Introduction to Programming Fortran 90

Handout One

September 16, 2013

Part I
Course Outline

1 Preliminaries

The aim of this course is to teach you how to write mathematical computer programs in FORTRAN 90. This will enable you to take a mathematical problem, implement a (computational) algorithm to solve the problem and translate your algorithm into a well written Fortran 90 program. To achieve this you must not only learn how to program in Fortran 90 but also the basics of good programming practice and structure, these will become clear as the course progresses.

Fortran 90 is a high level numerical programming language. What does this mean?

- **Programming language** : A method of communication that allows a programmer to provide a set of code that instructs a computer to perform a given task or collection of tasks.

- **Numerical** : Specifically designed and optimised to efficiently execute numerical algorithms and approximation techniques.

- **High level** : A high-level programming language provides a significant level of abstraction from the complicated inner workings of a computer. As opposed to a low level language it is far easier for humans to work with and learn. It uses terminology closer to our own spoken language than the complex assembly and binary code typically used to communicate with a computers architecture.

1.1 Course Structure

The nature of teaching computer programming means that traditional lectures are inappropriate. As a result this course will be mostly practical. This will involve you typing, compiling and testing codes for most of each session and all classes will be held in the microlab.

1.2 Course Handouts

Course notes will be handed out incrementally during the course introducing new material as it is required. Also there will be a few copies of 'Fortran 90 Course Notes' from a more advanced course held at Liverpool University, these are there for reference only and contain far more information than you need to pass this course.

1.3 Your Approach to the Course

There are a number of points arising from the structure of this course which you should try to stay aware of for the duration of the semester.

- There will be a lot of information in the course notes. Please make sure you read through all the notes carefully.

- This course has been designed so that a student with 'no' previous programming experience at all will be able to cope with the course content. During the course if it becomes clear that you are falling behind or struggling you *must* let me know. Unlike other lecture courses, this course will not be easy to revise at the last minute. The practical experience in this course is vital, so do attempt all the exercises and 'class projects' you encounter in the notes. Also, bear in mind that at first programming can seem a daunting and unforgiving task, however, it is often the case that with effort and determination to overcome a few hurdles it can actually be fun and a good challenge.

- Some of you may have had some previous programming experience. Don't make the mistake of going too quickly through the course material. It is inevitable that students will progress through the handouts at different speeds. There will be regular interruptions by me on programming style, common errors etc. which you cannot miss. If you think you really have run out of things to do during a practical session then let me know!

- These handouts will cover sufficient Fortran 90 to write adequate programs, but there will be important points which arise in discussion and are not included. You must make sure that you keep separate notes of additional subjects as they are discussed. Mostly this will involve keeping listings of your programs and any programs that I give you. These should be sufficiently clear so that they serve as references for projects and revision for the exam.

Part II
The Linux Operating System

Although you will be sat in front of a computer running Microsoft Windows you will not be using these machines to write, compile and run your Fortran 90 codes. The Microsoft Windows machine you are sat at will simply serve as an access point and you will use it to log into one of three machines owned by the Mathematics Department. These three machines are installed with an operating system called 'Linux'. Linux was devised to be a free version of UNIX and it has evolved into a very powerful and well used operating system.

Semantics :

- Anything inside square brackets, [ ], will be a key/mouse-press or a menu item

- Anything inside quotes, ' ', will be a program name application or title

- Anything inside curly brackets, { }, will be a Linux command statement. NOTE the curly brackets are NOT part of the Linux command so do not type them in at the terminal!

- Anything inside angular brackets, <>,which are inside curly brackets is an argument to the Linux command like a file or directory name. NOTE the angular brackets are NOT part of the Linux command so do not type them in at the terminal!

2 Logging in and out

You can log into your assigned fortran machine from the Microlab using the application 'putty' as follows.
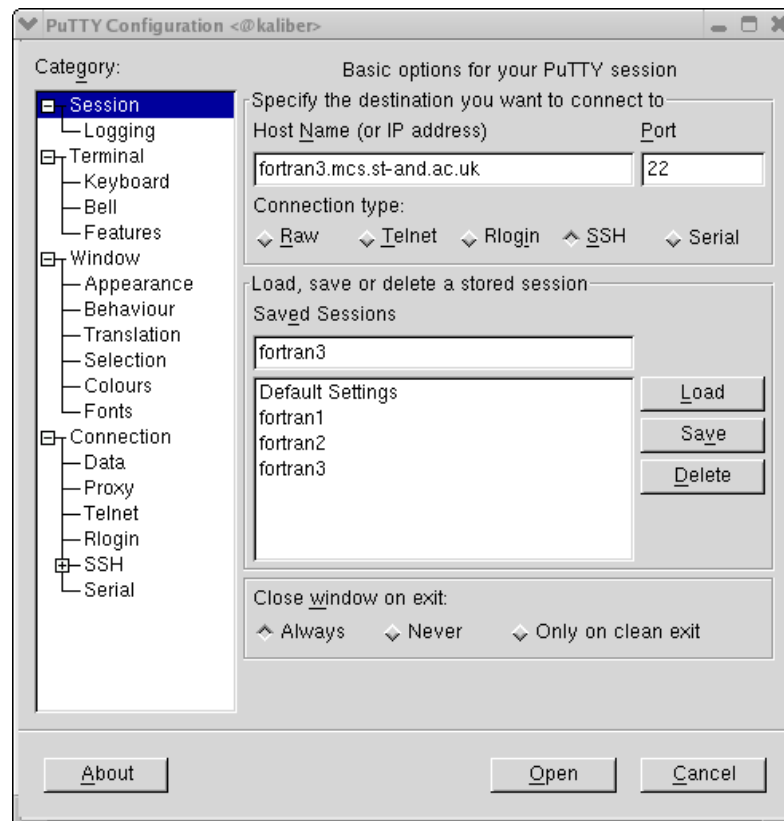

Figure 1: Putty Window.

- Now you have to start a terminal session so you can log into the 'remote host' which will be one of the three course machines **'fortran1'**, **'fortran2'** or **'fortran3'**.

- You can log onto your fortran machine using the application 'putty' found on the microlab machine. To do this click [**Start**]→[**Communications**]→[**Putty**].

- Once started, in the **'Category'** menu on the left, the **'Session'** option should be highlighted. On the right in the **'Host Name (or IP address)'** entry box, you need to type in the fully qualified domain name for your designated machine, for example **'fortran1.mcs.st-and.ac.uk'** if your account is on **'fortran1'**.

- Next click the **'Open'** tab, at the bottom right, you now should see a terminal pop up on your screen at which you can type your 'login name' and then your 'password' given to you with your notes.

  Once you have finished working you can log out of your fortran machine by holding down the '[Ctrl]' key and pressing '[D]' or typing 'exit' at the linux prompt in the terminal window.

2.1 Logging in remotely

If you wish you can use **'putty'** to log into the fortran machines from outside the microlab. You can download **'putty'** for free from (note the tilde)

'`http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe`'

Install putty and use the 'fully qualified domain name' of your assigned fortran machine in the 'host name (or IP address)' part of the putty session window. The full name will be one of fortran1.mcs.st-and.ac.uk , fortran2.mcs.st-and.ac.uk or fortran3.mcs.st-and.ac.uk. You login details will of course be the same.

3 Linux Operating System Environment

When you log into one of the fortran machines you will be using a computer operating system called Linux. In Linux unlike Microsoft Windows and the Apple Mac. you will be using a terminal to type in commands and run programs. In order to complete this course you will therefore need to know some basic Linux commands.

3.1 Your 'home area' and directory structure

When you first login you have control of the terminal in what is referred to as your 'home directory'. To draw an analogy, a directory in Linux can be understood in the same way as a folder in Microsoft Windows. Your 'home directory' is your 'highest level' directory and all other directories you create inside it will branch off it in a 'tree' like structure. There is a special character ' ~ ' (called **tilde**) in Linux to reference the user's home directory.

Inside your home area you can create files and directories, write Fortran codes and look at any data you generate from your codes. You can create as many subdirectories (branches) as you like in your home directory. You can then create new subdirectories in the subdirectories you have previously created and so on. The reason for structuring our home areas into subdirectories is so we can organise our files into separate related groups rather than have lots of different files all mixed up in a messy fashion in our home directory.

All these directories and subdirectories are known collectively together as your 'home area' or your 'account'
Only you (and the 'System Administrator') can see or work in your home area. Any work you do will NOT be visible to the other students on the course.
In your home directory you already have a directory **fortran**. It is in this directory that you will work though all the example codes in the handout. In order to reference the location of any file in your home area both a **pathname** and **filename** are required.
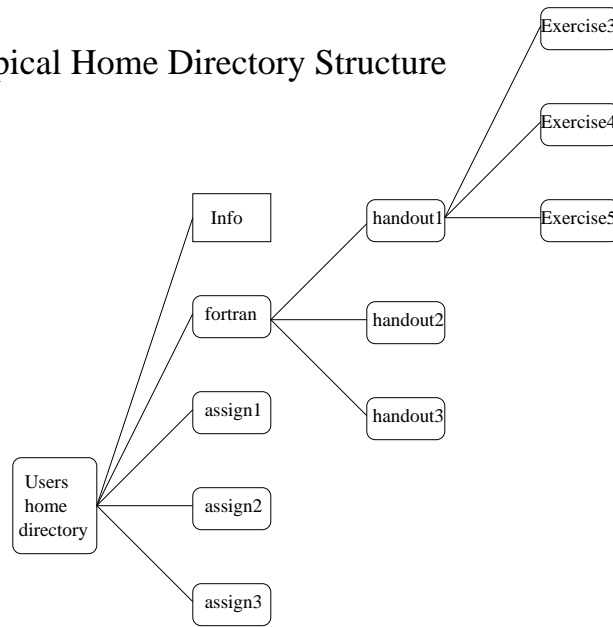
A Typical Home Directory Structure

Figure 2: What your directory structure could look like soon!

- The **pathname** provides the location of the directory in order to reference the file. For example

  `~/fortran/handout1/exercise3`

  is the pathname for the '`exercise3`' directory in the diagram relative to the user's home directory It is important to note that the pathname is always relative to some base directory. In this case the base directory is the user's home directory referenced by the `~`. This will become more clear later in the notes when you see some examples.

- The **filename** is simply the name of the file to be referenced.

So for example if in the `exercise3` directory you create a file called '`newtonraphson.f90`'. Then the file can be referenced by

`~/fortran/handout1/exercise3/newtonraphson.f90`

As you can now see the pathname given above is simply a list of the directories you would have to traverse, from the user's home directory, in order to reach the directory in which the file you need to reference resides. The directory names are delimited by the ' `/` ' character. Do not worry if you are a little confused here careful reading and studying of the examples presented later will help clarify any confusion.

4   Some Basic Linux Commands

Semantics :

- In the notes a Linux command will be encapsulated inside curly brackets, { }. The curly brackets are NOT part of the Linux command so do not type them in at the terminal!

- The Linux command name will appear in **bold** text.

- Any arguments supplied to a Linux command are encapsulated inside angular brackets, <>. The angular brackets are NOT part of the Linux command so do not type them in at the terminal!

4.1   The Terminal Prompt

When you login you will be presented with the terminal prompt. This will look similar to the line below

```
fortran1 jill>
```

The purpose of the prompt is to be informative. It has three pieces of information.

- 'fortran1' : The name of the machine the user is logged into.

- 'jill' : The name of the current working directory. This is jill because the user has just logged in and therefore control is in the user's home directory which is called jill.

- '>' : This is the delimiter and informs the user that any commands typed at the terminal will appear after this character.

4.2   LiSting your directory contents: {ls}

The command {ls <*pathname*> } will list the files and subdirectories in the directory referenced by '*pathname*'. The command {ls} without any arguments will list the current directory. Giving the {ls} command the argument {-l} ie. {ls -l} provides extra information including the date of last modification, file size and its owner which will be you! Note {ls -l} on its own would give you information on all the files in the directory. Consider a directory that has three fortran codes in it 'prog1.f90', 'prog2.f90' and 'prog3.f90'

```
fortran1 jill> ls -l prog1.f90
-rw-rw-r--    1 jill  jill        159 Feb  8 16:55 prog1.f90
fortran1 jill> ls -l *.f90
-rw-rw-r--    1 jill  jill        159 Feb  8 16:55 prog1.f90
-rw-rw-r--    1 jill  jill        109 Feb  8 16:55 prog2.f90
-rw-rw-r--    1 jill  jill        119 Feb  8 16:55 prog3.f90
fortran1 jill>
```

NOTE: the '*' is a wildcard character and is explained later in section 4.7 'removing files'.

NOTE: You will need to create some files to practice using these commands. You can create an empty file using the Linux command {touch}. The command takes one argument which is the filename of the empty file you want to create.{touch <*filename*>}. So to create an empty file 'test' and demonstrate some uses of the {ls} command consider the example below.

```
   fortran1 jill> ls
   fortran/  info@
   fortran1 jill> touch test
   fortran1 jill> ls
   fortran/  info@  test
   fortran1 jill> ls -l
   total 4
   drwxr-x--- 4 jill jill 4096 Oct 30 21:05 fortran/
   lrwxrwxrwx 1 jill jill   17 Oct 30 21:03 info -> /home/steveb/info/
   -rw------- 1 jill jill    0 Oct 30 21:19 test
```

4.3   Print Working Directory: {pwd}

The command {pwd} will print the 'pathname' for the current working directory. So, for example, if user 'jill' logs in, and before doing anything else, types {pwd} then hits return the terminal then returns the current working directory,

```
fortran1 jill> pwd
/scratch/home/mt4112/jill
fortran1 jill>
```

as user 'jill' has just logged in and has not moved out of his home directory. All the home directories are subdirectories of the '/scratch/home/mt4112' directory. Try doing a {**cd ..**} while in your home directory then an {**ls**}, you will see the directories of the all other students on the course. You will notice that you can not {**cd**} into other students' directories however.

4.4 Changing Directory: {cd}

The command {**cd** <*pathname*>} will move you from the current directory to the directory referenced by '*pathname*'. {**cd ..**} (two periods) will move you backup a directory level. Note also you can chain the two periods together as in {**cd ../..**} which will take you back up two directories. A useful tip is noting that your home directory can be referenced by the tilde character ' ~' so, wherever you are in the directory structure, typing {**cd ~/fortran**} will return you to the subdirectory 'fortran' off your home directory. Hitting the [**return**] key after typing {**cd**} or {**cd ~**} have the same effect and no matter where you are will place you back in your home directory.

4.5 MaKing a new DIRectory: {mkdir}

The command {**mkdir** <*pathname*>} will create a directory referenced by '*pathname*'. When you first log in to your account you are automatically placed in your home directory. From here you can start creating any subdirectories you feel may help in organising your files. So for example to create a new directory called '`handout1`' inside your '`fortran`' directory you could do the following.

Exercise One (Part I): Repeat the work by user jill below. Please ensure you understand each line and if not reread the relevent notes above.

```
fortran1 jill> pwd
/scratch/home/mt4112/jill
fortran1 jill> ls
fortran/  info@
fortran1 jill> cd fortran
fortran1 fortran> mkdir handout1
fortran1 fortran> ls
handout1/
```

4.6 ReMove DIRectory: {rmdir}

The command {**rmdir** <*dirname*>} will remove the directory '*dirname*' if it is empty. If the directory *dirname* is not empty the directory will not be deleted and the Linux terminal will give you an appropriate error message. To delete files in a directory, so you can then go ahead and remove the directory, {**cd**} into the directory and use the {**rm**} command, {**cd ..**} back out again to use {**rmdir**}.

4.7 ReMoving files: {rm}

The command {**rm** <*filename*>} will remove the file, *filename*, from the directory. In the Linux command statement replacing <*filename*> with <***> would remove all files in that directory **use this command with caution**. As a check you will be asked to confirm if you want to delete the file(s) in question, type [Y] or [N] as required then press return [↵]. Used in this way the '*' is referred to as a wildcard character.

Exercise One (Part II): Repeat the work by user jill on Fortran1 shown below. Make sure you understand exactly what is happening using the notes on Linux commands above. **Note** the error message when you first try to delete the directory 'testdir' because the directory is not empty and has two files in it.

```
fortran1 jill> mkdir testdir
fortran1 jill> cd testdir
fortran1 testdir> ls
fortran1 testdir> touch file1
fortran1 testdir> touch file2
fortran1 testdir> ls
file1  file2
fortran1 testdir> cd ..
fortran1 jill> rmdir testdir
rmdir: testdir: Directory not empty
fortran1 jill> cd testdir
fortran1 testdir> rm *
rm: remove regular empty file 'file1'? y
rm: remove regular empty file 'file2'? y
fortran1 testdir> ls
fortran1 testdir> cd ..
fortran1 jill> rmdir testdir
fortran1 jill> ls
fortran/  info@
```

4.8   CoPying Files: {cp}

The command {**cp**} is a very useful and important command. It allows us to copy files and if needed whole directories in our home area. It has various forms and the following examples in the exercise show its use. Please make sure you understand every line.

Exercise One (Part III): Repeat the work done by user jill below. If you are not sure what a command is doing reread the appropriate section above for clarification.

```
fortran1 jill> mkdir testdir
fortran1 jill> cd testdir
fortran1 testdir> touch file1 file2 file3 file4
fortran1 testdir> ls
file1  file2  file3  file4
fortran1 testdir> cp file4 filefour
fortran1 testdir> ls
file1  file2  file3  file4  filefour
fortran1 testdir> cp file3 filethree
fortran1 testdir> ls
file1  file2  file3  file4  filefour  filethree
fortran1 testdir> rm filefour filethree
rm: remove regular empty file 'filefour'? y
rm: remove regular empty file 'filethree'? y
fortran1 testdir> ls
file1  file2  file3  file4
fortran1 testdir> mkdir temp
fortran1 testdir> ls
file1  file2  file3  file4  temp/
fortran1 testdir> cp file1 temp
```

```
fortran1 testdir> cp file2 temp
fortran1 testdir> ls
file1  file2  file3  file4  temp/
fortran1 testdir> ls temp
file1  file2
fortran1 testdir> cd temp
fortran1 temp> ls
file1  file2
fortran1 temp> rm -f file1 file2
fortran1 temp> ls
fortran1 temp> cd ..
fortran1 testdir> rmdir temp
fortran1 testdir> ls
file1  file2  file3  file4
fortran1 testdir> rm -f file1 file2 file3 file4
fortran1 testdir> cd ..
fortran1 jill> rmdir testdir
```

note the use of the '**-f**' argument to the {**rm**} command. This arguments tell the {**rm**} command not to prompt the user to check if the file really needs to be deleted. Use with caution.

4.9 MoVing/renaming Files: {mv}

The command, {**mv** <*filename*> <*destination*>}, allows you to move file(s) from one directory to another or simply rename a file. The arguments to the command {**mv**} work in the same way as those for {**cp**}. Consider the example below

---

Exercise One (Part IV): Repeat the work done by user jill below. If you are not sure what a command is doing reread the appropriate section above for clarification.

```
fortran1 jill> mkdir testdir
fortran1 jill> cd testdir
fortran1 testdir> touch t1  t2  t3  t4
fortran1 testdir> mkdir test2
fortran1 testdir> ls
t1  t2  t3  t4  test2/
fortran1 testdir> mv t? test2
fortran1 testdir> ls
fortran1 testdir> ls test2
t1  t2  t3  t4
fortran1 testdir> cd test2
fortran1 test2> mv t4 t4_new
fortran1 test2> ls
t1  t2  t3  t4_new
fortran1 test2> rm -f t1  t2  t3  t4_new
fortran1 test2> cd ..
fortran1 testdir> rmdir test2
fortran1 testdir> cd ..
fortran1 jill> rmdir testdir
```

Notes : In the command {**mv t? test2/**} the question mark is called a 'wildcard', you have already seen one 'wildcard' the '*' which references all files, it actually references all files because the '*' represents

9

all combinations of any number of characters in a valid filename. The '?' is a different wildcard in that in only references *any single character.* So {**mv t? test2/**} means select all files that start with 't' and have one more character in their name and then move them into the directory 'test2' which is one level below the current directory. Note the '?' form of referencing would also work the {**cp**} command.

5   Command Editing Tips

Recalling Previous Commands: You can use the [↑] key to scroll through the history of commands that you have previously typed at the terminal window. Pressing [↑] once will recall the most recent command you typed and pressing [↑] twice will recall the second most recent and so on. The [↓] key can be used to scroll the commands back again. Using the arrow keys like this is **very** useful when you need to issue similar commands repeatedly as it saves on typing. Note that once you have recalled a command this way you can also edit it to do something different.

[Tab] key completion: This is a useful feature for completing commands, filenames etc. that you are typing in at the command line prompt. A good example of this is if you wanted to change directory to a directory that has a really long name, say 'fortran90 source files', instead of typing

```
fortran1 test> cd fortran90_source_files
```

Type,

```
fortran1 test> cd for
```

and before pressing return [←] press the [Tab] key. This will complete the directory name for you as long as there is no ambiguity in the completion. The [Tab] completion is also useful as it automatically avoids any spelling errors!

Copying and Pasting: It is often useful to be able to 'copy and paste' text that is already visible in a terminal window. You can do this by first selecting/highlighting the text. Hold down the [Left Mouse Button] and drag the mouse pointer across the text you want to select. Then, using the arrow keys, move the cursor to where you want to place the selected text and press [Right Mouse Button]. The selected text will then be pasted into the desired location.

Now you have read through the commands above, work through this exercise below by simply copying the work done by 'jane' on 'fortran1'. When you have finished you will have a directory 'fortran' in which you have made a directory 'handout2'. Now any examples and exercises you do from handout two can be kept in the 'handout2' directory.

Exercise Two: Move into your fortran directory and make a directory called 'handout2'. Make use of the commands {**pwd**} and {**ls**} to show where you are in your home area's directory structure and to list the files in a directory.

```
fortran1 jane> ls
fortran  info
fortran1 jane> pwd
/scratch/home/mt4112/jane
fortran1 jane> cd fortran
fortran1 fortran> pwd
/scratch/home/mt4112/jane/fortran
fortran1 fortran> mkdir handout2
fortran1 fortran> ls
handout2
fortran1 fortran> cd handout2
fortran1 handout2> pwd
/scratch/home/mt4112/jane/fortran/handout2
```

6 Typing/Editing Text Files: 'nano'.

Before starting to write a Fortran code you must become familiar with a terminal text editor. The actual Fortran codes you will be writing are just text files and so can be written in any text editor. The text editor we will use is a very simple editor called 'nano'.

The only things you absolutely must know about the 'nano' editor are

- You open a file to be edited in the 'nano' editor by typing {**nano** <*filename*>} where *filename* is the name of the file you wish to edit. If that file doesn't exist 'nano' will create an empty file of that name for you in the current working directory and open it for editing.

- You can edit the file by moving the cursor to the location in the file you want to edit. Anything you type will appear at that point. Pressing the backspace [←] key will delete the character immediately to the left of the cursor position.

- At the bottom of the editor is a menu of commands that you can use. To do any of these you press the [Ctrl] key and while holding it down also press the letter indicated.

- To exit the editor press [Ctrl]+[x]. You will be asked if you want to save your changes (it actually says 'Save modified buffer (ANSWERING 'No' WILL DESTROY CHANGES)' but this means the same!). If you do want to save then type [y], otherwise type [n]. You will then be asked to give the file a name. If you want to keep the same name just press return [↩].

- If you want to know more, look at the online help in nano by pressing [Ctrl]+[g]. To exit the help session press [Ctrl]+[x].

7 What is Fortran?

Fortran is the name of a programming language. Put simply, a Fortran program, is a **sequential** set of instructions for the computer to execute. It was designed and has evolved for scientific and mathematical programming. The version of Fortran taught here is 'Fortran 90'. Many Fortran 90 commands are omitted in this course so don't be worried by text books that have Fortran 90 commands you don't recognise. The material provided during the course will be sufficient for you to complete the coursework and sit the exam.

8 Creating A program Using 'nano' Editor

As mentioned above, to perform a mathematical calculation on a computer you must first create a sequential list of Fortran commands (known as the program) which the computer can subsequently be told to compile. The program will be stored in your account with the filename you provided.

To do this you must first create that file. 'hello_world.f90' is a suitable name for this first program. The '.f90' file extension identifies the file as Fortran 90 source code. To create a file with this name type {**nano hello_world.f90**} (followed by return [←]). This allows you to enter/edit the text of the program as in the simple text editor 'nano' . The command type {**nano hello_world.f90**} tells the computer to start the nano editor and load the file 'hello_world.f90' for editing. Since this file doesn't yet exist 'nano' creates it for you and of course at first it is just an empty file.

Exercise Three: Now in your directory 'fortran/handout1' make a new directory 'exercise3'. Move into 'exercise3' and use nano to create the empty file 'hello_world.f90', ie

```
fortran1 exercise3> nano hello_world.f90
```

Type the following Fortran 90 code, a simple 'hello world' program exactly as you see it below on the page. When you have typed in the program press [Ctrl]+[x], i.e. hold down the [Ctrl] key and press the [x] key. 'nano' will ask if you want to save the changes, type [y]. 'nano' will then ask if you want to call the program 'hello_world.f90'. This is in case you want to save it under a different name. 'hello_world.f90' is fine so just press return [←].

```fortran
PROGRAM hello_world
!*** program to say hello

  IMPLICIT NONE

  PRINT*,"-----------"
  PRINT*,"Hello World"
  PRINT*,"-----------"

END PROGRAM hello_world
```

Great! You have typed in your first Fortran 90 code (well for most of you). Now you can compile it. Compilation is necessary as the computer does not understand Fortran 90 code. The compilation process translates the Fortran code into machine code that the computer can process. To compile and then run the code type the two lines below! Do not worry about how they work yet this will be explained later, just try it.

```
fortran1 exercise3> f90 -o hello_world hello_world.f90
fortran1 exercise3> hello_world
 -----------
 Hello World
 -----------
fortran1 exercise3>
```

If you get an error message after the compilation command (the line starting with f90) then you have made a typing error in the program. Launch nano again {>nano hello_world.f90} and check carefully for the error. When you have your program working note how Fortran 90 prints out the lines in the same order you told it to in your code.

Exercise Four (I): As you did for exercise three create a directory for exercise four. Now type in the following Fortran 90 code which calculates the third power of 5 and will be explained below. Don't worry about mistakes, they can be corrected later using 'nano' (see Section 6). The reason for some of these words being in upper case is they are Fortran 'keywords', that is they will be recognised by the compiler and have a specific meaning as defined by the Fortran standard.

NOTE : Do NOT type in the line numbers on the left. They are only there to help explain the program in the next section.

```
1.    PROGRAM power3
2.    !*** program to calculate 3rd power of 5
3.
4.      IMPLICIT NONE
5.
6.      INTEGER :: a, b
7.
8.      a = 5
9.      b = a*a*a
10.
11.    PRINT*, a,b
12.
13.  END PROGRAM power3
```

9   How the program works

This section is not to explain all the fortran commands found in the code in any detail, but to give you a feel for how the code works.

Line 1 'PROGRAM power3' This defines the start of a program called 'power3'. The name of the program does not have to be the same as the name of the text file but it often makes sense to have them the same. The rules for allowable program names are the same as for variable names, the correct semantics of variable names will be dealt with in more detail later. Everything to the right of and on the same line as an exclamation mark (!) is called a comment. Comments are ignored by the compiler and are there only to help the person using the code to remember what bits of the code do. You must always comment your code.

Line 2 This line begins with '!' so the entire line is a comment and is ignored, you can write anything you like after a '!'. Note the use of asterisks to highlight the comment.

Line 4 'IMPLICIT NONE' is a command to turn off a particularly awful feature of an older version of Fortran. This will be explained in more detail later. For now just remember that this line *must* follow the program line of all of your codes. Notice that this line is indented by two spaces compared to the first line. This has no effect on the code and is there to make the code easier to read and look at. This will make more sense as you see more complex codes later where good indentation makes a great difference to the readability of codes.

Line 6 'INTEGER :: a, b' The program is going to use two variables (named a and b) which are both of type integer. All variables which a code uses must be defined in a similar fashion to this before they are used.

Line 8 'a = 5' sets the variable a equal to 5.

Line 9 '`*`' The asterix is the muliplication operator so '`b = a*a*a`' intructs the program to multiply `a` by itself twice i.e., take the cube of `a` , and set `b` equal to the answer. NOTE this has nothing to do with the '`*`' in the command '`PRINT*, a,b`' .

Line 11 This line prints the values of `a` and `b` so that you can see them on the screen. For now just remember that this writes the output to the screen so that you can see it!

Line 13 This tells the computer that the program contains no more lines and to finish.

10 Running the program

The code you should now have is in a form suitable for you to read. You must now convert the code into a form that can executed on the computer. This process is called compiling. For the program 'power3.f90' you have to enter the following commands

- {**f90 -o power3 power3.f90**} creates the executable file 'power3' from the source code file 'power3.f90'. Note that whatever your source code is called it must end in '.f90'. The first part of the command {**f90**} is the command to launch the Fortran compiler which as you would expect from the name compiles the Fortran text into machine code that the computer can execute. The {**-o power3**} part is a 'flag' to the compiler that tells it to name the executable code that it creates 'power3'. The last part {**power3.f90**} is the name of the fortran source code to be compiled (ie. your program).

- The program you have just created can now be run on the computer simply by entering the name of the executable you instructed the compiler to create. So in this case you would simply type {**power3**} at the command line prompt.

Exercise Four (II) : In your '`exercise4`' directory try compiling and running the program yourself using the commands

```
fortran1 exercise4> f90 -o power3 power3.f90
fortran1 exercise4> ls
power3*  power3.f90
fortran1 exercise4> power3
 5 125
fortran1 exercise4>
```

Now try again but change the source code and recompile it to calculate the cube of 6. Then do it again for 7, 8 and 9.

10.1   Compiler Errors

Consider the 'power3' code again.

```
1.    PROGRAM power3
2.    !*** program to calculate 3rd power of 5
3.
4.      IMPLICIT NONE
5.
6.      INTEGER :: a, b
7.
8.      a = 5
9.      b = a*a*a
10.
11.    PRINT*, a,b
12.
13.   END PROGRAM power3
```

If you typed the source code in correctly with no errors then the compilation process will go smoothly and the compiler will generate correct code. If you made a typing error that renders the code to be 'incorrect Fortran 90' the compilation will then fail. The compiler will generate some appropriate errors and print them to the screen. The error messages should be useful to you when trying to 'debug' your code.

Exercise Four (III) : In your 'exercise4' directory generate some deliberate errors in your 'power3.f90' code, to do this change line '11' to the code to make it incorrect by removing the asterisk '*'
PRINT,a
and change line '9' of your code to read,
c = a*a*a
and recompile. You should now see the error messages from the compiler.

```
fortran1 exercise4>  f90 -o power3 power3.f90
fortcom: Error: power3.f90, line 11: Syntax error, found ',' when expecting one
 of: ( * <IDENTIFIER> <CHAR_CON_KIND_PARAM> <CHAR_NAM_KIND_PARAM>
 <CHARACTER_CONSTANT> ...

  PRINT, a,b
-------^
fortcom: Error: power3.f90, line 9: This name does not have a type, and must
 have an explicit type.   [C]
  c = a*a*a
--^
compilation aborted for power3.f90 (code 1)
```

Where possible the compiler will give you the line number where the error occurs and a sensible message. It tells you here that the variable c used at line 9 has not been defined, which of course is true as we only defined a & b , that is what it means by 'This name does not have a type'. There is also a syntax error at line '11' (the missing asterisk). It tells us it found a ',' when expecting something else.

Explanation:

NOTE : When compilation fails, the compiler does not create an executable file for you to run for the obvious reason that it found the source code file 'power3.f90' to have errors in it. In the exercise above when you caused the compilation to fail the file 'power3' was not created. The file 'power3' still exists in your 'exercise4' directory because it has not been deleted since your previous successful compilation. You could check this by using the command {**ls -l**} and looking at the modification/creation time of the files. You will see that the source code file 'power3.f90' has a later modification time/date than that of the executable file 'power3'. Therefore subsequent attempted compilation of 'power3.f90' could not have created the current 'power3' file.