

Программирование

Лекция 8 ПРОЦЕДУРЫ

Петров Александр Владимирович
Фёдоров Станислав Алексеевич

**(по материалам Веренинова Игоря
Андреевича с изменениями и
дополнениями на Fortran 08 и UML)**

Сентябрь 2013

Назначение процедур

Разбиение исходной задачи на подзадачи

Устранение дублирования кода

Повторное использование написанного кода

Библиотеки готовых подпрограмм и функций

Раздельное написание программ

Виды процедур

```
module ИМЯ
```

```
...
```

```
contains
```

```
    модульные процедуры
```

```
end module
```

Модули

```
program ИМЯ
```

```
...
```

```
contains
```

```
    внутренние процедуры
```

```
end
```

Головная
программа

```
subroutine или function
```

```
...
```

```
contains
```

```
    внутренние процедуры
```

```
end
```

Внешние
процедуры

Модульные процедуры

Описываются в модулях
после оператора **contains**.

Обладают явным интерфейсом.

Имеют доступ ко всем объектам модуля
(типы данных, переменные, внутренние процедуры)
при несовпадении имён

Могут содержать
другие внутренние процедуры.

Внутренние процедуры

Описываются в головной программе
после оператора **contains**.

Обладают явным интерфейсом.

Имеют доступ ко всем объектам
головной программы
(типы данных, переменные, внутренние процедуры)
при несовпадении имён

Не могут содержать
другие внутренние процедуры.

Внешние процедуры

Описываются отдельно от головной программы или в других файлах.

Обладают неявным интерфейсом.

Обмен данными с головной программы происходит посредством формальных параметров.

Могут содержать другие внутренние процедуры.

Функции

тип **function** имя_функции (формальные
параметры)

типы формальных параметров

типы внутренних переменных

исполняемые операторы

имя_функции = вычисленное значение

end function имя_функции

Результатом может выступать также переменная описанная в операторе **result** после объявления функции.

Функции

Пример.

Функция представлена тригонометрическим рядом

$$f(x, N) = \sum_{k=1}^N \frac{\sin kx}{k}$$

Тип функции – вещественный;

формальные параметры –

x -вещественный, **k** -целый;

внутренние переменные – сумма (вещественный),
индекс суммирования (целый).

Функции

```
! *****
!  
!               Функция f(x)  
! *****
real function f(x,N)
!----- формальные параметры -----
    real x
    integer N
!----- внутренние переменные -----
    real sum
    integer k
!  
    sum = 0.0d0
    do k = 1,N
        sum = sum + sin(k*x)/k
    end do
    f = sum ! результат присвоили имени функции
end function f
```

Вызов функции

Вызов созданной функции
аналогичен вызову стандартной функции

```
program FX
  real res
  res = f(0.5, 100) ! вызов функции
  write(*,*) res
end
```

фактические
параметры
0.5d0, 100

```
! *****
!                                     функция f(x)
! *****
real function f(x,N)
  ...
end function f
```

формальные
параметры
x, N

Вызов функции

Вызов функции без параметров

```
program func2
```

```
  write(*,*) pi()
```

```
end
```

```
! ****
```

```
!           функция возвращает число Пи
```

```
! ****
```

```
real function pi()
```

```
  pi = 2.0*acos(0.0)
```

```
end function pi
```

Вызов функции

При вызове должны соответствовать:

- 1) тип функции и тип переменной, которой присваивается результат функции
- 2) тип формальных и фактических параметров

Подпрограммы

subroutine имя_подпрограммы (формальные
параметры)

типы формальных параметров

типы внутренних переменных

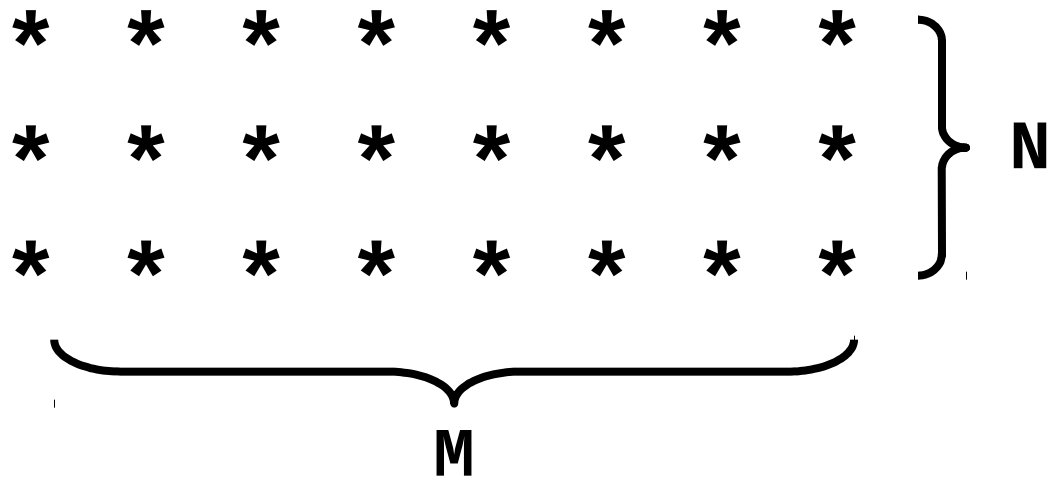
операторы описания

исполняемые операторы

end subroutine имя_подпрограммы

Подпрограммы

Подпрограмма печати
прямоугольной таблицы символов



формальные параметры –
M, **N** (целый), **CH** (символьный);

внутренние переменные –
индексы таблицы (целый).

Подпрограммы

```
! *****
!  
!           Подпрограмма печати таблицы символов  
! *****  
subroutine table(M,N,CH)  
! ----- формальные параметры -----  
    integer M,N  
    character CH  
! ----- внутренние переменные -----  
    integer i,j  
!  
    do k = 1,M  
        do j = 1,N  
            write(*,"(A,\)") CH  
        end do  
        write(*,*)  
    end do  
end subroutine table
```

Вызов подпрограммы

Оператор **call** вызывает подпрограмму

```
program sub
```

```
  call table(4,7, '*')
```

```
end
```

```
! *****
```

```
!           Подпрограмма печати таблицы символов
```

```
! *****
```

```
subroutine table(M,N,CH)
```

```
! ----- формальные параметры -----
```

```
...
```

```
end subroutine table
```

соответствие
фактических
и формальных
параметров

Внутренние переменные

Внутренние переменные доступны и используются внутри процедур.

Внутренние переменные получившие инициализацию являются статическими, т.е. память выделяется на этапе компиляции (неявно обладают атрибутом **static**).

Атрибут **automatic** устанавливает переменные автоматическими, т.е. память выделяется во время работы программы.

Внутренние переменные

```
program static_var
```

```
  call sub() ! M = 1  
  call sub() ! M = 2  
  call sub() ! M = 3
```

```
end
```

```
subroutine sub()  
  integer :: M = 0  
  M = M + 1  
  write(*,*) M  
end subroutine sub
```

```
program static_var
```

```
  call sub() ! M = 1  
  call sub() ! M = 1  
  call sub() ! M = 1
```

```
end
```

```
subroutine sub()  
  integer , automatic :: M  
  M = 0  
  M = M + 1  
  write(*,*) M  
end subroutine sub
```

Передача параметров

Формальные параметры

- 1) входные, `intent(in)`
- 2) выходные, `intent(out)`
- 3) входные/выходные, `intent(inout)`

```
subroutine sub(a, b, c)
  integer,    intent(in)      :: a
  real,       intent(out)     :: b
  character,  intent(inout)   :: c
  ...
end subroutine sub
```

По умолчанию все параметры имеют атрибут `inout`, т.к. передаются по ссылке.

Вид связи intent(in)

Принимают значение от соответствующего фактического параметра и не могут изменяться при выполнении процедуры.

Соответствующими фактическими параметрами могут быть выражения, переменные, значения, константы.

```
program param_in  
  call sub(100)  
end
```

```
subroutine sub(a)  
  integer, intent(in) :: a ! входной параметр  
  ...  
end subroutine sub
```

Вид связи intent(out)

Параметры передают свое значение соответствующему фактическому параметру и предназначены для вывода данных из процедуры.

Соответствующий фактический параметр должен быть переменной.

```
program param_out
  call sub(100) ! ошибка, ожидалась переменная
end

subroutine sub(a)
  integer, intent(out) :: a ! выходной параметр
  ...
end subroutine sub
```

Вид связи intent(inout)

Параметры могут как принимать значения от фактического параметра, так и передавать данные в фактический параметр.

Соответствующие фактические параметры должны быть переменными.

```
program param_inout
  integer :: q = 80
  call sub(q)
end

subroutine sub(a)
  integer, intent(inout) :: a ! входной/выходной
  ...                       ! параметр
end subroutine sub
```

Optional параметры

Атрибут **optional** устанавливает формальные параметры необязательными.

Функция **present** проверяет присутствие необязательного параметра при вызове процедуры.

При отсутствии проверки функцией **present** возможны ошибки выполнения для параметров с видом связи **intent(out, inout)**.

Процедура должна иметь явный интерфейс.

Optional параметры

```
program param_in
  real res
  res = logarithm(144.0, base = 12.0)
  write(*,*) res

CONTAINS
real function logarithm(a, base)
  real, intent(in)      :: a
  real, intent(in), optional :: base
  if (present(base)) then
    logarithm = log(a)/log(base)
  else
    logarithm = log10(a)
  end if
end function logarithm

END
```


Область видимости

Объекты описанные в головной программе доступны во внутренних процедурах, недоступны во внешних.

Объекты описанные во внутренней, внешней процедурах доступны только в них самих.

При совпадении имен объектов головной программы и внутренней или внешней процедур, объекты являются разными.

*Объекты – переменные, типы данных, внутренние процедуры (для головной программы или внешней процедуры)

Область видимости

```
program region
  integer a, b;   integer c, d
  ...
contains ! -----
  subroutine sub (a,b)
    integer q, p
    ...
  end subroutine sub
END
! *****

subroutine proc(a,b)
  integer a, b;   integer c, d
contains
  subroutine small(a,b)
    integer f, q
    ...
  end subroutine small
end subroutine proc
```

The diagram illustrates variable visibility in Fortran. Red arrows indicate the scope of variables 'a', 'b', 'c', and 'd'. Purple arrows indicate the scope of variables 'q' and 'p' within the 'sub' subroutine. A green dashed line separates the 'region' program from the 'proc' subroutine. A green asterisk line separates the 'proc' subroutine from the 'small' subroutine.

Область видимости

```
program array
  integer, parameter :: M = 10
  integer :: A(M) = [2,4,6,8,9,1,1,1,1,1]

  call PrintArray()

contains
  subroutine PrintArray() ! нет формальных параметров
    integer k             ! доступ к массиву A
    do k = 1,M             ! из головной программы
      write(*,"(i6,\)") A(k)
    end do
    write(*,*)
  end subroutine PrintArray

END
```

Оператор interface

```
program question  
  write(*,*) middle(3.0,2.0)  
END
```

```
real function middle(a,b)  
  real a,b  
  middle = (a+b)/2  
end function middle
```

По умолчанию тип функции **middle** – целый.
Функция объявлена как вещественная,
но головная программа об этом "не знает".

Как указать головной программе
про тип функции **middle** ?

Оператор interface

Определяет явно заданный интерфейс.
Используется для внешних процедур.
Внутренние процедуры обладают явно заданным интерфейсом.

interface

тип имя_функции(формальные параметры)

тип формальных параметров

конец описания функции

end interface

Оператор interface

Исправленная версия программы

```
program question
```

```
interface
```

```
  real function middle(a,b)
```

```
    real a,b
```

```
  end function middle
```

```
end interface
```

```
write(*,*) middle(3.0,2.0)
```

```
END
```

```
real function middle(a,b)
```

```
  real a,b
```

```
  middle = (a+b)/2
```

```
end function middle
```

Оператор return

Вернуть управление вызывающей процедуре или головной программе.

```
program break_function
```

```
write(*,*) div(3,0)
```

```
contains
```

```
integer function div(a,b)
```

```
integer a,b
```

```
if (b == 0) then
```

```
div = 0
```

```
write(*,*) "ERROR divided by zero"
```

```
return
```

```
else
```

```
div = a/b
```

```
end if
```

```
end function div
```

```
END
```

Процедуры как параметры

Универсализация процедур.

основная функция

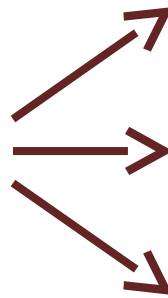
$$f(x) = \sum_{k=1}^{100} fun(k) \cdot x^k$$

$$p(x) = x + \sin x$$

$$q(x) = x - \cos x$$

$$s(x) = \sqrt{x} + x$$

функции
подставляемые
вместо функции *fun*



Процедуры как параметры

```
program param_func
external p, q, s

write(*,*) f(0.5, p) ! p(x) - фактический параметр
write(*,*) f(0.7, q) ! q(x) - фактический параметр
write(*,*) f(1.2, s) ! s(x) - фактический параметр
end

!*****

real function q(x) ! ----- функция q(x)
real x
  q = x-cos(x)
end function q

real function s(x) ! ----- функция s(x)
real x
  s = sqrt(x)+x
end function s
```

Процедуры как параметры

```
real function p(x) ! ----- функция p(x)
```

```
real x
```

```
  p = sin(x)+x
```

```
end function p
```

```
real function f(x, fun)
```

```
interface ! --- явный интерфейс для внешних функций
```

```
  real function fun(x)
```

```
    real x
```

```
  end function fun
```

```
end interface
```

```
real x
```

```
integer k
```

```
f = 0.0
```

```
do k = 1, 100
```

```
  f = fun(REAL(k))*x**k+f
```

```
end do
```

```
end function f
```

Процедуры как параметры

Оператор **external** объявляет, что перечисленные внешние процедуры передаются как параметры.

Если хотим передавать стандартные функции как параметры, то нет смысла писать

```
program param_func
  external mysin
  write(*,*) f(0.5, mysin)
end
! *****
real function mysin(x) ! ----- функция q(x)
real x
  q = sin(x)
end function mysin
```

Процедуры как параметры

Оператор **intrinsic** объявляет, что перечисленные стандартные процедуры передаются как параметры.

```
program param_func  
  intrinsic sin  
  write(*,*) f(0.5, sin)  
end  
...
```

Вместо указания функции в операторе **external** можно использовать блок **interface**.

Процедуры как параметры

```
program param_func
```

```
interface
  real function p(x)
    real x
  end function p
end interface
```

интерфейс
фактического
параметра-
функции

```
interface
  real function f(x, fun)
    interface
      real function fun(x)
        real x
      end function fun
    end interface
    real x
  end function
end interface
...
```

интерфейс
формального
параметра-
функции

Рекурсивные процедуры

Процедура вызывающая сама себя.

recursive – объявление рекурсивной процедуры.

Прямая рекурсия

proc ---> **proc** ---> **proc** --->

Косвенная рекурсия

proc ---> **sub** ---> **proc** ---> **sub** --->

Обязательна проверка окончания
рекурсивного вызова.

Рекурсивные процедуры

Рекурсивный вывод последовательности чисел.

```
program recurse
```

```
    call Number(10) !
```

```
contains
```

```
recursive subroutine Number(N)
```

```
integer N
```

```
    write(*,*) " N = ", N
```

```
    if (N == 1) return ! точка останова
```

```
    call Number(N-1) ! рекурсивный вызов
```

```
end subroutine Number
```

```
end
```

Рекурсивные процедуры

Результирующая переменная, предложение **result**.

```
program fact
```

```
  write(*,*) factorial(10)
```

```
contains
```

```
  recursive function factorial(p) result(k)
```

```
    integer, intent(in) :: p
```

```
    integer k
```

```
    if (p == 1) then
```

```
      k = 1
```

```
    else
```

```
      k = p * factorial(p - 1)
```

```
    end if
```

```
  end function
```

```
end
```


Чистые процедуры

Чистые процедуры – процедуры без побочных эффектов

```
program side_effect
  real :: s = 10.0

  write(*,*) (f(s)+f(s))/2.0 ! ожидалось 100
                              ! результат 110.5

contains
  real function f(x)
    real x
    f = x*x
    x = x+1
  end function f

end
```

Чистые процедуры

Ключевое слово **pure** объявляет процедуру чистой.

Для чистых процедур характерно

- все формальные параметры функций имеют вид связи **intent(in)**;
- все формальные параметры подпрограмм имеют вид связи **intent(in, out)** или **inout**;
- отсутствуют операторы В/В, **stop**.

Все встроенные функции являются чистыми.

Чистые процедуры

В операторе **forall** используются только чистые процедуры.

```
PROGRAM PURE_FORALL
real :: A(10) = -1.0
integer i

forall (i = 1:5, A(i) < 0)
  A(i) = F(A(i))
end forall

contains
  real pure function f(x)
    real, intent (in) :: x
    f = x*x
  end function f

END
```