Programming with **FORTRAN**

An-Yi Bu (NCU)

Sep 8, 2011

Why Fortran ?

- Especially designed for scientific computing.
- The oldest high-level programming language.

Outline

- Review Putty & Unix commands.
- The history of Fortran.
- How to program & compile Fortran codes.
- Tips for writing a program.
- Practice: Gauss Elimination.
- Recommend material.

Review Putty

- Putty settings.
- Connect to the workstation
  - IP: 140.113.22.122 (clw01)
    102 (clw02)
    103 (clw03)
    104 (clw04)
    105 (clw05)
- If you've already logged in the one of above machines, just type **ssh clwxx** to connect to the others.

Review Unix commands

New directory (建造子目錄)	**mkdir** dir_name
change directory (切換目錄)	**cd** dir_name **cd ..** (回到上一層) **cd** /usr/aybu
rename file (重新命名檔案) **move file** (移動檔案)	**mv** old new **mv** filename **./**dir
remove file (刪除檔案) **remove directory** (移除子目錄)	**rm** filename **rmdir** dir_name **rm –R** name
list contents in directory (顯示目錄內容)	**ls** **ls ***keywords **ls -al**

History

- **Fortran (Formula Translator):**
  the first (high-level) programming language invented by John Backus for IBM in 1954, and released commercially in 1957.

** **high-level language :** It's closer in appearance to human language.

- In 1976, Fortran77 is widely used for numerical and scientific computing.

- In 1991, the version Fortran90 added many new features to reflect the significant changes in programming.

History

- **Example (Machine, Assembly, and High-level)**

1) Machine Language

```
SF$0100001#%*F010SD#10@%&^&^*
```

2) Assembly Language

```
lw   $8              ! LOAD  basePay
add $8 $9 $17   ! ADD    overtimePay
sw   $8              ! STORE grossPay
```

3) High-Level Language

```
grossPay = basePay + overtimePay
```

Let's Start – Basic Format

- **Free Format**

```
! Ex1: My first Fortran code.
program main
2    write(*,*) "hello, world" &
3                    ,"; hello, NCTU"
end
```

1) ! : comment

2) 132 characters is allowed each line. Use "&" if more than 132 characters or needed.

3) line number can be added in front of each line

Let's Start – How to Compile

Now, do it by yourself :

- A simple exercise

1) Write hello.f90

2) Use gfortran or intel fortran to compile your code by

gfortran hello.f90 -o hello.out

ifort hello.f90 -o hello.out

3) **./a.out ./hello.out**

Let's Start –Variable Declare

- **integer, real or real*8, parameter,** complex or complex*8, character, logical.

```
! Ex2: Variable Usage
program variable
    implicit none    ! Each variable must be claimed
    integer :: a = 8  ! use :: to assign a value to a at
first
    real       b
    real*8   c
    b = 8
    c = a
    write(*,*) "(a, b, c) = ( ",  a  ,  ",",  b  ,  ",",  c  ,  " ) "
end
```

Let's Start –Variable Declare

- Remember that …

1) Name variables different from the Fortran commands.

(Ex. integer :: WRITE = 8 ; real * 8 :: PROGRAM)

2) Make variables names meaningful.

(Ex. integer month, real weight)

3) It's case-insensitive for Fortran

(Ex. character nctu ; character NctU)

4) The first character MUST be an English letter

(Ex. real time ; ~~integer 3day~~)

5) Avoid some "confusing" letters.

(Ex. I , l , O , 0)

Let's Start – Operators

- ＋、－、*、/ : basic operators
- ** : power
- ( ) : first priority

```
! Ex3
program operator
    implicit none
    integer :: a = 5
    write(*,*) 3**2 + 1 , 3 ** ( 2 + 1 )  ! 10 , 27
    write(*,*) 3 / 5                      ! 0
    write(*,*) 3. / 5 , 3 / 5.            ! 0.6,
0.6
    write(*,*) 3 / real(a)                ! 0.6
end
```

Let's Start – Input & Output

- write(* , *) : Output.

  position ↗ ↖ format

- read(* , *) : Input

```
! EX4
program iotest
    implicit none
    integer n
    write(*,*) “Input your student ID
number : ”
    read (*,*) n
    write(*,*) “Ur ID number is : ” , n
end
```

Let's Go Further – IF statement

- **Logical operators**

==	Equivalent
/=	Not Equivalent
>	Greater than
>=	Greater or Equivalent
<	Little than
<=	Little or Equivalent

```
If ( mod(n,2) = 0 ) then  ! Do you think this reasonable?
   write(*,*)  " n is an even number "
endif
```

Let’s Go Further – IF statement

- “==“ and “=“ are completely different !
- “=“ : **assign** right value to left.

Let's Go Further – IF statement

- **Usage :**

```
if ( condition 1 ) then
    …
else if (condition 2 ) then
    …
else
    …
end if
```

Note. Use **.or.** and **.and.** when there are more than one condition.

Let's Go Further – IF statement

```
! EX5 iftest
program iftest
    implicit none
    real s
    write(*,*) " Score: "
    read (*,*) s
    if ( s >= 90 .and. s <= 100 ) then
       write(*,*) s , " is A ! "
    else if ( s >= 80 .and. s < 90 ) then
       write(*,*) s , " is B ! "
    else if ( s >= 70 .and. s < 80 ) then
       write(*,*) s , " is C ! "
    else
       write(*,*) " Too bad, BYE! "
    endif
end
```

Let's Go Further – IF statement

- Nested if statement

```
if ( … ) then
    if ( … ) then
    elseif ( … ) then
    endif
endif
```

Let's Go Further – IF statement

```
! EX6 iftest2
program nestedif
    implicit none
    real x,y
    write(*,*) "Input (x,y): "
    read(*,*) x , y
    if  ( x > 0 ) then
      if ( y > 0 ) then
         write(*,*) "The first Quadrant!"
      elseif( y < 0 ) then
         write(*,*) "The fourth quadrant!"
      endif
    elseif ( x < 0 ) then
                Do it by yourself !!!
    endif
end
```

Let's Go Further – Do Loop

- **Usage**

(1) Single loop

```
do  i = begin , end ( , increment )
    …
end do
```

(2) Double loop

```
do  i = begin 1, end1 ( , increment )
    do  j = begin 2, end2 ( , increment )
        …
    end do
end do
```

Let's Go Further – Do Loop

```
! EX7. 1 + 3 + ... + 99 = ?
program looptest
    implicit none
    integer sum, i
    sum = 0    ! Initialize sum
    do i = 1 , 99, 2
        sum = sum + i
    enddo
    write(*,*) sum
end
```

DIY : (For & if) write a code to compute the last two digits of 23^1437. [Hint: you may use "mod(a,b)" to compute the remainder of a/b.]

Let's Go Further – Array

- **1-dimention array :**

  datatype :: arrayname(size)

integer, real, complex, logical

MUST be a parameter!

- **2-dimention array**

  datatype :: arrayname(row , col)

Let's Go Further – Array

- **Initialize array (similar to Matlab)**

  For A is a matrix:

  A = 5

  A = (/1,2,3/)

  A = B

  A = B+C , A = B-C, A = B*C, A=B/C

  A = sin(real(B))

  A(3:5) = 5

  A(:) = B(:,2)

Let’s Go Further – Array

```
! Ex8.
program arraytest
    implicit none
    integer, parameter :: row = 3 , col = 3
    integer i
    real :: matrixA( row , col )
    write(*,*) “ Input matrix A =”
    ! read(*,*) matrixA
    do i = 1 , row
        write(*,*)"the",i,"-th row:"
        read(*,*)matrixA(:,i)
    enddo
    do i = 1 , row
        write(*,*) matrixA(:,i)
    enddo
end
```

DIY : write a code to compute C = A + B

Let's Go Further – Array

- **Dynamic array**

  In some situation, you don't know the size of a matrix before executing a code. You can allocate the size later.

- **Usage**

```
integer :: size  ! Do not use parameter type
datatype, allocatable :: arrayname(:)
        …
    read(*,*) size
allocate( arrayname(size) )
deallocate( arrayname )
```

Let’s Go Further – Array

```
! Ex9.
program dyarray
    implicit none
    integer :: student , i
    real, allocatable :: grade(:)
    write(*,*) “How many students?“
    read(*,*) student
    allocate(grade(student))
    do i = 1, student
        write(*,*) “grade = ”
        read(*,*) grade(i)
    enddo
    do i = 1, student
      write(*,*) i,"-> grade = ",grade(i)
    enddo
    deallocate(grade)
end
```

Recommend Materials

- **Fortran 95** 程式設計, 彭國倫, 碁峰出版社