Handout Four

October 15, 2013

1 Characters & Strings in Fortran

1.1 Declaration

In handout two of your notes (section 2.3) you were briefly shown how to declare various character types. A table similar to the following was given as correct examples of character declarations. There are two types of character declarations, a 'single' character or a 'string' of characters. The 'string' of characters is achieved by including the ('LEN') length specifier to the declaration statement. You should also be aware that when dealing with 'CHARACTER' types the single quote character (')or the double quote character (") can be used as the delimiters for the character type.

```
1. CHARACTER :: char1,char2,letter="I"
2. CHARACTER, PARAMETER :: a="a",b="b",c="c"
3. CHARACTER (LEN=6) :: class1="MT3501", class2="MT3502",class3="MT3503"
4. CHARACTER (LEN=*), PARAMETER :: headline="Wigan won the Premier League?"
5. CHARACTER (LEN=5), PARAMETER :: name="Sarah"
```

- The first declaration declares three single 'CHARACTER' variables the third of which is initialised to be the letter 'I'.
- The second declaration statement declares three single 'CHARACTER' objects and declares them with the 'PARAMETER' attribute setting them to the first three letters of the alphabet. Remember data types declared with the 'PARAMETER' attribute can not have there contents changed anywhere in the program, often referred to as constants.
- The third declaration statement declares and initialises three 'CHARACTER' string variables of length six characters.
- The fourth declaration declares a 'CHARACTER' string parameter. The use of the asterisk denotes that the length of the string parameter will be determined by the compiler. This is useful as you do not have to count the characters in the string and explicitly state the number in the declaration.
- The fifth and final declaration is also a 'CHARACTER' string parameter but the length of the string is specified as five and the length not determined by the compiler (note an asterisk could have also been used here instead of the nuber '5').

1.2 Concatenation & substrings

In Fortran there is only 'one' intrinsic operator for the 'CHARACTER' type and it is dyadic (requires two operands). The operator is the 'concatenation' operator '//' (two front slashes) and is used to append one 'CHARACTER' type to another ie. `"abc"//"def"` takes the string `"def"` and appends it to the string `"abc"` to give a single string `"abcdef"`. This is made clear in the next example code.
'Substrings' can also be extracted from strings by indexing the strings through a pair of appended brackets. So consider the example declaration 'four' above for the string parameter 'headline'. Then '`headline(5:10)`' would reference the fifth to the tenth (inclusive) characters of the string which would

be ‘`United`’. The colon can be seen as globally referencing all possible characters in the extent of the string but is then limited by the integer delimiters ‘5’ and ‘10’. Consider the next bit of code, what do you think will be printed out to screen? Try running it to check if you were correct.

Exercise One : Consider the next bit of code. what do you think will be printed out to screen? Type it up and run it to check if you were correct.

```
PROGRAM characters

  IMPLICIT NONE

  CHARACTER (LEN=*), PARAMETER :: headline="Man United will win the league?"
  CHARACTER (LEN=*), PARAMETER :: fname="Steve", lname="Smith"
  CHARACTER (LEN=11) :: fullname

  ! *** Example of concatenation of two strings ***
  fullname=fname//lname
  PRINT*,fullname

  !*** Concatenation of a string a character and a string ***
  fullname=fname//" "//lname
  PRINT*,fullname

  ! *** Example of a substring ***
  PRINT*,headline(5:10)

END PROGRAM characters
```

1.3 Character intrinsic functions

Fortran has a set of intrinsic functions for dealing with ‘`CHARACTER`’ data types. Described below are some you may find useful.

- ‘`str1=TRIM(str2)`’ : Inputs a string ‘`str2`’ and returns a string ‘`str1`’ which is the same as the input string but with any trailing blanks removed. ie `TRIM("Hello    ")` would return `"Hello"`. .

Exercise Two : Consider the following code. What do you think will be printed out to screen? Type it up and run it to check if you were correct.

```
PROGRAM characters2

  IMPLICIT NONE

  CHARACTER (LEN=*), PARAMETER :: fname="Wayne", lname="Rooney"
  CHARACTER (LEN=20) :: fullname !** NOTE 20 characters!!!

  fullname=fname//" "//lname !** Concatenation
  PRINT*,fullname," will score in Euro-2016!"
  PRINT*,TRIM(fullname)," will score in Euro-2016!"

END PROGRAM characters2
```

In the above exercise the character string variable '`fullname`' is declared to have a length of '20' characters but the name '`"Wayne Rooney"`' uses only twelve of these characters, the eight remaining characters in '`fullname`' are trailing blanks but will still be printed to the screen unless removed with the '`TRIM`' function.

- '`str1=ADJUSTL(str2)`' : Inputs a string '`str2`' and returns a string '`str1`' which is the same as the input string but with any 'leading blanks' removed and appended as 'trailing blanks'.

.

Exercise Three : Consider the following code. What do you think will be printed out to screen? Type it up and run it to check if you were correct.

```
PROGRAM characters3

  IMPLICIT NONE

  CHARACTER (LEN=*), PARAMETER :: fname="     Paul", lname="Scholes"
  CHARACTER (LEN=20) :: fullname

  fullname=fname//" "//lname
  PRINT*,fullname," will score in Euro-2004!"
  PRINT*,TRIM(fullname)," will score in Euro-2004!"
  PRINT*,TRIM(ADJUSTL(fullname))," will score in Euro-2004!"

END PROGRAM characters3
```

In the above exercise the string '`fname`' has been declared with a series of 'leading blanks'. The 'nested' function calls of '`ADJUSTL`' and '`TRIM`' are used in combination to remove all blanks!

- '`str1=ADJUSTR(str2)`' : Inputs a string '`str2`' and returns a string '`str1`' which is the same as the input string but with any 'trailing blanks' removed and inserted as 'leading blanks'.
- '`int=LEN(str2)`' : Inputs a string '`str2`' and returns the length of the string (Including trailing blanks) as an integer.
- '`int=LEN_TRIM(str2)`' : Inputs a string '`str2`' and returns the length of the string (Excluding trailing blanks) as an integer and will have the same effect as the statement '`int=LEN(TRIM(str))`'.
- '`int=INDEX(str1, str2)`' : Inputs two strings '`str1`' & '`str2`'. The function checks to see if '`str2`' is a 'substring' of '`str1`'. If '`str2`' is a substring of '`str1`' then '`INDEX`' returns an integer which is the starting index of '`str2`' in '`str1`' else it returns the integer zero.

Other intrinsic functions, for string & character manipulation, exist in Fortran but are not needed for this course.

2 The '`PRINT`' Statement In Detail

So far some of your programs will have produced a rudimentary table of numbers. You can write programs that produce better looking output by using some of the formatting features of the '`PRINT`' statement. In all the examples so far you have created output by using the command '`PRINT*`'. The asterisk tells the code to write the output using Fortran's default formatting. This is called '*free format*' output. The default formatting rules are not always the best way to present the output. It would therefore be useful to know how to control the output format. You can replace Fortran's default '*free format*' formatting with your own by replacing the asterisk with a 'formatting string'. There are a lot of options available for formatting output.

2.1 Descriptor Symbols

A formatting string is built up of descriptor statements and/or sub-strings to be written to the output device, which is by default the screen. A descriptor statement is a group of characters that allow you format a particular data item. You can tailor the structure of the data output to your taste to improve the visual appearance. For the purpose of this course the output will be to the screen or to a text file. The common descriptor symbols are given in the table below.

- '`d`' - Number of digits to the right of the decimal place in a '`REAL`' number.
- '`m`' - Minimum number of 'digits' to be displayed. Padded out with zeros.
- '`n`' - Number of spaces to skip
- '`r`' - **Repeat Count** No of times a descriptor is to be repeated
- '`w`' - **Field Width** Number of characters wide to use for data

2.2 The '`INTEGER - I -`' Descriptor

This takes the form '`rIw.m`' where the meanings of `r`, `w` and `m` are given in section 2.1. Integer values are right justified in their fields. If the field width is not large enough to accommodate an integer then the field is filled with asterisks. Not all the descriptor symbols have to be used in a descriptor statement. Typical usage would be in a statement like:

```
PRINT "(3I6)",i,j,k
```

Which would print each of the '`INTEGERS`' i,j and k in fields of width six characters.

2.3 The '`REAL - F -`' Descriptor

This takes the form '`rFw.d`' where the meanings of `r`, `w` and `d` are given in section 2.1. Real values are right justified in their fields. If the field width is not large enough to accommodate the real number then the field is filled with asterisks. Not all the descriptor symbols have to be used in a descriptor statement. If a '`d`' symbol is given and then the real number will be rounded off if necessary (not truncated) or padded with zeros if there are not enough digits. Typical usage would be in a statement like:

```
PRINT "(F12.3)",pi
```

Which would print the constant 'pi' in a field of width 12 characters with three decimal places.

2.4 The '`REAL - E -`' Descriptor

Real numbers in fortran can be displayed in exponential notation. This is where a number is displayed as a value between 0.1 and 1 (with appropriate sign) and multiplied by a power of 10. The '`E`' descriptor statement takes the form '`rEw.d`' where the meanings of `r`, `w` and `d` are given in section 2.1. Real values are right justified in their fields. If the field width is not large enough to accommodate the real number then the field is filled with asterisks. Not all the descriptor symbols have to be used in a descriptor statement. If a '`d`' symbol is given and then the real number will be rounded off if necessary (not truncated) or padded with zeros if there are not enough digits. You have to be more careful when working with '`E`' descriptors regarding the '`w`' width symbol. For example to print out a real with four decimal places a field width of at least eleven is needed. One for the sign of the mantissa, two for the zero, four for the mantissa and four for the exponent itself ie.

```
-0.xxxxE+xx
```

In general the width field must satisfy the expression

$$w \geq d + 7$$

Typical usage would be in a statement like:

```
PRINT "(E10.3)",123456.0
```

which gives

'0.123E+06'

2.5 The 'REAL - ES -' Descriptor

This takes the form 'rESw.d' where the meanings of r, w and d are given in section 2.1. The 'E' descriptor described above differs slightly from the traditional well known 'scientific notation'. Scientific notation has the mantissa in the range 1.0 to 10.0 unlike the E descriptior which has the mantissa in the range 0.1 to 1.0. Real values are right justified in their fields. If the field width is not large enough to accommodate the real number then the field is filled with asterisks. Not all the descriptor symbols have to be used in a descriptor statement. If a 'd' symbol is given and then the real number will be rounded off if necessary (not truncated) or padded with zeros if there are not enough digits. You have to be more careful when working with 'ES' descriptors regarding the 'w' width symbol. In general the width field must satisfy the expression

$$w \geq d + 7$$

Typical usage would be in a statement like:

```
PRINT "(ES10.3)",123456.0
```

which gives

'1.235E+05'

2.6 The 'CHARACTER - A -' Descriptor

This takes the form 'rAw' where the meanings of r and w are given in section 2.1. Character types are right justified in their fields. If the field width is not large enough to accommodate the character string then the field is filled with the first 'w' characters of the string. Not all the descriptor symbols have to be used in a descriptor statement. Typical usage would be in a statement like:

```
PRINT "(a10)",str
```

Which would print out the character string 'str' in a field width of 10 characters.

2.7 The 'Space - X -' Descriptor

This takes the form 'nX' where 'n' is the number of desired spaces.
Typical usage would be in a statement like:

```
PRINT "(5X, a10)",str
```

Which would print out five blank spaces then the character string 'str' in a field width of 10 characters.

2.8 The 'Newline - / -' Descriptor

This takes the form '/' and forces the next data output to be on a new line. Typical usage would be in a statement like: `PRINT "(/,5X, a10)",str`

Which would print out a blank line then five blank spaces then the character string 'str' in a field width of 10 characters. NOTE how descriptor statements are separated with commas in the format string!!!

’

Exercise Four : Type in the example code below and work out what is happening with the help of the explanations in the bullet points following the code.

```
PROGRAM output_formats

  IMPLICIT NONE

  REAL :: c = 1.2786453e-8, d = 0.6574893e2
  INTEGER :: n = 200289, k = 45, i = 2
  CHARACTER (LEN=5) :: str="Hello"

  !*** Example of PRINT statements. Explanations in main text below.
  !*** Note the next to lines are to help you judge which column
  !*** the output is in on the screen.

  PRINT "('    5   10   15   20   25   30   35')"
  PRINT "('----|----|----|----|----|----|----|')"

  PRINT "(i6)", k
  PRINT "(i6.3)", k
  PRINT "(3i10)", n, k, i
  PRINT "(i10,i3,i5)", n, k, i
  PRINT "(a10)",str
  PRINT "(f12.3)", d

  PRINT "('----|----|----|----|----|----|----|')"

  PRINT "(e12.4)", c
  PRINT '(/,3x,"n = ",i6, 3x, "d = ",f7.4)', n, d


END PROGRAM output_formats
```

Note that ALL of the formatting information is enclosed in parentheses and then held in a string.

- "`(i6)`" means that an integer is going to be written and 6 columns are set aside to write the integer into. Always make sure that you have enough columns set aside for the integer to fit into and note that if the integer uses less than six columns then it is right justified, that is its right most digit is in the sixth column!
- "`(3i10)`" means that 3 integers are going to be written and each is going to have 10 columns set aside for it. Since none of the integers are that long each is placed as far right in these 10 columns as they can go i.e. right justified.
- "`(i10, i3, i5)`" means that the first integer is going to be placed into 10 columns, the second into 3 and the last into 5.

- "`(a10)`" means '`CHARACTER`' data is to be output in a right justified field of 10 columns.
- "`(f12.3)`" : here the 'f' means that it's a floating point number i.e. a real number. 12 columns will be allocated for the display of the number in its natural form with 3 digits after the decimal place e.g. 0.6574893e2 would be written as 65.748. The number is truncated after the 3rd decimal place.
- "`(e12.4)`" : the `e` also means that the number is a real number but this time the output will be in *exponential* format. This means that the output will be in 12 columns and to 4 significant figures e.g. 1.2786453e-8 is output as 0.1279E-07. Note that here although there are only 4 significant figures in the mantissa a minimum of 10 columns are needed.
- `'( /,3x,"n = ",i10, 3x, "d = ",f12.4)'` : here the '/' (front-slash) means start a new line before outputting the next item. Both `3x`'s mean leave 3 blank spaces. The `i10, f12.4` have been explained above. Anything enclosed by `" "` in a format statement is written to the output exactly as written. Note that the symbol `"` is not the same as typing `'` twice but is a separate character. You get this symbol by holding down the shift (⇑) key and pressing the quotes key (top left of the shift key). As a result the last '`PRINT`' statement will produce `n = 200289 d = 65.7489`.

Common errors are not allowing enough columns and trying to write a real number using an integer format etc. so be careful about these points.

Exercise Five : : Copy your McLaurin ($\sin(x)$) series code into your '`handout4/exercise5`'. Change it so that it prints out at each iteration the number of terms used so far, the approximation and the error in the approximation. Also before any iterations, print out headings for the columns. Your output should look similar to the output below.

```
 Enter the value of x you require:
3.0
 Enter the number of terms you require:
10

Terms      Approx     Error

  1      3.000000   2.85888
  2     -1.500000   1.64112
  3      0.525000   0.38388
  4      0.091071   0.05005
  5      0.145313   0.00419
  6      0.140875   0.00025
  7      0.141131   0.00001
  8      0.141120   0.00000
  9      0.141120   0.00000
 10      0.141120   0.00000

 The approximation is for x =  3.000000
 Number of terms in the approximation =  10
 The approximation =   0.141120
 The true value is =   0.141120
```