Version 2015-1
as of 15 Sep 2015

Introduction to Scientific Programming

Data Types

THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

Intrinsic Data Types

- Numeric Data Types
  - **`integer`**
  - **`real`**
  - **`complex`**

- Non-numeric
  - **`character`**
  - **`logical`**

Intrinsic Data Types

- type-spec [attribute-spec] :: entity-declaration list
- *Short:* Type [Optional attributes] :: Variables

```
integer[ kind selector ]              kind selector
real[ kind selector ]                 kind=<kind-value>
complex [ length selector ]
logical[ length selector ]            length selector
character[ length selector ]          len=<length-value>
```

– Attributes: *parameter, allocatable, dimension, intent, optional, save, pointer, target*

`implicit none` - a digression

- If **`implicit none`** is not specified in a program and/or a subprogram block (subroutine, function)
  - variables with names that begin with the letters **i-n** are **`integer`** by default
  - variables with names that begin with **a-h** or **o-z** are of type **`real`**
- Using **`implicit none`** is advisable (at least for beginners)
  - misspelled variable names will be detected by the compiler
  - Use **`implicit none`** in all your code/homework: programs! *(and functions and subroutines)*
  - All integers in examples will start with letters **i-l**

Intrinsic Data Types - `integer`

```
integer                                  :: i    ! Default 4 bytes
integer(kind=4)                          :: j    ! Explicitly 4 bytes
integer(8)                               :: k    ! Explicitly 8 bytes

integer, parameter                :: lng=selected_int_kind(16)
integer(kind=lng)                 :: l

i = 5; j = 6; k = 7_8; l = 2_lng

print *, huge(i), huge(j)      ! huge() is a build-in function and
                   ! returns the largest value of the                    !
argument type
```

Output:

```
 2147483647    9223372036854775807
```

- **`selected_int_kind(n)`** returns the kind value needed to specify precision to **`n`** decimal places
- Good for portability
- Constants can be defined to arbitrary precision, e.g., **`2_lng`**

Intrinsic Data Types - `real`

- 32-bit by default, with a range [10^{-38},10^{38}]; 64-bit [10^{-308},10^{308}] (and negative!)

```
real                                  :: x    ! Default 4 bytes
real(4)                               :: y    ! Explicitly 4 bytes
real(8)                               :: z    ! Explicitly 8 bytes

integer, parameter             :: db=selected_real_kind(12,99)
real(kind=db)                  :: r

x = 5.; y = 6.; z = 7._8; r = 2_db    ! Multiple statements in one line
print *, huge(y), tiny(y)      ! tiny() returns smallest number
print *, huge(z), tiny(z)
```

Output:
```
 3.4028235E+38  1.1754944E-38
 1.797693134862316E+308  2.225073858507201E-308
```

- **selected_real_kind(n,m)** returns the kind value needed to specify precision to **n** decimal places and exponent up to **m**
- Good for portability
- Constants can be defined to arbitrary precision, e.g., **2_db**

Intrinsic Data Types - `complex`

- 32-bit by default, same as default real

```
complex :: i, pt
...
i  = (4.0, 1.0) ! Real part, imaginary part
pt = 4.0        ! What does this do?

print *, 'i  = ', i
print *, 'pt = ', pt
```

Output:

```
 i  =  (4.000000,1.000000)
 pt = (4.000000,0.0000000E+00)
```

Intrinsic Data Types - `character` (String)

- Length specifiers
- String can have a length of zero

```
character(len=10)  :: first, last ! String of max length 10
character(len=20)      :: full        ! String of max length 20

first = ''      ! String with no content '          '
first = 'John'      ! 4 letters + 6 trailing blanks 'John      '
last  = 'Doe'
full  = first           ! Assignment
full  = first // last ! Assignment with concatenation
print *, full
full  = trim(first) // ' ' // trim(last) ! trim() cuts off trailing print
*, full                 ! blanks
                        ! // concatenates strings
```

Output:

```
John      Doe
John Doe
```

Intrinsic Data Types - `logical` (Boolean)

- Can assume one of two values: `.true.` or `.false.`
- Can be assigned an expression. Its value is the same as the truth value of the expression

```
logical                 :: b1, b2, b3
logical, parameter              :: always_true = .true.

b1 = .true.
b2 = .false.
b3 = b1 .and. b2                ! Relational operator .and.
print *, 'b1, b2, b3 = ', b1, b2, b3
b3 = b2 .or. always_true        ! Relational operator .or.
print *, 'b1, b2, b3 = ', b1, b2, b3
```

Output:

```
 b1, b2, b3 = T F F
 b1, b2, b3 = T F T
```

In Class Lab

Write a program that accepts three numbers, (a, b, and c) from the keyboard and your name (name)

- The program will the say hello to you i.e. “Hello, Jim”
- It will then calculate the volume of a sphere with a being the radius. V = (4/3) * pi * a^3 (NOTE: the 2 *’s are used for exponent, i.e. a^3 would be a**3.0)
- Calculate the volume of a cube with ‘a’ being the length, ‘b’ being the height, and ‘c’ being the width.
- BONUS: multiply a, b, and c. Divide by 7. Take your answer and convert it to an integer.

We gratefully acknowledge the sponsorship of Chevron Corporation, whose generous support of TACC has made possible this Scientific Computing Curriculum and other student-focused initiatives.

License