Variables, Expressions, Conditions, and All That

Variables

- Variables are placeholders for *locations in memory.*
  - Variables always have a *type* even if you don't have to declare it
  - The *primitive types* correspond (more or less) to the types defined in hardware, specifically integers, floating-point (single and double), and characters.
  - Many languages define more basic types including Booleans, strings, complex numbers, and so forth.

Types

- The computer has distinct **types** that are internally quite distinct. Each type has a set of **operators** defined on it.
  - Implicit typing
    - Historical, DO NOT use in new code
    - By default, variables declared beginning with letters I-N are integers; all others are reals (single precision floating point)
  - Explicit typing
    - Variables are declared by type

Numeric Types: Integers

- Integer
  - Quantities with no fractional part
  - Represented by sign bit + value in *binary*
    - *Computers do not use base 10 internally*
    - Default integers are of size 32 bits
  - Maximum integer is is 2^{32}-1 (signed)
  - Compiler extension (in most compilers)
    - INTEGER*8 (old declaration style) is a 64-bit integer
    - Will show another method when we learn about KIND

Floating Point

- Floating point single precision
  - Called REAL in Fortran
  - Sign, exponent, mantissa
  - 32 bits in nearly all languages
  - IEEE 754 defines representation and operations
  - Approximately 6-7 decimal digits of precision, *approximate* exponent range is 10^{-126} to 10^{127}

Double Precision

- Double precision floating point
  - Sign, exponent, mantissa
  - 64 bits
    - Number of bits NOT a function of the OS type! It is specified by the IEEE 754 standard!
  - Approximately 12-13 decimal digits of precision, approximate exponential range 10^{-1022} to 10^{1023}
  - In Fortran the default literal is single precision. Double precision literals *must* include a d/D exponent indicator.

Complex

- A complex number consists of 2 reals enclosed in parentheses
  - z=(r,i)
  - Most compilers provide the
  - COMPLEX DOUBLE PRECISION extension as a variable type

Logical

- Values can be .true. or .false. (periods required)
  - Are not necessarily represented by integers; internal representation is up to the compiler

Non-Numeric Types: Character

- Character
  - 1 byte (8 bits) per single character
- A character has a fixed length that must be declared at compile time

```
character(len=8) :: mychar
```

- In subprograms a character of unspecified length may be passed

```
character(len=*) :: dummy
```

Variables and Literals

- Literals aka constants
  - Specified values e.g.

```
3
3.2
3.213d0 (Fortran double precision)
“This is a string”
.true.
(1.2,3.5) (Fortran complex)
```

- Variables
  - Have a type but the value must be assigned
  - Variables are assigned *locations in memory* by the compiler or interpreter

Type Conversions

- If a variable is of one type but it needs to be of a different type, it is necessary to do a *type conversion* aka a *cast*.
- An expression with more than one numeric type is said to be *mixed*.

```
N=20*3.5/11.d0
```

- Most compilers will automatically cast numeric variables to make mixed expressions consistent. The variables are promoted according to their rank. Lowest to highest the types are integer, float, double, complex.
- Almost no languages can or will automatically cast non-numeric types to numerics.

Type Conversions (Continued)

- Explicit casting among numeric types

```
R=real(I)
I=int(R)
Z=cmplx(r1,r2)
```

Non-Numeric ⇔ Numeric

- Fortran has a peculiar way to do this called internal read/write
- Convert numeric to character

```
character(len=4)  :: age
integer           :: iage
 iage=39
 write(age,’(i4)’) iage
```

- Convert character to numeric

```
 age=‘51’
 read (age,’(i4)’) iage
```

Fortran Declarations

- First statement should be

```
PROGRAM myname
```

- Then follow it immediately with

```
IMPLICIT NONE
```

Declare variables with syntax

```
INTEGER                :: I, J
REAL                   :: R, S, T
DOUBLE PRECISION       :: D
DOUBLE COMPLEX         :: Z
LOGICAL                :: FLAG
CHARACTER (len=20)     :: C
```

- All caps are not required but I use them to emphasize the keywords.

Arithmetic Operators

- Operators defined on integers, floats, and doubles
- + - add subtract
- * / multiply divide
- ** exponentiation
- Operator Precedence is:
- ** (* /) (+ -)
- All languages evaluate left to right by precedence unless told otherwise with parentheses

Integer Operators

- Fortran: 2/3 is always zero! Why?
  - Because 2 and 3 are integers so / is an integer operation that yields an integer result
  - Remainder comes from mod(a,r) or modulo(a,r)
    - mod and modulo are NOT THE SAME for negative numbers

Logical/Boolean Operators

- Negation
  - .not.

    .not. flag
- AND
  - .and.
- OR
  - .or.

NonNumeric Operators

- Strings/Characters
  - There are many (some of which require function calls)
  - Fortran
    - Concatenation //
  - Substring extraction
  - S(1:3) First character is counted as 1 and the last one in the substring is the upper bound. This expression extracts characters 1 to 3. Fortran counts from 1.

Conditional Operators

- Conditional operators represent *relationships*. They can be defined on any type. Most commonly we use numerical conditional operators.
- These compare two numerical values for equality, non-equality, greater than, less than, greater than or equal to, less than or equal to.

Comparison Operators

- Numeric
  - Fortran has two sets, one with letters and one with symbols. Note that /= has a / for "not"
    - .eq. ==
    - .ne. /=
    - .lt. <   .gt. >   .le. <=   .ge. >=

Expressions

- Expressions are combinations of variables, literals, and operators and/or functions that can be evaluated to yield a value of one of the legal types.
- Examples

```
a+3*c
sqrt(abs(a-b))
A .or. B
```

Conditional Expressions

- Conditional expressions evaluate to *true* or *false*.

- `x > 2.0`
- `y .gt. 0.0 and y .lt. 1.0`
- `N == 0 .or. N .eq. 1`

Conditional Operator Precedence

- Like arithmetic operators, conditionals have a precedence. This may be somewhat language dependent but an example might be:
- greater/less outrank equal
- equal outranks and
- and outranks or

Statements

- A statement is the “sentence” of the language. It contains one complete instruction.
- Fortran peculiarity: statements are *executable* or *non-executable*. Non-executable statements are instructions to the compiler (variable declarations, interfaces, etc.) Executable statements perform some action. All non-executable statements must precede the first executable statements in a program unit.

Comments

- Fortran
  - Old style: c or C in first column, entire line
  - New style (free format) !
    - Anything to the right of ! is ignored to the end of the line

Making Choices

- Computer programs really can't do that many things. They can
  - Assign values to variables (memory)
  - Make decisions based on comparisons
  - Repeat a sequence of instructions over and over
  - Call subprograms
- Decisions are one of the fundamental programming constructs

Conditionals Cause Branching

- IF (comparison operation evaluating to Boolean) do something ELSE do something else
- IF (comparison) do something ELSE IF (comparison) do some other thing ELSE default behavior
- WARNING: In nearly all languages the comparison *short circuits*, i.e. once it determines T or F of the comparison it does not do any more evaluations. Don't rely on a compound comparison operation to evaluate a function or set a variable.

Fortran Syntax

elseif and else are optional

```
if ( comparison ) then
     code
elseif ( comparison) then
    more code
else
     yet more code
endif
```

Fortran SELECT CASE

- Many else ifs can become confusing.

```
SELECT CASE (expression)
      CASE(:value0)   ! Expression <= value0
            code
      CASE(value1)
            code
      CASE(value2)
            code
      CASE(value3:)    ! Expression >=value3
            code
      CASE DEFAULT   ! Optional
            code
END SELECT
```