PROGRAMMING WITH FORTRAN 1

I don't know what the programming
language of the year 2000 will look like,
but I know it will be called FORTRAN.
Charles Anthony Richard Hoare

Basic Introduction to Linux

1. In Linux folder is called directory
2. Files are stored in a hierarchical tree structure of directories(folders) and subdirectories.
3. When you open konsole, you will find yourself working in your home directory.
4. Remember the case sensitiveness of what you type on the konsole (terminal)

Rules for naming files and directories

1. The filenames can be up to 255 characters (or bytes) long
2. Files and directories names are case sensitive
3. Upper and lower case letters, numbers, dot (.), underscore (_) symbol
4. Blank space are allowed but it is very difficult use those files
5. The symbol / (root directory) is reserved as separator between files and directories and therefore should not be used

Cont. rule for naming files

In common with other operating systems, UNIX often names files with names of the form

<filenamebody>.< extension>

where the extension (usually no more than three or four letters) indicates the file type. For example

- .txt : text file
- .html (or .htm) : INTERNET page file
- .tex : document in the TeX (or LaTeX)
  .ps : postscript file

Meaning of some commands

1. 'mkdir' for creating a directory (folder)
2. 'cp' for copy
3. 'ls' for directory list
4. 'pwd' for showing the current path or current working directory
5. 'cd <dirname>' for changing current directory to <dirname>
6. 'cat<filename>' for listing the contents of a file. You can also 'less' or 'more' for listing

Cont. meaning of some commands

1. 'rm <filename>' for delete (remove) file <filename>
   !! USE WITH CAUTION !!
2. 'mv' for moving or renaming a file or directory
3. 'rm -r' for removing directory
4. 'df' for displaying the amount of available disk space
5. 'date' for displaying the date and time on your system
6. 'Exit' for exiting the konsole (terminal)
7. 'man' to the manual page (of command)

Introduction to Emacs

Emacs provides a mode that makes it easy to edit Fortran code, providing context sensitive indentation and layout.

Notation

It is very important to become familiar with these notations used in all standard Emacs documents.

1. C-x = press the Ctrl key, hold it and press x.
2. M-x = press the Alt key.ho;d it and press x.
3. C-M-x = press and hold Ctrl key, then add Alt and then x.
4. RET = blueEnter key
5. SPC = the Space bar
6. ESC = the escape key
7. C-x-C-f = press Ctrl and keep it down while you press x first and then f.

Note - at times Shift key is used in combination with Ctrl or Alt keys

Basic Emacs commands

C-x C-f - opens a file and shows it in the current buffer
C-x C-s - saves the buffer
C-x C-w - writes the buffer in a different file (Save as)
C-x C-c - quites emacs
C-a - jump to the beginning of the current line
C-e - jump to the end of the current line
M-f - move forward one word
M-b - move backward one word
M-< - move to the top of the buffer
M-> - move to the bottom of the buffer
C-d - delete the character at the cursor
C-k - delete the text from the position of cursor to the end of the current line
M-d - delete forward until the end of the next word
M-Del - delete backwards until the beginning of a previous word

Basic Emacs commands conti

C-r - search backward (searching towards the top of the current buffer)
C-s - search forward (searching towards the end of the current buffer)
M-% - replace forward
C-SPC - mark the beginning of the text for copying/moving/deleting
C-w - cut the text from buffer to the clipboard
C-y - paste the text from the clipboard at the position of the cursor
C-_ - undo the last change
ESC ESC ESC - cancel the last operation

Saving Document

After executing C-x C-s command, the minibuffer at the bottom of your screen ask for the file name, type the name and the extension. To replace an existing file, press the TAB key during writing the file name and Emacs automatically adds the rest of the name.

Course Objective

To learn Fortran 90 program so that you can use it to solve scientific problems.

Good News

No previous programming experience is required.

What is computer program?

A computer program is a set of instructions for the computer to execute sequentially. The particular set of rules for coding the instructions to a computer is called a programming language.

Steps involved-

- Initially the program source code is written in plain text by the programmer using text editor. Fortran 90 files typically have extension (.f90).
- This file is then compiled (by existing application, called the compiler) to translate the plain text of the source code to binary code for the processor. The executable files have extension (.exe).
- The binary executable file is then run to get the output/results in plain text.

Fortran Background

Fortran (FORmula TRANslator) was the first high-level programming language proposed in 1953 by John Backus and the first program was run in April 1957. Fortran is use mainly within the scientific community for solving problems that has a significant arithmetic content.

Versions/Standards

- Fortran 66 - first standard
- Fortran 77 - character handling
- Fortran 90 - array and modular programming
- Fortran 95 - functional programming
- Fortran 2003 - object-oriented and generic programming
- Fortran 2008 - concurrent programming.

Fortran Program Structure

```
[PROGRAM  name ]

[IMPLICIT NONE]

[Declaration-part]

[execution-part]

[subprogram-part]

END [ PROGRAM   name]
```

Everything in square brackets [ ] is optional. However, it is good programming practice to put the name of the program in both header and END statements.

The program name must :

- Begin with a letter (upper/lower case)
- It contains letters. digits 0-9, and underscore characters
- Maximum of 31 characters.

Features of Fortran Program 1/8

IMPLICIT NONE

This is very important non-executable fortran statement used to disable the default typing provisions of Fortran. When this statement is used in the program, any variable that does not appear in an explicit type declaration statement is considered error. If omitted, fortran uses implicit declaration where a variable is considered integer if the first letter of its names lies in the range of I - N and will be declared to be a real variable otherwise.

IMPLICIT NONE should appear after the Program statement and before any type declaration statement.

IMPLICIT NONE will be use in this course to write safe programs.

Features of Fortran Program 2/8

STATEMENT

- Fortran statement layout is one-per-line separated with blank lines for clarity.
- There may be more than one statement per line, separated by a semicolon(;) e.g
  A = 1; B = 10; C = 5
- Line may be up to 132 characters long.
- If the statement is too long to fit on one line, the continuation character (&, ampersand) is used as last character and then statement continue on next line; e.g

  ```
  Rad = Deg * 3.14  &
  /180.0
  ```

  The above is equivalent to Rad = Deg * 3.14/180.0

Features of Fortran Program 3/8

Fortran Data type

- A Fortran 90 data type may be an integer, real, logical, complex, and character string
- An integer data type is a string of digits with an optional sign: 12, -20, 0, +2333
- A real data type has two forms decimal and exponential

  The decimal form consist of string of digits with exactly one decimal point. e.g 1.4, -13.5, +0.04, 33333.5

In the exponential form, start with an integer/real. followed by a E/e, followed by an integer (i.e., the exponent). e.g 15E3 (12x10^3), *equivalent to* 15*e*3 (15x10^3)

Features of Fortran Program 4/8

Fortran Data type cont.

- A logical data type is either .TRUE. or .FALSE.
  Note that the periods surrounding TRUE and FALSE are required!
- A complex data type consists of two numeric constants separated by comma and enclosed in parentheses. e.g (1.,0.) equivalent to $1 + 0i$ and (0.7071,0.7071) equivalent to $0.7071 + 0.7071i$
- A character/string data type is a string of characters enclosed between two double or two single quotes. e.g "abc" produces abc
  The length of a character string is the number of characters between the quotes including blank space.

Features of Fortran Program 5/8

Variables

- A variable is an entity that is used to store a value, comparable to the use of variable in algebra.
- Must be between 1 and 31 characters in length. The first character must be a letter and the remaining if any, may be letters, digits or underscore.
- The variables are case insensitive and therefore NAME and naMe are the same.
- Fortran 90 does not have reserved words and therefore Fortran keywords can be used as variable names but avoid it.
- The type of a variable is declared in the declaration statement.
- Variable can be given a value during the execution of the program - by assignment or by a READ statement.

Features of Fortran Program 5/8

Valid Variables names

x
my_variable
Results17
STUDENT

Invalid Variables names

1x	first character is a number
my$variable	'$' is not letter, numeral or underscore
my_really_really_long_variable_name	more than 31 characters

Features of Fortran Program 6/8

Declaration of Variables

- Fortran 90 uses the following for variable declaration, where type-specifier is one of the fortran following keywords: INTEGER, REAL, LOGICAL, COMPLEX and CHARACTER, and list is a sequence of identifier separated by commas.

  type-specifier :: list

  Example:
  INTEGER :: x, total
  REAL :: average, z_mean
- CHARACTER variables require the string length
- The keyword CHARACTER must be followed by the length attribute (LEN = integer_value), where integer_value is the length of the string.

Features of Fortran Program 7/8

Declaration of Variables cont

- If the length of the string is 1 then the CHARACTER can be declared without the LEN attribute.
  EXAMPLE:
  - CHARACTER (LEN = 20) :: answer, name
    Variables answer and name can hold up to 20 characters.
  - CHARACTER :: phy
    Variable phy can hold only ONE character (i.e length is 1)
  - There is another where of declaring CHARACTER length when PARAMETER attribute is used. The compiler assumed the length from the parameter and you do not have to state the length. The symbol * is used for the length of character.
    EXAMPLE ::
    CHARACTER (LEN = *), PARAMETER :: dir = "my program"

Features of Fortran Program 8/8

Variables Attributes

Various Fortran attributes may be specified for variables in their type-declaration statement.

- The PARAMETER attribute dictates that a constant is being defined. A variable declared with this attribute may not have its value changed within the program unit.
- The PARAMETER attribute is used after the data type keyword, followed by variable name followed by a = and then value for the variable.
  EXAMPLE :
  REAL, PARAMETER :: PI = 3.143
  INTEGER, PARAMETER :: TOTAL = 10

Features of Fortran Program 9/8

Variable Initialization

- A variable receives its value with
  - Initialization: It is done once before the program runs.
  - Assignment: It is done when the program executes an assignment statement.
  - Input: It is done with a READ statement.
- The variable can be initialized in their type-declaration statement
  EXAMPLE:
  REAL :: x = 0.05
  INTEGER :: gravity = 10
- It can also be initialized before execution statement.

Features of Fortran Program 10/8

Standard input/output (I/O) statement

- Fortran uses the READ * as standard input statement to read data into variable from the keyboard.
- Read statement is of the form: READ *, input_list
- If READ * has n variables in its list, there must be n Fortran declared constants.
- Each entered constant must have the type of the corresponding variable.
- Data items are separated by spaces and may spread into multiple lines.
- The asterisk * can be change to specify the data format. The format is of the form READ "(input_format)", input_list

Features of Fortran Program 11/8

Standard input/output (I/O) statement cont

- Fortran uses the PRINT * as standard output statement to display data output on the screen.
- Print statement is of the form: PRINT *, output_list
- To print undeclared string in your program put the string statement in between double quotes. e.g PRINT*, "This is fortran class" displays This is fortran class
- PRINT*, without output_list produces a blank line
- A formatted PRINT statement is of the form: PRINT "(output_format)", output_list

Features of Fortran Program 10/8

Program Comments

- Comments start with a exclamation mark (!)
- Everything following (!) will be ignored by the compiler
- Always comment your code so you can understand it later and also for others to understand when reading your code.

Example :

```
output = input1 + input2                ! sum the inputs
```

```
PROGRAM first_program
! This program tries to illustrate some basic features of fortran
INTEGER :: x, y
```

Features of Fortran Program 11/8

Cases and Indentation

- Except within character strings, Fortran is completely case-insensitive.
- Fortran keywords such as REAL, DO, END, IF and intrinsic functions such as SIN, COS , TAN should be in “UPPER CASE“ letters and user created entities such as variable names should be in lower case.
- 'Indentation', this is essential for program readability. It is common to indent a program's content by 2 spaces from its header and END statement, and also indent the statements contained within, for example, Fortran keywords such as IF constructs or DO loops by a similar amount.

Operators in Fortran 1/6

- There are four types of operators in Fortran 90: arithmetic, relational, logical and character.

Arithmetic Operator

Order of Precedence	Operator	Meaning
[1]	**	Exponentiation operator
[2]	/	Division operator
[2]	*	Multiplicative operator
[3]	+	additive operator
[3]	-	Minus operator

- Within the same level of precedence, evaluation will proceed from left to right for all the operators with the exception of exponentiation where evaluation proceeds from right to left.
- Parentheses (...) can be used to change the default order of precedence.

Operators in Fortran 2/6

Relational Operator

- Relational operators are used to test the relationship between two numeric operands for that specific operator.
- Expression involving these operators will reduce to either '.TRUE.' or '.FALSE.'.

Operator	Meaning
$<$ or .LT.	Less than
$>$ or .GT.	Greater than
$==$ or .EQ.	Identically equal to
$/=$ or .NE.	Not equal to
$<=$ or .LE.	Less than or equal to
$>=$ or .GE.	Greater than or equal to

Operators in Fortran 3/6

Logical Operator

- Logical operators act on logical expressions to create a largest logical expression.
- Parentheses (...) can be used to change the default order of precedence.

Order	**Operator**	**Meaning**
[1]	.NOT.	Negative the following operand
[2]	.AND.	.TRUE. if both operands are .TRUE.
[3]	.OR.	.TRUE. if at least one operand is .TRUE.
[4]	.EQV.	.TRUE. if both operands reduce to the same
[4]	.NEQV.	.TRUE. if both operands are different

Operators in Fortran 4/6

Character Operator

- There is only one character operator, concatenation symboled //.
- If strings A and B have lengths m and n, then concatenation A // B is a string of length m+n
- It is use to join two separate characters.

Example:
'Man' // 'chester' gives 'Manchester'
CHARACTER(LEN = 7) :: word_1 =“missing”
CHARACTER(LEN = 3) :: word_2 = “you”
CHARACTER(LEN = 10) :: ans
ans = word_1 // word_2   ! missingyou

Operators in Fortran 5/6

Character substrings

- A consecutive portion of a string is a substring.
- To use substrings, one may add an extent specifier to the CHARACTER variable.
- An extent specifier has the following form:

  (integer_exp1 : integer_exp2)

- The first and the second expressions indicates the start and end: (3 : 8) means 3 to 8.
- If A = "abcdefg", then A (3 : 5) means A's substring from position 3 to position 5 (i.e., "cde")
- If the first extent specifier is missing, the substring start from the first character, and if the second extent specifier is missing, the substring ends at the last character.

Fortran Intrinsic Function 6/6

Function	Meaning	Data type
SQRT(x)	square root of x	Real
COS(x)	cosine of x	Real in radian
SIN(x)	Sine of x	Real in radian
TAN(x)	Tangent of x	Real in radian
ASIN(x)	Sine inverse	Real
ACOS(x)	Cosine inverse	Real
ATAN(x)	Tangent inverse	Real
EXP(x)	e^x	Real
ALOG(x)	$\log_e(x)$	Real
ALOG10(x)	$\log_{10}(x)$	Real
ABS(x)	Absolute value of x	Real, Integer
MAX(x1,x2 ....)	Maximum value of x vector	Real, integer
MIN(x1,x2 ....)	Minimum value of x vector	Real, integer
MODULO(x, y)	x modulo y	Real, integer
MOD(x, y)	remainder when x is divided by y	Real

Data type Conversions 6/6

Function	Meaning
INT(x)	Truncate a real number to an integer
NINT(x)	Round a real number to the nearest integer
REAL(x)	Convert to real
CEILING(x)	Nearest integer greater than or equal to x
FLOOR(x)	Nearest integer less than or equal to x
CMPLX(x)	Real to complex
FRACTION(x)	Fractional part of real

Write and Run fortran program 1/3

Creating a fortran program

- Either open a pre-existing source code or create it and save using text editor and should have a name that ends in .f90.
- A text editor is a type of program used for editing plain text files.
- Examples of some text editors are (emacs, gvim, vi, gedit, kile, xemacs etc)
- On command line and inside your program directory, open the file by typing:
- emacs prog1.f90 where prog1 is the filename and emacs text editor
- Emacs shows blank page since prog1 is a new file.

Write and Run fortran program

Compiling fortran program

- The role of fortran compilers is to translate the plain text of the source code to binary code for the processor.
- Some free available compilers are (g95, gfortran, ifort, etc)
- During compilation the program is checked for syntax errors and, if none are found, an executable file is created
- The default name for the executable file will be a.out but you can also specify a name with different extension.
- In the same directory containing the program file type of one these commands:

  1. g95 -o prog1.exe prog1.f90 this produces executable file called prog1.exe
  2. g95 prog1.f90 this produces executable file called a.out

Write and Run fortran program

Running fortran program

- If compilation step is carried out successfully, then the executable prog1.exe or a.out will appear when the files are listed with ls.
- You can then run the program with the command ./prog1.exe or ./a.out
- The ./ before the file name tells the computer that the executable file is in the current directory.

Example program 1/5

Product of two variables

```
PROGRAM firstprogram                                  ! start of program
  !This program finds product of two real variables
  IMPLICIT NONE                 ! All variables used must be declared
  REAL :: x,y,z                                       ! Declare variables
  x = 5.1                                              ! Assign x
  y = -17.2                                            ! Assign y
  z = x*y                                              ! Assign (x*y) to z
  PRINT*,'The product of x and y is', z                ! Print out z
END PROGRAM firstprogram                               ! end of program
```

Example program

Radius and perimeter of cycle

```
PROGRAM maths                                  ! start of program
  !This program finds the radius and perimeter of a cycle
  IMPLICIT NONE                     ! All variables used must be declared
  REAL, PARAMETER :: pi = 3.143             ! parameter declaration
  REAL :: rad, area, peri                  ! Declare variables to be used
  PRINT*, 'Enter the radius of the cycle'
  READ*, rad
  area = pi*rad**2                          ! assign area
  peri = 2*pi*rad                           ! assign peri
  PRINT*,'The area of a cycle of radius', rad 'is', area
  PRINT*,                                    ! print blank line
  PRINT*,'The perimeter of a cycle of radius', rad 'is', peri
END PROGRAM maths                             ! end of program
```

Formatted Output 1/7

- Lots of 1/O related features in Fortran - not all will be considered for now
- Fortran PRINT statements do formatting by preceding the variables to be printed with (format descriptor).
- Format descriptor are put in parentheses.

Symbols used with format descriptor

Symbol	meaning
c	Column number
d	Number of digits to right of the decimal places of real
m	Minimum number of digits to be displayed
n	Number of spaces to skip
r	Repeat count - the number of times to use a descriptor
w	Field width - the number of characters to use

They are applied to both output and input

Formatted Output 2/7

Integer Output - The I Descriptor

Integer values are right justified in their fields. This means that integers are printed out so that the last digit of the integer occupies the rightmost column of the field.
The I descriptor has the general form

rIw or rIw.m

where r, w, and m have the meanings given in the above table.

Example

```
INTEGER :: x = 12345
PRINT'(I5)', x          !print integer in field width of 5
```

If the integer is too large to fit into the field in which it is to be printed, then the field is filled with asterisk.

Formatted Output

Real Output - The F Descriptor

Real values are printed right justified within the fields. If the specified field width is less than the number to be printed, the number is rounded off before it is displayed. On the other hand, if the specified width is more than the number to printed, extra zeros are appended to the right of the decimal point.
The I descriptor has the general form

rFw.d

where r, w, and d have the meanings given in the table.

Example:

```
REAL :: x = 1234678
PRINT'(F8.3)', x     !print real of field width 8 with 3d.p
```

This will produce 1234.678

Formatted Output

Real Output - The E Descriptor

Real data can be printed in exponential form using the E descriptor. Real values displayed by E descriptor are normalized to a range between 0.1 and 1.0. For example 4.096 x 10^3 will appears as 0.4096 E+04 using the E descriptor
The E descriptor has the general form

rEw.d

The width d of E descriptor must satisfied the expression
$w \geq d + 7$ where d has the same meaning given in the table
Example:

```
REAL :: w = 23456
PRINT'(E11.5)', w
```

This produce 0.23456E+05 as output.

Formatted Output 5/7

True Scientific Notation - The ES Descriptor

The ES descriptor is exactly the same as the E descriptor, except that the number to be printed is displayed with mantissa in the range between 1 and 10

The ES has the form:

rESw.d

The formula for the minimum width of an ES descriptor is the same as that of E but the leading zero in E descriptor is replaced by a significant digit in ES.

Example:

```
REAL :: w = 23456
PRINT'(ES11.5)', w
```

This produce 2.34560E+04 as output.

Formatted Output

Character Output - The A Descriptor

A descriptor has the form:

    rA or rAw

where r and w have the meanings given in the table

The rA descriptor displays character data with the same width as the number of characters being displayed whiles the rAw displays character data with width w. The rA are printed right justified and if the width of the field is shorter than the length of character variables only first w characters of the variable are printed.

Example:

CHARACTER(LEN = 7) :: dept = 'physics'

PRINT'(A7)', dept

This produce physics as output.

PRINT'(A3)', dept

This produce phy as output.

Formatted Output

Horizontal Spacing - The X Descriptor

The X Descriptor is used to add one or more blanks space between two values on the output line.

The X Descriptor has the form:

nX

where n is the number of blanks to insert.

Example:

```
CHARACTER(LEN = 7) :: dept = 'physics'
PRINT'(A7)', dept
```

This produce physics as output.

```
PRINT'(2x,A7)', dept
```

This produce   physics as output (ie moves two space to the right)

Fortran 90 Control Structures

Three block constructs

1. IF Statement and IF Constructs
2. DO
3. CASE

All can be nested.

Fortran 90 Control Structures

IF statement and Construct

An IF Statement/Construct allows part of a program to be executed if certain condition is true.
Relational and logical operators are used to specify then condition
Fortran 90 has IF statement or logical IF and IF construct

- IF Statement or Logical IF
- IF Constructs

  Three form of IF constructs

  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSE IF-END IF

Fortran 90 Control Structures 3/5

Logical IF or IF Statement

The logical IF or IF Statement executes or does not execute a statement based on the value of a logical expression.
The syntax of the logical IF statement is:

IF (<logical-expression>) < action-statement>

The action-statement is executed if logical-expression is true. This is used if there is only one action statement. It can also be used to exit from a loop.

Example

```
IF (mark > 70) PRINT*, 'A'
```

Fortran 90 Control Structures 4/5

IF Construct - (IF-THEN-END IF)

This form simply performs a single test and if the test result is true then a series (or a single) Fortran statements that lie in the 'body' of the construct are executed.
The syntax is of the form:

```
IF (< logical-expression >) THEN
   < action-statement1 >
   < action-statement2 >
   <        etc        >
END IF
```

Remember to do indentation to make your code readable !!!

Fortran 90 Control Structures

IF Construct - (IF-THEN-ELSE-END IF)

This form of IF Construct allows the program to take action on a series of Fortran statement if the condition is true as well as if it is false. This is done by the addition of the 'ELSE' keyword.
The syntax is as follows:

```
IF (< logical-expression >)  THEN
   < Statements if condition is true >
ELSE
   < Statements if condition is false >
END IF
```

Remember to do indentation to make your code readable !!!

Fortran 90 Control Structures

IF-THEN-ELSE-END IF Example

```
PROGRAM if_prog1                        ! start of program
  IMPLICIT NONE                        ! all variables must be declared
  INTEGER :: x, y                         ! x, y declared as integers
  PRINT*, 'Please enter two integers'       ! print prompt
  READ*, x, y                     ! read in x and y
  IF (x < y) THEN
   PRINT*, 'The first number is the smaller.'
   x = y - x
   PRINT*, 'The difference is ', x
  ELSE
   x = y - x
   PRINT*, ' The difference is negative', x
  END IF
END PROGRAM if_prog1
```

Fortran 90 Control Structures

IF Construct - (IF-THEN-ELSE IF-END IF)

This form of IF Construct allows the programmer to produce a 'mutually exclusive' list of options. Note there can be any number of 'ELSE IF' branches in the construct and also 'ELSE' is optional. The syntax is as follows:

```
IF (< logical-expression1 >) THEN
   < Statements if condition1 is true >
ELSE IF (< logical-expression2 >) THEN
   < Statements if condition2 is true >
ELSE IF (< logical-expression3 >) THEN
   < Statements if condition3 is true >
ELSE
   < Statements all conditions are false >
END IF
```

Fortran 90 Control Structures

DO and DO WHILE loops

If a block of code is to to performed repeatedly it is put inside a DO loop

There are two basic types of DO loops:

- Deterministic DO loops

  The number of times the section is repeated is stated explicitly.

- Non-deterministic DO loops

  The number of repetitions is not stated in advance.

Fortran 90 Control Structures 2/5

DO loops - (Deterministic DO loop)

The number of times the section is repeated is stated explicitly.
The syntax is as follows:

```
DO control-var = initial, final [ , step ]
  < Statements >
END DO
```

- control-var is an INTEGER variable, initial, final and step are INTEGER expression; however, step cannot be zero.
- If step is omitted, its default value is 1
- statement are executable statement of the DO
- The control-var runs from initial in increments of step until it reaches the value final and then the loop terminates.
- If step is positive, this DO counts up; if step is negative, this DO counts down

Fortran 90 Control Structures 3/5

DO loops - (Deterministic DO loop)

If step is positive

- The control-var receives the value of initial
- If the value of the control-var is less than or equal to the value of final, the statements part is executed. Then, the value of step is added to control-var, and goes back and compares the value of control-var and final.
- If the value of control-var is greater than the value of final, the DO-loop completes and the statement following END DO is executed.

Fortran 90 Control Structures 4/5

DO loops - (Deterministic DO loop)

If step is negative

- The control-var receives the value of initial
- If the value of control-var is greater than or equal to the value of final, the statements part is executed. Then, the value of step is added to control-var, goes back and compares the value of control-var and final.
- If the value of control-var is less than the value of final, the DO-loop completes and the statement following END DO is executed.

Fortran 90 Control Structures

DO loops - (Non-deterministic DO loop)

The number of repetitions is not stated in advance. The enclosed section is repeated until some condition is or is not met. Note that this loop can be infinite without the condition.
This may be done in two alternative ways:

- The first requires a logical reason for stopping looping
- The second requires a logical reason for cycling looping.

Non-deterministic DO-loops are particularly good for:

1. summing power series (looping stops when the absolute value of a term is less than some given tolerance)
2. single-point iteration (looping stops when the change is less than a given tolerance)

Fortran 90 Control Structures 3/5

Non-deterministic Do loop - (with Exit condition)

In these, the loop is terminated by a conditional exit statement, ie. an IF statement which executes exit action.
The syntax is as follows:

```
DO
  < statement body >
  IF (< logical-expression > ) EXIT
  < statement body >
END DO
```

The loop continues until some logical expression evaluates as TRUE. Then it jumps out of the loop and continues with the code after the loop. In this form a TRUE result tells you when to stop looping.

Fortran 90 Control Structures

Non-deterministic Do loop - (with Cycle condition)

The CYCLE statement starts the next iteration (ie. executing statements again).
The syntax is as follows:

```
DO
  < statement body >
  IF (< logical-expression > ) CYCLE
  < statement body >
END DO
```

The action of the IF statement if it is TRUE is to cause the program to cycle immediately to the next iteration of the loop, skipping any remaining statements in the current cycle.

Fortran 90 Control Structures

Non-deterministic Do loop - (Exit and Cycle Example)

```
PROGRAM exit_cycle_exam
  IMPLICIT NONE
  INTEGER :: i
  INTEGER, PARAMETER :: istart = 1, iend = 5, ilimit=3
  DO i = istart, iend
   IF (i == ilimit) EXIT
   PRINT*, i
  END DO
END PROGRAM exit_cycle_exam
```

Please re-run the program by replacing the EXIT with CYCLE and exam the difference in the output.

Fortran 90 Control Structures

Nested DO Loops

If one loop is completely inside another one, the two loops are called nested loops.
For each control-var of the outer loop, the entire inner loop has to be completed before next iteration in the outer loop.
When Fortran compiler encounters an END DO statement, it associates that with the innermost currently open loop. Therefore, the first END DO statement closes the inner loop and the last the outermost loop.

Fortran 90 Control Structures

Nested DO Loops -Example

```
PROGRAM nest_exam
 IMPLICIT NONE
 INTEGER :: x, y
  INTEGER, PARAMETER :: xstart = 1, xend = 2
  INTEGER, PARAMETER :: ystart=1, yend=4
  REAL :: results
  PRINT*,' x y results'
  DO x = xstart, xend
   DO y = ystart, yend
    result = (x*1.0)/(y*1.0)
    PRINT*, x, y, result
   END DO
  END DO
PROGRAM nest_exam
```

Fortran 90 Control Structures 4/5

DO WHILE construct

The DO WHILE loop always exits at the top of the loop construct as this is where the test takes place.
The syntax is as follows:

```
DO WHILE (< logical-expression >)
  < body statements >
END DO
```

The body statements in the DO WHILE block are executed for as long as the result of the logical-expression remains TRUE.

Fortran 90 Control Structures

SELECT CASE Construct

The SELECT CASE statement chooses which operations to perform, depending on the set of outcomes (selector) of an expression than can be integer, character or logical, but not real. The general form is:

```
SELECT CASE (expression)
  CASE (case_selector_1)
    < statements_1 >
  CASE (case_selector_2)
    < statements_2 >
  CASE DEFAULT
    < statements_default >
END SELECT
```

Fortran 90 Control Structures 4/5

SELECT CASE Construct

If the expression is in the range of values included in case_selector_1, then $<$ statements_1 $>$ is executed and then move to the next case_selector to execute statements in that block. The CASE DEFAULT (optional)is used whenever the value expression is outside of all the case selector.
The SELECT CASE construct is alternative to an IF-ELSE IF - END IF construct.

Fortran Program Units

- A fortran program can be built up from a collection of program units
- There are four types of program unit in the Fortran
  1. Main program
  2. Functions
  3. Subroutines
  4. Modules
- Each program must contain one (and only one) main program
- A subprogram or procedure (functions and subroutines) are computations that can be "called" (invoked) from the main program
- Relevant functions and subroutines can be put together into a module.

Fortran Subprogram 1

- Fortran 90 has two types of subprograms, functions and subroutines.
- Subprograms are independent blocks of Fortran which perform a specific task.
- They may be written and compiled separately, and are joined to a main program when the program is linked.
- One main advantage of subprograms is that they may be written and tested as isolated units to ensure that they operate correctly.
- Another advantage of subprograms is that they are portable, i.e it can be use in later programs whenever you need it.
- Functions and subroutines may be internal (i.e. contained within and only accessible to one particular program unit) or external (and accessible to all).

Fortran Subprogram 2

Functions syntax 1

- Function returns a single computed result via the function name.
- A Fortran function, or function subprogram has the following syntax:

```
type FUNCTION function-name (arg1, arg2, . . ., argn)
   IMPLICIT NONE
   [ declaration part ]
   [ execution part ]
END FUNCTION function-name
```

- type is a Fortran 90 type (e.g., INTEGER, REAL, LOGICAL, etc).
- function-name is a Fortran 90 identifier.
- arg1, ..., argn are dummy arguments

Fortran Subprogram 3

Functions syntax 2

- Function receives inputs from the outside world via its dummy arguments, does some computations, and returns the result with the function-name
- The function-name must appear in one or more assignment statement like this:

  $$\text{function-name} = \text{expression}$$

  where the result of expression is saved to the name of the function
- Note that function-name cannot appear in the right-hand side of any expression.
- The variables listed in the argument list in the function definition are called dummy variables. Their names do not have match the names used in parts of the program where the function is called.

Fortran Subprogram 4

Functions syntax 3

Note that a function without dummy arguments is declared with empty parentheses ( ) e.g

```
type FUNCTION function-name ( )
   IMPLICIT NONE
   [ declaration part ]
   [ execution part ]
END FUNCTION function-name
```

- If the function is written in the same file as the main program, it must appear after the end of the main program.
- The arguments used when the function is actually invoked are called actual arguments. They may be variables (e.g. x, y), constants (e.g. 1.0, 2.0) or expressions (e.g. 3.0 + x, 2.0/y), but they must be of the same type and number as the dummy arguments.

Recursive Functions syntax 3

- A function that calls itself (directly or indirectly) is called a recursive function.
- If the function is recursive then the FUNCTION keyword in the top line must be proceeded by the keyword RECURSIVE.
- Its declaration is of the form:

```
type RECURSIVE FUNCTION function-name (arg1, . . . , argn)
  IMPLICIT NONE
  [ declaration part ]
  [ execution part ]
END FUNCTION function-name
```

Functions Intent Declaration 4

- It is a good practice to declare whether dummy arguments are intended as input or output using the INTENT attribute.
- The intent of a variable in a function/subprogram can have one of three values: in, out or inout
  INTENT(IN) - The dummy argument is used only to pass input data to the subprogram and its content does not change.
  INTENT(OUT) - The dummy argument is a variable and is used only to return results to the main program when the subprogram is invoked.
  INTENT(INOUT) - The dummy argument is used both to pass input data to the subprogram and also to return results to the main program when he subprogram is invoked.
- Us the intent attribute to help make sure that variables cannot be changed accidentally in a function.

Fortran Subprogram 6

Subroutine syntax 1

Subroutine is used in computations that involves the return of several values of different types. It takes values from its formal arguments, and returns some computed results with its formal arguments.
Subroutine syntax is of the form:

```
SUBROUTINE subroutine-name (arg1,arg2, ..., argn)
 IMPLICIT NONE
 [ declaration part ]
 [ execution part ]
END SUBROUTINE subroutine-name
```

If a subroutine does not require any dummy arguments, the first line is written with or without empty parentheses ( ) but the keyword subroutine and its name should be there.

Fortran Subprogram 7

To call Subroutine 2

- The subroutine is invoked by the CALL statement in the main program
- The CALL statement may have one of the three forms:
  1. CALL subroutine-name (arg1, arg2, ..., argn)
  2. item CALL subroutine-name ( )
  3. CALL subroutine-name
- The last two forms are equivalent and are used for calling subroutine without dummy arguments.
- The order and type of the actual arguments in the arguments list in the CALL statement must match the order and type of the dummy arguments declared in the subroutine.

Fortran Subprogram 7

Recursive Subroutine 3

- It is permissible for subroutine also to invoke itself.
- recursive subroutine must have the keyword recursive it its first line.
- Its declaration is of the form:

```
RECURSIVE SUBROUTINE subroutine-name (arg1,arg2, ..., argn)
 IMPLICIT NONE
 [ declaration part ]
 [ execution part ]
END SUBROUTINE subroutine-name
```

Fortran 90 Modules 1

- Modules exist so that anything required by more than one program unit may be package in a module and made available were needed.
- They are designed to hold declaration of variables, data and subprograms which are to be made available to other program unit.
- A program may use any number of modules, with the restriction that each must be named separately.
- A module looks like the main program, except that it does not have the executable part. Hence, a main program must be there to use modules
- A module consists of two part: a specification part for the declaration statements, and a subprogram part.

Fortran 90 Modules 2

Module Syntax

- Fortran 90 module has the following syntax:

```
MODULE module-name
  IMPLICIT NONE
  [ Declaration part ]
CONTAINS
  [ internal functions/subroutines ]
END MODULE module-name
```

- Modules can contain just the declaration part or the subprogram part, or both.
- Any program unit that makes use of the module's contents should include the following statement immediately after the program name or other program unit statement:

```
USE module-name
```

Fortran 90 Modules

Some features of Module

- After USE module-name statement, the variables and subprograms contained in the module-name can be used in the program unit.
- Note that it is not necessary to declare the module variables in the main program. The USE statement is sufficient.
- If a module is contained in the same file as the main program, it must come before the main program.
- Modules cannot USE themselves either directly (module A uses A) or indirectly (module A uses module B which uses module A)
- To restrict program unit to access only variables that it requires from module, the ONLY qualifier is used. e.g
  USE module-name, ONLY: var1, var2
  Only var1 and var2 of the module variables are accessible.

Fortran 90 Modules 4

Some features of Module

- The (USE module-name, ONLY) statement safeguards the module elements that are not needed by making them inaccessible to the program.
- It can also make programs more transparent, by showing the origin of the dtatobjects or subprograms, particularly if the program uses several modules.
- If the name of an item that is accessed from a module conflicts with another name elsewhere, one may use the "renaming" feature of USE.
- For each identifier in USE to be renamed, the the syntax: USE module-name, name-in-program => name-in-module
- In this program, the use of name-in-program is equivalent to the use of name-in-module in the module

Fortran 90 Modules 5

Module accessibility - Public and Private

- By default, everything in a module is publicly available, that is, the USE statement in the main program makes available all of the code in the module
- PUBLIC and/or PRIVATE statements (or attributes) may be used to restrict access to some variables, declaration statement or subprograms in a module
- Everything that is declared PRIVATE is not available outside the module unit, PUBLIC is the opposite.
- As a statement PUBLIC or PRIVATE can set the default for the module, or can be applied to a list of variables or module names.
- As an attribute PUBLIC or PRIVATE can control access to the variables in a declaration list.

Module accessibility - Public and Private

- To specify PRIVATE and PUBLIC, do the following:
  PUBLIC :: name-1, name-2, ..., name-n
  PRIVATE :: name-1, name-2, ..., name-n
- The PRIVATE statement without a name makes all entities in a module private. To make some entities visible outside the module declare them as PUBLIC.
- PUBLIC and PRIVATE may also used in the declaration statement. e.g
  INTEGER, PRIVATE :: sum1, avg1
  where sum1 and avg1 are visible within the module
  INTEGER, PUBLIC :: sum2, avg2
  where sum2 and avg2 are available within/outside the module

Fortran 90 Modules 6

Compiling Programs with Module

- The module may be in the same source file as other units or it may be in a different file.
- To enable compiler to operate correctly, the module must be compiled before any program units that USE it. Hence,
  1. if it is in a different file the module must be compiled first;
  2. if it is in the same file the module must come before any program units that USE it.
- Suppose a program consists of the main program mypro.f90 and a module mymod.f90, then compilation syntax is as follows:

  ```
  compiler_name -o mypro mymod.f90  mypro.f90
  ```
- Run the mypro executable file (./mypro.exe) to get the output

Compiling Programs with Module

- Since modules are supposed to be designed and developed separately, they can also be compiled separately to object codes.
- Compile the module with the syntax:
  `compiler_name -c mymod.f90`
- The -c option causes the file to be compiled but not linked; this creates mymod.o and mymod.mod files.
- Compile the main program with the syntax:
  `compiler_name -c mypro.f90`
- Once the two parts have compiled successfully, link them together with the command:
  `compiler_name -o mypro mymod.f90  mypro.f90`
- Run the mypro executable file (./mypro.exe) to get the output