A No Frills Introduction to Fortran Programming

Kevin Keay

Apr 27 2007

1. Introduction

1.1. What is Fortran?

Fortran (*For*mula *tran*slator) is a scientific programming language which began life in the late 1950s. The language was designed to help scientists and engineers perform complex computational tasks. Basically it was a way of getting a computer to operate on mathematical formulae and equations in various fields, like meteorology, and produce 'answers' to practical problems. It has evolved over the ensuing decades to become the leading scientific language. Much of the world's library of scientific programs is in Fortran although this will change with the increasing popularity of C, which is suitable for any programming task, not just for science. In fact, many of today's Fortran compilers, which turn Fortran source code into an executable piece of software that actually runs on a computer, are written in C and translate the Fortran code into equivalent C routines. Many scientific organizations, like the Australian Bureau of Meteorology, continue to use their older Fortran code since it is too expensive and inefficient to rewrite the existing code in C, especially if it is working correctly. Of course, many new routines are being written in C and the two languages may be interfaced too. Hence Fortran programs will still be compiled and used in the future.

Today there are two common versions of Fortran which are widely used: Fortran 77 and Fortran 90 (the numbers are supposed to reflect the years in which they appeared). The former is available for all platforms (PC, Sun, iMac) while the latter is traditionally associated with supercomputers (Cray, NEC), which have more than one CPU, since it features parallel and vector processing (ways to make a program run faster).

This guide is a very brief introduction to Fortran 77. In general, Fortran source code and system commands e.g. `g77`, will be presented in `Courier` font, to distinguish them from other text. We will look at the compilation and running of a simple example program. A *brief* discussion of software to read and graph some common weather data formats will be given, together with their location. Finally, some suggested reading is outlined.

1.1. Fortran 90

There is not a lot of experience with Fortran 90 in our group. However there are Fortran 90 compilers on atlas (Sun) and zeus (DEC Alpha) – both are called: `f90`. Kevin Keay has limited experience with Fortran 90 but should be approached initially with any queries. He has compiled a few Fortran 90 programs written elsewhere. Fortran 90 allows the user to operate on arrays and vectors in a simpler way than Fortran 77 – the code looks quite different! Most of you probably won't need to use it but there is a possibility that you might obtain code from outside the group written in this way.

1.2. Other programming languages

For your research you are most welcome to use other languages such as C. We recommend Fortran due to the wealth of experience within our group and it is not too difficult to learn. For your information the free C compiler `gcc` is available on all operating systems. The Suns also have the compiler `cc`. Kevin Keay has written *some* programs in C but is not an expert!

2. A simple example

We will consider a simple Fortran program to illustrate the basic concepts involved.

2.1. Basic structure of the program

Here is a very simple Fortran program. It consists of a set of *statements* that start with the `program` statement and finish with the `end` statement. This set of statements, or the Fortran *source code*, is saved in a text file e.g. first.f . To avoid compilation 'hassles' it is a good idea to use the extension .f for your source code files. Also, save as 'text-only', not as a Word document.

Each statement begins in column 7 and ends no greater than column 72 (the ruler just before the example is to make this clear – this ruler does *not* appear in the source code file). There are a couple of exceptions to this. Firstly, if you have a long statement you place a character in column 6 of the next line (often a + or *) and continue the rest of the statement. Secondly, you may need to use *statement numbers* associated with the `goto` and `format` statements. These numbers may go anywhere in columns 1-5. They are actually labels of your choice and have not connection with the line number in the source code file. Thirdly, you may put *comments* anywhere in your program. These have the character c in column 1 followed by any text. You may also use an exclamation mark (!) after a statement – an in-line comment.

```
         111---------------------------------------------------------777
123456789012---------------------------------------------------------012

      program first
c
c * Declaration section
      implicit none   ! Need to declare all variables
      real x,y
c
c * Executable section
      write(*,*)'My first program'
      write(*,*)'Enter a number: '
      read(*,*)x

      y= x**2 +4*x -12

c * Free format output
c   In Mardling, print is used instead of write
c   i.e. write(*,*) == print *,
      write(*,*)'x=',x,' y= ',y

c * User-specified formatted output
      write(*,10)x,y
10    format('x=',F8.4,2X,'y=',F9.5)
      end
```

The program comprises two sections: declaration and executable. The *declaration* section consists of statements that *define* variables and parameters used in your program. These immediately follow the `program` statement (this just has the name of your program) and come before all executable statements. The *executable* section consists of statements that actually *do something* e.g. the `read` statement reads in data from the keyboard or a data file. The executable statement: `y= x**2 +4*x -12` computes the variable `y` from the value of `x`. Finally, we have the `end` statement.

We may also regard the executable section as comprising three generic blocks: input, processing and output. The *input* block is the `read` statement – this allows us to input data into the program. The statement: `y= x**2 +4*x -12` is the *processing* block. Here we perform operations on (process) `x` to get `y`. Finally, the `write` statements constitute the output block – we need to view our results, in this case the value of `y`.

2.2. What does this example program do?

Well, it prints a couple of informative messages on the computer screen (`write`) and waits for you to enter a real (floating point) number (`read`) e.g. 2.5 (note that if you enter 2 it is interpreted as 2.0). The statement: `read(*,*)` means read from the keyboard and the number will be interpreted from its declaration i.e. real. This number is assigned to the variable `x` and then the statement: `y= x**2 +4*x -12` computes the variable `y` – in this case, `y` would be 4.25. Finally, the values of x and y are written on the screen (`write`). For illustration, this is done in *free format* and then by a *user-defined* `format`. The first approach involving `write(*,*)` is convenient - this means write to the screen in a default way - while the second approach using the two statements `write(*,10)x,y` and `10      format` allows greater control over the written output. Note that the connecting statement number or label (10) for the second approach.

2.3. Compiling the program

To compile this program we need a *Fortran compiler*. This is a piece of software which reads the Fortran source code file (first.f) and turns it into an executable program (software) which runs on your computer. The executable program is often called an *executable* or a *binary*. On most of the computers systems in the School of Earth Sciences there is a public domain Fortran 77 compiler called `g77`. Traditionally, the compiler is called `f77` but on some machines (especially the Suns) there is a proprietary Fortran compiler (`f77`) as well as the 'free' one (`g77`). In general, `f77` may have some different options to `g77`.Note: On the Linux PCs and Windows XP, `g77` is usually aliased to `f77` so the names actually refer to the same compiler.

Hence to compile our example on any computer system:

```
g77 -o first first.f
```

The option `-o` allows us to specify the output name of the executable. By default i.e. without the `-o` option, it is called `a.out` (or `a.exe` on Windows PCs).

The compiler produces one or more binary *objects* with the extension `.o` corresponding to your source code In our case, there is one called first.o but it is not retained unless the `-c` option of the `g77` compiler is used. When the program is compiled some pre-packaged routines which are stored as *libraries* are *linked* with the compiler binary output (objects) from your source code to create an executable. To show the creation of the object, the above compilation could be performed in two steps:

```
g77 -c first.f            (this creates the binary object first.o)
g77 -o first first.o      (this links the object to the system libraries to create
                          the executable)
```

An extension of this approach is used in *makefiles* involving the `make` command. (This will be touched on in the Lab Session). Some simple examples of these are *intrinsic* (inbuilt) functions e.g. sin(x). Depending on the complexity of your program, a host of libraries may be required. For instance, to incorporate NCAR Graphics into your Fortran program (like `conmap`) the NCAR Graphics libraries need to be available to the compiler so that the graphics routines are linked with your code.

The executable will only run on your current operating system e.g. an executable compiled on a Linux PC will not run on a Windows XP PC or an iMac. However, in principle you can transfer the source code (first.f) to other machines and compile it there – and it should work! This is the real power of programming.

2.4. Running the program

To run the executable, just type in the name: `first` . On the screen you will see the messages:

```
 My first program
 Enter a number:
```

followed by a prompt. Enter a number from the keyboard e.g. 2.5. Then some more output text is written to the screen:

```
 x=  2.5 y=   4.25
x=  2.5000  y=  4.25000
```

The above is the screen output on a PC running Windows 2000. The program was compiled with g77 under the Cygwin Linux emulation software. (This software is available on the PCs running Windows XP in the PC Lab.)
The first line uses free format and is chosen by the compiler. The second line is specified by the user in the format statement. Firstly, we write the text: `x=` on the screen. The value of the variable `x` is written in a field of 8 columns with 4 decimal places (`F8.4`), followed by a white space of 2 columns (`2X`). Finally, we write the text: `y=` and the value of the variable `y` in a field of 9 columns with 5 decimal places (`F9.5`). In both fields `F` denotes floating point (real) numbers corresponding to the declaration of `x` and `y` as real variables.

2.5. Some remarks on the example

See sections 2-6 of Reference [2] for some additional information that may help to understand the example. Although the above program is quite simple it nevertheless illustrates the basics of Fortran programming (in fact, any programming language). Extensions of the example might include reading from a data file, doing some processing e.g. computing the mean or a Fourier transform, and outputting to another data file. The input and output files may have a special format, perhaps even *binary* i.e. not viewable as text. Much of our weather data is in binary formats – usually they result in smaller files and are read more quickly than text files if there is a lot of data.

3. Software to read or graph some common weather data formats

This is a brief list of software which may be used in conjunction with Fortran programs to read and graph data provided in some common weather data formats. Note: This section is still under development.

3.1. conmap

The conmap (CMP) format is a simple binary one that we use in our group. Many people in our group have written programs which use this format. Some people (ex-students!) at CSIRO Atmospheric Research and the Australian Bureau of Meteorology also use it. See section 10 of Reference [2]. It is a convenient format for simple grided data e.g. global MSLP, and was devised so that weather data could be plotted with the free NCAR Graphics software. (There are some manuals discussing NCAR Graphics in the Unix Lab). We have Fortran programs e.g. `conmap`, which incorporate the NCAR Graphics libraries and produce contour maps of variables like MSLP – options include colour shading. To compile your own programs with NCAR Graphics you need to use the `ncargf77` command. For initial queries see Kevin Keay. Note: On the Linux PCs, `conmap` is the same as `conmap_kk` on a Sun. The latest version is called `conmap7`.The latter operating system has an older version called `conmap` while `conmap_kk` (written by Kevin Keay) has some extra features. The latest Linux version is called `conmap7`. For usage: `conmap7 --help`

3.2. NetCDF

Many weather data processing centres provide their products in NetCDF format which is supported across many scientific fields i.e. it is a general format. Unfortunately, we don't have any *general* programs to read NetCDF files – no one has been game enough to write one! Some options are:

1. Kevin Keay has a couple of Fortran programs which may be modified for the dataset being used. These programs output grided data in CMP (conmap) format. A good one to use for reanalysis products is `read_nc2cmp`. To compile your own programs that use NetCDF routines you need to include a special declarations (or *header*) file called netcdf.inc in your Fortran source code and specify the NetCDF library using the option `-lnetcdf` with the `g77` command. For initial queries see Kevin Keay.
2. Tim Butler wrote a C program called `read_ncep` to read NetCDF files from NCAR. The program was modifed by Kevin Keay to allow less than one year of data. The input files should contain 6 hourly data – the output files are in conmap format.To get limited help type: `read_ncep` . Note: This program is currently available on the Suns only.
3. Tim Butler also has an approach involving Perl to read NetCDF files. This is useful for 'peculiar' NetCDF files.
4. The free software GrADS (`grads`) can read many NetCDF files via the `sdfopen` command. You can also perform data processing and plot data as maps with `grads`. The software has an interactive X-Windows interface but may also be run in batch (script) mode.
5. Matlab (`matlab`) can read many NetCDF files. However, some data processing centres like NCEP often use a linear transformation on their data – `scale` and `offset`. These can be found using the command `ncdump -h` *`netcdffile`*. If they are other than 1 and 0 respectively you need to perform the transformation: `y = scale*x + offset` where `x` in the array read into `matlab` and `y` is the output array. There are extensive graphing capabilities within `matlab`.

3.3. GRIB

The GRIB format is an official WMO standard. In fact, data processing centres like NCEP and ECMWF store their data in this format and then convert to NetCDF for outside use. Some options are:

1. The free program `wgrib` by Wesley Ebisuzaki of NOAA can decode most GRIB files except for the reduced N80 Gaussian grid of ECMWF. The output is a file called dump and if you redirect the screen output of `wgrib` to a file called hdr (say), then the `readgribn7` program by Kevin Keay may be used to turn dump into a set of conmap files. The latter Fortran program is not general. It has been devised for the common 2.5x2.5 degree and 1.875 degree Gaussian grids of NCEP (both) and ECMWF (former) but can be easily modified for other datasets.
2. The free program `xconv` can be used to convert GRIB files to NetCDF files. See section 3.2 for NetCDF reading options. This program is especially useful for the reduced N80 grid mentioned in (1) since it can interpolate to a regular grid. It has an interactive X-Windows interface. There is also a script-based approach for converting a whole batch of files. Kevin

Walsh has been experimenting with this and the scripts have been passed on to Kevin Keay. For further information on `xconv` see: http://www.met.rdg.ac.uk/~jeff/xconv/ .

3.4. Location of programs and packages

In order to access the programs discussed above, as well as others e.g. cyclone tracking software, you need to specify their locations. This can be done with the PATH environment variable or via an `alias` command. As part of the distribution of this material there is a zip file called zdotfiles.zip which contains some *dotfiles*, named as such since they begin with a dot (.). If you already have a file called .cshrc in your home directory then you might want to rename it i.e. `mv .cshrc .cshrc.bak` . Note: To list dotfiles you need to use: `ls -a` . Then in your home directory e.g. /home/kevin, type: `unzip zdotfiles` . You may add your own customisations if you wish – perhaps you have some already in your current dotfiles. This procedure will allow you to access *most* programs on the Suns or Linux PCs with ease. The names of the programs from sections 3.1-3.3 are:

- `conmap` - on a Linux PC `conmap` is equivalent to `conmap_kk` on a Sun.
- `conmap7` - latest Linux version of conmap.
- `grads`
- `matlab`
- `read_nc2cmp`
- `readgribn7`
- `xconv`
- `wgrib`

Unless indicated, the same name is used on Linux PCs and a Sun.

4. References

4.1 Suggested reading

This document is also in: intronotes.pdf
It is recommended that you also read the following documents in this order:

1. A Simple Fortran Primer – Rosemary Mardling, Monash University (1997) (used with permission of the author). See: primer.pdf
2. Supplementary Notes for An Introduction to Fortran Programming – Kevin Keay (2006). See: suppnotes.pdf

4.1 Further information

You may obtain further information from the following sources.

3. Professional Programmer's Guide to Fortran 77 - Clive Page, University of Leicester (1995) (public domain). See: reference.pdf
4. See: `man g77`, `info g77` and `g77 --help` on the Linux PCs and Suns. Some of these help pages may not be on other operating systems. On the Suns see: `man f77` for help on the Sun `f77` compiler (not the same as `g77`). For Fortran 90 see: `man f90` on atlas (Sun).
5. Sun Fortran manuals (Unix Lab Room 408).