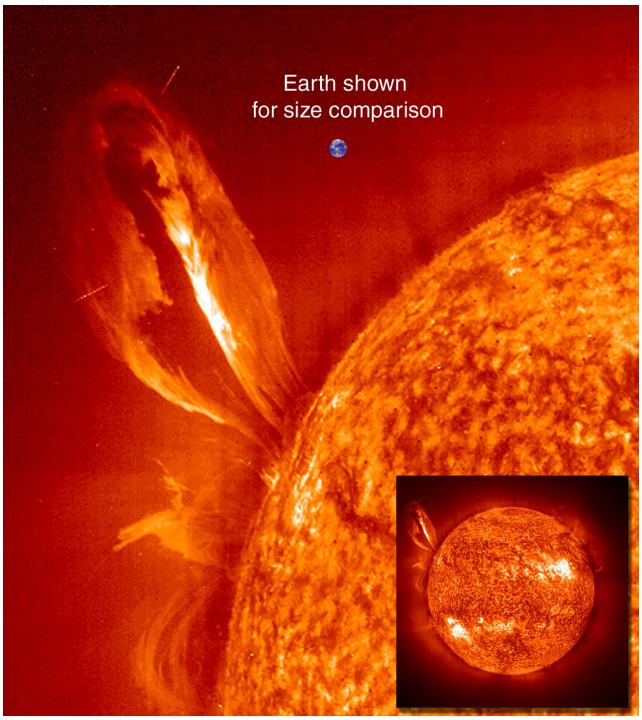
Getting Started With Linux and Fortran – Part 3

by Simon Campbell



[A Coronal Mass Ejection (CME) from our star. Taken by the SOHO spacecraft in July 1999. *Image Credit: NASA and European Space Agency*]

ASP 3012 (Stars)

Computer Tutorial 3

Contents

1 More Funky Linux Tricks

1.1 Linux Cut & Paste

Linux has a very nice cut and paste method. All you do is highlight the text you want - don't use the copy button in any of the menus and don't use control-c – just highlight.

Then you just take the cursor to where you want to paste the text and use the **middle mouse button** to paste, without using menus. If your mouse doesn't have a middle button I've found that pressing **both mouse buttons at once** usually works the same way. When you get used to this in Linux, it's hard to go back to the Windoze cut and paste!

1.2 Linux Filesystem Tips

1.2.1 The Linux Filesystem

For us humans, the Linux filesystem is similar to other filesystems, such as the Windows one - it's just a way of organising files into directories (called 'folders' in Windoze).

You can use a Windows-like navigator if you want (all Linux distributions have them) but here I'll explain how to move around using the command line. It is a little bit more tricky to move around the filesystem using the command line at first - but it soon becomes easier than drag & drop and clicking on folders!

Basically the filesystem is made up of a hierarchy of directories. The very first directory is just called '/' and everything else goes from there. For example you could **cd /** then **cd home** (if the computer has a /home/ directory) and you would be in /home/.

From there you could **cd joebloggs/asp3012/codes/** and you would be in the directory /home/joebloggs/asp3012/codes (you can always use **pwd** to check where you are).

Notice that you can include as many directory levels as you want in a cd command, so you don't have to move one directory at a time.

1.2.2 Extra *cd* Commands

Using **cd** by itself just takes you back to your 'home' (default) directory. However, it can do more than this. Try **cd ..** ('cd space dot dot'). It takes you *up* to the next highest directory. For example, say you were in the directory '/home/joebloggs/asp3012/codes/' then if you typed **cd ..** you would now be in '/home/joebloggs/asp3012/'.

The other main cd extension is **cd -** ('cd space minus'). This command takes you back to the last directory that you were in. So, continuing the same example, if we now typed **cd -** we would end up back where we started.

Another way to use the **cd ..** command is to do something like: **cd ../asp3012/notes/**. Here the cd .. command lets you go up one directory and then *down* into the /notes/ directory. You can also go two or more levels up like this: **cd ../../joebloggs/**.

All this takes a little getting used to but once you've mastered it you will be able to navigate around your directory structure very quickly.

1.2.3 Extra *ls* Commands

You already know about the list command **ls**, it just shows you what files are in the current directory. There are also some handy extensions to **ls** such as **ls -l** ('ls space minus little L'). This command shows a more lengthy (hence the '**l**') list of the files, giving more information like the size of your files and when they were modified.

Another handy one is **ls -lrt** which lists the files in reverse (**r)** time (**t**) order, so the most recent file is at the bottom of the list - very handy when you've got lots of files and want to find the one you just saved.

1.2.4 Moving Files Around

Moving and copying files around the filesystem is straightforward once you've got the hang of changing directories. The **mv** and **cp** commands can take the same directory arguments as the **cd** command. For example, if you wanted to copy a file from your current directory into the directory above, you just type **cp *filename2 ../filename***2. The copy of the file will be called *filename2*. If you want the new copy to have the same file name, just type: **cp *filename ../.*** – the dot after the forward slash just says 'use the same file name' and saves you typing it again. Moving files is exactly the same.

You can move files around even if you are not in the same directory as the file. For example, say you wanted to move a file from **/home/joebloggs/codes/** to **/home/**. All you do is type **mv /home/joebloggs/codes/*filename1* /home/*filename2*** . That way you can move (and copy) files from anywhere to anywhere, as long as you remember where they are :) You can also replace *filename2* with a dot if you want it to be the same name.

2 Basic F77 Part 3: The Calculator Program

This program introduces the concepts the **If-Then** logical construct, **mathematical operators** and **Functions** in Fortran 77. It simulates a (very) simple calculator. It is possible to extend the program to do more complex calculations.

The full program is in Figure 1. It has been broken up into its main parts so you can see the flow of the program. I will go through the program on the board and explain what each command does in more detail. Again, if you have seen all this before you can go on to the computer related questions on **Sheet 3**. Questions 3, 4, and 5 are about solving Differential Equations with various numerical schemes (see John Lattanzio's webpage). Or you could start the **assignment**.

2.1 Beginning the Program and Reading in from Keyboard

The program begins in the normal way by declaring a name for the code and then declaring all the variables it will be using.

The next step is to read some things in from the keyboard. Because it is mimicking a calculator we need to tell the program what numbers we want to operate on and then the operation we want it to perform (multiply, add, etc.).

Here we use real*4 variables for the numbers *(num1, num2* and *answer)*. For the operator we are using a character variable called *operation*. We read all these values in from the keyboard using **read(5,*)**. The *unit number* 5 is always the keyboard (the screen is number 6).

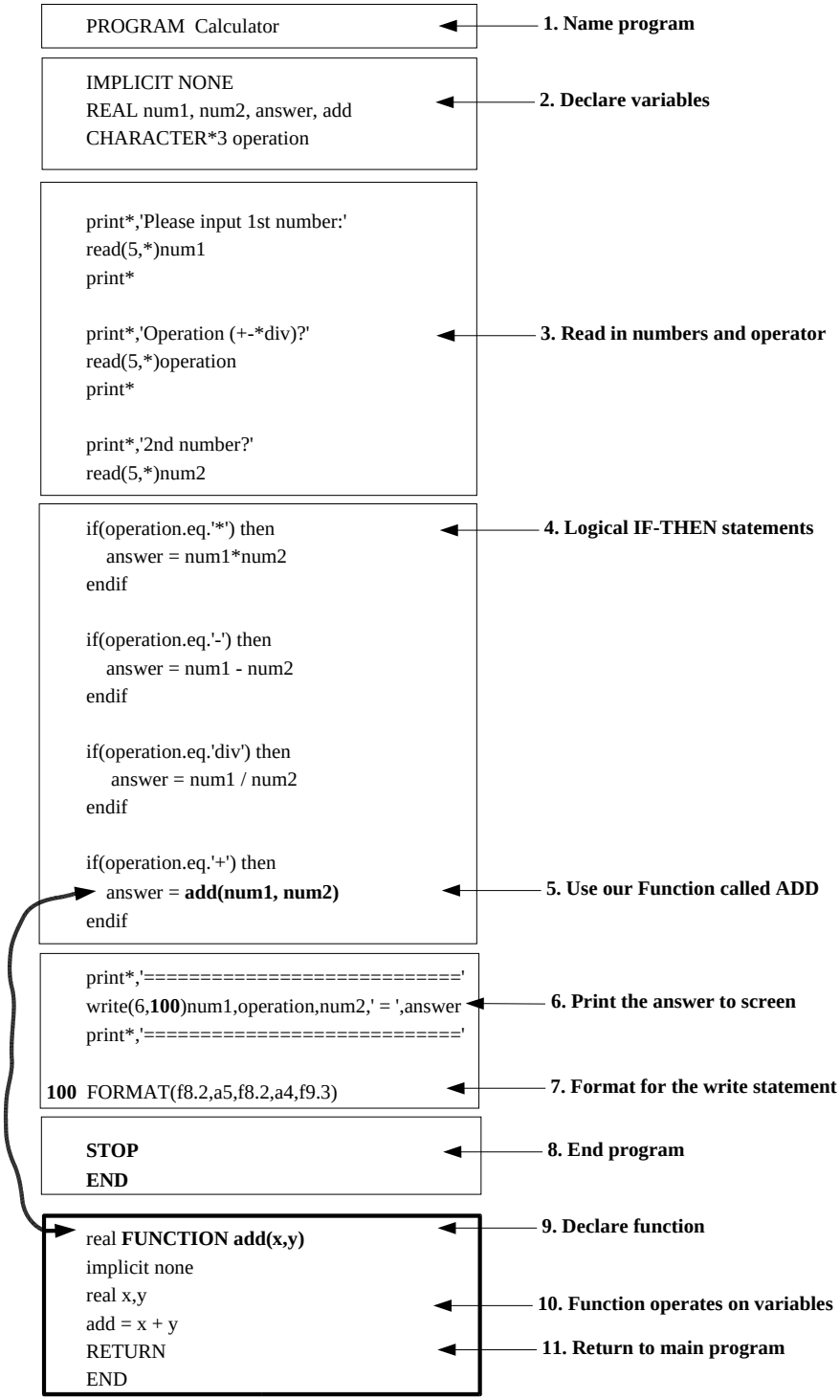


Figure 1: This computer code that simulates a simple calculator has been broken up into its main functional pieces. See text for details on particular commands.

2.2 If - Then Constructs

Once we have the values stored in the computer we can operate on them. However we first have to somehow tell the computer *which* operation to do.

We shall use the **If-Then** construct to choose between the operations. An **if** statement always contains some sort of test. Here we compare the character value of the variable *operation* with one of the four operations our code can do. The layout of an *if* statement is like so:

```
if (this test is true) then
   do this
   do this too
endif
```

The test is always a logical question like *'is x > y?'*. In Fortran you would write this as: **if(x.gt.y) then** ...

In the Calculator code we test all four operators. If the operator text matches in the *if* statement then the code goes and operates on the numbers and stores the result in the variable ***answer*** before moving on to the next part of the code.

2.3 Format Statements

Fortran 77 has its own way of formatting the output of numbers and characters. It uses the **format** statement. Each format statement is assigned a number which you can refer to when reading or writing from the code. To define a format you just do this:

```
100 format(some details in here)
```

The number can be (almost) anything. The main thing to remember is to put the number *inside the 6 blank spaces* at the start of the line. The 'f' for format must then start in at least the 7th column. The details in the brackets tell the computer the exact format. In this code we use **f8.2** and **a5**. The f8.2 means *'format the number as a Float eight significant figures long with two decimal places'*. Note that the 'significant figures' includes the decimal point and the sign (plus/minus). The a5 means *'character format five characters long'*. There are more format details but I won't go into them now.

To use the format statement you just use a write (or read) statement that has the format's number in it like so:

```
write(6,100)var1, var2
```

The variables will then be written with the nice format called **100**.

So now we've printed out the answer, we just **END** the program as usual - but we haven't quite finished...

2.4 Mathematical Operators

Since Fortran is used mostly by scientists, it has stacks of built-in mathematical operators and functions. Here are just a few:

Command	Effect
+	Add(!)
–	Subtract
/	Divide
*****	Multiply
******	Raise to the power of (eg. 5 squared = 5**2)
log10(*value)*	Gives the log (base 10) of *value*
sin(*value)*	Give the sine of *value*, where *value* is in radians

2.5 Functions

You may have noticed that in the addition *if* statement we did not just do *answer = num1 + num2*. Instead we have **answer = add(num1,num2)**. This was done to introduce Fortran functions.

A function is like a little sub-program that does a task that my be needed many times. It saves typing the commands many times if you can just put them in a function and then just use the function like another Fortran command.

Here we have a very simple function that just adds two numbers. We ***pass*** the numbers to the **add** function, it does the addition, then it sends the answer back. The answer comes back through the variable **add** – which is the function. This way we can just say **answer = add(num1,num2)** and answer will be assigned the value of **num1 + num2**. Normally functions are more complicated :)

2.5.1 How to Write a Function

A function is like a mini program. If you look at the code readout in Figure 1 you can see that the function has almost all of the main sections of a proper code. One main difference is that you *declare* a function like a variable (rather than giving it a program name). In this case **add** is a real*4 function. It then has its own list of variable declarations, then some code that does the operations, then an **end** statement. Another key difference is the **return** command. When it gets to this point it will return to the main code which will then continue running normally (see arrow in code readout).

Note: The function must be *outside the main program* – it comes ***after*** the main program **end** statement.

2.6 Compiling and Running Your Program

2.6.1 Compiling

Save your Fortran file (something like *calc*.f*)*. This time we aren't using PgPlot so we don't need to link to the PgPlot libraries. Just compile as normal:

f77 -o calc.run calc.f

The first command (f77) is the name of the compiler. The -o (little O) tells f77 what you want the ***executable*** (the version of your program that the computer can understand) to be called. *calc.f* is just your F77 program.

Now, if you list the contents of the directory, typing:

ls

Then you should see *'calc.run'* as well as *'calc.f'*.

2.6.2 Running

To run the *calc.run* program just type:

./calc.run

The program is very simple, so go easy on it! :)

3 Bored?

If this is all too easy for you then you may want to go on and try the **computer related questions on Sheet 3**. Questions 3, 4, and 5 are about solving Differential Equations with various numerical schemes (see John Lattanzio's webpage).

I also believe you have the **Stars assignment** now – so you could start on that! :)

ps. John Lattanzio's webpage is wrong on sheet 1. Here it is:

http://www.maths.monash.edu.au/~johnl/astro/ASP3012/stars.html

Mine also has some info:

http://www.maths.monash.edu.au/~scamp/tutes/asp3012/index.htm

4 Want Linux for Your Computer?

If you want to practice with Linux (and programming) at home, or just want to give it a go, I can recommend these 'distros', which you can download for free:

- **http://fedora.redhat.com/** (Fedora Core - I have this on my laptop)
- **http://www.ubuntulinux.org/** (Ubuntu)
- **http://www1.mandrivalinux.com/en/ftp.php3** (Mandriva (was Mandrake))
- **http://www.debian.org/** (Debian)

If you're really keen - or don't have a broadband connection to download the installation cds - I can burn some copies for you :)