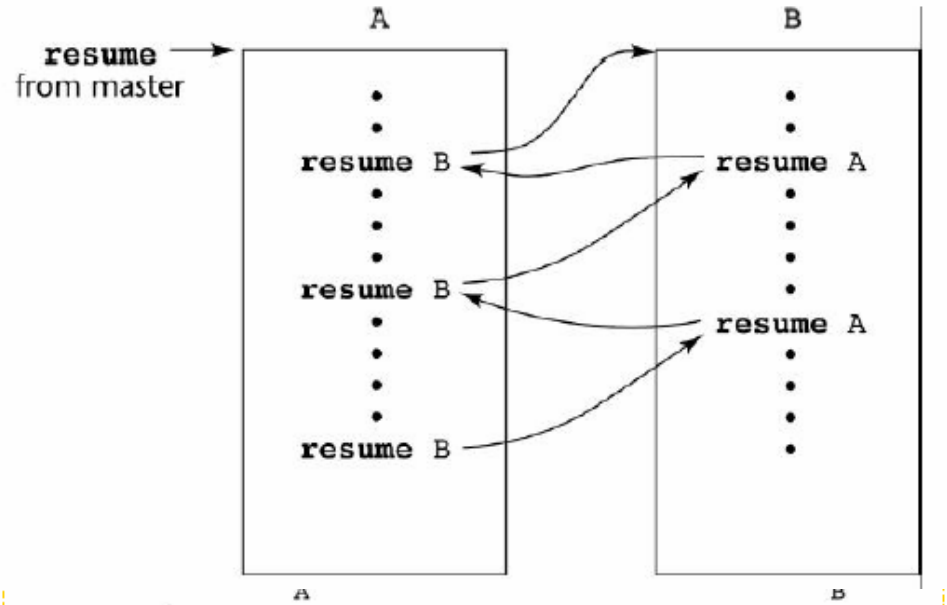
Sobrecarga de operadores

- Operadores sobrecarregados – operadores que têm múltiplos significados, isto é, estão definidos para diversos tipos de operandos.
  - Ex.: 25 + 40 (int + int) e 25.0 + 40.0 (real + real)
- Os operados sobrecarregados devem ter protocolos diferentes.
- Quase todas as linguagem de programação possuem operadores sobrecarregados.
- Programadores podem criar novos significados para operadores em C++ e ADA (esta facilidade não existe em Java).
  - Sobrecarga definida pelo programador é bom ou não?
    - A sobrecarga é boa se a legibilidade não for afetada.

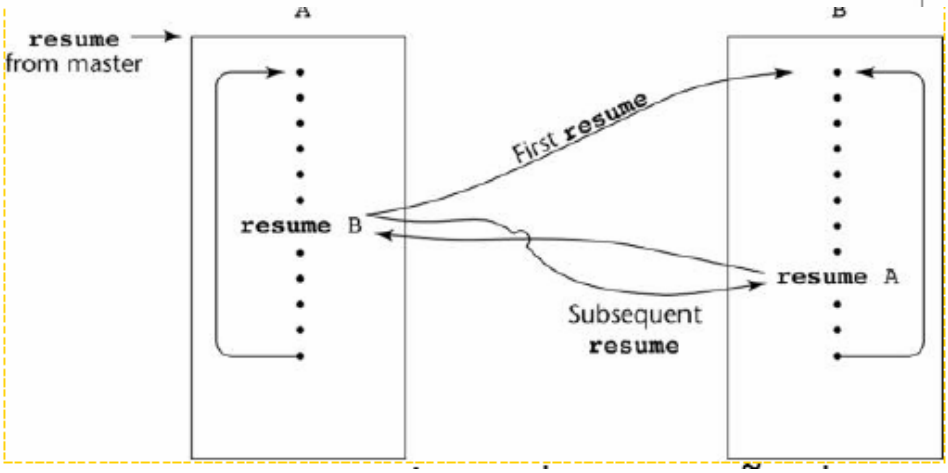
Co-rotinas

- Uma co-rotina é um subprograma que possui múltiplas entradas.
- É um tipo especial de subprograma. Em vez da relação mestre-escravo entre o subprograma chamador e o chamado, as co-rotinas estão em uma base mais igual.
  - Enquanto os procedimentos são executados do começo ao fim, uma co-rotina executa a partir do ponto em que foi suspensa até a próxima instrução que suspenda sua execução
- Quanto uma co-rotina é ativada, através de outro subprograma, esta é executa parcialmente sendo suspensa quando retorna o controle, podendo ser reativada de onde parou (se invocada novamente).
- A invocação de uma co-rotina é designado de retomada (resume).
- A primeira retomada de uma co-rotina é para seu início mas, invocações subseqüentes iniciam imediatamente após o último comando executado.
- Tipicamente, co-rotinas repetidamente retomam a execução entre si, possivelmente em laço infinito.

Execução de duas co-rotinas sem laços.



Execução de duas co-rotinas com laços.



- Co-rotinas fornecem um mecanismo de execução de unidade de programas quase concorrentes.
- A execução de co-rotinas é intercalada e não sobreposta.
- Co-rotinas existem em Simula 67 (1967), Bliss (1971), interLisp (1975) e Modula-2 (1985).