COMP 208
Computers in Engineering

Lecture 04

Jun Wang
School of Computer Science
McGill University

Fall 2007

Review

- Fortran is case insensitive
  - it is a convention to use all capital letters for keywords
- Comments
  - from ! to end of line
- Output with the WRITE statement

```
WRITE (*,*) 2007
WRITE (*,*) "Hello World!", 2007
```

- Variable declaration
  - variables are symbols referring to data stored in memory
  - variable must be declared before its use

```
type-specifier :: list-of-names
```

```
INTEGER :: month, year
```

Review

- FORTRAN has 5 intrinsic (built-in) data types:
  - `INTEGER`, `REAL`, `COMPLEX`, `LOGICAL`, `CHARACTER`
- Identifiers: start with letter, followed by letters, digits, or underscores
- Input with READ

```
READ (*,*) year
```

Integer vs. string

```
PROGRAM example
  IMPLICIT NONE
  INTEGER :: year = 2007

  WRITE (*,*) "year"
  WRITE (*,*) year

END PROGRAM example
```

"year" is a string literal

year is an integer variable

```
year
2007
```

Sequence

```
PROGRAM sequence
  IMPLICIT NONE

  WRITE (*,*) "Hello there!"
  WRITE (*,*) "How are you?"
END PROGRAM sequence
```

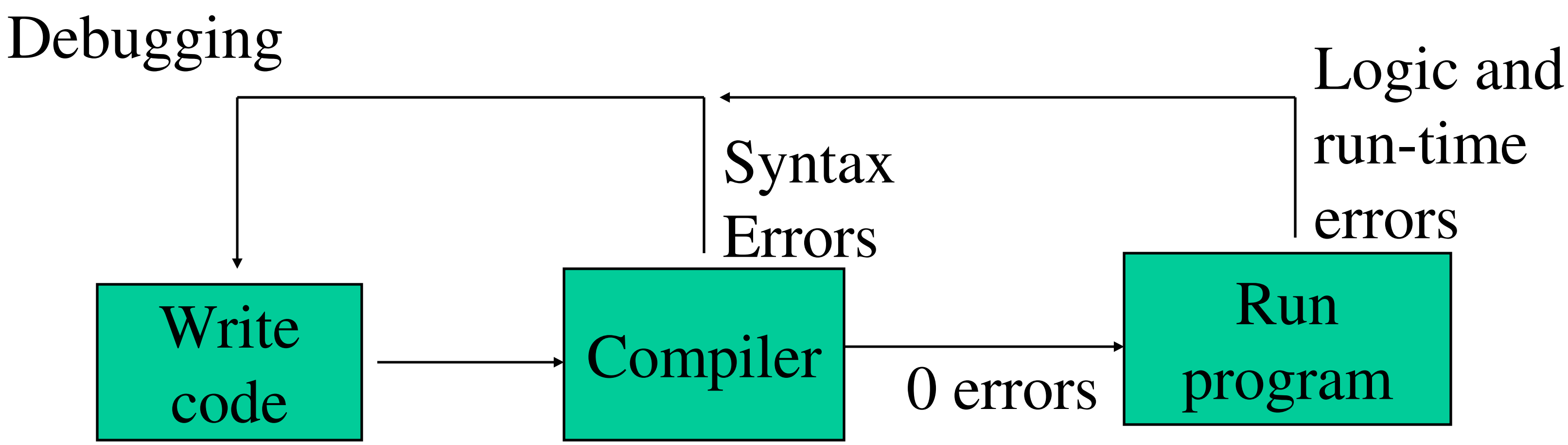
```
Hello there!
How are you?
```

```
PROGRAM sequence
  IMPLICIT NONE

  WRITE (*,*) "How are you?"
  WRITE (*,*) "Hello there!"
END PROGRAM sequence
```

```
How are you?
Hello there!
```

Development Life Cycle



- The process of finding and correcting programming errors is known as debugging.
- Compiler only produces machine code when there are no syntax errors in the program

Programming Errors

- A program can have three types of errors
  1. The compiler will find problems with syntax and other basic issues (*compile-time errors*)
     - If compile-time errors exist, an executable version of the program is not created
  2. A program may run, but produce incorrect results (*logical errors*)
     - d = b*b + 4.0*a*c
  3. A problem can occur during program execution, and causes a program to terminate abnormally (*run-time errors*)
     - Divide by zero
     - Wrong data type

Arithmetic Expressions

An arithmetic expression is formed using the operations:

```
+    (addition)
-    (subtraction)
*    (multiplication)
/    (division)
**   (exponentiation)
```

use x * y instead of xy for multiplication

- Arithmetic expressions compute numeric values
- The basic form is:

```
operand1 op operand2
```

where the 2 operands can be numbers, variables, or expressions

C does not have **

```
! ------------------------------------------------
! Compute B*B-4*A*C
! ------------------------------------------------
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```

Assignment Statement

The assignment statement has syntax:

```
variable = expression
```

Semantics

1. Evaluate the expression
2. Store the result in the variable

Assignment Statement

Watch your step!

New programmers often forget:

- The statement reads “backwards”
  - That is, the variable is on the left, not the right
  - The expression is evaluated before the value is stored in the variable
  - Any value that was in the variable before is replaced

The = sign means assignment; it does not mean equality!

Input

- The original algorithm was generic, that is, it was designed to work for any values of `a`, `b` and `c`
- Those values had to be provided by the user of the program

```
input a, b, c
d <- b*b - 4*a*c
```

- The input command tells the computer to take the values (from the user) and put them into `a`, `b` and `c`

```
! ---------------------------------------------------
! Compute B*B-4*A*C
! ---------------------------------------------------
PROGRAM Discriminant
  IMPLICIT NONE
  REAL :: a, b, c
  REAL :: d

! read in the coefficients a, b and c

  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the discriminant d

  d = b*b - 4.0*a*c

! display the results

  WRITE(*,*) 'The discriminant is ', d

END PROGRAM Discriminant
```

The READ Statement

The Fortran statement used to input values is the READ statement

It tell the computer to wait until the user provides n values and then puts them into the n memory locations indicated by the variables

Syntax:

READ (*,*) $\mathbf{var_1}$, $\mathbf{var_2}$, . . ., $\mathbf{var_n}$

READ Statement Semantics

Semantics:

- Wait for the person using the program to type values to be stored in the variables
- If not enough values are provided the program will just wait until all the values are there
- Input values must be the same type as the corresponding variables
- Data must be separated by commas or blanks
- Extra input values on that line are ignored

Input vs. output

```
PROGRAM example
   IMPLICIT NONE
   INTEGER :: year = 2007

   WRITE (*,*) year
   WRITE (*,*) year + 3
   WRITE (*,*) 3
END PROGRAM example
```

```
PROGRAM example
   IMPLICIT NONE
   INTEGER :: year

   READ (*,*) year     ! ok
   READ (*,*) year + 3   ! Error
   READ (*,*) 3      ! Error
END PROGRAM example
```

- The expressions in a WRITE statement can be numbers, variables, or expressions.
- The expressions in a READ statement must be simple variable names.

The Speed of Light

- How long does it take light to travel from the sun to earth?
- Light travels 9.46×10^{12} km a year
- A year is 365 days, 5 hours, 48 minutes and 45.9747 seconds long
- The average distance between the earth and sun is 150,000,000 km

Elapsed Time

```
PROGRAM light_travel
  IMPLICIT NONE
  REAL :: light_minute, distance, time
  REAL :: light_year = 9.46 * 10.0 ** 12

  light_minute = light_year / (365.25 * 24.0 * 60.0)
  distance = 150.0 * (10.0 ** 6)
  time = distance / light_minute

  WRITE (*,*) "Light from the sun takes ", time, &
              "minutes to reach earth."

END PROGRAM light_travel
```

& is the line continuation symbol

Watch out for ambiguity

Let’s look at an expression from our program

```
light_minute = light_year / (365.25 * 24.0 * 60.0)
```

What if the expression didn’t have parentheses?

```
light_minute = light_year / 365.25 * 24.0 * 60.0
```

Watch out for ambiguity

How about another expression?

```
distance = 150.0 * 10.0 ** 6
```

What value is assigned to distance?

```
(150.0 * 10.0) ** 6 ?
150.0 * (10.0 ** 6) ?
```

Precedence Rules

- Every language has rules to determine what order to perform operations
- These rules try to mimic the conventions we learn growing up
- For example, in FORTRAN ** comes before *
- In an expression, all of the **’s are evaluated before the *’s

Precedence of arithmetic operators

high

( )

**

* /

+ −

low

- () has the highest precedence, followed by **
- multiple **'s are evaluated right to left
- multiple * and / are evaluated left to right
- multiple + and - are evaluated left to right

Precedence Rules

- First evaluate operators of higher precedence

```
3 * 4 – 5 → 7
3 + 4 * 5 → 23
```

- For operators of the same precedence, use associativity. Exponentiation is right associative, all others are left associative

```
5 – 4 – 2 → -1 (not 3)
2 ** 3 ** 2 → 2**(3**2) → 2**9 → 512
```

Precedence of Operators in FORTRAN

Operators in order of precedence and their associativity:

Arithmetic

- ****** right to left
- ***, /** left to right
- **+, -** left to right

Relational

- **<, <=, >, >=, ==, /=** no associativity

Logical

- **.NOT.** right to left
- **.AND.** left to right
- **.OR.** left to right
- **.EQV., .NEQV.** left to right

Another Example

- Last lecture we looked at the problem of finding the roots of a quadratic equation
- We focused on the discriminant
- Here is a program that computes the roots

```
! ----------------------------------------------------
! Solve Ax^2 + Bx + C = 0
! ----------------------------------------------------
PROGRAM QuadraticEquation
  IMPLICIT NONE

  REAL :: a, b, c
  REAL :: d
  REAL :: root1, root2

! read in the coefficients a, b and c
  WRITE(*,*) 'A, B, C Please : '
  READ(*,*) a, b, c

! compute the square root of discriminant d

  d = SQRT(b*b - 4.0*a*c)

! solve the equation

  root1 = (-b + d)/(2.0*a) ! first root
  root2 = (-b - d)/(2.0*a) ! second root

! display the results

  WRITE(*,*) 'Roots are ', root1, ' and ', root2

END PROGRAM QuadraticEquation
```

Data Types

- In the examples we have declared the variables to be of type REAL
- That is, each variable can hold a real number
- What is a real number?
  - In Mathematics?
  - In FORTRAN?

Real Numbers (literals)

Real numbers can be expressed in 2 forms:

decimal form:

`1.23`

`123. (or 123.0), -123. (or -123.0)`

`.123 (or 0.123), -.123 (or -0.123)`

exponential form:

`1.0E-3 (0.001)` (exponential can be negative)

`150.0E6`

But not:

`1,000.000` (comma not allowed)

`12.0E1.5` (exponential must be integer)

Real Numbers (representation)

A real value is stored in two parts

1. A mantissa determines the precision
2. An exponent determines the range

Real numbers are typically stored as

- 32 bits (4 bytes): type REAL

Accuracy of Real Numbers

- REAL numbers:
  - Mantissa represented by 24 bits gives about 7 decimal digits of precision
  - Exponent represented by 8 bits gives range from 10^{-38} to 10^{38}

2 + 2 = ???

- Be careful not to expect exact results with real numbers

```
program roundoff
  implicit none
  real :: x, y
  x = 100.00002
  y = x*x - x
  write (*,*) x/y * (x -1)
end program roundoff
```

2 + 2 = ???

- What result do we expect?

$$\frac{x}{x^2 - x} \times (x-1)$$

- What result do we get?

```
>gfortran -fimplicit-none -W -Wall
          "roundoff.f90" -o "roundoff.exe"
>Exit code: 0
>roundoff
0.9999999
>Exit code: 0
```