Fortran中的数字精度的探究

一。Fortran中的数字类型

Fortran中的数字类型分为整数(Integer)，实数（Real）和复数（Complex）三种类型。复数类型由实部和虚部的两个实数组成，这里我们不多讨论。
通常能够在两种情况下见到数字，其一是标定一变量的类型，Integer 、Real 和Complex 均用于定义其所示类型的变量；其二则是直接作为数字常量出现，如 12、12.008、1.2E5，这种方式在C++中被称之为字面值常量。

1.整数类型

通常在Fortran中所指的整数类型都是具有符号的，可以等同于C++中的int类型，默认情况下写出的整型数字也是这个类型。无符号数作为一种新的数据类型在F95开始被引入，为 UNSIGNED 类型，不在本文的讨论范围内。有兴趣的同学可以参考链接.

1.1 整型变量

使用Integer定义整型变量，和C++的int类型相同用于表示整数类型。在默认的情况下，其定义的数据长度为4字节。在需要的情况下，我们也可以人为指定其数据长度以增加其储存上限。

1.2 整型数字

整形数字常数直接出现在源码中，其也存在默认的长度，默认长度和整型变量一样为4字节。如果给出的数字超出默认的长度，则会出现截断，但这在使用中是无意义的，因此编译器和IDE会报错。

```
write(*,*) 'integer num =',98765432109876543210
```

```
Error: Integer too big for its kind at (1). This check can be disabled with the option '-fno-range
```

通过添加编译参数 -fno-range-check 可以阻止编译器进行整数的上限检查，但是运行后的结果如下，这在实际计算中被视作错误的结果。

```
integer num =  -450461974
```

1.3 整型的表示范围

INTEGER类型根据其类型长度不同具有不同的数据表示范围，在Fortran中我们可以使用内置的huge函数得到某一整数的表示范围上限，即能表达的最大的数。

[sample code 使用huge函数可以得到integer类型]

```
    integer(kind = 1) ii
    write(*,*) 'integer huge=',huge(ii),huge(55455)
```

实际上，整数(INTEGER类型)的表示范围和其所占字节数的关系如下，令k为所占字节数：

$$[-2^{8k-1}, 2^{8k-1}-1]$$

1.4 人为指定整型变量和数据的空间大小

实际使用中，我们可以根据自身需求来灵活调整整形数据的大小，Fortran90之后可以通过设定种别（KIND）来修改变量的空间占用大小。对于Fortran中的整型数字常量，我们也可以在其后显式追加KIND后缀来人为指定kind数目，即直接加下划线再加KIND值。在下面的代码中我们将整数类型的变量声明为不同KIND的数目，对整型常数也给定种别，然后可以观察他们的表示范围上限。

```
INTEGER i ! 默认kind = 4
integer(kind = 1) ii !亦可写作 INTEGER*4 ii
integer(kind = 2) iii
integer(kind = 4) iiii
integer(kind = 8) iiiii
write(*,*) 'integer huge=',huge(i),huge(ii),huge(iii),huge(iiii),huge(iiiii),huge(55455),huge(5545
```

预期的结果应该如下所示：

```
 integer kind=  4       1           2           4           8           4           8
 integer huge=  2147483647  127  32767  2147483647  9223372036854775807  2147483647  9223372036854
```

关于FORTRAN-KIND，也请参考。

2.实数类型（浮点数类型）

在Fortran中，实数类型作为浮点数储存，和C++中的浮点数（float和double）类似。 // reference

2.1 浮点类型的表示方法

和C++相同，Fortran也使用业界统一的 IEEE-754标准,微软官方的C++文档也有相关可参考描述。通常根据所占据的字节数不同，可以分为单精度浮点数Single（在Fortran为REAL*4*，*C/C++中的float*）、*双精度浮点数Double（在Fortran中为REAL*8，C/C++中的double）。另外还有半精度（2字节）、四元精度（16字节）和双扩展精度浮点数Double-Extended（10 字节），这些格式作为可选实现，在某些编译器和某些语言中被实现为数据类型，不做过多讨论。

其中单精度浮点数使用四字节储存，即32位，其中包含一位符号位，8位指数位，23位有效数位；双精度浮点数使用八字节储存，即64位，其中包含一位符号位，11位指数位，52位有效数位。

详细的浮点数表示方式可以参考以上的参考文档，或者实机测试。

2.2 浮点数变量

对于双精度来说，早期 Fortran 提供了 Double precision 关键字来定义它。就新语法来说，我们不再建议使用这个关键字。而统一使用REAL类型和 Kind 进行定义。例如：

```
! 老式的写法
Real(4) :: rV1
Real(8) :: rV2
Double precision :: rrV3 , rrV4
! 新写法
Real( Kind = 4 ) :: rV1  , rV2 !// 单精度变量
Real( Kind = 8 ) :: rrV3 , rrV4 !// 双精度变量
REAL*8 :: a ! 和Real(Kind=8)等价
REAL b  ! 默认KIND=4 即单精度和C++ float等价
```

2.3 浮点数常数

同样的，在代码中直接出现的字面值形式的实数数字，即被视作浮点数常数，例如“3.1415926535”等，在默认情况下，Fortran将其的精度视作单精度，换言之，在代码中直接出现的数字，在被编译器接受并储存的时候会被作为单精度数据储存为常数。下面的示例代码中可以看出来这一点。

```
real(kind=8) a
a = 1.1234567890123456789 !default kind=4
write(*,*) 'real',a,1.1234567890123456789,1.1234567890123456789_8
```

在默认gfortran下，程序运行的预期的结果如下，第二个数据为默认浮点常数输出，第三个为人为限定kind=8。可以看到前两个数字输出都出现了精度损失，但第一个的输出位数确能达到KIND=8的要求。

```
real 1.1234568357467651 1.12345684 1.1234567890123457
```

对比三个数字的输出可以发现，其实第一个数字和第二个数字一样，都是在解析浮点常数时出现的精度丢失的。在这个过程中，虽然赋值操作和输出操作支持的精度为双精度（KIND=8），但由于浮点常数从代码源文件进入内存时即只有单精度，实际的整个操作下来有效精度只有单精度（KIND=4），这才导致我们看到精度损失。因此如果对赋值操作有精度需求，需要对等号左右人为给定合适的KIND。

二。常用的种别函数介绍

在FORTRAN90以后，提供了一些内部函数进行种别KIND相关的操作。

KIND(X)：函数KIND用于查询变量的种别，它返回X的种别值，X可以为变量和常量均可。如：KIND(0)返回值是整型的标准种别值，KIND(0.)、KIND(.FALSE.)、KIND(“A”)分别返回实型、逻辑型、字符型的标准种别值。

SELECTED_REAL_KIND([n],[m])：该函数返回实型变量（浮点数变量）对所取的值范围和精度恰当的种别值。其中n是指明十进制有效位的位数，m指明值范围内以10为底的幂次。例如：SELECTED_REAL_KIND(6,70)的返回值为8，表示一个能表达6位精度、值范围在－1070—＋1070之间实型数的种别值为8。硬件不满足时会得到异常返回值：-1(当精度位数达不到时)，-2(当数值范围达不到时)，-3(两者都达不到时)。

对于浮点数X，它的精度和范围也可通过内部函数PRECISION(X)和RANGE(X)查出。

SELECTED_INT_KIND([m])：该函数返回整型变量对所取的值范围恰当的种别值。m同样为以10为底数的幂次。

通常我们推荐对变量和常数做适当的KIND限定，尤其是对于需要高精度的数据，应该给定必要的KIND限制，以免出现不必要的精度损失。基于以上的种别函数，我们可以更好的指定KIND，通常为了保持程序的跨平台性能，可以将KIND作为参数给定便于修改。

三。通过编译器参数控制数据长度

实际使用中，不同的硬件设备和不同的编译环境，可能具有不同的数据长度（KIND）的默认值，或者存在某种需求，需要修改代码中未显式声明种别的数据，则可以通过编译器参数来进行修改。

1.调整默认数据长度

在GNU的gfortran中，可以使用如下的参数对数据进行控制。

- -fno-range-check
  此参数会禁止编译器对编译中的常数表达式的范围进行检查。例如遇到$\frac{1}{0}$时，如果不进行检查将会对结果赋值为+Infinity，对于超过HUGE的常数表达式也不会进行错误提示，而是直接进行溢出处理，例如代码 `write(*,*) -129_1` 将输出127.
- -fdefault-integer-8
  此参数将设置默认的整型数据INTEGER和逻辑型数据为8字节的宽度，对于没有人为限定KIND的整数常数也生效，但是不会对已经有KIND限定的整型变量和数据生效。
- -fdefault-real-8
  此参数将会把浮点数的默认数据占用设定为8字节。对于没有人为限定的浮点数常数也生效。但是，这个参数也会将FORTRAN的DOUBLE PRECISION 类型定义为16字节（在旧的FORTRAN中可以通过此类型来声明双精度浮点）。除非同时使用 -fdefault-double-8 参数。本参数不对已经有KIND限定的数据生效。
- -fdefault-double-8
  此参数会将FORTRAN中的DOUBLE PRECISION 的默认类型设置为8字节，由于通常我们以REAL来声明，因此使用到此DOUBLE PRECISION的情况多出现在老旧的代码中，建议更换为REAL并使用kind限定以达到表示双精度浮点数的目的。此参数可以配合-fdefault-real-8参数使用以阻止DOUBLE PRECISION类型的宽度提升。当然，对已经限定种别的数据此参数亦不生效。

关于以上的参数有一个很危险的场景：

```
real(kind=8) a
a = 1.12345678901234567890
write(*,*) 'dsign',dsign(a,a)
```

以上代码使用 gfortran 默认参数可以直接编译，只是由于浮点数常数的默认KIND为4，因此在赋

值给变量a的时候会出现精度损失，这个在上文已经提到，通常我们通过人为限定种别的情况下来避免精度损失。但是对于既有的代码直接修改可能比较繁琐，可以尝试使用参数 -fdefault-real-8 对无种别限制的浮点数指定KIND。但实际编译中，如果添加参数-fdefault-real-8 则会出现以下的报错。

```
   43 |     write(*,*) 'dsign',dsign(a,a)
      |                            1
Error: ‘a’ argument of ‘dsign’ intrinsic at (1) must be double precision
```

这个问题咋一看非常不合常理，针对double类型特化的函数dsign为什么对a的类型不能接受？但是经过试验之后才发现，其实dsign所谓的针对double其实是针对的Fortran内置的DOUBLE PRECISION类型，而不是固定C++的double或者IEEE的双精度浮点数（double），因此这个时候dsign所需要的参数的类型为DOUBLE PRECISION类型，亦或者使用REAL(KIND=16)的数据也能够满足要求。
因此，针对以上的情况，如果确实需要使用-fdefault-real-8 进行浮点数的默认种别的指定，那么搭配参数-fdefault-double-8 可以使得dsign函数能够如原预期一样运行。当然，也可以使用其他的方法完成所需的目的。

从以上的情况，我们也能够得到经验，在编码过程中，为保证程序的正确性和安全性，对于有特定精度需求的数据，应该尽量人为给定种别，除非你能够接受其被编译器自动限定为某些规定中的种别。对于内置的函数，除非确定自己所需的类型，否则应尽量使用通用类型的函数。

2.强制调整数据长度

gfortran也提供了一些参数用于强行修改特定类型的kind，在某些极其特定的情况下会被使用，通常不需要在意，如：

- -finteger-4-integer-8 此参数将强行将代码中的所有INTEGER(KIND=4)的变量强行转化为INTEGER(KIND=8)，如果转换失败则报错。
- -freal-4-real-8
- -freal-4-real-10
- -freal-4-real-16
- -freal-8-real-4
- -freal-8-real-10
- -freal-8-real-16

更多的参数请参考GFortran编译器的说明文档。

Reference

oracle对usigned整数的介绍

oracle对于Fortran浮点数的介绍

IEEE-754 浮点数标准,

微软官方的C++文档的浮点数介绍

gfortran中对KIND的说明

GNU-gfortran-文档