



Monitoria 28 set

Diferenças entre `function` e `subroutine`

Quando um bloco de código tem uma função específica e será constantemente reutilizado, surge a oportunidade de utilizar uma função ou subrotina.

A Fortran `function` is similar to a mathematical function, which takes one or many parameters as inputs and returns a single output value.

A Fortran `subroutine` is a block of code that performs some operation on the input variables, and as a result of calling the subroutine, the input variables are modified.

Fonte.



Passagem de valores por referência ou por cópia



Uma **função** é mais simples de utilizar, aceitando argumentos e retornando um valor

Pode ser utilizada dentro de expressões como o `write`, `if()`



Uma **subrotina** pode ser mais difícil de utilizar, por não retornar um valor específico é preciso controlar as alterações feitas nos valores passados. **Não** pode ser usada dentro de expressões, apenas com o *statement* apropriado `call`

```
function func(i) result(j)
  integer, intent (in) :: i ! input
  integer               :: j ! output

  j = i**2 + i**3
end function

program main
  implicit none
  integer :: i
  integer :: func !retorno da função declarado no código

  i = 3
  print *, "sum of the square and cube of", i, "is", func(i)
end program
```

O argumento de retorno é implicitamente `intent (out)`

O tipo do retorno da função deve estar declarado no código!

```
subroutine square_cube(i, isquare, icube)
  integer, intent (in) :: i          ! input
  integer, intent (out) :: isquare, icube ! output

  isquare = i**2
  icube   = i**3
end subroutine

program main
  implicit none
  external square_cube ! external subroutine
  integer :: isq, icub

  call square_cube(4, isq, icub)
  print *, "i,i^2,i^3=", 4, isq, icub
end program
```

Exemplo de recursão:

```
recursive function fact(i) result(j)
  integer, intent (in) :: i
  integer :: j
  if (i==1) then
    j = 1
  else
    j = i * fact(i - 1)
  end if
end function fact
```

Exercicio 5

Leitura de matrizes, do [repositório do Heitor](#):

```
program power
  implicit none
  integer, parameter :: dp = kind(0.d0)

  integer :: n      ! matrix size
  integer :: i, j

  real (kind = dp), allocatable, dimension(:, :) :: A, B   ! matrix

  ! read matrix size, allocate matrix, read matrix
  read(*, *) n
  allocate(A(n,n), B(n,n))

  read(*, *) A
  do i = 1, n
    print *, (A(i, j), j = 1, n)
  end do

  do i = 1, n
    read(*, *) (B(i, j), j = 1, n)
  end do
  do i = 1, n
    print *, (B(i, j), j = 1, n)
  end do


  deallocate(A, B)
end program power
```

Resulta em:

```
$ ./a.out
3
1 2 3
4 5 6
7 8 9
    1.0000000000000000      4.0000000000000000      7.0000000000000000
    2.0000000000000000      5.0000000000000000      8.0000000000000000
    3.0000000000000000      6.0000000000000000      9.0000000000000000
1 2 3
4 5 6
7 8 9
    1.0000000000000000      2.0000000000000000      3.0000000000000000
    4.0000000000000000      5.0000000000000000      6.0000000000000000
    7.0000000000000000      8.0000000000000000      9.0000000000000000
```

Veja o exercício 5 do projeto 1 da turma de 2018

Descrição do projeto:

 https://github.com/heitorPB/fiscomp-2018/blob/master/projeto1/p1_fiscomp18.pdf

 https://github.com/heitorPB/fiscomp-2018/blob/master/projeto1/proj1_fiscomp18.pdf

Código de ajuda:

fiscomp-2018/1e.f90 at master · heitorPB/fiscomp-2018

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/heitorPB/fiscomp-2018/blob/master/projet01/1e.f90>

heitorPB/fiscomp-2018

Material do Curso de Introdução a Física Computacional de 2018 - IFSC - USP

1 Contributor

0 Issues

1 Star

0 Forks

