

Программирование

Лекция «Типы данных»

Петров Александр Владимирович

Фёдоров Станислав Алексеевич

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

Типы данных

Стандартные

`integer` целый

`real` вещественный

`complex` комплексный

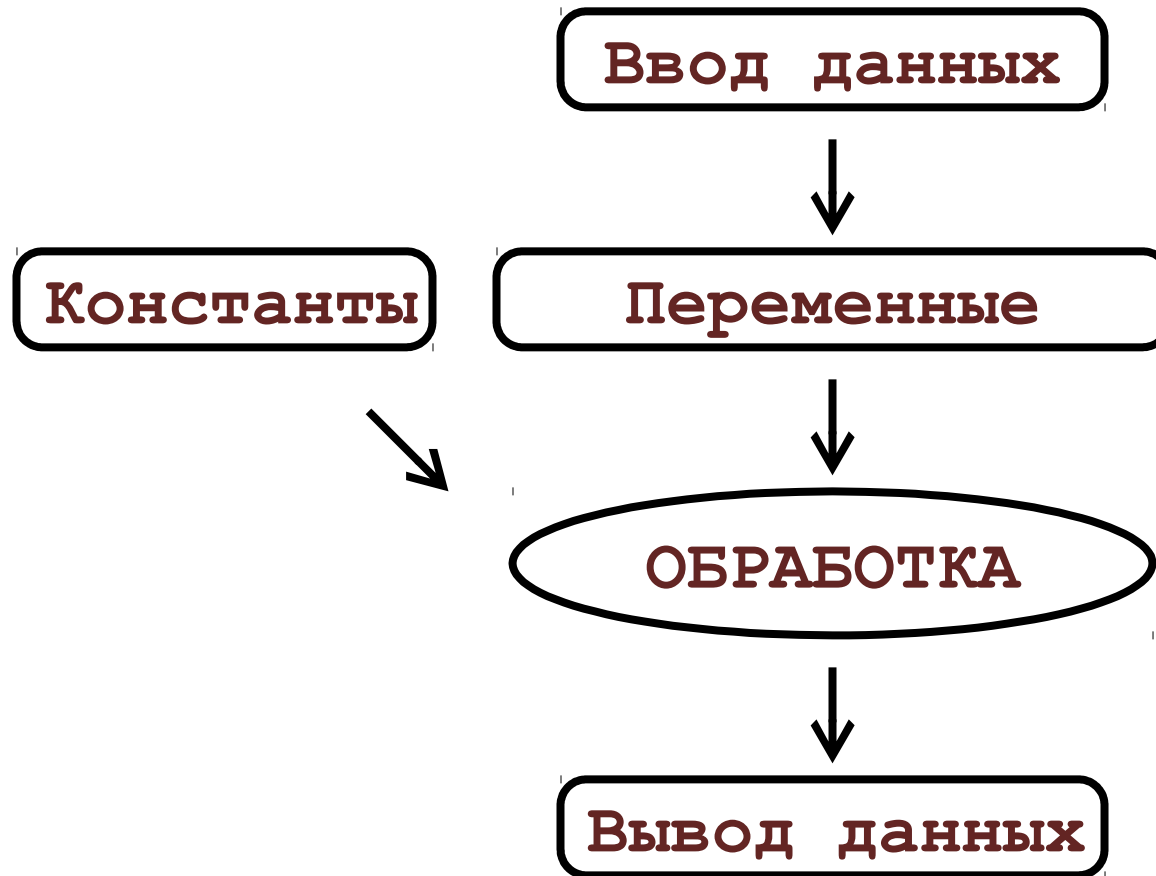
`logical` логический

`character` символьный

Производные

`type` (имя типа)

Переменные и константы



Значения переменных могут изменяться,
константы содержат всегда оно и тоже значение.

Задание имён

- * Латинские буквы **A..Z, a..z**
(маленькие и большие не различаются).
- * Цифры и знак подчеркивания **со 2-й позиции.**
- * Длина имени **не более 63** символа.

R_0
Vortex
a1
DistPol
Правильно

2pressure
func(x)
sd:q
w.x
Неверно

Используйте осмысленные имена!

Целочисленный тип

тип	длина (байт)	диапазон
<code>integer(1)</code>	1	-128 .. 127
<code>integer(2)</code>	2	-32768 .. 32767
<code>integer(4)</code>	4	- 2^{31} .. $2^{31}-1$
<code>integer(8)</code>	8	- 2^{63} .. $2^{63}-1$

Номера, счетчики, переменные циклов,
границы и индексы массивов.

Целочисленный тип

! ----- способы объявления переменных

```
integer(1) a5
integer(4) nomer
integer index
integer*2 b2
```

! ----- способы объявления констант

```
integer, parameter :: T0 = 500
integer(2), parameter :: fact = 10100
integer(8), parameter :: QW = 3**20
```

! ----- следует объявлять переменные и константы

```
integer, parameter :: I_ = 4
integer(I_), parameter :: IntConst
integer(I_) :: IntVar
```

```
index = 1000      ! переменным присвоили значения
a5 = B'1000101    ! двоичное представление
b2 = O'347'        ! восьмеричное
c7 = Z'AAB'        ! шестнадцатеричное
```

Инициализация переменных

```
program unknown  
  integer koef  
  write(*,*) koef  
end
```

Инициализация переменной –
объявление + присваивание значения.

```
integer :: a = 10 ! инициализация переменной  
integer :: s = 2**8+3**7+4**6 ! арифметические операции
```

Вещественный тип

тип	длина (байт)	точность (знаков)	диапазон
<code>real(4)</code>	4	7	$1.2 \cdot 10^{-38}$.. $3.4 \cdot 10^{+38}$
<code>real(8)</code> или <code>double precision</code>	8	15	$2.3 \cdot 10^{-308}$.. $1.7 \cdot 10^{+308}$
<code>real(16)</code>	16	33	$3.4 \cdot 10^{-4932}$.. $1.1 \cdot 10^{+4932}$

Переменные используемые для математических вычислений.

Вещественный тип

```
use ISO_Fortran_Env
! ----- переменные
real(4) :: p = 3.14159 ! 3.14159_4
real(4) :: s = 0.00001 ! .00001 или 1E-5

real(4) :: A = 6.79E+15
real(4) :: B = -9.0E-10

real(8) :: q = 123456789D+5
double precision :: f = +2.7843D0

real(16) :: p1 = 123456789Q4000
real(16) :: p2 = -1.23Q-400

! ----- константа
real, parameter :: pressure = 1e+10

! ----- следует объявлять переменные и константы
integer, parameter :: R_ = REAL32, R2_ = REAL64, R4_ = REAL128
```

Комплексный тип

тип	длина (байт)	точность (знаков)
<code>complex(4)</code>	8	7
<code>complex(8)</code>	16	15
<code>complex(16)</code>	32	33

```
! ----- переменные
complex(R_) c1
complex :: i1 = (0, 1) ! мнимая единица

! ----- константа
complex, parameter :: z = (2, 3) ! 2+3i
```

Переменные для обработки комплексных данных
(корни уравнений, преобразования Фурье).

Арифметические операции

операция	название	порядок	выполнение
**	степень	1	←
*	умножение	2	→
/	деление	2	→
-, +	знак числа	3	←
+	сложение	4	→
-	вычитание	4	→

Целочисленная арифметика

Деление целого числа на целое – результат целое.

```
S = 1/3 + 1/3 + 1/3    ! S = 0.0  
P = 16**(1/4)          ! P = 1.0
```

Запись целого числа в вещественной форме.

```
S = 1.0/3.0 + 1./3. + 1./3    ! S = 1.0  
P = 16**(1.0/4)              ! P = 2.0
```

Деление целого числа на нуль – ошибка выполнения.

```
m = 2/3  
k = n/m    ! деление на нуль
```

Переполнение значения.

```
integer(1) :: bt = 127  
bt = bt+1    ! bt = -128
```

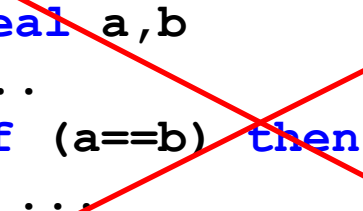
Вещественная арифметика

Действительные числа представлены
с определенной точностью.

$$a + (b+c) \neq (a+b) + c$$
$$(a+b)^2 \neq a^2+2ab+b^2$$

```
a = 1.0/3; b = 4.0/7
write(*,*) (a+b)**2           ! 0.8185941
write(*,*) a**2+2*a*b+b**2    ! 0.8185942
```

Не рекомендуется сравнивать на равенство
вещественные числа!



```
real a,b
...
if (a==b) then
...
```

```
real, parameter :: eps=1.E-5
real a,b
...
if (abs(a-b)<eps) then
...
```

Вещественная арифметика

Переход от типа с меньшей точности к большей может привести к погрешности.

```
program lost_precision
  real(R_) :: a = 1.03
  real(R2_) b

  b = a

  write(*,*) "a = ", a      ! 1.030000
  write(*,*) "b = ", b      ! 1.02999997138977

end
```

Вещественная арифметика

Не следует использовать в одном выражении значения, различие между которыми превышает число значащих цифр.

```
program arifm

  real(R_) :: a = 1.E+10, b = -1.E+10, c = 5.0

  write(*,*) a+b+c    ! 5.000000
  write(*,*) a+c+b    ! 0.00000000E+00

end
```

Вещественная арифметика

Деление вещественного числа на нуль –
бесконечность.

```
real(R_) a,b  
a = 1.0/0.0      ! результат Infinity  
b = -1.0/0.0     ! -Infinity
```

Результат Nan "нет числа" – недопустимый результат.

```
real(R_) a,b  
a = (-2.0)**0.34 ! Nan  
b = asin(2.0)    ! Nan
```

Логическая функция **IsNan(x)**
проверки на значение Nan.

Смешанная арифметика

Автоматическое приведение типов по схемам

"целый \rightarrow вещественный \rightarrow комплексный"

"от меньшей разрядности \rightarrow к большей"

Исключение – операция возведение в степень.

Запись $2^{**}5$ равносильна $2*2*2*2*2$
 $2^{**}5.0$ равносильна `exp(5.0*ln(2.0))`

Преобразование типов может приводить
к появлению погрешности !

Снижение погрешности



Не вычитайте близкие числа.



Не делите большие по модулю числа на малые.



Сложение (вычитание) длинной последовательности чисел начинайте с меньших чисел.



Уменьшайте число операций.



Используйте алгоритмы, для которых известны оценки ошибок.



Не сравнивайте на равенство вещественные числа.

Арифметические выражения

Математика	Fortran
$2a + 3(b + c)$	<code>2*a+3* (b+c)</code>
$\frac{a + b}{c + d}$	<code>(a+b) / (c+d)</code>
$\frac{a}{b \cdot c \cdot d}$	<code>a/ (b*c*d) или a/b/c/d</code>
$\sqrt[7]{a^5}$	<code>a** (5.0/7.0)</code>
$a^2 + b^5$	<code>a*a + b**5</code>

Используйте дополнительные переменные для повышения читаемости кода программы.

Преобразование числовых типов

Приведение к целому типу

`int(a, kind)`, `kind = 1, 2, 4, 8`

Приведение к вещественному типу

`real(a, kind)`, `kind = 4, 8, 16`

Приведение к комплексному типу

`cmplx(a, kind)`, `kind = 4, 8, 16`

`a` – целого, вещественного или
комплексного типов.

```
integer :: a = 10
real(R4_) s
s = real(a, R4_) ! привели к типу real(R4_)
```

Операция присваивания

$k = k+1$! увеличение значения на 1
 $k = k-1$! уменьшение значения на 1
 $k = 2*k$! увеличение в 2 раза
 $k = k/2$! уменьшение в 2 раза
 $k = -k$! смена знака

$s = s+k$! увеличение s на k
 $s = s-k$! уменьшение s на k
 $s = s*k$! увеличение s в k раз
 $s = s/k$! уменьшение s в k раз

$tmp = a$! поменяли местами значения переменных a и b
 $a = b$
 $b = tmp$

Логический тип

тип	длина (байт)	значения
<code>logical(1)</code>	1	.TRUE. .FALSE.
<code>logical(2)</code>	2	
<code>logical(4)</code>	4	
<code>logical(8)</code>	8	

```
! ----- переменные
logical(4) :: st = .FALSE.
logical    :: res = .TRUE.
```

Переменные-флаги, проверки наступления событий,
конструкции **if**.

Операции отношения

Операция	Имя
> или .gt.	больше
< или .lt.	меньше
== или .eq.	равно
/= или .ne.	не равно
>= или .ge.	больше либо равно
<= или .le.	меньше либо равно

Не используйте устаревшие нотации операций: **.gt.** и т. д.

logical position

position = 3<5 **! .TRUE.**

position = 3==0 **! .FALSE.**

Операция AND

Логическое умножение, конъюнкция.

$1 \text{ .and. } 1 = 1$	$0 \text{ .and. } 1 = 0$
$1 \text{ .and. } 0 = 0$	$0 \text{ .and. } 0 = 0$

Операция OR

Логическое сложение, дизъюнкция.

$$1 \text{ .or. } 1 = 1$$

$$0 \text{ .or. } 1 = 1$$

$$1 \text{ .or. } 0 = 1$$

$$0 \text{ .or. } 0 = 0$$

Операция XOR

Логическое исключающее "или", строгая дизъюнкция.

$$1 \text{ .xor. } 1 = 0$$

$$0 \text{ .xor. } 1 = 1$$

$$1 \text{ .xor. } 0 = 1$$

$$0 \text{ .xor. } 0 = 0$$

Операция NOT

Логическое отрицание, инверсия.

`.not. 1 = 0`

`.not. 0 = 1`

Операции эквивалентности

$$1 \text{ .eqv. } 1 = 1$$

$$1 \text{ .eqv. } 0 = 0$$

$$1 \text{ .neqv. } 1 = 0$$

$$1 \text{ .neqv. } 0 = 1$$

$$0 \text{ .eqv. } 1 = 0$$

$$0 \text{ .eqv. } 0 = 1$$

$$0 \text{ .neqv. } 1 = 1$$

$$0 \text{ .neqv. } 0 = 0$$

СИМВОЛЬНЫЙ ТИП

Объявления переменной для хранения 1 символа.

```
character key  
character(1) ch  
character(LEN=1) symbol
```

Объявления строки для хранения 100 символов.

```
character(100) str  
character word*100 ! Устаревшее.  
character(Len=100) path  
integer, parameter :: CH_ = Selected_Char_Kind("ISO_10646")  
character(Len=10, Kind=CH_) ISO_string  
character(CH_) WiT ! Какой это тип данных?
```

Имена файлов, обработка клавиш,
внутренние файлы, любая текстовая информация.

СИМВОЛЬНЫЙ ТИП

Объявления СИМВОЛЬНЫХ КОНСТАНТ.

```
character, parameter :: key = 'A'  
character(1), parameter :: ch = "Q"  
character(100), parameter :: str = "C:\"  
character(len=11), parameter :: path = "D:\data.txt"
```

Присваивание символьных значений.

```
str = ' It's very good! ' ! ' ' 1 апостроф  
adr = '"TEXT" ' ! "TEXT"
```

С - строки

Символьная константа заканчивающаяся символом С.

```
character(100) cstr
```

```
cstr="Fortran & C++"C    ! cstr - C-строка
```

Управляющие символы в С-строках:

\\ – слеш;

\a – звуковой сигнал;

\b – на 1 символ назад;

\n – новая строка;

\r – возврат каретки;

\t – горизонтальная табуляция;

и другие.

Операции со строками

// - конкатенация (сцепление, соединение) строк.

```
character a*5, b*2, c*20
...
a = 'AAAAA'
b = '...'
c = a//b//a    ! AAAAA..AAAAA
```

Обращение к подстроке, нумерация с единицы.

```
character (100) str, substr
str = '1234567890'
substr = str(1:3)    ! 123
```


Процедуры обработки строк *

Процедура	Описание
<code>len(str)</code>	длина строки
<code>len_trim(str)</code>	длина строки без хвостовых пробелов
<code>index(str, sub)</code>	номер первого вхождения строки substr в строку str
<code>iachar(ch)</code>	ASCII-код символа
<code>achar(code)</code>	возврат символа с кодом <code>code</code>
<code>getcharqq()</code> **	возврат нажатого символа
<code>peekcharqq(x)</code> **	определение нажатия клавиши

Ввод/вывод

Дескрипторы данных

Дескриптор	Тип	Представление
nIw [.m]	Целый	Целое число
nFw.d	Вещественный	F-форма
nEw.d	Вещественный	E-форма
nLw	Логический	T и F , .T и .F , .TRUE. и .FALSE.
nAw	Символьный	Строка символов

n – число повторений;

w – количество выводимых символов;

m – число ведущих нулей;

d – число цифр после десятичной точки.

Примеры вывода данных

```
integer :: a = 10, b = 20, c = 30
real    :: s = 1.237, p = 1.87342E+10
complex :: k = (0.0,1.0)
logical :: st = .true.
character :: key = 'A'
```

```
write(*,"(3i4)") a,b,c      ! ^^10^^20^^30
write(*,"(f10.5)") s        ! ^^^1.23700
write(*,"(E10.2)") p        ! ^^0.19E+11
write(*,"(2f5.1)") k        ! ^^0.0^^1.0
write(*,"(L2)") st          ! ^T
write(*,"(A4)") key          ! ^^A
```

```
write(*,"(I4)") 1000000      ! **** ошибка
write(*,"(F5.4)") 123.456    ! ***** ошибка
```

Примеры ввода данных

```
program prog
  integer X, Y
  character(100) str

  write(*, "(A,\) ") "Enter coordinates x,y "
  read(*, "(2I4) ") X, Y
  write(*, "(A,I4) ") "Summa = ", X+Y
                                ! 4^^^5 результат 9
                                ! 4^^^^^^^^^^5 результат 4

  write(*, "(A,\) ") "Path..."
  read(*, "(A) ") str
  write(*, "(A) ") str(1:3)
                                ! Path...C:\DOCUM\1.txt
                                ! результат C:\

end
```

Ввод/вывод

Описатели управления:

- nX** – вывод **n** пробелов;
- SP** – вывод знака "+" в числовых данных;
- SS** – не выводить знак "+";
- S** – восстановление действия дескриптора **SS**;
- Tn** – абсолютная табуляция;
- TRn** – относительная правая табуляция;
- TLn** – относительная левая табуляция;
- BN** – игнорировать пробелы;
- BZ** – интерпретировать пробелы как нули;
- /** – переход на следующую строку;
- ** – не переходить на следующую строку.

Обработка ошибок

`write(*,*,ERR = целочисленная метка) ...`
`read (*,*,ERR = целочисленная метка) ...`

```
program check_error
integer k

read(*,*,ERR = 100) k  ! если введен недопустимый символ

write(*,*) k*1000
stop

100 stop "ERROR"
end
```

Так не делаем! Используем код возврата.

```
read(*,*,iostat=iostat) k  ! если введен недопустимый символ
if (iostat == 0) &
    ! Ошибок не было.
```

Умолчания о типах данных

По умолчанию все объекты программы, имена которых начинаются с букв `i, j, k, l, m, n` или `I, J, K, L, M, N` являются типа `integer`.

Все остальные объекты имеют тип `real`.

Оператор `implicit` изменяет правила умолчания.

`implicit integer (A-B), logical (C-D)`

`implicit none` – все имена должны быть объявлены явно.

Ссылки и адресаты

Ссылка – переменная, связанная с другой переменной, называемой адресатом.

При обращении к ссылке будет происходить обращение к адресату и наоборот.

```
integer, pointer :: p    ! ссылка  
integer, target  :: a    ! адресат
```

Ссылки позволяют создавать динамические структуры данных - списки, стеки, деревья, очереди.

Ссылки и адресаты

Операция => прикрепление ссылки к адресату.

```
program prog
integer, pointer :: p
integer, target :: a
  a = 100
  p => a      ! прикрепили ссылку к адресату
  write(*,*) p

  p = 100     ! a = 100
  a = 500     ! p = 500
end
```

Все изменения, происходящие с адресатом, дублируются в ссылке.

Ссылки и адресаты

Массивные указатели

```
real, pointer :: a_ptr(:)
real, target :: a_trg(5) = [1,2,3,4,5]
a_ptr => a_trg
print*, a_ptr
end
```

Результат

1.000000 2.000000 3.000000 4.000000 5.000000

Ссылки и адресаты

Функция `associated(pt, addr)` возвращает `.TRUE.` если ссылка `pt` прикреплена к адресату `addr`.

```
program prog
integer, pointer :: p1, p2, p3
integer, target :: a,b
  a = 100;  b = 2; p1 => a;  p2 => a
  write(*,*) associated(p1,p2)    ! TRUE
  write(*,*) associated(p1)
  write(*,*) associated(p2,a)
  p1 => b
  write(*,*) associated(p3)        ! FALSE
  write(*,*) associated(p1,p2)
  write(*,*) associated(p1,a)
end
```

Ссылки и адресаты

Оператор **nullify** - открепление ссылки от адресата.

```
program prog
integer, pointer :: p1, p2
integer, target :: a

a = 1000;  p1 => a;  p2 => a
! если к адресату прикреплены две ссылки,
! то отсоединим последнюю

if (associated(p1,p2)) nullify(p2)

write(*,*) associated(p1), associated(p2) ! T, F

end
```

Целочисленные указатели

Целочисленный указатель – переменная целого типа, содержащая адрес некоторой переменной, называемой адресной переменной.

```
real a           ! базируемая переменная
pointer (p,a)    ! p – целочисленный указатель
                 ! на переменную типа real

character ch      ! ch – базируемая переменная
pointer (pc,ch)  ! pc – целочисленный указатель
                 ! на тип character
```

Целочисленный указатель и базируемая переменная используются совместно.

Целочисленный указатель часто используется для обращения к функциям языка С.

Целочисленные указатели

Функция **LOC** вычисляет адрес переменной.

```
program arrow
integer a      ! базируемая переменная
pointer(p,a) ! указатель переменной a на целый тип

integer :: b = 100

p = loc(b)      ! меняем адрес переменной a
a = 500          ! базируемой переменной поместим в b
                ! значение 500

write(*,*) "address = ", p, & ! 5038080
          " value = ", a, & ! 500
          " b      = ", b, & ! 500

end
```