Scientific Fortran for Beginners

Introduction

Welcome to Fortran!

If you've not downloaded gfortran or another compiler, please see https://fortran-lang.org/en/learn/os_setup/install_gfortran/

Fortran Source Files.

The standard modern Fortran file extension is `.f90` . You might sometimes see `.f95` or `.f03` for specific Fortran standards.

If you happen to see `.f` or `.for` file extensions, you're dealing with old FORTRAN, which looks quite different.

Hello World!

A Fortran file is made up of a list of statements, each taking place on its own line. There's no need for curly braces or indentation, but you'll probably want to use indentation.

```
program main
    implicit none
    print *, "Hello, World!"
end program main
```

Fortran is also case insensitive, so `PROGRAM MAIN` is equivalent to `program main` or `pRoGrAm MaIn`

Compilation

Once you have written a Fortran file, it needs to be compiled. If you have installed gfortran this is done in the following way.

```
$ gfortran myprogram.f90 -o myprogram.out
```

You can then run the binary like so.

```
$ ./myprogram.out
```

Variables

Fortran is a statically typed language, variables must be defined with a type and can only be assigned to values of that type. Here are a few examples:

```
integer :: i, j, ij, jsq
logical :: i_less_than_j

i = 12
j = 17
i_less_than_j = i < j
ij = i * j
jsq = j ** 2
```

Constants are defined using the keyword `parameter` .

Floating point numbers come in single and double precision forms. It's standard practice to define the **kinds** of these using the following syntax.

```
integer, parameter :: dp = kind(1.0d0), &
    & sp = kind(1.0e0)
real(sp) :: f1
real(dp) :: f2

f1 = 1.0_sp
f2 = 1.0_dp
```

Arrays

Arrays are assigned using parentheses `()` . To make an array of three dimensions you would:

```
    real(dp) :: array_3d(3)

! Set all elements to zero.
    array_3d = 0._dp

! Set the first element to 1.
    array_3d(1) = 1._dp

! Set multiple elements.
    array_3d = (/1._dp, 0._dp, 0._dp/)
```

Flow Control

If statements use the following syntax.

```
if (some_logical) then
    ! Your first branch in here.
else if (another_logical) then
    ! Your second branch in here.
end if
```

Basic for loops, or do loops as they are known in Fortran use the following syntax:

```
do x = 1, n
    ! Your repeating code in here.
end do
```

While loops:

```
do while (some_logical)
    ! Your repeating code in here.
end do
```

Exercise 1

Write a program which calculates the dot product of two vectors, and prints it. Use the following two vectors:

$$(1, 30, 1.2, -12) \cdot (100.05, 0.00034, 6, 67)$$

Subprograms

Fortran supports both functions (those which `return` a value) and subroutines (those which don't return a value). We'll start with function syntax as its what many of you may be most familiar with.

```fortran
program main
    implicit none

    integer x

    x = 12
    print *, x
    x = add_two(x)
    print *, x

contains
    integer function add_two(x)
        integer, intent(in) :: x
        add_two = x + 2
    end function add_two
end program main
```

And now subroutines.

```
program main
    implicit none

    integer x

    x = 12
    print *, x
    call add_two(x)
    print *, x

contains
    subroutine add_two(x)
        integer, intent(in out) :: x
        x = x + 2
    end subroutine add_two
end program main
```

A Small Project: Coulomb's Law

The second part of this workshop is about creating a little graph of two charged particles acting under Coulomb's law.

A quick refresher: the electrostatic force felt by a charged particle due to another charged particle is

$$\mathbf{F}_1 = \frac{q_1 q_2}{4\pi\varepsilon_0 {|\mathbf{r}_{12}|}^2}\hat{\mathbf{r}_{12}},$$

where $\mathbf{r}_{12}$ is a vector in the direction of particle 1 from particle 2.

Writing to a file

Fortran doesn't have any plotting tools that are as easy to use as matplotlib (in my opinion), for example, so its common practice to write out the data to a file and then to write scripts that analyse the data.
This is also good practice if you wish to save on time spent generating data. The syntax for opening a file, writing to it line by line, and closing it is as follows.

```
open(unit = 1, file = "myfile.data", status = "new")
write(1,*) "3.14"
close(1)
```

Exercise 2: Coulomb's Law

Back to the project, your code should:

- Create a file named `coulomb_repulsion.data` .
- Define two electrons, with positions and charges.
- Calculate the Coulomb force felt by the moving electron at 1,000 points in space as $\mathbf{r}_{12}$ is varied between $(0.1, 0.1, 0.1)$ mm and $(1.1, 1.1, 1.1)$ mm.
- Write each force on a new line in `coulomb_repulsion.data`

In your simulation please use the following constants:

$$\pi = 3.14,\ \ \varepsilon_0 = 8.85 \times 10^{-12}\text{F/m},\ \ e = 1.60 \times 10^{-19}\text{C}$$

Here's the coulomb repulsion equation again:

$$\mathbf{F}_1 = \frac{q_1 q_2}{4\pi\varepsilon_0 |\mathbf{r}_{12}|^2}\hat{\mathbf{r}_{12}},$$