Problema (tolo)

Dadas três matrizes calcule o produto de todas as combinações de três matrizes duas a duas (sem usar as operações pré-definidas do FORTRAN).

Especificação

⟶ n,a,b,c   matrizes quadradas de ordem n.

⟵ a×b, a×c, b×c

Algoritmo

```
alg
  ler(n)
  ler(a,b,c)
  calcular(a×b)
  calcular(a×c)
  calcular(b×c)
  escrever(a×b,a×c,b×c)
fimalg
```

¡ É necessário repetir o cálculo três vezes mudando só as matrizes intervenientes!

¡ Trabalho repetitivo, com o consequente aumento da possibilidade de ocorrência de erros de escrita!

Solução

Efectuar o cálculo do produto de duas matrizes num sub-algoritmo que possa receber parâmetros para dessa forma se adaptar a diferentes situações.

Temos então:

```
sub-alg prod_matrizes(inteiro:n,tabelas:x,y,z)
  para i <- 1 até n
    para j <- 1 até n
      z(i,j) <- 0
      para k <- 1 até n
        z(i,j) <- z(i,j) + x(i,k)*y(k,j)
      fimpara
    fimpara
  fimpara
fimsub-alg
```

sub-programa escrito em termos genérico, escrito em função dos seus parâmetros formais.

```
alg
  ...
  prod_matrizes(n,a,b,ab)
  prod_matrizes(n,a,c,ac)
  prod_matrizes(n,b,c,bc)
  ...
fimalg
```

programa contendo a chamada dos sub-programas através de argumentos concretos (parâmetros reais).

¿ Sub-algoritmos, quando ?

- Quando é necessário explicitar uma determinada tarefa repetidas vezes (eventualmente com mudanças nos intervenientes).
- Quando o problema a resolver é complexo, de forma a melhor estruturar a solução encontrada.

¿ Sub-programas, como ?

- escrever sub-programas (como se de um programa se tratasse) em que se especifica o que entra (dados) e o que sai (resultados), isto é, especificando quais são os *parâmetros formais*.
- escrever um programa (principal) e "chamando" os sub-programas (pelo seu nome) fornecendo a este os dados e esperando dele os resultados, isto é, especificando quais são os *parâmetros reais*.

¿ Em FORTRAN, como ?

O FORTRAN permite a escrita de unidades independentes umas das outras (em termos sintácticos) que depois podem ser combinadas através da escrita de um programa.

- sub-programas – `Subroutine`
- funções – `Function`

Procedimentos (`Subroutine`) em FORTRAN

Declaração

`Subroutine` < *identificador*> (< *lista_de_parâmetros*>)
< *secção_de_declarações*>
< *secção_de_instruções*>
[`Return`]
`End Subroutine`

Os parâmetros são ditos parâmetros formais dado que o sub-programa é escrito em função deles enquanto o programa principal tem o seu conjunto próprio de identificadores.

Cada procedimento, assim como o programa principal são entidades sintacticamente independentes que:

- podem ser escritas em ficheiros separados, permitindo assim a compilação (parcial) separada:

```
> f90 -c subprograma.f90
> f90 -c programa.f90
> f90 programa.o subprograma.o
```

- podem ser escritos num mesmo ficheiro, nesse caso a instrução de compilação é a usual:

```
> f90 programa.f90
```

As instruções com a opção de compilação `-c`, têm o efeito de gerar somente o código binário correspondente ao `sub-programa.f90` e ao `programa.f90` (criando os ficheiros `sub-programa.o` e `programa.o`), depois é necessário "juntar" esses "pedaços" para criar o programa final.

Invocação (ou Chamada)

`Call` < *identificador*> (< *lista_de_parâmetros*>)

Os parâmetros são ditos parâmetros reais dado que o programa que faz a chamada é escrito em função deles e são eles que contêm a informação a manipular.

Notas:

- A comunicação entre o programa e os sub-programas é feita através dos parâmetros formais/reais.
- Os sub-programas podem ser invocados (mais do que uma vez) pelo programa principal ou por outros sub-programas, só os sub-programas recursivos é que se podem chamar a si próprios.
- Quando um sub-programa é invocado a execução do programa é interrompida, passando o controlo para o sub-programa, quando este acaba o controlo retoma para o programa no ponto após a invocação do sub-programa.

Um exemplo: dados dois inteiros efectuar a sua soma.

Especificação:

$\longrightarrow$ i,j

$\longleftarrow$ res=i+j

Programa:

```
Program principal
  Implicit None
  Integer :: i,j,res

  Write (*,*) "Introduza dois Inteiros"
  Read (*,*) i,j
  Call somar(i,j,res)      ! invocar o sub-programa somar
  Write (*,*) "O resultado é: ",res
End Program

Subroutine somar(arg1,arg2,res)   ! par. formais
  Implicit None
  Integer,Intent(IN) :: arg1,arg2 ! par. entrada
  Integer,Intent(OUT) :: res      ! par. saída

  res=arg1+arg2
End Subroutine
```

Notas:

- Na altura da invocação do sub-programa é estabelecida a correspondência entre parâmetros reais e formais, (todos do tipo `Integer`).

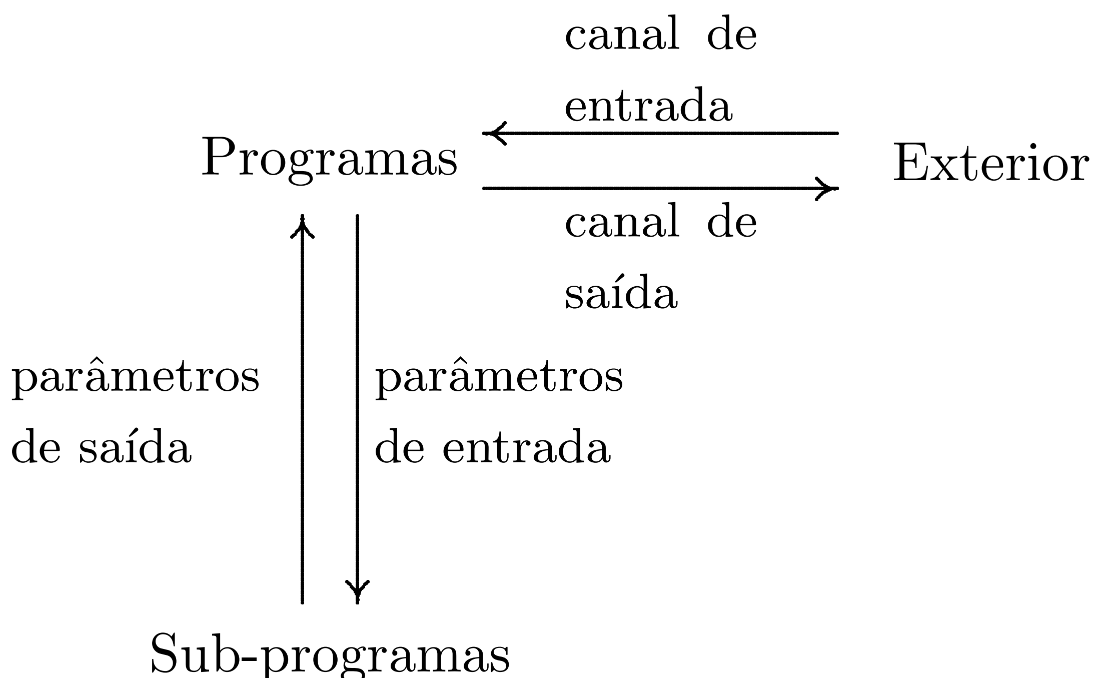
```
    Call somar(i,j,res)
            ↓   ↘   ↖
subroutine somar(arg1,arg2,res)
```

- Ligação entre os parâmetros:
  - `Intent(IN)` - variáveis (só) de entrada.
  - `Intent(OUT)` - variáveis (só) de saída.
  - `Intent(INOUT)` - variáveis de entrada e saída.

  A ligação é feita sempre por *referência*, no entanto o compilador faz a verificação do tipo da ligação tendo em conta a informação do sub-programa (declaração dos parâmetros formais).

- Compilação – `f90 -o somadois somadois.f90`.
- Execução – `somadois`.

Os sub-programas devem ser pensados como blocos independentes que comunicam entre si através dos parâmetros formais/reais. A comunicação com o exterior (leituras e escritas) devem ser feitas somente pelo programa principal, excepção a esta regra são os sub-programa especializados na leitura e/ou escrita.



Metodologia de Programação Estruturada e Descendente

- Decompor o programa global em sub-programas específicos.
- Resolver cada um dos sub-programas.
- Combinar as soluções parciais para formar a solução global.

O processo de decomposição repete-se para os sub-programas até se atingir sub-programas tão simples que a sua solução se possa expressar em poucas linhas de uma dada linguagem de programação. Atingido esse ponto passa-se à fase de resolução e combinação das diferentes soluções.

Vejamos então como resolver o problema já referido da multiplicação de três matrizes entre si.

Problema

Dadas três matrizes quadradas de ordem n, a, b, e c calcular ab, bc, e ac.

Especificação

$\longrightarrow$ n,a,b,c matrizes quadradas de ordem n.

$\longleftarrow$ a×b, a×c, b×c

Algoritmo

```
alg
  ler_matriz(n,a)
  ler_matriz(n,b)                (1)
  ler_matriz(n,c)
  calcular(n,a,b,a×b)
  calcular(n,a,c,a×c)             (2)
  calcular(n,b,c,b×c)
  escrever(n,a×b)
  escrever(n,a×c)                 (3)
  escrever(n,b×c)
fimalg
```

Sub-problema (1) – leitura de matrizes

$\longrightarrow$

$\longleftarrow$ n, a (com $a \in M(n,n)$)

```
sub-alg ler_matriz(inteiro: n, tabela: a)
  ler(a(i,j), (1≤i≤n, 1≤j≤n))
fimsub-alg
```

Sub-problema (3) – escrita de matrizes

$\longrightarrow$ n, a (com $a \in M(n,n)$)

$\longleftarrow$

```
sub-alg escrever_matriz(inteiro: n, tabela: a)
  escrever(a(i,j), (1≤i≤n, 1≤j≤n))
fimsub-alg
```

Sub-problema (2) – Produto de matrizes

$\longrightarrow$ n, a, b (com $a, b \in M(n,n)$)

$\longleftarrow$ n, ab (com $ab \in M(n,n)$)

```
sub-alg  prod_matrizes(inteiro:n,tabela:x,y,z)
  para i <- 1 até n
    para j <- 1 até n
      z(i,j) <- 0
      para k <- 1 até n
        z(i,j) <- z(i,j) + x(i,k)*y(k,j)
      fimpara
    fimpara
  fimpara
fimsub-alg
```

O Sub-programa (2) em FORTRAN, `mult_mat.f90`

```
Subroutine mult_mat(n,x,y,xy)

  Implicit None

  Integer, Intent(In) :: n
  Integer, Intent(In),Dimension(n,n) :: x,y
  Integer, Intent(Out),Dimension(n,n) :: xy
  Integer :: i,j,k

  Do i=1,n
     Do j=1,n
        xy(i,j)=0
        Do k=1,n
           xy(i,j)=xy(i,j)+x(i,k)*y(k,j)
        End do
     End do
  End do
End Subroutine mult_mat
```

Notas:

- Parâmetros de entrada, `Intent(IN)`, não podem sofrer modificações;
- Parâmetros de saída, `Intent(OUT)`, só servem para transportar valores para o programa de chamada;
- A declaração das matrizes `Dimension(n,n)` estabelece uma dimensão máxima para as matrizes no sub-programa de $n \times n$, no entanto como a declaração real (aquela que faz a reserva de memória) é a do programa principal, o valor de n nunca pode ser superior ao valor declarado no programa principal;
- Variáveis locais `i,j,k`, a sua existência está limitada ao tempo de execução do sub-programa.

O Sub-programa (1) em FORTRAN, `ler_mat.f90`

```
Subroutine ler_mat(n,x)

 Implicit None

 Integer, Intent(IN) :: n
 Integer, Intent(OUT),Dimension(n,n) :: x

 Integer :: i,j

 Write(*,*) "Introduza (por linhas) os elementos da matriz"
 Do i=1,n
    Read (*,*) (x(i,j),j=1,n)
 End Do
End Subroutine ler_mat
```

O Sub-programa (3) em FORTRAN, `escrever_mat.f90`

```
Subroutine escrever_mat(n,x)

 Implicit None

 Integer, Intent(IN) :: n
 Integer, Intent(IN),Dimension(n,n) :: x

 Integer :: i,j

 Do i=1,n
    Write (*,*) (x(i,j),j=1,n)
 End Do
End Subroutine escrever_mat
```

O Programa principal, `mult_3_matrizes.f90`

```
Program mult_matrizes

 Implicit None

 Integer :: n
 Integer, Allocatable, Dimension(:,:) :: a,b,c,ab,bc,ac
 Integer :: falta_espaco=1

 Do While (falta_espaco==1)
   Write (*,*) "Qual e a dimensao das matrizes?"
   Read (*,*) n
   Allocate(a(n,n),b(n,n),c(n,n),stat=falta_espaco)
   Allocate(ab(n,n),bc(n,n),ac(n,n),stat=falta_espaco)
 End do
 Write (*,*) "Ler a matriz a (por linhas)"
 Call ler_mat(n,a)
 Write (*,*) "Ler a matriz b (por linhas)"
 Call ler_mat(n,b)
 Write (*,*) "Ler a matriz c (por linhas)"
 Call ler_mat(n,c)
 Write (*,*) "Calcular os Produtos"
 Call mult_mat(n,a,b,ab)
 Call mult_mat(n,b,c,bc)
 Call mult_mat(n,a,c,ac)
 Write (*,*) "Escrever a matriz ab"
 Call escrever_mat(n,ab)
 Write (*,*) "Escrever a matriz bc"
 Call escrever_mat(n,bc)
 Write (*,*) "Escrever a matriz ac"
 Call escrever_mat(n,ac)
End Program mult_matrizes
```

Notas:

- Comunicação Programas↔Sub-programas feita exclusivamente através dos parâmetros de entrada/saída;
- Comunicação Exterior↔Programa feita pelo programa principal e por sub-programas especializados na leitura e/ou escrita (e só esses).
- Clara distinção entre tarefas.
- Declaração dinâmica das matrizes.
- Compilação separada:
  - `f90 -c ler_mat.f90`
  - `f90 -c escrever_mat.f90`
  - `f90 -c mult_mat.f90`
  - `f90 -c mult_3_mat.f90`
  - `f90 -o mult_3_mat mult_3_mat.o ler_mat.o escrever_mat.o mult_mat.o`

Funções (implementação de funções)

Funções (em FORTRAN) são sub-programas que devolvem um só valor, o qual está associado ao nome do sub-programa. Uma função é usada (chamada) numa expressão.

Declaração

```
Function <identificador> (<lista_de_parâmetros>)
    <secção_de_declarações>
    (...)
    <nome> = <expressão>
    (...)
End Subroutine
```

Comunicação Programa ↔ Sub-programas

- <*lista_de_parâmetros*> - valores **só** de entrada
- <*nome*> - um valor (saída).

Utilização/Chamada

Numa expressão através do nome da função e da lista de parâmetros reais.

Notas:

- A correspondência entre parâmetros reais e parâmetros formais segue as mesmas regras que no caso dos procedimentos.
- A lista de parâmetros pode ser vazia.
- O tipo da função (tipo de saída) tem de ser declarado tanto no programa de chamada assim como na declaração da função (assume-se `Implicit none`).

  <tipo> `Function` *<identificador>* `...`

  ou

  `Function` *<identificador>* `...`
  <tipo> `::` *<identificador>*

- No corpo da função tem de existir pelo menos uma instrução na qual seja definido o valor de saída da função.
- O nome da função é um identificador que não pode ser usado do lado direito de uma instrução de atribuição (a menos de chamadas recursivas).

Exemplo: função factorial

Definição recursiva

$$\begin{aligned} n! &= n(n-1)! \\ 0! &= 1 \end{aligned}$$

Definição não recursiva

$$n! = n(n-1)(n-2)\ldots 1 = \prod_{i=1}^{n} i$$

O programa (não recursivo) em FORTRAN

```
Integer Function factorial(n)
  Implicit None
  Integer, Intent(In) :: n
  Integer :: i,aux
  aux = 1
  Do i=2,n
     aux=aux*i
  End Do
  factorial=aux
End Function factorial

Program testa_factorial
  Implicit None
  Integer :: n,factorial
  Write (*,*) "Qual o valor de n?"
  Read (*,*) n
  Write (*,100) n,"!=",factorial(n)
100 Format(1x,I2,A2,I10)
End Program testa_factorial
```

Efeitos co-laterais

Dado que a comunicação entre programas e sub-programas ("procedimentos e funções") é feito por referência podemos usar nas funções parâmetros de saída. No entanto e devido à forma como as funções são usadas (chamada de dentro de uma expressão) tal pode ser desastroso, em qualquer dos casos é sempre considerado um mau estilo de programação.

¿Porquê desastroso?

$$y + f(x) \stackrel{?}{=} f(x) + y$$

Em matemática sim! Propriedade comutativa da adição.

Em FORTRAN talvez...

```
real Function f(x,y)

  Implicit None
  real, Intent(IN) :: x
  real, Intent(OUT) :: y      !!! PERIGO!!!

  y=3
  f=x+y
End Function f

Program efeitos_co_laterais

  implicit None
  real :: x,y,f

  Write (*,*) "Qual o valor de x?"
  Read (*,*) x
  y=-2
  Write (*,*) y+f(x,y)
  y=-2
  Write (*,*) f(x,y)+y
End Program efeitos_co_laterais
```

Exemplo de execução:

```
    Qual o valor de x?
2
    3.000000         8.000000
```

A mesma operação com dois resultados diferentes!?

A propriedade comutativa da adição não é válida!?

¡Temporização!

- `y+f(x,y)`, o valor de y que participa na adição ainda não foi alterado.
- `f(x,y)+y`, o valor de y que participa na adição já foi alterado por efeito da chamada da função f.

Conclusão:

¡Não usar (nunca) parâmetros de saída nas `functions`!

Recursão

O FORTRAN 90/95 permite a implementação de funções recursivas, isto é, funções em que a própria função entra na definição do corpo da função.

Por exemplo a função factorial

$$n! = \begin{cases} 0! & = & 1 & \text{caso de base} \\ n! & = & n(n-1)! & \text{caso recursivo} \end{cases}$$

É necessário especificar:

- um **caso recursivo** em que a definição da função (ordem n) é feita em termos da própria função (ordem $n-1$) mas de forma a que se "caminhe" para o caso de base.
- um **caso de base** no qual se define um valor concreto para a função num determinado ponto (ordem 0).

Em FORTRAN 90/95 isto é possível dado ser possível chamar uma `function` de dentro da própria `function`.

Exemplo: a implementação recursiva da função factorial.

É Necessário adicionar o qualificativo `Recursive` ao cabeçalho da função.

```
Recursive Integer Function factorial(n)

  Implicit None

  Integer, Intent(IN) :: n

  If (n==0) Then
     factorial=1                     ! caso de base
  Else
     factorial=factorial(n-1)*n      ! caso recursivo
  End If
End Function factorial

Program testa_factorial

  Implicit None

  Integer :: n,factorial

  Write (*,*) "Qual o valor de n?"
  Read (*,*) n
  Write (*,100) n,"!=",factorial(n)
100 Format(1x,I2,A2,I10)
End Program testa_factorial
```