# Introduction to Modern Fortran

## *Fortran Language Rules*

Nick Maclaren

**nmm1@cam.ac.uk**

March 2014

## Coverage

This course is modern, free–format source only
[ If you don't understand this, don't worry ]
The same applies to features covered later

Almost all old Fortran remains legal
Avoid using it, as modern Fortran is better
This mentions old Fortran only in passing

See the OldFortran course for those aspects
It describes fixed–format and conversion
Or ask questions or for help on such things, too

## Important Warning

Fortran's syntax is verbose and horrible
It can fairly be described as a historical mess
Its semantics are fairly clean and consistent

Its verbosity causes problems for examples
Many of them use poor style, to be readable
And they mostly omit essential error checking

- Do what I say, don't do what I do

Sorry about that . . .

## Correctness

Humans understad linguage quite well even when it isnt stroctly correc

Computers (i.e. compilers) are not so forgiving

- Programs must follow the rules to the letter

- Fortran compilers will flag all syntax errors

Good compilers will detect more than is required

But your error may just change the meaning
Or do something invalid (“undefined behaviour”)

## Examples of Errors

Consider (N*M/1024+5)

If you mistype the ‘0’ as a ‘)’: (N*M/1)24+5)

You will get an error message when compiling

It may be confusing, but will point out a problem

If you mistype the ‘0’ as a ‘–’: (N*M/1–24+5)

You will simply evaluate a different formula

And get wrong answers with no error message

And if you mistype ‘*’ as ‘8’?

## Character Set

Letters (A to Z and a to z) and digits (0 to 9)
Letters are matched ignoring their case

And the following special characters
_ = + − * / ( ) , . ’ : ! " % & ; < > ? $
Plus space (i.e. a blank), but not tab
The end–of–line indicator is not a character

Any character allowed in comments and strings

- Case is significant in strings, and only there

## Special Characters

_ = + − * / ( ) , . ’ : ! " % & ; < > ? $

slash (/) is also used for divide
hyphen (–) is also used for minus
asterisk (*) is also used for multiply
apostrophe (’) is used for single quote
period (.) is also used for decimal point

The others are described when we use them

## Layout

- Do not use tab, form–feed etc. in your source
Use no positioning except space and line breaks

Compilers do bizarre things with anything else
Will work with some compilers but not others
And can produce some very strange output

Even in C, using them is a recipe for confusion
The really masochistic should ask me offline

## Source Form (1)

Spaces are not allowed in keywords or names
INTEGER is not the same as INT EGER

HOURS is the same as hoURs or hours
But not HO URS – that means HO and URS

- Some keywords can have two forms

E.g. ENDDO is the same as END DO
But EN DDO is treated as EN and DDO

⇒ END DO etc. is the direction Fortran is going

# Source Form (2)

- Do not run keywords and names together

```
INTEGERI,J,K       –    illegal
INTEGER   I,J,K    –    allowed
```

- You can use spaces liberally for clarity

```
INTEGER   I   ,   J   ,   K
```

Exactly where you use them is a matter of taste

- Blank lines can be used in the same way

Or lines consisting only of comments

# Double Colons

For descriptive names use underscore
largest_of, maximum_value or P12_56

- Best to use a double colon in declarations

Separates type specification from names

```
    INTEGER :: I, J, K
```

This form is essential where attributes are used

```
    INTEGER, INTENT(IN) :: I, J, K
```

## Lines and Comments

A line is a sequence of up to 132 characters

A comment is from ! to the end of line
The whole of a comment is totally ignored

```
    A = A+1    ! These characters are ignored
    ! That applies to !, & and ; too
```

Blank lines are completely ignored

```
    !
    ! Including ones that are just comments
    !
```

## Use of Layout

Well laid–out programs are much more readable
You are less likely to make trivial mistakes
And much more likely to spot them!

This also applies to low–level formats, too
E.g. 1.0e6 is clearer than 1.e6 or .1e7

- None of this is Fortran–specific

## Use of Comments

Appropriate commenting is very important
This course does not cover that topic
And, often, comments are omitted for brevity

“How to Help Programs Debug Themselves”
Gives guidelines on how best to use comments

- This isn’t Fortran–specific, either

## Use of Case

- Now, this IS Fortran–specific!

It doesn't matter what case convention you use

- But DO be moderately† consistent!

Very important for clarity and editing/searching

For example:
UPPER case for keywords, lower for names
You may prefer Capitalised names

† *A foolish consistency is the hobgoblin of little minds*

## Statements and Continuation

- A program is a sequence of statements
Used to build high–level constructs
Statements are made up out of lines

- Statements are continued by appending &

```
    A = B + C + D + E + &
        F + G + H
```

Is equivalent to

```
    A = B + C + D + E + F + G + H
```

## Other Rules (1)

Statements can start at any position

- Use indentation to clarify your code

```
IF (a > 1.0) THEN
    b = 3.0
ELSE
    b = 2.0
END IF
```

- A number starting a statement is a label

```
10 A = B + C
```

The use of labels is described later

## Other Rules (2)

You can put multiple statements on a line

```
    a = 3 ;   b = 4 ;    c = 5
```

Overusing that can make a program unreadable
But it can clarify your code in some cases

Avoid mixing continuation with that or comments
It works, but can make code very hard to read

```
    a = b + c ; d = e + f + &
        g + h
    a = b + c + & ! More coming ...
        d = e + f + g + h
```

## Breaking Character Strings

- Continuation lines can start with an &

Preceding spaces and the & are suppressed

The following works and allows indentation:

```
        PRINT 'Assume that this string &
                &is far too long and complic&
                &ated to fit on a single line'
```

The initial & avoids including excess spaces
And avoids problems if the text starts with !

This may also be used to continue any line

## Names

Up to 31 (now 63) letters, digits and underscores

- Names must start with a letter

Upper and lower case are equivalent
DEPTH, Depth and depth are the same name

The following are valid Fortran names

A, AA, aaa, Tax, INCOME, Num1, NUM2, NUM333,
N12MO5, atmospheric_pressure, Line_Colour,
R2D2, A_21_173_5a

## Invalid Names

The following are invalid names

| | |
|---|---|
| 1A | does not begin with a letter |
| _B | does not begin with a letter |
| Depth$0 | contains an illegal character '$' |
| A−3 | would be interpreted as subtract 3 from A |
| B.5: | illegal characters '.' and ':' |

A_name_made_up_of_more_than_31_letters

too long, 38 characters

## Compiling and Testing

We shall use the gfortran under Linux
PWF/MCS/DS Windows does not have a Fortran
Using any Fortran compiler is much the same

Please ask about anything you don't understand
Feel free to bring problems with other Fortrans
Feel free to use gdb if you know it

Solutions to exercises available from
Fortran/Answers

## Instructions

If running Microsoft Windows, CTRL–ALT–DEL
Select Restart and then Linux
Log into Linux and start a shell and an editor
Create programs called prog.f90, fred.f90 etc.

- Run by typing commands like

```
nagfor –C=all –o fred fred.f90
./fred
```

- Analyse what went wrong
- Fix bugs and retry

## Instructions

- Run by typing commands like

```
gfortran -g -O3 -Wall -Wextra -o fred fred.f90
./fred
```

- Analyse what went wrong
- Fix bugs and retry