

Программирование

Лекция «Арифметические операции и округление»






Петров Александр Владимирович
Фёдоров Станислав Алексеевич

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

Ранг и типы арифметических операций

- ❑ Если операнды арифметической операции имеют один и тот же тип, то результат операции имеет тот же тип.
- ❑ Если операнды операции имеют различный тип, то результат операции имеет тип операнда наивысшего ранга.

Ранг типов арифметических операндов (в порядке убывания):

`complex(8)` `complex(4)` `real(8)` `real(4)` `integer(4)` `integer(2)`
`integer(1);`     

Примеры (1)

Пример 1

```
integer(2) :: a = 1, b = 2
```

```
real(4) :: c = 2.5
```

```
real(8) d1, d2
```

```
d1 = a / b * c ! 0.0_8
```

```
d2 = a / (b * c) !0.2_8
```

Пример 2

```
integer :: a = 8, b = 3
```

```
real :: c
```

```
c = 2.0**(a / b) ! 4.0
```

```
c = 2.0**(float(a) / float(b)) ! 6.349604
```

Примеры (2)

Пример 3

```
real(4) :: a = 4
```

```
real(8) :: b = 4, x
```

```
x = dsqrt(a)
```

```
x = dsqrt(b)
```

! Ошибка: тип параметра должен быть real(8)

! Верно

Искажение точности (1)

Пример 1

```
real(4) :: a = 1.11
```

```
real(8) :: c
```

```
c = a
```

```
print *, a           ! 1.110000
```

```
print *, c           ! 1.110000014305115
```

Если сразу начать работу с типом `real(8)`, то точность сохраняется.

Пример 2

```
real(8) :: c
```

```
c = 1.11_8           ! или c = 1.11d0
```

```
print *, c           ! 1.1100000000000000
```

Следует:

```
real(R2_) :: c
```

```
c = 1.11_R2_         ! или c = 1.11d0
```

Искажение точности (2)

Искажение может произойти и при переходе к низшей разновидности типа.

Пример 3

```
integer(2) :: k2 = 325
```

```
integer(1) :: k1          ! -128 <= k1 <= 127
```

```
k1 = k2
```

```
print *, k2              ! 325
```

```
print *, k1              ! 69
```

Ошибки округления (1)

При вычислении выражений с вещественными и комплексными операндами возникает ошибка округления.

```
real :: x = 0.1
do
    ! Бесконечный цикл
    print *, x
    x = x + 0.1
    if (x == 1.0) exit
end do
```

Корректное завершение цикла можно обеспечить следующим образом:

```
real :: x = 0.1
do
    print *, x
    x = x + 0.1
    if (abs(x - 1.0) < 1.0e-5) exit ! x практически равен 1.0
end do
! abs(x - 1.0) возвращает |x - 1.0|
```

Нельзя сравнивать вещественные числа на предмет точного равенства или неравенства, а следует выполнять их сравнение с некоторой точностью.

Ошибки округления (2)

Влияние ошибок округления можно снизить, правильно формируя порядок вычисления.

```
real(4) :: x = 1.0e+30, y = -1.0e+30, z = 5.0
```

Сумма равна 5.0. Найдем и выведем сумму следующим образом:

```
print *, x + (y + z)      ! 0.000000e+00
```

Ошибка! Правильной является такая последовательность вычислений:

```
print *, (x + y) + z      ! 5.000000
```


Элементные функции преобразования типов данных (1)

Часто требуется явное преобразование типов, например чтобы избежать целочисленного деления или правильно обратиться к функции.

```
integer :: a = 2, b = 3  
print *, sin(float(a + b))    ! -9.589243e-01
```

`int(a [, kind])` — преобразовывает параметр **a** в целый тип с параметром разновидности **kind** путем отсечения значения **a** в сторону нуля. Если параметр **kind** отсутствует, то результат имеет стандартный целый тип.

Элементные функции преобразования типов данных (2)

Аналогичные преобразования, но с фиксированным типом результата выполняются следующими функциями:

| | | |
|----------------------|-------------------------------------|-------------------------|
| <code>int1(a)</code> | Целый, вещественный или комплексный | <code>integer(1)</code> |
| <code>int2(a)</code> | Целый, вещественный или комплексный | <code>integer(2)</code> |
| <code>int4(a)</code> | Целый, вещественный или комплексный | <code>integer(4)</code> |
| <code>hfix(a)</code> | Целый, вещественный или комплексный | <code>integer(2)</code> |
| <code>jfix(a)</code> | Целый, вещественный или комплексный | <code>integer(4)</code> |

Элементные функции преобразования типов данных (3)

izext(a), **jzext(a)**, **zext(a)** — преобразовывает логические и целые значения в целый тип с большим значением параметра разновидности. Преобразование выполняется путем добавления нулей в свежие биты результата.

izext(a) **logical(1)** , **logical(2)** ,
 integer(1) , **integer(2)**

integer(2)

jzext(a) **logical(1)** , **logical(2)** ,
 logical(4) , **integer(1)** ,
 integer(2) , **integer(4)**

integer(4)

zext(a) **logical(1)** , **logical(2)** ,
 logical(4) , **integer(1)** ,
 integer(2) , **integer(4)**

integer(4)

Элементные функции преобразования типов данных (4)

`real(a [, kind])` – преобразовывает параметр `a` в вещественный тип с параметром разновидности `kind`.

□ Если параметр `kind` отсутствует, то результат имеет вещественный тип по умолчанию.

□ Если параметр `a` комплексного типа, то результат вещественный с параметром разновидности типа `kind`, если `kind` задан, и с тем же параметром разновидности типа, что и `a`, если `kind` опущен.

```
integer :: a = 10  
real(8) s
```

```
s = real(a, 8)      ! привели к типу real(8)
```

`dbble(a)`, `dfloat(a)` – преобразовывает целый, вещественный и комплексный параметр `a` в вещественный тип `real(8)`.

Элементные функции преобразования типов данных

(5)

`cmplx(x [, y] [, kind])` — преобразовывает целые, вещественные или комплексные параметры в комплексный тип с параметром разновидности `kind`. Если параметр `kind` опущен, то результат `complex(4)`.

- ❑ Если `y` задан, то `x` — вещественная часть комплексного результата.
- ❑ Параметр `y` не может быть задан, если `x` комплексного типа.
- ❑ Если `y` задан, то он является мнимой частью комплексного результата.
- ❑ Если `x` и `y` являются массивами, то они должны быть согласованными.

```
complex z1, z2
complex(8) z3
z1 = cmplx(3)           ! Возвращает 3.0 + 0.0i
z2 = cmplx(3, 4)        ! Возвращает 3.0 + 4.0i
z3 = cmplx(3, 4, 8)     ! Возвращает число complex(8) 3.0d0 + 4.0d0i
```

`dcmplx(x [, y])` — выполняет те же преобразования, что и функция `cmplx`, но тип результата всегда `complex(8)`.

Элементные функции преобразования типов данных (6)

logical(L, [, kind]) – преобразовывает логическую величину из одной разновидности в другую. Результат имеет такое же значение, что и **L** и параметр разновидности **kind**.

□ Если **kind** отсутствует, то тип результата **logical**.

Элементные функции преобразования типов данных (7)

`ichar(c)`

`character(1)`

`integer(4)`

`iachar(c)`

`character(1)`

`integer(4)`

`char(i[, kind])`

целый

`character(1)`

`achar(i)`

целый

`character(1)`