Our Strategy for Learning Fortran 90

- We want to consider some computational problems which build in complexity.
  - evaluating an integral
  - solving nonlinear equations
  - vector/matrix operations
  - fitting data
  - numerical differential equations
- We want to investigate some algorithms (or methods) for solving these problems.
- We want to learn enough Fortran 90 to implement these methods.
- At first we will be writing very simple codes but as the semester goes on, we will see how to write more general codes using object-oriented techniques.
- We will also look at ways to output the results of our computer programs.
- Before we begin to look at a particular problem we want to get familiar with some basic fortran syntax.

Fortran Basics

- Fortran 90 is a free format language which means that you can start or end a statement in any column
- Fortran 77 is not free format
- Fortran is not case sensitive

`Program` is the same as `program` is the same as `PROGRAM`

- A simple code will consist of the following

- program statement
- opening comment statements for documentation of program
- `implicit none` statement (to be discussed later)
- declaration statements
- executable statements
- end program statement

Comments statements should be added throughout program for readability and clarity.

- Many editors use text coloring to make your code more readable and to find errors more easily.
- The `program` and `end program` , declaration are usually one color
- Comments are usually printed in another color.
- We will also see that do loops, conditionals are highlighted in a specific color.
- Read and write statements are highlighted.

Program Statements

- First statement of each computer program is a program statement
- Syntax
  - the word `program` followed by at least one space followed by the name of the program
  - for example, the program which uses Monte Carlo for determining π might be called

```
program  mc_pi
```

  - the name of the program does not have to be the same as what you call the program but typically you would call this program mc_pi.f90

- `end program` statement is last statement of your program
- Syntax
  - the words `end program` followed by the name of the program
  - the program name must be the same as in the initial program statement
  - for our example, the end program statement is

```
end program mc_pi
```

- For example, if your code consists of the following you will get an error message upon compilation. Why?

```
program test1

implicit none

      ⋮

end program test
```

Comment Statements

- Comment statements are essential for documentation.
- You will write many codes during this course and by the end of the semester (let alone next year) you will have forgotten what a code does or how you are implementing an algorithm.
- Comment statements will help you (and others) figure out what your code does.
- Fortran ignores comment statements - they are for human use only
- Fortran must know which statements are to be executed and which statements are just there for your information.

- How does fortran know when a statement is a comment statement?
  - An exclamation mark `!` tells fortran to ignore what comes after it
  - The exclamation mark is often at the beginning of a line if you are adding a sentence describing what the code does

    ```
    !  This program sums the first n integers.
    ```

  - Other times you may want to add a comment at the end of a line

    ```
    average = total / n !  compute average of n numbers
    ```

  - Note that in this case the compiler ignores everything after `!`

Declaration Statements

- When we write a code we use variables.
  - For example, the integer `n` may be the total number of integers we are summing.
  - Or `average` may be our average of n numbers which is a real number
- Each variable must be assigned a memory location.
- In order for the compiler to do this, it must know what type of variable it is. For example, an integer requires less storage than a real number.
- Consequently we must declare each variable that we use.
- In Fortran 77 programmers used to declare that any variable starting with i through n were automatically integers. This is now considered bad programming practice.

Implicit None Statement

- We will always use the declaration

```
implicit none
```

- This statement appears before statements declaring our variables as real, integer, etc.
- Its purpose is to tell the compiler that we will declare every one of the variables we use.
- This is very useful in debugging a code.
- For example, if we have a variable named `psi` which we have declared as a real number and in a statement we accidentally type `phi` then the compiler will give us an error that says undefined variable.
- If you do not use this statement then the compiler will assume that any variable beginning with i, j, k, l, m, n you do not declare is automatically an integer; variables beginning with the remaining letters are automatically reals.

Data Types

There are 5 basic data types

- integer
- real
- complex (we will not be using complex arithmetic)
- character
- logical

Every variable name you use must be declared to be one of these five types because the compiler needs to associate this variable with a memory location.

For example

```
integer :: n
```

declares the variable n as an integer.

Integers

- An integer is a whole number
- It can be positive, negative or zero
- Examples

$$0 \qquad 5 \qquad -45 \qquad 1024 \qquad -234567$$

- Syntax for declaring a variable n as an integer

```
integer :: n
```

- Format for declaring variables m, n as integers (in one statement)

```
integer :: m, n
```

- It is usually better programming practice to declare each variable separately
- Counters for recursive loops will always be integers.

Real Numbers

- We will use the terminology real number or floating point number interchangeably.
- Real numbers will always contain a decimal point but no commas.
- Real numbers may be entered using standard decimal expression or in exponential notation.
- Examples

$$0.0 \qquad 0.5231 \qquad -45.1 \qquad 1024.79$$

$$0.0E0 \qquad 5.231E-1 \qquad -4.51E1 \qquad 1.02479E3$$

- Syntax for declaring a variable `average` a real number

```
real :: average
```

- Format for declaring variables a, b as reals (in one statement)

```
real :: a, b
```

- It is usually better programming practice to declare each variable separately
- Later we will talk about precision which will tell the computer how many significant digits to use to store a floating point number.

Characters

- A character (sometimes called a string) is a sequence of symbols from the fortran character set.
- For us, characters will usually include letters, numbers, periods, underscores, and blanks. See page 24 of your text for a complete list of the fortran character set. Note that your author left off the underscore character " _"
- Examples

  `approx_pi.txt` (13 characters) `jane doe` (8 characters including blank)

- Syntax for declaring a variable filename as a character

$$\texttt{character(len = 20) :: filename}$$

- Note that the syntax `(len=20)` allocates a maximum of 20 characters for the string filename

Logicals

- There are two logical constants in fortran:

```
.true.    .false.
```

- So when you declare a variable as a logical, you are saying that it can only have the value true or false
- Syntax for declaring a variable flag_converge as a logical

```
logical :: flag_converge
```

- As an example we might initially set `flag_converge=.false.`, perform some iterations of an algorithm and when convergence is acheived we set `flag_converge=.true.`. That way we can test this variable to check if convergence has been achieved.
- Note that when we set a logical to be either true or false the syntax requires that we use a period before and after true or false.

The Parameter Statement

- Sometimes we want to declare a variable in such a way that it can never be changed throughout the entire program.
- For example, we might want to define π or some physical parameters in our problem.
- Fortran allows us to do this when we declare the variable through the parameter statement.
- Examples of syntax for the parameter statement declaration

```
real, parameter ::  pi = 3.14159
integer, parameter ::  two = 2
```

Continuation of a statement

- If a statement extends over more than one line you can continue the statement by putting the symbol & at the end of the line
- For example, the following two statements are equivalent

$$\texttt{b} = \texttt{a} + 7$$

$$\texttt{b} = \texttt{a} \quad \&$$

$$+7$$

- When writing a code in an editor you don't want to make the statements too long

More than one expression on a line

- Almost all the time, we will have a single expression on a single line of the file.
- Sometimes, however we may have several assignments that we want to put on the same line of the file for compactness.
- This can be done by separating the expressions by a semicolon

$$\texttt{x1} = 1.0; \qquad \texttt{x2} = 1.0; \qquad \texttt{x3} = 1.0$$

- This should be used sparingly because it reduces the readability of your code which makes debugging harder.

The Assignment Statement

- The assignment statement just assigns a value to a variable; the variable may be integer, real, etc.
- For example, if `radius` is declared real and `n` is declared an integer then we could have

  ```
  radius = 5.0
  n = 5
  ```

- If `n` is declared an integer and we write

  ```
  n = 5
  ```

  then xxxx
- If `radius` is declared real and we write

  ```
  radius = 5
  ```

  then xxxx

Writing a fortran program to use as a basic scientific calculator

- One of the simplest (but not too useful) programs to write is to perform calculations like a scientific calculator
- For example, we could compute the value of

$$\sin\frac{\pi}{9} + e^{0.234} + (2.345 - \sqrt{4.5})^5$$

- Fortran has some built-in functions which are called intrinsic functions for trig functions, square roots, exponentials, logs, etc. - just like you have keys on your calculator.
- The first thing we need to know is what are the symbols for numeric operations - adding, multiplying, exponentiation, etc.
- We also have to be concerned about order of operations. However, just like on paper, if you use parentheses freely then you can minimize problems. For example, does this expression mean $\frac{7}{2}$ or $3 + \frac{4}{2} = 5$?

```
3 + 4 / 2
```

Numeric Operations

- In fortran the numeric operations use standard symbols except exponentiation.

addition	+		
subtraction	-		
multiplication	*		
division	/		
exponentiation	**	(not ^)	(irritating, I know)

- For example, if we want to compute

$$(5.1)^6 - \frac{908.5}{7.36}$$

we type

```
5.1 **6 - 908.5/7.36
```

Mixed Arithmetic

- Most of the time, we will be performing numeric operations on variables. For example, we may want to compute $\frac{a}{b}$ where a, b have been defined.
- Care must be taken to distinguish between dividing two integers and dividing two real numbers.

`5/4` is not the same as `5.0/4.0`

- Here's what happens if you compute 5/4 using integer arithmetic and then using floating point arithmetic.

```
program mixed

implicit none

integer ::  n, m

real ::  u, v

n = 5

m = 4

print *, 'Integer division'

print *, '5/4 = ', n/m

u = 5.

v = 4.

print *, 'Floating point division'

print *, '5./4.  = ', u/v
```

```
end program mixed
```

OUTPUT

Integer division

$5/4 = 1$

Floating point division

$5./4. = 1.250000$

- Moreover, we will see that if we use mixed-arithmetic (i.e., trying to combine integers and reals) we can sometimes get in trouble.
- Using mixed arithmetic is considered bad programming practice. In this class you will lose points on your program if you use it!
- For example, instead of writing `2.0 +3 / 7` you should use `2.0 + 3.0/7.0` If you write, e.g., `2 + 3.1` then most compilers will give you 5.1 as the result but you should still use `2.0+3.1`.

Order of Operations

- Fortran has a set of rules for the order in which operations are computed. The evaluation is the way we normally expect in mathematical expressions.
- The priority rules for parentheses-free expressions are:

  1. perform all exponentiation; if more than one go right to left
  2. multiplications and divisions are performed next - left to right
  3. additions and subtractions are performed last - left to right

- We can modify the standard priority rules by including parentheses.
- The order of priority for parentheses is inner to outer.
- The rule of thumb you should follow, is when in doubt, use parentheses.
- Parentheses can also make complicated expressions easier to read.

Examples

- The expression

$$\texttt{x} + \texttt{y} * \texttt{z}$$

  means to multiply `y` times `z` and add the result to `x` .
- It is not the same as the expression

$$(\texttt{x} + \texttt{y}) * \texttt{z}$$

  In this case the parentheses take precedence and `x` is first added to `y` and the result multiplied by `z` .
- In the expression

$$\texttt{x} + \texttt{y} \texttt{**2} * \texttt{z}$$

  exponentiation takes precedence so it is done first; then the multiplication and finally the addition. Thus the expression means to first square `y` , then multiply the result times `z` and add the result to `x` .
- Note that the following expressions are not the same

$$\texttt{2.0 ** 3 **2} = 2^9 = 512 \qquad (\texttt{2.0 **3}) \texttt{**2} = 8^2 = 64$$

- Here's an example of an expression using nested parentheses

$$4.1 * ((5.5 \ / \ 2.35) **4 \ / \ 2.0)$$

  We first perform the division 5.5/2.35 (inner most parentheses) , then the expression inside the outer parentheses is done using priority rules - raise the result to the fourth power and then divide by two . Finally we multiply result by 4.1

- Warning - a programming error that we often get is mismatched parenthesis. This means that we have inadvertently missed a left or right parenthesis. For debugging ease, I like to put a space between the parenthesis and the quantities to be computed so that they are more obvious.

- In the classwork you will see this compilation error when you debug a code.

Basics of UNIX

- We know how to create a file but how do we delete one?
- How do we create folders, move files around, combine files, etc?
- To do these operations we will use `UNIX` commands for this.
- We'll show you some basic commands in UNIX.
- Check out the Resources section of the website to see on-line documentation for UNIX.

Classwork/Homework

- Modify the code `calculator.f90` to compute the following quantity:

$$(\pi + 2)^3 - \frac{4.1}{5}$$

  The correct answer is 135.10295 Note that you should call the result `value` because that is what is in the print statement.

- Now purposely edit your code so that you have mismatched parentheses. Try to compile the code and see the error you get. Correct the syntax.

- Now add statements to your code to compute and print out the second quantity

$$\frac{8.1 - 7.6^2}{3\pi}$$

  which has a value of -5.2690902

- You should demonstrate to us that you have a properly compiled and executed code by the start of next class. It does not have to be handed in.

- Review the UNIX tutorial on the website.