

ATSC 212

# FORTRAN - part 1

Roland Stull  
[rstull@eos.ubc.ca](mailto:rstull@eos.ubc.ca)



1

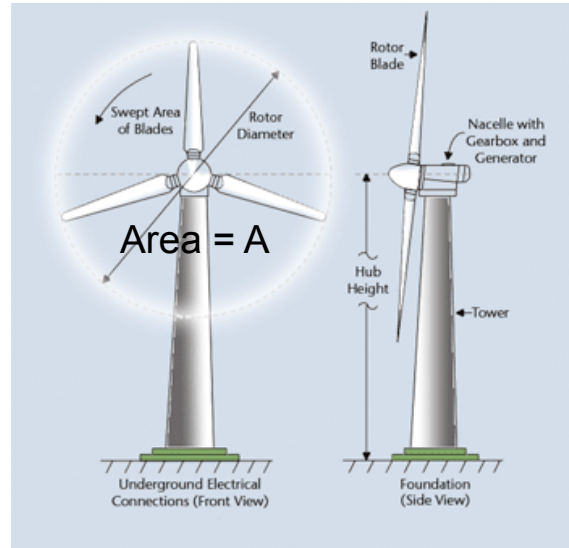
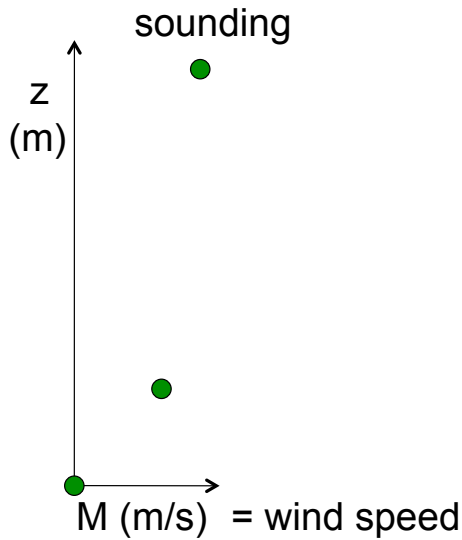
## Goals for Today's Lab

- Intro to evolution of fortran
- Learn the “emacs” text editor.
- Write & run a simple fortran program -- the start of a larger program to calculate wind power.
- Learn about compiler error messages, and tips for debugging.
- Get experience with fortran syntax and control structures.
- Learn & use version control.
- Learn & use top-down programming.



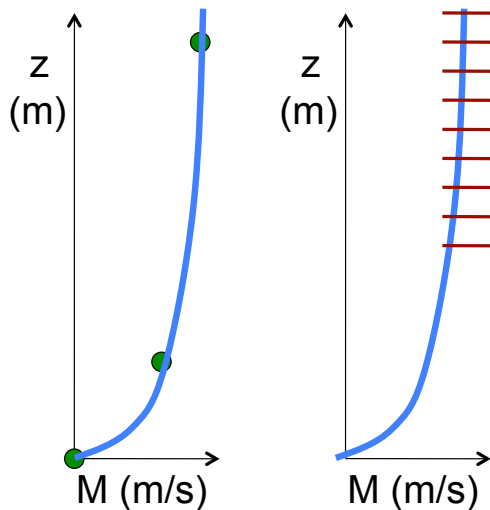
2

# Discussion on blackboard of plan of attack for the wind power calculation = $0.5 \cdot \rho \cdot A \cdot M^3$

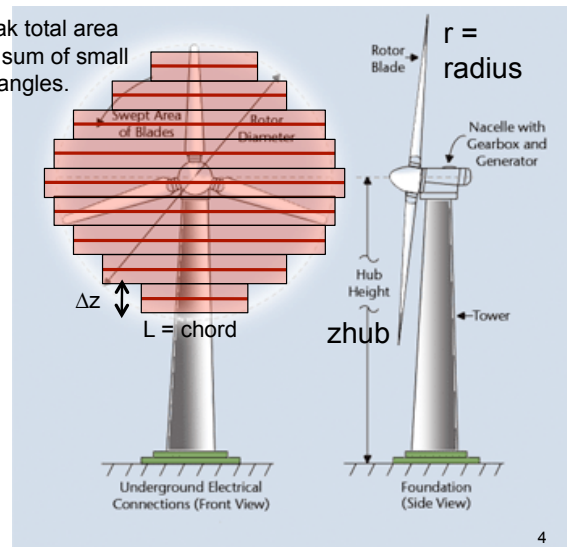


Drawing of the rotor and blades of a wind turbine, courtesy of ESN

# Discussion on blackboard of plan of attack for the wind power calculation = $0.5 \cdot \rho \cdot A \cdot M^3$



Break total area into sum of small rectangles.



Drawing of the rotor and blades of a wind turbine, courtesy of ESN



## Useful Formulas:

$$\text{Power (W)} = 0.5 \cdot \rho_{\text{air}}(\text{kg/m}^3) \cdot A(\text{m}^2) \cdot [M(\text{m/s})]^3$$

where A is disk area swept out by the turbine blades

M is wind speed

$\rho_{\text{air}}$  is air density.

$$\rho_{\text{air}} = \rho_0 \cdot e^{-z/H}$$

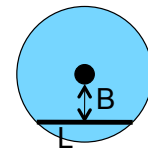
where  $\rho_0 = 1.225 \text{ kg/m}^3$ , and scale height  $H = 8550 \text{ m}$

The length L of a circle's chord that is distance B

from the circle center is:  $L = 2 \cdot [r^2 - B^2]^{1/2}$

where r = circle radius, and  $B = z_{\text{hub}}(\text{agl}) - z(\text{agl})$ .

(agl = above ground level).



5

## FORTTRAN = FORmula TRANslation




- Designed to solve scientific equations.
- Was the first high-level language (HLL). [Reads like English.](#)
- Is independent of the particular processor (i.e., is portable). [Differs from Assembly Language.](#)
- Is an “imperative” language. [Do this. Then do that.](#)
- Is a “procedural” language. [Breaks tasks into subroutines & fnts.](#)
- A compiler reads the standard FORTRAN code ([ascii text, written by humans](#)) as input, and produces specialized machine code ([binary](#)) as output. ([Each processor needs a different compiler.](#))
- We then “run” or “execute” the compiled code.
- Modern compilers “optimize” the code. [Make it run fast.](#)

6

# FORTRAN evolution

(see [timeline of programming languages](#))



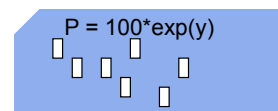
- FORTRAN I (released in 1957 by IBM)
  - FORTRAN II (1958, separate compile & link)
  - FORTRAN IV (1961. Machine independent.)
  - FORTRAN 66 (First ASA standardized version.)
  - FORTRAN 77 (ASA standardized in 1977)
- 
- FORTRAN 90 (a major upgrade. Modern.  Includes array math. Adopted as standard by ANSI and ISO.)
  - FORTRAN 95 (a minor change from F90)
  - FORTRAN 2003 (a moderate upgrade, with oop)
  - FORTRAN 2008 (a minor change; official in 2010)

(see a nice summary in [Wikipedia](#))

7

## The old days...thru F77

- Input was via computer cards.
- Output was to a line printer.
- Batch jobs. Not interactive.
- Programmers served the computer, not the computer serving the programmers.
- Column alignment of code



## The new days... F90/95/2003/2008

8

# FORTRAN Tutorial



- An excellent FORTRAN tutorial was created by Stephen Brooks, Univ. of St. Andrews, Scotland.
- Copies of his tutorials are presented on our web page.
- Google can find other good tutorials, eg: <http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortran.html>
- See the Resources link on our course web page for access to a full FORTRAN language manual.
- Also see Wikipedia "Fortran language features"

9

# Steps in FORTRAN programming



- Design algorithms and program flow (e.g., using a flow chart).
- Write/Edit the FORTRAN code (which is just an ascii text file) on a text editor, & save as a “**source**” code file.
- Compile the source code into binary “**object**” files, by running a FORTRAN compiler program.
- Link the compiled object files to other compiled subroutines or libraries, if needed, to create an “**executable**” binary file. (“Make” files are scripts that tell the computer which compiled files and libraries to combine and link together.)
- Run the resulting executable.

10

# Editing: Program Editors



Some ascii text editors use GUI interface (g) with mouse.

“Program editors” are ascii text editors that can color-code (cc) statements for different programming languages.

Examples of editors for different computer systems:

- [MacOSX](#): TextEdit(g), TextWrangler(g,cc)
- [PC Windows](#): NotePad(g) [DON'T use WordPad]
- [Linux](#): VI(cc), Emacs(g,cc) [We will use both in this course.]

Some commercial software provides a full “programmers environment” (editors, compilers, debuggers, profilers)

- Visual Studio (MicroSoft, for Windows machines)
- CodeWarrior (for many systems)
- Absoft (for Mac, Windows, and linux)

11

# Editing: An example



First, write the “source” code. Follow along with the instructor. Open emacs to do this.

```
! Estimate wind power
program windpowermain
  write(*,*) "Welcome to Wind Power"  !welcome user
end program windpowermain
```

Then, save the file with suffix “.f95”

For example: `wp01.f95`

Notes:

Comments start with an exclamation point.

[Good programming practice to document WHILE you write your code.]

FORTTRAN is case insensitive. X and x are the same variable.

Hit the “Return” or “Enter” key at the end of each line. No special character is used at the end of each line.

Each line can be up to 132 characters long.

If you need a longer line, end the first line with &

& and start the continuation line also with the ampersand.

12

# Compiling: Compiler Programs



- FORTRAN compilers exist for almost all computers, including desktop PCs and Macs.
- Some are VERY expensive, but have VERY nice editing and debugging environments.
- Some free FORTRAN compilers that run on most platforms (linux, Mac, PC) are available:
  - <http://ftp.g95.org/>
  - <http://gcc.gnu.org/wiki/GFortran>  
produced by the GNU organization.
- We will use **gfortran** in this course.

13

# Compiling & Running under linux



Example. You should follow along:

```
> gfortran wp01.f95 -o runwp01 #invoke the compiler
> ./runwp01 #run the executable
Welcome to Wind Power #this is the output
> #the next linux prompt
```

Notes:

“**gfortran**” is the name of the fortran 95 compiler.

It takes the text file “**wp01.f95**” as input.

The “**-o**” option tells the compiler that you will provide a name for the output file.

I have named the output executable file “**runwp01**”.

(Although not needed, some programmers like to name executable files with suffix “.exe”. Such as “**runwp01.exe**”<sup>14</sup> )

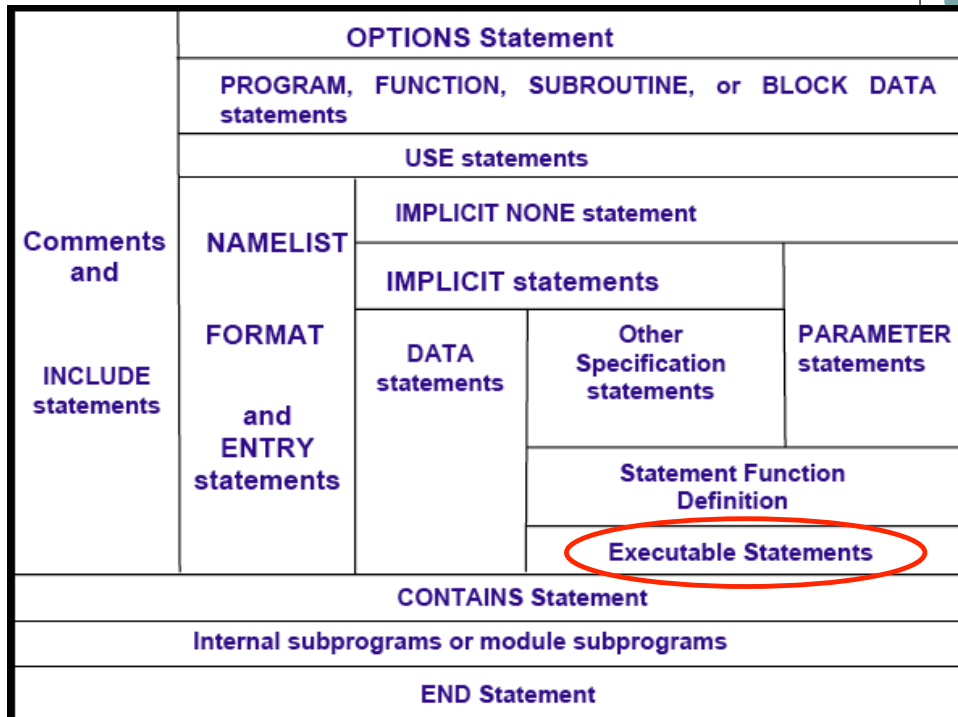
# Some Elements of FORTRAN



- Variables (including array variables)
- Operators (assignment, math, logical)
- Conditionals
- Loops
- Functions & subroutines (& built-in functions)
- I/O: input from keyboard & output to screen
- File Handling

15

# Order of Statements



16



# Variables: Type Declarations



Although FORTRAN does not require that variables be declared before you use them, it is VERY good practice to do so. To enforce such “strong typing” of variables, you should always declare “`implicit none`” first.

Reals are floating point numbers (with a decimal `3.14` and optionally with as scientific notation `8.99E-6` which means  $8.99 \times 10^{-6}$ ).

Integers are whole numbers.

Characters are strings of ascii characters of length 0 or more, in quotes. “`line`”

Logicals are boolean variables such as `.false.` or `.true.`

```
implicit none          !impose strong typing
real :: e              !vapour pressure (kPa)
real :: p = 101.325    !total pressure (kPa), initialized.
real, parameter :: epsilon = 0.622    !constant. Can't change.
integer :: nlevel      !number of sounding levels
character (len=80) :: inputline !string of input characters
logical :: done = .false.    !a flag indicating if done
```

17

# Try it – Type declarations



First, do “save as” with name “wp02.f95”. This allows us to create a new version of the code.

**Version Control** is Good Programming Practice.

Next, add code as shown in black (follow along with instructor), and save.

```
! Estimate wind power

!===== main program =====
program windpowermain

! declare variables
implicit none          !enforce strong typing
real :: power = 0.0    !power (W) outout from turbine

!set up
write(*,*) "Welcome to Wind Power"    !welcome user

!save results
write(*,*) "power = ", power    !display result

end program windpowermain
```

18

## Try it. Compile and run.



```
➤ gfortran wp02.f95 -o runwp02 #invoke the compiler
➤ ./runwp02 #run the executable
Welcome to Wind Power #this is output
power = 0.000000 #this is more output
➤ #the next linux prompt
```

Good.

Next, lets look at error messages and debugging.

19

## Try it – Finding & fixing errors



First, do "save as" with name "wp03.f95", to create a new version.

Next, change the code as shown, save, compile, & execute.

```
! Estimate wind power

!===== main program =====
program windpowermain

! declare variables
implicit none #enforce strong typing
real :: power = 0.0 #power (W) outout from turbine

!set up
write(*,a) "Welcome to Wind Power" #welcome user

!save results
write(*,*) "power = ", power #display result

end program windpowermain
```

20

## Try it – Error Messages



Your output might look like:

```
wp03.f95:11

  write(*,a) "Welcome to Wind Power" !welcome user
          1
Error: Symbol 'a' at (1) has no IMPLICIT type
```

It tells you:

- 1) which program had the error: which line of code (line 11) had the error.
- 2) it displays a copy of the offending line, and then under it uses "1" to point to the error.
- 3) it explains the reason for the error.

21

## Try it – Debugging



Next fix that first error. Then make a different error by forgetting to write the ending set of quotes. Change the code as shown, save, compile, & execute.

```
! Estimate wind power

!===== main program =====
program windpowermain

! declare variables
implicit none                !enforce strong typing
real :: power = 0.0          !power (W) output from turbine

!set up
write(*,*) "Welcome to Wind Power" !welcome user

!save results
write(*,*) "power = ", power    !display result

end program windpowermain
```

22

## Try it – More Error Messages



Your output might look like:

```
wp03.f95:11  
  
  write(*,*) "Welcome to Wind Power !welcome user  
            1  
Error: Unterminated character constant beginning at (1)
```

It tells you:

- 1) which program had the error: which line of code (line 11) had the error.
- 2) it displays a copy of the offending line, and then under it uses "1" to point to start of the section that had the error.
- 3) it explains the reason for the error.

**Note:** These errors can be caught in editors with colored highlighting of syntax.

23

## Version Control – good programming practice



One of the reasons for saving previous working versions of the code, is that you can always revert back to a previous good version if you screwed up the new version so bad that you can't fix it easily.

Also, by making only small changes to the new version, you can more easily isolate the likely places where the error could be. This speeds debugging.

Lets do it. Just delete the version 3 (wp03.f95) from your editor, and open version 2 (wp02.f95). Then immediately save it as a new version 3 (wp03.f95).

To encourage this, the markers for this course will need to see ALL versions in your directory, for you to earn full marks.

24

# Variables: Arrays



!Here is how you can declare 1-D array variables:

```
real, dimension(16) :: temperature
integer, dimension(10) :: digits
character (len=100), dimension(120) :: poem
```

!Or, for a 2-D array:

```
real, dimension(120,2) :: sounding
```

!Then, you can reference any array element in a 1-D array by:

```
integer :: i, d
real :: T
character (len=100) :: line
i = 3
T = temperature(i)
d = digits(i)
line = poem(i)
```

25

# Wind Energy



This program will read the wind data from a meteorological sounding as shown below. Thus, we can anticipate that we will need to have arrays of heights, wind directions, wind speeds, and lines in the sounding.

	05	10	15	20	25	30	35	40	45	50	55	60
71109	YZT	Port	Hardy	Observations at 12Z 29 Jan 2007								
	PRES	HGHT	TEMP	DWPT	RELH	MIXR	DRCT	SKNT	THTA	THTE	THTV	
	hPa	m	C	C	%	g/kg	deg	knot	K	K	K	
	1025.0	17	0.0	-0.4	97	3.64	245	2	271.2	281.1	271.8	
	1018.0	73	3.6	2.0	89	4.36	203	3	275.3	287.3	276.1	
	1000.0	218	3.0	1.2	88	4.19	95	4	276.1	287.8	276.9	
	997.0	242	2.8	1.0	88	4.14	91	4	276.2	287.7	276.9	
	989.4	305	4.5	-1.4	66	3.50	80	4	278.5	288.4	279.1	
	978.0	399	7.0	-5.0	42	2.71	61	4	281.9	289.8	282.4 <sup>6</sup>	

## Try it ...



Add the following type declarations to the "declare variables" part of your `wp03.f95` code, save, compile, and run.

```
integer, parameter :: maxlines = 120 !max sounding lines that can be captured
real, dimension(maxlines) :: zmsl !array of heights MSL (m)
real, dimension(maxlines) :: speed !array of wind speed (knots)
character (len=100), dimension(maxlines) :: sounding !holds the whole sounding
```

Your output from `./runwp03` should still say:

```
Welcome to Wind Power
power = 0.000000
```

27

## Operators



Operators allow you to perform actions on variables.

Assignment: `=`

Mathematical: `+`, `-`, `*`, `/`, `**` (unary `+`, `-`)

Logical:

<code>==</code>	or	<code>.eq.</code>
<code>/=</code>	or	<code>.ne.</code>
<code>&gt;</code>	or	<code>.gt.</code>
<code>&gt;=</code>	or	<code>.ge.</code>
<code>&lt;</code>	or	<code>.lt.</code>
<code>&lt;=</code>	or	<code>.le.</code>
<code>.and.</code>		
<code>.or.</code>		
<code>.not.</code>	(unary)	
<code>.eqv.</code>	(logically equivalent...)	
<code>.neqv.</code>	... or not)	

28

# SUBROUTINES

("helper" functions)



```
program somemath
  implicit none           !Enforce strong typing
  real :: x,y,f,p        !Declare variables
  . . .                  !prompt user to enter x and y
  call factorial(x,f)     !find the factorial of x
  call power(x,y,p)      !find x to power y
  . . .                  !display results on screen
end program somemath

subroutine factorial(x,f)
  real, intent(in) :: x   !the input variable
  real, intent(out) :: f = 1.0 !initialize factorial
  . . . !calculate f as x factorial
end subroutine factorial  !returns factorial in f
```

29

## Try it ...

First, Save As "wp04.f95" to create a new version. As an example of "top-down" **good programming practice**, add the following subroutine calls to your main program. Save.



```
!set up
  call welcome
  call getturbinespecs
  call getsounding

!compute wind power
  call findpower

!save results
  call saveresults
```

30

## Try it ...



Next, add subroutine "stubs" that don't do anything except announce that they've been called (to help you debug the program). For example:

```
!=====
subroutine welcome
  implicit none          !enforce strong typing
  write(*,*)
  write(*,*) "Welcome to Wind Power"
end subroutine welcome

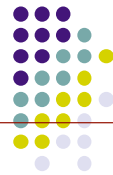
!=====
subroutine getturbinespecs
  implicit none          !enforce strong typing
  write(*,*)
  write(*,*) "getturbinespecs: Get specs of the wind turbine."
end subroutine getturbinespecs
```

You can write the other stubs. Then save into wp04.f95, compile, run, fix, and save again.

31

## Try it...

Your output from runwp04 should say:



```
Welcome to Wind Power

getturbinespecs: Get specifications of the wind turbine.

getsounding: Get the file holding the input sounding

findpower: Calculate the wind power.

saveresults: Write to disk and screen the wind power.

power =      0.000000
```

32



# FUNCTIONS



`!Will be discussed next week.`

33

## READ from keyboard and WRITE to screen



`READ (*,*) variable1, variable2, etc.`

`WRITE (*,*) variable3, variable4, etc.`

`!The first * in the read/write statement defaults to the standard input (keyboard) or output (screen). The second * specifies "list directed", unformatted, reads and writes.`

34

## WRITE (more details)



In a write statement, the arguments in the parentheses are:

```
write (unit number, format) stuff, to, be, written
```

Some examples:

```
write (*,*) "Wind speed (m/s)= ", M, " Temp(K)= ", T
!       where * unit number = default = computer screen,
!       and * format = list directed (format is based on
!       the type declarations of the stuff to be written)

write(*,"(F8.2)") T
!       Writes a real number to the screen, formatted to print
!       into 8 columns, with 2 digits right of the decimal point.
!       Example: bb273.15  where "b" is a blank space

write(1,"(a)") "Hello world"
!       Which writes to a previously-opened file (unit 1),
!       in an alphanumeric (character string) format.
!       File Handling will be explained next week in class.
```

35

## Useful Code Segment for Prompting User to Enter Input



```
character (len=50) :: name
...
...
write(*,"(a)",advance="no") "Type in your name: "
read(*,*) name
```

```
!This code segment prompts the user to type in something,
!and allows the user to respond by typing on the same line.
!
```

```
!The extra advance="no" specification in the write statement
!prevents the automatic line-feed from happening. It applies
!only to the one write statement in which it is specified.
```

36

## Try it ...



First, Save As "wp05.f95".

Modify subroutine **getturbinespecs** to prompt the user for the hub height "zhub" and turbine radius "r".

After reading "zhub" and "r", echo (write) those value to the screen (to keep the user happy by confirming the values).

You can either start on your own, or follow along as I write the code.

**Hint: Don't forget to declare the new variables in this subroutine before you use them.**

Save, compile, debug, run, save.

37

## Try it ...



First, Save As "wp06.f95".

Modify subroutine **getsounding** to prompt the user to enter the name "soundingfilename" of the file holding the sounding.

Also, echo (write) the filename to the screen.

You start on your own, and I will follow along later.

**Hint: Don't forget to declare any new variable in this subroutine before you use it.**

Save, compile, debug, run, save.

38

# Read from a file (on disk, etc.)



```
INTEGER :: ero,err

OPEN(1,FILE="filename", STATUS="old", ACTION="read", IOSTAT=ero)
IF (ero .NE. 0) STOP "Can't open file."

READ(1,*, IOSTAT=err) variable1, variable2, etc.
IF (err .NE. 0) BLAH !e.g., EXIT a loop

CLOSE(1)

!(ero=0 if successful, positive if failure).
! In the OPEN statement, instead of a character string
! "filename", you can have a character variable there,
! which holds the file name.
!(err=0 if successful, -1 if end of file, -2 end of record,
! positive if failure)
! For example:
integer :: ero
character (len = 30) :: studentroster
...
studentroster = "ubc_atsc212_classlist.txt"
open(1,file=studentroster,status="old", action="read", iostat=ero)
if (ero .ne. 0) stop "Can't open file."
```

39

# More File Commands & Info



## For the OPEN statement:

ACTION can be "read" or "write". (If the ACTION word is missing, than both read & write is assumed.) **Good programming practice:** for input files, specify "read" only, to avoid accidentally overwriting any important info.

STATUS can be "old", "new", "replace", "scratch", or "unknown". **Use old for input files, and replace for output files.**

## More file commands:

WRITE(1,\*) "blah" !for writing to a disk file that you had previously opened as unit 1.

REWIND(1) !move to beginning of the file that you had previously opened as unit 1.

BACKSPACE(1) !move back one line in the file that you had previously opened as unit 1.

40

## Try it ...



First, Save As "wp07.f95".

Modify subroutine **getsounding** to open the old file that the user specified, then write to the screen the value of the error flag (don't do the "if" test yet), and finally close the file.

You start on your own, and I will follow along later.

**Hint: Don't forget to declare any new variable in this subroutine before you use it. (such as the error flag variable, which is an integer)**

Save, compile, debug, run, save.

**Hint: If your program can't open the file, be sure that the file (such as darwin.txt) is in the same folder as your program, and don't forget to type the .txt suffix as part of the file name the user types in.**

41

## Control Structures: CONDITIONALS



**Version 1:**

```
if (logical expression) blah
```

**Version 2:**

```
if (logical expression) then
  blah
  blah
  blah
end if
```

*!this is a "block" of statements*

**Examples:**

```
if (T < 273.) write(*,*) "It's cold outside."
```

```
if (i == 5) then
  T = temperature(i) + 273.15      !temperature in K
  E = sigma * (T**4)              !Stefan Boltzmann
  write(*,*) E                    !output to screen
endif
```

42

## More Conditionals



The **IF THEN ELSE** statement. The **ELSE IF** is optional:

```
if (logical expression) then
  blah
  blah
else if (different logical expr) then
  blah
  blah
else if (different logical expr) then
  blah
  blah
else
  blah
  blah
end if
```

**Example:**

```
if (T<0.) then
  write(*,*) "It's cold."
elseif (T>40.) then
  write(*,*) "It's warm."
else
  write(*,*) "It's mild."
endif
```

43

## Try it ...



First, Save As "wp08.f95".

Modify subroutine **getsounding** to check the file-opening error flag after trying to open the file, and if OK than write to the screen that the file successfully opened. If not OK, then write to the screen about this failure, offer a hint on what to do next time, and stop the execution.

You can follow along as I code it.

Save, compile, debug, run, save.

```
if (ero .ne. 0) then !can't open the file
  write(*,*) " Sorry. Can't find the file: ", soundingfilename
  write(*,*) "   Don't forget to add a suffix .txt if needed."
  stop "Bye."
else
  write(*,*) " Good. Successfully opened file: ", soundingfilename
  write(*,*)
endif
```

44

## Try it ...



First, Save As "wp09.f95".

Modify subroutine **getturbinespecs** to check that the turbine radius is less than the hub height, because if the turbine blade is too long, then it will hit the ground. [Good programming practice to check for unphysical or unreasonable values.] If  $r$  is greater than or equal to  $z_{hub}$ , then tell what the problem is to the user, and allow the user to enter a new radius  $r$ .

You can code this on your own, and I will follow later. Save, compile, debug, run, save.

45

## ...and More Conditionals



An example of the **CASE** statement:

```
integer :: T           !temperature (°C)

select case (T)
case (:-1)
  blah                !for T ≤ -1
case (0)
  blah                !for T = 0
case (1:20)
  blah                !for 1 ≤ T ≤ 20
case (25, 32, 47)
  blah                !for any of the listed T
case default
  blah                !for all other T
end select
```

46

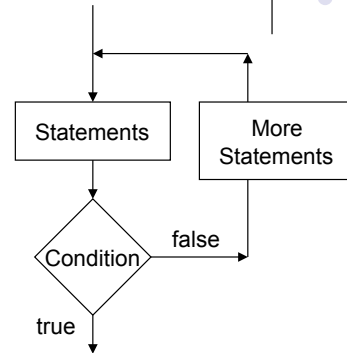
# Control Structures: Loops

Loops allow us to take repeated actions until a condition is met.

```
do                                !This is a "while" loop
  blah
  blah
  if (logical expression) exit
  more blah
  more blah
end do
```

Example:

```
rh = 80.                          !Initialize the relative humidity (%)
do                                  !Repeat the following statement block
  rh = rh - 25.                    !Decrement the relative humidity
  if (rh<=0) exit                 !Stop repeating if humidity is invalid
  y = x / rh                       !Perform some calculation
enddo
```



47

## Try it ...

First, Save As "wp10.f95".

Modify subroutine **getsounding** to read each line of the sounding file, and display the line to the screen as it reads it.

Hint: read each line as a character string called "line".

You can follow along as I code it. Save, compile, debug, run, save.

```
character (len=100) :: line        !one line in the sounding file
integer :: err                     !error flag for reading a file

. . .

write(*,*) "===== "
do                                  !read all lines in the file
  read(1,"(a)", iostat=err) line    !try to read a line
  if (err .ne. 0 ) exit             !couldn't read the line
  write(*,"(a)") trim(line)         !echo the line that was read
enddo
write(*,*) "===== "
write(*,*)
```

48

Note: the "trim" function removes any trailing blanks from the string.



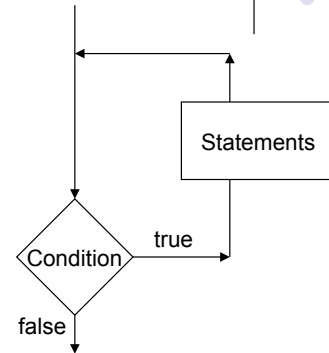


# Control Structures: more Loops



Loops allow us to take repeated actions till a condition is met.

```
!This is a "do while" loop
do while (logical expression)
  blah
  blah
end do
```



Example:

```
rh = 80.           !Initialize the relative humidity (%)
do while (rh >= 0) !Repeat the following statement block
  rh = rh - 25.    !Decrement the relative humidity
  y = x / rh       !Perform some calculation
enddo
```

49

## Try it ...



First, Save As "wp11.f95".

Modify subroutine **getturbinespecs** to replace the `r < zhub` conditional statements with a **do while** loop that allows the user to keep entering `r` values until a good `r` value is finally entered.

You can code it. I will follow along later.  
Save, compile, debug, run, save.

50

## ...and More Loops: the "counting do" loop

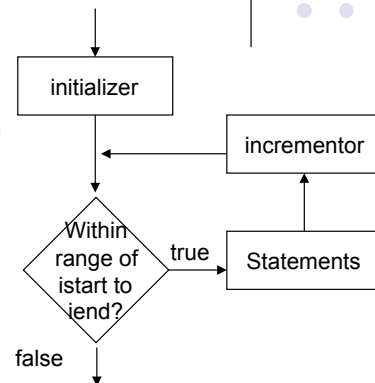
For repeated calculations where some index or value increases or decreases with uniform increments:

!(if increment is missing, 1 is assumed)

```
do index = istart, iend, increment
  blah
  blah
  blah
  blah
end do
```

Example:

```
jend = 25           !number of grid points across BC
do j=1,jend,2       !for every second grid point
  M = wind(j)       !extract the wind speed from the array
  accumDrag = accumDrag + CD*(M**2) !accumulated air drag
enddo
```



51

## Try it ...

First, Save As "wp12.f95".

Modify subroutine **getsounding** so that after the whole file was read (but before you close the file), you rewind the file back to the beginning, and then read and display only the first 5 lines of the file. These are the header lines that don't have sounding numbers in them.

You can code it. I will follow along later.  
Save, compile, debug, run, save.



52

# Loop Control Structures: CYCLE & EXIT



A loop with known starting and ending values and increments, but from which you might want to skip some calculations, or exit the loop early.

```
do index = istart, iend, increment
  blah
  blah
  if (conditional expr) cycle      !skip remaining statements
                                   !but remain inside loop

  blah
  blah
  if (conditional expr) exit      !exit immediately from loop
  blah
  blah
end do
```

53

## Summary



- FORTRAN is a programming language.
- It is designed for scientists and engineers.
- It is multi-paradigm: imperative, procedural, structural & object-oriented.
- It is a compiled language.
- It is used extensively in NWP & other number-crunching jobs in meteorology, physics, & engr.
- You can use any ascii text editor to write the FORTRAN code -- we use emacs here.

Any Questions so far?

54