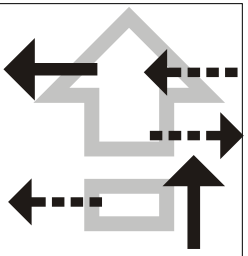
Глава 8



Структура программы

Как правило, программы не представляются единым блоком, а разделяются на логически завершенные подзадачи, размещаемые в различных *программных компонентах.* В Фортране выделяется три вида программных компонентов: *главная программа, внешняя подпрограмма, модуль* (начиная с Фортрана 90).

Главная программа является обязательным компонентом всякой программы, предназначенной для исполнения, ей операционная система передает управление для выполнения программы.

Главная программа организует работу программы в целом, используя вызовы *подпрограмм* — выделенных частей программы, возвращающих управление в место вызова. Подпрограммы могут быть *внешними*, то есть выделенными в самостоятельную программную единицу, или *внутренними* (внутренние подпрограммы появились в Фортране 90), встроенными в главную программу или внешнюю подпрограмму.

Элементы программы, используемые несколькими программными единицами — блоки описаний, модульные подпрограммы, интерфейсы внешних подпрограмм — объединяются в *модули*, обеспечивающие унифицированное представление этих элементов в различных компонентах. Модули включаются в другие программные единицы оператором `use`.

Порядок операторов

Во всех компонентах программы должен соблюдаться строго определенный порядок следования операторов. В соответствии с правилами каждый компонент разделяется на девять последовательных уровней; каждый вид операторов должен размещаться на одном или нескольких определенных уровнях и нигде больше. Предписываемый порядок размещения операторов показан в табл. 8.1.

Таблица 8.1. *Порядок следования операторов в программном компоненте*

Номер уровня	Операторы
1	`program`, `subroutine`, `function`, `module`
2	`use`
3	`implicit none`, `format`
4	`implicit`, `parameter`, `format`
5	`paramater`, `data`, различные операторы описания, `format`
6	исполняемые операторы, `format`
7	`contains`
8	описания внутренних или модульных подпрограмм
9	`end`

Главная программа

Главная программа может включать заголовок, операторы присоединения модулей, операторы описания, исполняемые операторы, описания внутренних подпрограмм. Единственным обязательным оператором главной программы является оператор `end`. Все элементы главной программы должны следовать в том порядке, в каком они перечислены. Внутренние подпрограммы отделяются от предшествующих им исполняемых операторов оператором `contains`.

Приведем пример. Программа `hypotenuse` (листинг 8.1) вычисляет длину гипотенузы по введенным значениям длин катетов. Вычисление квадратного корня из суммы квадратов выделено во внутреннюю подпрограмму.

Листинг 8.1. Вычисление длины гипотенузы

```
program hypotenuse
 implicit none
 real a, b, c
 write(*, *) 'введите длину катета a'
 read(*, *) a
 write(*, *) 'введите длину катета b'
 read(*, *) b
 c = hlength(a, b)
 write(*, *) 'длина гипотенузы c = ', c
 contains
        real function hlength(x, y)
         real x, y
         hlength = sqrt(x * x + y * y)
        end function hlength
end program hypotenuse
```

Атрибуты операторов `end` не являются обязательными, однако их наличие значительно улучшает читаемость программы.

Внешние подпрограммы

Подпрограмма, как внешняя, так и внутренняя, обычно выполняет одну из логически завершенных подзадач общей задачи, решаемой программой. Описание внешней подпрограммы размещается вне других программных компонентов. Структура внешней подпрограммы отличается от структуры главной программы наличием обязательного заголовка. Заголовок подпрограммы состоит из обязательного оператора-указателя типа подпрограммы (`subroutine` или `function`), обязательного имени подпрограммы и необязательного списка формальных параметров. Оператор-

указатель типа зависит от наличия возвращаемого подпрограммой значения. Подпрограмма, не обязательно возвращающая значения, описывается оператором `subroutine` и вызывается оператором `call`. Подпрограмма, возвращающая значение (подпрограмма-функция), описывается оператором `function`; в этом случае в заголовке подпрограммы может указываться тип возвращаемого значения. Обращение к подпрограмме-функции выполняется по имени функции со списком фактических параметров, которое включается в выражение (см. листинг 8.1). Возвращаемое значение подпрограммы-функции может быть и массивом (начиная с Фортрана 90).

Программа `hypotenuso` (листинг 8.2) также вычисляет длину гипотенузы по введенным значениям длин катетов, но в этом примере вычисление квадратного корня из суммы квадратов выделено во внешнюю подпрограмму.

Листинг 8.2. Вычисление длины гипотенузы

```
program hypotenuso
 implicit none
 real :: a, b, c, hlength
 write(*, *) 'введите длину катета a'
 read(*, *) a
 write(*, *) 'введите длину катета b'
 read(*, *) b
 c = hlength(a, b)
 write(*, *) 'длина гипотенузы c = ', c
end program hypotenuso

real function hlength(x, y)
 real x, y
 hlength = sqrt(x * x + y * y)
end function hlength
```

В следующем примере (см. листинг 8.3) вычисление длины гипотенузы по введенным значениям длин катетов представлено в виде внешней подпрограммы, содержащей внутреннюю подпрограмму вычисления квадратного корня из суммы квадратов.

Листинг 8.3. Вычисление длины гипотенузы

```
subroutine hypotenuss(a, b, c)
 implicit none
 real, intent(in) :: a, b
 real, intent(out) :: c
 c = hlength(a, b)
 contains
        real function hlength(x, y)
         real :: x, y
         hlength = sqrt(x * x + y * y)
        end function hlength
end subroutine hypotenuss
```

Модули

Модули — третий вид программных компонентов, которые используются для объединения данных глобального характера. В модули включаются описания общих для нескольких подпрограмм переменных, производных типов и подпрограмм. Модуль состоит из обязательного заголовка, необязательных операторов описания модульных переменных и модульных подпрограмм и завершающего оператора `end`. Описания модульных переменных от описаний модульных подпрограмм отделяются оператором `contains`.

Модуль `mconst` (листинг 8.4) содержит набор именованных констант. Компонентам, использующим эти константы, будет достаточно только подключить этот модуль оператором `use`.

Листинг 8.4. Пример модуля

```
module mconst
 implicit none
 real, parameter :: pi = 3.1415926, e = 2.7182818, phi = 1.618034
end module mconst
```

Модульные подпрограммы, если область их видимости не ограничивается специально, являются внешними подпрограммами. Они также могут включать в себя внутренние подпрограммы. Оператор `end`, завершающий описание модульной подпрограммы, должен содержать указание имени подпрограммы.

Модуль `rpoint` (листинг 8.5) содержит константы-координаты точки на плоскости и описание функции преобразования координат при повороте вокруг нее на заданный угол.

Листинг 8.5. Модуль с описанием функции

```
module rpoint
implicit none
real, parameter :: a = 10.5, b = 12.7
contains
        subroutine rotation(x, y, alpha)
         real, intent(inout) :: x, y
         real, intent(in) :: alpha
         real :: lx, ly
         lx = x - a
         ly = y - b
         x = lx * cos(alpha) - ly * sin(alpha) + a
         y = lx * sin(alpha) + ly * cos(alpha) + b
        end subroutine rotation
end module rpoint
```

Модули подключаются к другим программным компонентам оператором `use`. Этот оператор должен следовать сразу после заголовка компонента.

Пример подключения и применения модуля приведен в листинге 8.6.

Листинг 8.6. Пример подключения и применения модуля

```
program moduser
 use rpoint
 use mconst
```

```
 implicit none
 real :: r, x, y, alpha
 write(*, *) 'введите радиус'
 read(*, *) r
 write(*, *) 'длина окружности радиуса ', r, ' равна ', 2 * pi * r
 write(*, *) 'введите координаты точки на плоскости'
 read(*, *) x, y
 write(*, *) 'введите угол поворота в радианах'
 read(*, *) alpha
 write(*, *) 'при повороте на угол ', alpha, &
  ' относительно точки ', a, b
 write(*, *) 'точка с координатами ', x, y
 call rotation(x, y, alpha)
 write(*, *) 'преобразуется в точку с координатами ', x, y
end program moduser
```

Оператор `use` допускает введение переименований при подключении. Это позволяет обойти возможные конфликты имен локальных и модульных переменных. Кроме того, с помощью директивы `only` оператора `use` можно ограничить доступ к модульным переменным.

Программа `modrename` использует константы `phi` и `pi` из модуля `mconst` и собственную переменную с именем `phi`. Модуль `mconst` подключается с переименованием параметра `phi` и ограничением доступа (модульная именованная константа `e` при этом подключении остается недоступной).

Листинг 8.7. Пример подключения модуля с переименованием и ограничением доступа

```
program modrename
 use mconst, only: mphi => phi, pi
 implicit none
 real :: l, phi
 write(*, *) 'введите длину отрезка l'
 read(*, *) l
 phi = l * mphi
```

```
 write(*, *) 'длина окружности радиуса ', l, ' равна ', 2 * pi * l
 write(*, *) 'прямоугольник идеально пропорционален, ', &
  ' если при высоте ', l
 write(*, *) 'он имеет ширину ', phi
end program modrename
```

Модули удобно использовать для хранения описаний производных типов и операций над ними.

Модуль `modpoint3d` (листинг 8.8) содержит описание производного типа `point3d`, представляющего точку трехмерного пространства. В модуле определяется и подпрограмма `distance`, возвращающая расстояние между двумя точками `point3d`.

Листинг 8.8. Пример модуля

```
module modpoint3d
implicit none
type :: point3d
        real :: x, y, z
end type point3d
contains
        real function distance(a, b)
         type(point3d), intent(in):: a, b
         distance = sqrt((a%x - b%x)**2 + &
          (a%y - b%y)**2 + (a%z - b%z)**2)
        end function distance
end module modpoint3d
```

Программа `mostfar` (листинг 8.9) конструирует объекты типа `point3d` и выбирает из них точку, наиболее удаленную от заданной.

Листинг 8.9. Пример использования модуля `modpoint3d`

```
program mostfar
 use modpoint3d
 implicit none
 type(point3d), :: current, center,prmax
```

```
 parameter(center = point3d(1, 1, 1))
 real :: x, y, z, rmax = 0, r
 character :: seq = 'y'
 do
        if(seq == 'y'.or.seq == 'y') then
                write(*, *) 'введите координаты x, y, z'
                read(*, *) x, y, z
                current = point3d(x, y, z)
                r = distance(current, center)
                if(rmax < r)then
                        rmax = r
                        prmax = current
                        write(*, *) 'самая удаленная точка: ',&
                         prmax%x, ' ', prmax%y, ' ', prmax%z
                endif
                write(*, *) 'новая точка? (y/n) '
                read(*, *) seq
        else
                exit
        endif
 enddo
end program mostfar
```

При необходимости можно разграничить элементы модуля на *публичные* и *приватные*, доступные только внутри модуля. Это можно сделать, назначая элементам атрибуты `public` и `private` соответственно либо используя одноименные операторы. По умолчанию все элементы модуля считаются публичными.

Внутренние подпрограммы

В отличие от внешних и модульных подпрограмм *внутренние подпрограммы* не могут содержать вложенных подпрограмм. В остальном их форма не отличается от формы внешних подпрограмм. Кроме того, внутренним подпрограммам доступны все элементы компонента-носителя. Внутренняя подпрограмма, однако, определяет свою область видимости для переменных и меток.

Внутренняя подпрограмма `reusex` определена программой `usesxy` (листинг 8.10). Переменные `x` и `y`, описанные программой-носителем, доступны внутренней подпрограмме, однако подпрограмма `reusex` определяет собственную переменную с именем `x`.

Листинг 8.10. Пример использования внутренней подпрограммы

```
program usesxy
 implicit none
 real :: x = 5, y = 6
 write(*, *) 'program usesxy: x=', x, ' y = ', y
 call reusex
 contains
        subroutine reusex
                real :: x = 22
                write(*, *) 'subroutine reusex: x = ', x, ' y = ', y
        end subroutine reusex
end program usesxy
```

Внутренние подпрограммы доступны только из компонента-носителя.

Параметры подпрограмм

Подпрограммы, как правило, описывают наборы действий, используемые неоднократно с различными значениями переменных, над которыми эти действия выполняются. Исходные значения и результаты работы подпрограммы могут передаваться подпрограмме через *параметры*. Параметры, перечисляемые в описании подпрограммы, называются *формальными.* При обращении к подпрограмме на место формальных параметров подставляются переменные с определенными значениями — *фактические* параметры. Типы формальных и фактических параметров должны совпадать.

Программа `factipars` (листинг 8.11) задает фактические параметры подпрограммы `formipars`, которая выводит на экран значения переданных ей параметров.

Листинг 8.11. Пример использования подпрограммы

```
program factipars
 implicit none
 real :: x
 integer :: num
 character(len = 11) :: name
 character :: seq = 'y'
 do
        if(seq == 'y'.or.seq == 'y') then
                write(*, *) 'введите число вещественного типа'
                read(*, *) x
                write(*, *) 'введите число целого типа'
                read (*, *) num
                write(*, *) 'введите слово (не более 11 символов)'
                read(*, *) name
                call formipars(x, num, name)
                write(*, *) 'новый набор? (y/n) '
                read(*, *) seq
        else
                exit
        endif
 call formipars(0.5e0, 1000, 'конец обеда')
 enddo
end program factipars

subroutine formipars(a, b, word)
 implicit none
 real :: a
 integer :: b
 character(len = 11) :: word
 write(*, *) 'параметр a = ', a, ' параметр b = ', b,&
   ' параметр word = ', word
end subroutine formipars
```

Использование параметров для обмена данными с подпрограммой дает возможность контролировать действия подпрограммы над ними. Атрибут в описании формальных параметров подпрограмм `intent` позволяет выделить входные, выходные и смешанные параметры. Параметры, отмеченные как входные, не могут использоваться в левой части оператора присваивания и в качестве фактического параметра другой подпрограммы. Выходные параметры теряют свои значения при входе в подпрограмму. Выходные и смешанные параметры обязаны быть переменными.

Программа `user` (листинг 8.12) вызывает подпрограмму `subuser`, передавая ей два входных параметра и получая результат выполнения этой подпрограммы через третий параметр.

Листинг 8.12. Пример использования подпрограммы

```
program user
 implicit none
 real :: a, b, c
 a = 1.45e0
 b = -0.8e0
 call subuser(a, b, c)
 write(*, *) 'синус суммы углов a = ', a, ' и b = ', b, ' равен ', c
end program user

subroutine subuser(x, y, c)
 implicit none
 real, intent(in) :: x, y
 real, intent(out) :: c
 c = sin(x + y)
end subroutine subuser
```

Подставлять фактические параметры на места соответствующих формальных параметров (*позиционный* метод передачи параметров) при вызове подпрограммы необязательно. Если подпрограмма имеет *явный интерфейс* (см. далее *разд. "Интерфейсы" этой главы*), будучи, например, внутренней или модульной, то можно использовать *ключевой* метод передачи параметров.

При ключевом методе каждый фактический параметр передается с ключом, в качестве которого используется формальный параметр. Порядок параметров при этом методе не имеет значения. Можно смешивать оба метода, передавая начальную часть списка параметров позиционным методом, а оставшуюся — ключевым. Формальные параметры, отмеченные атрибутом `optional`, могут отсутствовать в списке фактических параметров вообще. В таком случае, для "опознания" переданных необязательных параметров должен использоваться ключевой метод передачи параметров. Наличие или отсутствие необязательного параметра может быть проверено встроенной функцией `present`.

Третий формальный входной параметр внутренней подпрограммы `nonmnd` необязательный. В зависимости от наличия этого параметра в фактических параметрах возвращается различный по смыслу результат. Программа `usesit` (листинг 8.13) вызывает `nonmnd` с различным числом входных параметров.

Листинг 8.13. Пример использования необязательных параметров

```
program usesit
 implicit none
 integer :: a, b, c, r
 character(len = 10) :: comment
 write(*, *) 'введите три целых числа > 0'
 read(*, *) a, b, c
 call nonmnd(a, b, c, r, comment)
 write(*, *) 'результат:', r, ' ', comment
 write(*, *) 'введите два целых числа > 0'
 read(*, *) a, b
 call nonmnd(x = a, y = b, re = r, comm = comment)
 write(*, *) 'результат:', r, ' ', comment
 contains
        subroutine nonmnd(x, y, z, re, comm)
         integer, intent(in) :: x, y, z
         optional z
         integer, intent(out) :: re
         character(len = 10), intent(out) :: comm
```

```
         comm = 'площадь'
         re = abs(x * y)
         if(present(z)) then
               comm = 'объем'
               re = abs(re * z)
         endif
        end subroutine nonmnd
end program usesit
```

В качестве параметра подпрограмме может быть передано не только значение, но и имя внешней или модульной подпрограммы. Параметр-подпрограмма должен отмечаться оператором `external` или описываться в интерфейсном блоке (см. далее).

Подпрограмма-функция `dothis` (листинг 8.14) возвращает значение функции, заданной параметром, от переданного аргумента.

Листинг 8.14. Пример использования функции

```
program usedothis
 implicit none
 real :: x = 2.0
 real, external :: quadrat, qroot, quadsin
 write(*, *) 'x = ', x
 write(*, *) 'x в квадрате = ', dothis(x, quadrat)
 write(*, *) 'корень квадратный из x = ', dothis(x, qroot)
 write(*, *) 'синус x в квадрате = ', dothis(x, quadsin)
 contains
        real function dothis(x, funcname)
         real :: x
         real, external :: funcname
         dothis = funcname(x)
        end function dothis
end program usedothis

real function quadrat(r)
 real :: r
 quadrat = r * r
end function quadrat
```

```
real function qroot(r)
 real :: r
 qroot = sqrt(r)
end function qroot

real function quadsin(r)
 real :: r, s
 s = sin(r)
 quadsin = s * s
end function quadsin
```

Подпрограммы в общем случае не могут вызывать сами себя. Чтобы разрешить это, нужно объявить подпрограмму *рекурсивной*, включив в ее заголовок префикс `recursive` и дополнение `result`, указывающее переменную, в которой помещается результат.

Рекурсивная внутренняя подпрограмма `factorial` используется программой `getfact` (листинг 8.15) для вычисления факториала целого числа.

Листинг 8.15. Пример использования рекурсивной подпрограммы

```
program getfact
 implicit none
 integer :: arg
 1      write(*, *) 'введите положительное целое число < 10'
 read(*, *) arg
 if(arg > 10.or.arg < 0) goto 1
 write(*, *) arg,' != ', factorial(arg)
 contains
        recursive integer function factorial(n) result(f)
         integer :: n
         if(n == 1) then
                f = 1
         else
                f = n * factorial(n - 1)
         end if
        end function factorial
end program getfact
```

Интерфейсы подпрограмм

Для использования подпрограммы, необходимо знать ее *интерфейс* — тип подпрограммы, список ее формальных параметров и их типы. Внутренние и модульные подпрограммы обладают *явным*, то есть доступным на этапе компиляции, интерфейсом.

Проверить же соответствие описаний фактических и формальных параметров внешних и формальных подпрограмм на этапе компиляции невозможно. Исправить этот недостаток можно с помощью *интерфейсов*, содержащих копии заголовков внешних подпрограмм с описаниями их формальных параметров.

Программа `undebt` (листинг 8.16) использует внешнюю подпрограмму `subby`, считая, что последней требуется три фактических параметра вещественного типа. В описании `subby` использует четыре параметра: два целого и два вещественного типов.

Листинг 8.16. Пример использования подпрограммы

```
program undebt
 implicit none
 real :: a, b, c
 a = 12.3; b = 4.2; c = 5.5
 write(*, *) 'a + b + c = ', (a + b + c)
 call subby(a, b, c)
end program undebt

subroutine subby(a, b, c, x)
 real :: c, x
 integer :: a, b
 write(*, *) 'subby: a + b + c ', (a + b + c)
end subroutine subby
```

Описание интерфейса состоит из обязательного *заголовка* `interface`, за которым следует *тело интерфейса* — копия заголовка внешней подпрограммы с описаниями ее формальных параметров. Описание интерфейса завершается оператором `end interface`. Интерфейсные блоки размещаются среди операторов описания.

Эффекта, произведенного несоответствием формальных и фактических параметров, не будет, если в программу `undebt` (листинг 8.17) включить описание интерфейса подпрограммы `subby`. В этом случае несоответствие типов будет обнаружено при компиляции программы.

Листинг 8.17. Пример описания интерфейса

```
program undebt
 implicit none
 real :: c, x
 integer :: a, b
 interface
        subroutine subby(a, b, c, x)
         real :: c, x
         integer :: a, b
        end subroutine subby
 end interface
 a = 12.3; b = 4.2; c = 5.5
 write(*, *) 'a + b + c = ', (a + b + c)
 call subby(a, b, c, x)
end program undebt

subroutine subby(a, b, c, x)
 real :: c, x
 integer :: a, b
 write(*, *) 'subby: a + b + c = ', (a + b + c)
end subroutine subby
```

С помощью именованных интерфейсных блоков можно объединять несколько подпрограмм под единым именем. В результате по общему имени будет вызываться подпрограмма с подходящим набором параметров. В именованных интерфейсах все подпрограммы должны быть либо функциями, либо процедурами. Имена специфических подпрограмм могут совпадать с родовым именем. Не допускается полного совпадения формальных параметров подпрограмм одного именованного интерфейса, вне зависимости

от различий их позиций в списке формальных параметров и различий в названии подпрограмм.

Модуль `union` (листинг 8.18) содержит описание *родового интерфейса*, в котором под общим названием объединены схожие по смыслу, но различные по типам используемых параметров внешние подпрограммы-функции.

Листинг 8.18. Пример модуля с описанием родового интерфейса

```
module union
 implicit none
 interface gamma
        function rgamma(x)
         real :: rgamma, x
        end
        function igamma(x)
         integer :: igamma, x
        end
 end interface
end module union

function rgamma(x)
 real :: rgamma, x
 rgamma = 1.0 + x * x
end function rgamma

function igamma(x)
 integer :: igamma, x
 igamma = 1 + x * x
end function igamma
```

Программа `genby` (листинг 8.19) использует модуль `union` и обращается к родовому имени функций `rgamma` и `igamma`.

Листинг 8.19. Программа, использующая модуль `union`

```
program genby
 use union
 implicit none
```

```
 integer :: i = 5
 real :: r = 0.5e1
 write(*, *) 'gamma(', i, ') = ', gamma(i)
 write(*, *) 'gamma(', r, ') = ', gamma(r)
end program genby
```

При помещении в именованный интерфейсный блок модульных подпрограмм нет необходимости повторять в нем их заголовки и описания, достаточно перечислить их названия в директиве `module procedure`. В этом случае удобнее поместить именованный интерфейсный блок в модуль.

С помощью именованного интерфейсного блока с модульными процедурами можно переопределять встроенные операции и операцию присваивания для производных типов.

В модуле `mod3d` (листинг 8.20) переопределяются присваивание и встроенная операция умножения для типа `point3d`.

Листинг 8.20. Пример использования модуля `modpoint3d`

```
module mod3d
 use modpoint3d
 implicit none
 function vm(x, y)
  type(point3d), intent(in) :: x, y
  type(point3d) vm
  vm = point3d(x%y * y%z - y%y * x%z, x%x * y%z - &
        y%x * x%z, x%x * y%y - y%x - x%y)
 end function vm

 function ass3d(x)
  type(point3d), intent(in) :: x
  type(point3d) ass3d
  ass3d = point3d(x%x, x%y, x%z)
 end function ass3d

 interface assignment(=)
  module procedure ass3d
```

```
 end interface

 interface operator(*)
  module procedure vm
 end interface
end module mod3d
```

Области видимости имен и меток

Внутренние подпрограммы и интерфейсные блоки имеют собственные пространства имен, и потому можно не заботиться о совпадении имен их переменных с именами компонента-носителя. Имена, используемые внутри интерфейсов, доступны только внутри интерфейсов и больше нигде. Поэтому описывать интерфейсы можно простым копированием заголовков подпрограмм и их блоков описаний. Внутренней же подпрограмме доступны все переменные компонента-носителя, если только такие же имена не встречаются в ее собственном блоке описания: имена из внутреннего блока описания перекрывают имена носителя.

Имена модульных переменных и подпрограмм доступны из всех компонентов присоединяющей программы или подпрограммы, если только он не описывает собственных элементов с такими же именами.

В отношении меток соблюдается следующее правило: все компоненты обладают собственным пространством меток, и нет никакой необходимости заботиться об их уникальности.

Задачи

Задача 8.1

Напишите программу, считывающую два вещественных числа и вычисляющую синус и косинус суммы этих чисел. Оформите вычисление сначала в виде внутренней, а затем внешней подпрограммы. Используйте свойства внутренних подпрограмм.

Задача 8.2

Напишите программу, вычисляющую на заданном отрезке с указанным шагом все значения аргумента и функции, заданной параметром подпрограммы. Функция может быть одной из следующих:

1. $y = \dfrac{1}{1+x^2}$.
2. $y = \dfrac{1}{1+\ln(x)}$.
3. $y = 1 + x^2 \ln(x)$.

Задача 8.3

Напишите программу интегрирования произвольной функции двух переменных по заданной прямоугольной области плоскости. Используйте рекурсивные подпрограммы. Проверьте работу программы для функции $z = \dfrac{1}{y+x^2}$.

Задача 8.4

Создайте модуль, описывающий комплексный тип как производный. Определить для этого типа все встроенные операции, используя именованные интерфейсы. Проверьте работу модуля, используя встроенный тип `complex`.

Задача 8.5

Опишите подпрограмму, конструирующую объекты типа `point3d` из трех необязательных входных параметров любого числового типа, кроме комплексного. Отсутствие параметра равносильно нулевому значению обозначаемой им координаты.

Задача 8.6

Опишите тип `point4d`, представляющий собой точку четырехмерного пространства. Определите для этого типа встроенные операции сложения и умножения, а также присваивание.

Задача 8.7

Без помощи компьютера определите, какие ошибки в использовании параметров могут привести к неудаче при попытке компиляции следующей программы:

```
program notvery
 implicit none
 real :: x, z
 real, parameter :: y = 5
 x = 2
 call subpro(x, z, y)
 call subpro(5, 21 + 8, x)
end program notvery

subroutine subpro(a, b, c)
 real, intent(in) :: a
 real, intent(out) :: b
 real, intent(inout) :: c
 a = 2.18 / (b - c)
 c = a * a + b
end subroutine subpro
```

Задача 8.8

Без помощи компьютера определите, будет ли откомпилирована следующая программа и какие значения будут напечатаны, если она будет выполнена?

```
program main
 use numbers
 implicit none
 real :: x, z
 call subpro(x)
 z = y
 write(*, *) x, ' ', y
 contains
    subroutine subpro(a)
     implicit none
```

```
      real :: a
      a = x + y
    end subroutine subpro
end program main

module numbers
 implicit none
 real, parameter :: x = 10, y = 5
end module numbers
```

Задача 8.9

Определите, какие из следующих именованных интерфейсов описаны с ошибкой?

1.
```
interface grabeer(i, x, y)
 function rabber(i, x, y)
  real :: y, x
  integer :: i
 end

 function gabber(x, ig, y)
  integer :: ig
  real :: x, y
 end

end interface
```

2.
```
interface grabeer(i, x, y)

subroutine rabber(i, x, y)
 real :: y, x
 integer :: i
end

function gabber(x, ig, y)
 integer :: ig, x
 real :: y
```

```
    end

    end interface
```

3.
```
    interface grabeer(i, x, y)

    subroutine rabber(i, x, y)
     real :: y, x
     integer :: i
    end

    subroutine grabber(x, ig, y)
     integer :: ig, x
     real :: y
    end

    end interface
```