

Computers in Chemistry – Lecture IX

Prof. Dr. Stephan Irle
Quantum Chemistry Group
Nagoya University

1

Get this lecture online

- Please go to: <http://qc.chem.nagoya-u.ac.jp>
- Click on “Teaching”
- Click on “PPT” link of “9.1 Lecture IX – Subroutines and Arrays in FORTRAN”
userid: **qcguest**, password: **qcigf!**

limit.f90 (form the sum of increasingly large integers to a specified limit)

8.1 Lecture VIII - Functions in FORTRAN (PPT)

8.2 Example programs: temp2.f90 (Fahrenheit to Celsius temperature conversion), temp2ext.f90 (same but with an external function definition)

2

7.1 Subroutines I

- **Programming with Subroutines**
- Program units designed to perform particular tasks under the control of some other program unit.
- **Look like FUNCTIONS but do not return a value.**
- FORTRAN 90 allows **three types of subroutines**: internal, module, and external subroutines
- May not return a value to all, or may return more than one value modified in the argument-list.
- **A function is referenced by its name alone, whereas a subroutine is referenced by a CALL statement**

3

7.1 Subroutines II

- The form of a subroutine subprogram is:

```
subroutine heading  
Specification part  
Execution part  
END SUBROUTINE statement
```

- Like functions, the subroutine can be contained in the same or a different .f90 file. For simplicity, we will only consider the case where one .f90 source code file contains both **program** and **subroutine(s)**.

4

7.1 Subroutines III

- Subroutine heading is a SUBROUTINE statement of the form:
`SUBROUTINE subroutine-name (formal-argument-list)`
- Or, for a recursive subroutine,
`RECURSIVE SUBROUTINE subroutine-name (formal-argument-list)`
- “subroutine-name” is a legal Fortran identifier, “formal-argument-list” is an identifier or list (possibly empty, in which case **we do not need “()”**) of identifiers separated by commas.
- A subroutine is referenced in the program by a CALL statement of the form:
`CALL subroutine-name (actual-argument-list)`
- Subroutines can contain CALL statements themselves. However, they cannot call themselves unless they are specified as a “recursive” subroutine (see previous page).

5

7.1 Subroutines V

- Specification part of a subroutine has the same form as that of a regular program. It must declare:
The type of each formal argument appearing in the “list-of-arguments” as well as variables that appear in the subroutine
- The execution part of a function subprogram is similar to a regular program, but unlike a function **it does NOT require a statement:**
`function-name = expression`
- The last statement of a function subprogram should be:
`END SUBROUTINE subroutine-name`

7

7.1 Subroutines IV

- Variables in the “formal-argument-list” are called “**formal**” or “**dummy arguments**” and are used to pass information to the function subprogram.
- Note: Different program languages have different default ways of passing information from the main program to the subprogram.
- FORTRAN: “**pass-by-reference**” (use a memory pointer)
- C/C++ and Java: “**pass-by-value**” (the value cannot be changed by the subprogram)

6

7.1 Subroutines VI

- Example: download a program to convert temperature from Fahrenheit to Celsius units, this time using an external subroutine, temp3.f90, and compile and run it in an X-Windows terminal by:
 - cd Downloads
 - gfortran -o temp3.x temp3.f90
 - ./temp3.x

8

7.1 Subroutines VII

- Sample run:

```

$ ./temp3.x
Enter temperature in Fahrenheit:
32
 32.00000 is equivalent to  0.000000 in Celsius
More temperatures to convert (Y/N)?
y
[stephan@hawk ~]$ ./temp3.x
Enter temperature in Fahrenheit:
32
 32.00000 is equivalent to  0.000000 in Celsius
More temperatures to convert (Y/N)?
y
Enter temperature in Fahrenheit:
212
 212.0000 is equivalent to 100.0000 in Celsius
More temperatures to convert (Y/N)?
y
Enter temperature in Fahrenheit:
-22.5
 -22.50000 is equivalent to -30.27778 in Celsius
More temperatures to convert (Y/N)?

```

9

8.1 Arrays I

- An array is a **data structure**
- Arrays can be 1-dimensional (vector), 2-dimensional (matrix), or multi-dimensional

```

REAL, DIMENSION(2) :: Vector, Rotated_Vector
REAL, DIMENSION(2,2) :: Rotation_Matrix

```

- Arrays are often used in vector calculus, linear algebra, data processing, etc.
- Ideal for computation, especially in combination with counter-controlled DO loops

10

8.1 Arrays II

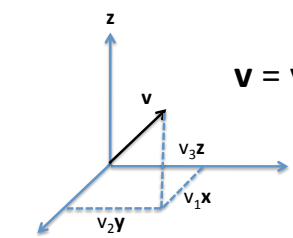
- A vector needs to be defined in a space. Typically, this is three-dimensional Euclidean space \mathbf{R}^3 where the three base vectors are orthogonal on each other (form 90° angles with each other):

$$\mathbf{x} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

11

8.1 Arrays III

- Then, any vector \mathbf{v} in \mathbf{R}^3 can be mathematically expressed as a *linear combination* of these three vectors:



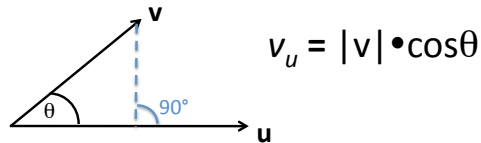
$$\mathbf{v} = v_1 \cdot \mathbf{x} + v_2 \cdot \mathbf{y} + v_3 \cdot \mathbf{z} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

a, b, c are called "vector coefficients"

12

8.1 Arrays IV

- A scalar projection v_u of a vector \mathbf{v} on another vector \mathbf{u} is given by:



- A scalar product between two vectors corresponds to their “inner product”:
 $\mathbf{v} \cdot \mathbf{u} = |\mathbf{v}| \cdot |\mathbf{u}| \cdot \cos\theta.$

13

8.1 Arrays V

- In two dimensions, the scalar (“dot”) product is given by:

$$\mathbf{v} \cdot \mathbf{u} = v_1 * u_1 + v_2 * u_2 = |\mathbf{u}| * |\mathbf{v}| * \cos\theta$$

- **Properties of the scalar (“dot”) product:**
 - a) If the two vectors are “orthogonal” (form 90° angles), their scalar product is 0!
 - b) If the two vectors are identical (form 0° angles), their scalar product is the square of its magnitude, $|\mathbf{v}|^2$

14

8.1 Arrays VI

- Task: Write a program that reads two 2-dimensional vectors \mathbf{v} and \mathbf{u} , and then calculates and prints their scalar product.
- Note: Please try to use a subroutine to compute the scalar product for any two vectors.
- **Good luck. This concludes today’s lecture.**

15