Introduction to Fortran 90, Part II

Brian T. Smith, UNM

Andrew C. Pineda, UNM

CS 471/Math 471

Source Forms

- Source form is the form (set of rules) to write a program
- There are two source forms
  - fixed source form
    - Fortran 77 (card) form with a few extensions
    - line limit is 72 chars. (statements start in 7th col)
  - free source form
    - a form with no artificial boundaries
    - line limit is 132 characters

Free Source Form

- Upper and lower case are permitted and are indistinguishable in source text except when in character strings, comments, and I/O
  - Smith is the same as SMITH and sMiTh
  - no blanks allowed within a token (statement keyword, identifier, multi-character delimiter)

Free Source Form Cont'd

- Every statement begins with a statement keyword except an assignment statement

```
REAL  X
CALL  FOO
 print *, ...
 end
```


Free Source Form Cont'd

- Identifiers begin with a letter and can have up to 31 letters, digits, underscores

```
X
FOO_BAR
Internet2
One word  -- invalid
__This_is_invalid
2two -- invalid
```


Free Source Form Cont'd

- Statements can begin in any column and can be up to 132 characters long without continuation
- Continuation is indicated with an & at the end of the line to be continued
  - up to 39 continuation lines are allowed
- Comments beginning with ! in any position
  - may be at the end of a statement (or part)

Source Form Examples

- Continued line:
  - Valid continuation
    - print *, ‘This is a long line’, a, b, & ! Valid
      c, d, e
  - Invalid continuation
    - print *, “This is wrong”, lon & ! Invalid
    - g_variable_name, another_ & ! Invalid
      long_identifier_name

Parts of a Fortran Program

- A Fortran program consists of a main program, and zero or more subprograms.
- Each program unit consists of:
  - program unit header
  - specification part
  - execution part
  - internal part
  - end program unit statement

Program Header

- Marks the beginning of the unit
- Specifies the name of the unit and any objects, known as arguments, that are available for use (input or output) from the calling unit

Specification Part

- Specifies the data objects and other entities that are manipulated by the program unit
- Specifies the data environment in which the computation is performed by the executable part -- that is, the attributes of the data objects, objects made accessible to the program unit

Data

- Data consists of:
  - type
  - values
  - forms for constants
  - operators to manipulate data
- Types -- intrinsic or user-defined
  - intrinsic (all parameterized)
    - integer, real, complex, character, logical

Parameterized Types

- What is it?
  - a parameter (integer) in the type declaration that specifies “flavors” of a particular intrinsic type
    - single precision and double precision are both real types but just different sizes
  - the parameter distinguishes “sizes” of the type
    - short (16 bits) and long (32 bits) integers

Why Parameterized Types?

- Support available "flavors" of types as have occurred in practice
  - different sized integers (8, 16, 32, and 64 bit)
  - different sized reals (32, 64, and 128 bit)
  - different sized logicals (8, 16, and 32 bit)
  - different sized complexes (64, 128, and 256 bit)
  - different sized characters (8 and 16 bit)

Integer Type

- Integer type specification statements:

```
integer  I, J
integer  ::  K
integer, dimension(10,10) :: P
integer( selected_int_kind( 6 ) )  Q
```

  - :: is required when there is more than one attribute and optional otherwise
  - above, Q is of type integer and of a kind that supports integers as large as 10^6

Logical Type

- Logical type specification statements

```
logical  L
logical(2)  LL
```

  - L is default logical
  - LL is logical of kind 2, usually 2 bytes long (IBM xlf)

Character Type

- Character type specification statements

```
character(24)   C
character  A_SINGLE_CHAR
character(3,2)  KANJI_STR
```

  - C is a string 24 characters long of default character
  - A_SINGLE_CHARACTER is 1 character long
  - KANJI_STR is a string 3 characters long of kind 2

REAL Type

- Real type specification statements

```
REAL  R
real( selected_real_kind(10,100) )  D
double precision, dimension(3) ::  DD
```

  - R is of default real type
  - D is of a real type with a kind that supports at least 10 decimal digits of precision and a range of 10^{100}
  - DD is a double precision 1-D array of size 3

Complex Type

- Complex specification statements
  - a pair of reals (real and imaginary parts)

```
complex  C
complex( kind(D) )  CD
```

  - C is of type default complex
  - CD is of type complex of the same kind as D

Integer Values

- Zero, positive, negative whole numbers
- Ordered, but a maximum in practice
  - 3, -3, 123456789, ...
- Fortran form of integers
  - 3, -3, 123456789, 123456789123_8
  - _8 specifies kind 8 for the constant
- Integer model for all representation values
  - sign, magnitude, radix, maximum (min=-max)

Integer Model

- For a given kind of integers, find integers q and r such that any (most) representable integers i can be represented as:

$$i = s\sum_{k=0}^{q-1} d_k r^k$$

where s is ± 1 and $0 \le d_k < r$.

Integer Model Cont'd

- Once q and r are chosen, every integer of the form:

$$i = s\sum_{k=0}^{q-1} d_k r^k, s = \pm 1, 0 \le d_k < r$$

must be representable with this kind.
- For Intel 80486 PC using the Salford Fortran 90 compiler, q is 31 and r is 2 for the default integer type.(7 and 2 for kind=1)

Inquiries for Integer Model

- Intrinsic **digits**(integer) returns q for integer
- Intrinsic **radix**(integer) returns r for integer
- Intrinsic **huge**(integer) returns $r^q - 1$

Logical Values

- Only two values -- all are representable
- Values are true and false -- they are NOT ordered
  - e.g. true, false, 0, 1, T, F
- Form of values
  - .true., .true._1, .false., .false._4
  - _1 and _4 are kind values for the logical values

Character Values

- Character (and strings of characters)
  - letters, digits, underscore, special characters
    - e.g. 0, A, a, _, (, ., &, !
  - they are ordered lexically by a processor-defined collating sequence
  - Fortran's form of character values and strings
    - e.g. '0', 'A', "a", "abc", 'ABC', 2_'~^alpha', ...
    - the kind number and underscore precedes the quoted character string

Real Values

- Mathematical real values (zero, neg., pos.)
  - e.g., 1.0, -0.0, π, *e*, 10^4, 1/3 (one-third)
- They are ordered
- Fortran's form for real values
  - e.g., 1.0, -0.0e0, 1.0d4, 1.1_8
  - 1.1_8 is real of kind value 8
  - no form for π, *e*, one-third
- most real numbers are not representable without approximation - 0.1 = .000110011...(2)

Real Model

- For a given kind of real, find integers *q*, *b*, $e_{\min}$, and $e_{\max}$ such that any (most) representable real numbers *r* can be represented as:

$$r = 0, \quad or \quad r = sb^e \sum_{k=1}^{q} d_k b^{-k}$$

$$where \quad s = \pm 1, \quad e_{\min} \le e \le e_{\max}, \quad 0 \le d_k < b, \quad d_1 \ne 0$$

Real Model Cont'd

- Once *q*, *b*, $e_{\min}$, and $e_{\max}$ are chosen, every number of the form:

$$r = 0, \quad or \quad r = sb^e \sum_{k=1}^{q} d_k b^{-k}$$

$$where \quad s = \pm 1, \quad e_{\min} \le e \le e_{\max}, \quad 0 \le d_k < b, \quad d_1 \ne 0$$

must be representable in this type of real and kind.

Real Model Cont'd

- On an Intel 486 PC using the Salford f90 compiler, the values of these parameters for the default real kind are 24, 2, -125, and 128 (*q*, *b*, $e_{\min}$, and $e_{\max}$)
- For double precision on the same system, the values are 53, 2, -1021, and 1024 (*q*, *b*, $e_{\min}$, and $e_{\max}$)

Inquiries for the Real Model

- Intrinsic **digits**(real) returns q
- Intrinsic **radix**(real) returns b
- Intrinsic **minexponent**(real) returns $e_{\min}$
- Intrinsic **maxexponent**(real) returns $e_{\max}$
- Intrinsic **huge**(real) return (1-b(-p))*b^emax
- Intrinsic **tiny**(real) returns b^(emin-1)
- Intrinsic **precision**(real) returns the dec. prec

Complex Values

- Mathematical complex values
  - e.g., 1.2, -i, 0.1 + 1.1i
- They are unordered, but can be compared for equality and inequality
- Fortran's form for complex values
  - e.g., (1.2,0.0), (0,-1), (0.1d0,1.1d0), (1.1_8, -1.1_8)
  - the last complex constant has kind value 8
- Most are not representable without approx.

Operators and Operations

- For integer, real, and complex types:
  - unary: +, -
  - binary: +, -, *, /, **
- For logical type
  - unary: .not.
  - binary: .or., .and., .eqv., .neqv.
- Comparison of logicals -- use .eqv. and .neqv.

Operators Cont'd

- Character type
  - binary: //
- Relational operators for all types that are ordered (real, integer, character)
  - binary: >, >=, <, <=, ==, /=
  - synonyms: .gt., .ge., .lt., .le., .eq., .ne.
- Complex can use = = and /=

Kinds of Intrinsic Types

- All intrinsic types have "flavors" with different representations
  - 16-bit integers, 32-bit integers
  - single precision (float in C), double precision
- Within a type, the different flavors are designated by different kind numbers, but the kind numbers are not standard

Real Kind Numbers

- IBM xlf (or xlf90) compiler
  - real(4) R
  - real(8) D
  - double precision DD ! Same type, kind as D
- NAG f90 compiler
  - real(1) R
  - real(2) D
  - double precision DD ! Same type, kind as D

User-Defined Types

- The user can define his/her own type consisting of an arbitrary collection of components (fields) of intrinsic, the same, or other user-defined types
- Often call derived-types
- Useful to collect data together of nonhomogeneous types
- A means of language extension: quaterions

Example of Derived Type

```
type RATIONAL
  integer NUMER, DENOM
end type RATIONAL
```

- Represents rational numbers NUMER/DENOM where NUMER and DENOM are integers
- Declare a variable of type RATIONAL
  - type(RATIONAL) R

Access Components

- Access components using the qualifier %
  - R % NUMER   ! Numerator of R
  - R % DENOM   ! Denominator of R
- Existing and new operators can be defined for a user-defined type
  - e.g. the operator + with rational operands can add two rational numbers together
  - e.g. the operator .multiply. does multiplication

Derived-Types Cont'd

- Essentially, everything you can do with objects of intrinsic type you can do with a user-defined type, provided operator definitions are provided

Derived-Type Assignment

- Assignment between data objects of the same derived types is defined by default
  - for non-pointer components, component-by-component
    - may be scalar or array assignment
    - defined by intrinsic assignments
  - for pointer components, uses pointer assignment

DT Assignment Cont'd

- The default assignment may be overridden
  - uses a generic assignment interface
  - references a subroutine to provide the definition
    - as a module procedure

```
interface  assignment(=)
   module procedure  <list_of_names>, …
end interface
```


DT Assignment Cont'd

- As an external procedure

```
interface  assignment(=)
   subroutine  <name>( L, R )
      <type>, intent(in) :: R
      <type>, intent(out) :: L
   end subroutine
      …
 end interface
```


Constructing Derived-Type Values

- The type definition creates a derived-type constructor of the same name as the type
- With the RATIONAL type, the value rational(2,3) creates the rational number 2/3
- The constructor is the name followed by a parenthesized list of values of the same type as the components (fields)

Exercise

- On the xlf compiler, print the following values (values from intrinsic functions) for various kinds for each of the intrinsic types:

Intrinsic	Type	Kind Numbers
digits	integer	1, 2, 4, 8
	real	4, 8, 16

Exercise Cont'd

- Repeat the above for the functions **range, huge,** and **radix**
- Repeat for reals only using the functions **digits, radix, range, precision, minexponent, maxexponent, epsilon, huge**, and **tiny**

A "Real" Example Program

```
program  MY_SQRT_PROGRAM

   ! This program computes an approximation to the square
   ! root of a real number read from standard input and
   ! prints the result to standard output.  The program is
   ! written in Fortran 90 following the free source form
   ! specified by the Fortran 90 standard.
```

- Make sure there are no identifiers declared implicitly.

```
implicit  none
```


A "Real" Example Cont'd

- Declare the names used as intrinsic procedures (not necessary to do this).

```
intrinsic  abs, epsilon, selected_real_kind, sqrt
```

- Determine the kind number WP for a real type that has at least 6 decimal digits of accuracy.

```
integer, parameter :: WP = selected_real_kind( 6 )
```


A "Real" Example Cont'd

- Declare various data objects:

```
!  A        the value read for which the approximate
!           square root is to be computed.
!  X        the current iterate (or approximation) to
!           the square root of A
! PREV_X    the previous iterate
! FUDGE  a fudge factor to make sure that iteration
!           terminates.
real(WP)  A, PREV_X, X
real(WP), parameter :: FUDGE = 2.0_WP
```


A "Real" Example Cont'd

- Read in the value of A.

```
read *, A
```

- If A is negative, the result is complex; do not try to compute the approx. square root.

```
if( A > 0.0_WP )  then
   ...
elseif( A == 0.0_WP )  then
   ...
else
   ...
endif
```

A “Real” Example Cont’d

- A is positive; approximate the square root. Initialize the iteration process. The method used is Newton's method $x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$

where *f* is the polynomial $x^2 - a$.

```
X = A
PREV_X = 0.5_WP*X
```


A “Real” Example Cont’d

- Perform the Newton iteration; stop when two consecutive iterates are close enough.

```
do while( abs( X - PREV_X ) > &
          FUDGE*abs(X)*epsilon(X) )
   PREV_X = X
   X = 0.5_WP*(X + A/X)
end do
```


Exceptional Cases

- Handling the exceptional cases:

```
elseif( A == 0.0_WP )  then
  ! The square root of zero is zero.
  X = 0.0_WP
else
  print *, 'The square root of a negative number &
           & complex and this', &
            'program does not compute it.  &
            & Terminate execution.'
   stop
 endif
```


End Of The Program

- Print the value of the approximate square root and the vendor's intrinsic square root. How close are they?

```
print *, 'My approximate square root of ', A, ' is ', X
print *, "The vendor's approximate square root is ",&
      sqrt(A)
```

- A main program stops when an END statement is encountered.

```
end program MY_SQRT_PROGRAM
```