OO Fortran Exercises

Adrian Jackson

February 27, 2018

Contents

1 Introduction

The exercises in this sheet are designed to help with your understanding of the Fortran object-oriented, or object-oriented like, features that we are discussing in this course. The exercises on this sheet are small, self contained, practicals, and go hand in hand with the percolate practical we are also undertaking on the course.

The first practical is simply to familiarise you with getting access to, and writing programs, on ARCHER. After this the practicals aim to highlight the main features discussed in the associated lectures.

2 Getting going on ARCHER

The aim of this exercise is to get you used to logging into the ARCHER resource, using the command line and an editor to manipulate files. ARCHER is a large scale parallel resource, so jobs would generally be submitted through a batch system. However, as we are not creating parallel programs in this course we can do all our development and execution of programs on the ARCHER login nodes.

You can find more details on ARCHER and how to use it in the User Guide at:

- http://www.archer.ac.uk/documentation/user-guide/

ARCHER has three compiler suites available for uses:

- Cray
- Intel
- GNU

For these exercise we will be using the Cray compilers, these are loaded by default when you log on the system.

2.1 Log into ARCHER frontend nodes and run commands

You should have been given a guest account ID – referred to generically here as `guestXX` and password. (If you have not, please contact a demonstrator.)

These credentials can be used to access ARCHER by connecting to

```
ssh -X guestXX@login.archer.ac.uk
```

with the SSH client of your choice (`-X` ensures that graphics are routed back to your desktop). Once you have successfully logged in you will be presented with an interactive command prompt.

For more detailed instructions on connecting to ARCHER, or on how to run commands, please see the Appendix.

3 Introduction to Fortran

1. **CFD calculation:** Compile and run the CFD practical. You do not need to alter anything in the code at this point.
2. **Hello World:** Write a Fortran 95 program to write out Hello World on the screen. Compile and run the program.

3. **Area calculation:** Write a program to read in a radius and calculate the area of the corresponding circle and volume of the corresponding sphere. Try out the program with different inputs.

   Area of a circle: $area = \pi r^2$

   Volume of a sphere: $volume = \frac{4\pi r^3}{3}$

   Use the value 3.14159 for π

   You can use the following code as the template for the program:

```
  PROGRAM Area_and_Vol
!...Add specification part
  WRITE(*,"(A)") "Type in the radius: "
  READ*, radius
  !...Add code to calculate area and volume
  WRITE(*,"(A26,F5.1,A4,F6.1)") &
"Area of circle with radius ",&
radius, " is ", area
  WRITE(*,"(A28,F5.1,A4,F6.1)") &
"Volume of sphere with radius ",&
radius, " is ", volume
END PROGRAM Area_and_Vol
```

4 Modules

1. **Triangles:** Write a module that contains functions that will calculate the area and angles in a right angled triangle if passed the integer lengths of the three sides of the triangle as arguments, and will calculate the length of the hypotenuse if passed the integer lengths of the other two sides as arguments. Import that module into a program and test it.
2. **Polymorphism:** Extend your shapes module so that it can also calculate the same quantities if passed real numbers rather than integers. You should use interfaces for the procedures so you can call them by the same name regardless of the type of number passed in the main program.
3. **More shapes:** Write a new module that contains functions that calculate the area of a rectangle given the length of two sides. Use this in the program, along with the triangle module to calculate the area of half the rectangle (hint: use the triangle functionality to do this calculation).

5 Derived Types

1. **Triangles again:** Create a derived type that represents a right angled triangle using the lengths of the three sides. Use this in your triangle module.

2. **Vector:** Create a derived type that represents a vector and also stores the length of the vector. Write procedures that use that data type to calculate the sum and dot product of two vectors. Time how long it takes to undertake these operations for, say, 10,000 elements (you may need to add a loop to repeat the calculations enough times to give a sufficiently trustworthy timing).
3. **Vector again:** Create a derived type that represents a vector point, and use this to create a further derived type that represents a vector. Compare its performance to the previous type you created.
4. **Fields:** Create a derived type that represents a particle in a 2d field that has a position and velocity. Create a derived type that represents a set of particles (i.e. field). Assuming that the velocity stored in the particle type represents the distance a particle will move in one timestep simulate the particles moving for a number of timesteps and time how long it takes for the simulation to run.
5. **Fields again:** Now create a particle set derived type that stores the same information without using the particle type (i.e. a set of arrays for the velocity and position of the particles). Run the same simulation and see if there is a difference in performance from the last exercise. *Additional task:* Consider how you split the particles into different domains (i.e. if you wanted to use more than one process to run the simulation), and what impact this could have on the implementation you would choose. *Additional task:* Extend the particle datatype to 3d.
6. **CFD:** Complete the CFD practical.

6 Classes

1. **Even more shapes:** Implement type bound procedures for the shape classes that you created in the derived type exercises. Try unlimited polymorphism functionality to have single functions that will calculate the required values for integer or real versions of the shapes. Uses these as before and measure the performance of this approach.
2. **Even more vectors:** Likewise convert the vector modules you had before, adding type bound procedures and evaluate the performance.
3. **Even more fields:** Add the procedures used by the fields functionality to the fields type. Extend the procedures for that they can take a 1d, 2d, and 3d particles, checking at run time which class has been passed. *Additional task:* Add a constructor for the class to initialise the particles when the class is created.

7 Inheritance and Overloading

1. **People:** Create set of classes that represent different levels of staff in an organisation. Use an abstract class to define the required basic functionality and data. You can assume that more senior staff have different functionality to junior staff. Test operations on your different levels of staff.

2. **Vector Operator:** Overload the $+$ and $-$ operators to accept two vectors and produce a single vector as a result.
3. **Shapes:** Define an abstract shape type with deferred procedures that triangle implements. Extend triangle create a rectangle type that performs the same functionality you had before.
4. **Field Hierarchy:** Implement a class hierarchy for 1d, 2d, and 3d particles, and fields that use them. Why super- and sub-classes will you need. How will the classes be constructed? Does this new hierarchy still give the same performance as you had with the previous implementations of fields.

8 Appendix

8.1 Detailed Login Instructions

8.1.1 Procedure for Mac and Linux users

Open a command line *Terminal* and enter the following command:

```
local$ ssh -X guestXX@login.archer.ac.uk
Password:
```

you should be prompted to enter your password.

8.1.2 Procedure for Windows users

Windows does not generally have SSH installed by default so some extra work is required. You need to download and install a SSH client application - PuTTY is a good choice:

- http://www.chiark.greenend.org.uk/s̃gtatham/putty/

When you start PuTTY you should be able to enter the ARCHER login address (login.archer.ac.uk). When you connect you will be prompted for your user ID and password.

8.2 Running commands

You can list the directories and files available by using the *ls* (LiSt) command:

```
guestXX@archer:~> ls
bin  work
```

You can modify the behaviour of commands by adding options. Options are usually letters or words preceded by '-' or '–'. For example, to see more details of the files and directories available you can add the '-l' (l for long) option to *ls*:

```
guestXX@archer:~> ls -l
total 8
drwxr-sr-x 2 user z01 4096 Nov 13 14:47 bin
drwxr-sr-x 2 user z01 4096 Nov 13 14:47 work
```

If you want a description of a particular command and the options available you can access this using the *man* (MANual) command. For example, to show more information on *ls*:

```
guestXX@archer:~> man ls
Man: find all matching manual pages
 * ls (1)
   ls (1p)
Man: What manual page do you want?
Man:
```

In the manual, use the spacebar to move down a page, ‘u’ to move up, and ‘q’ to quit and exit back to the command line.

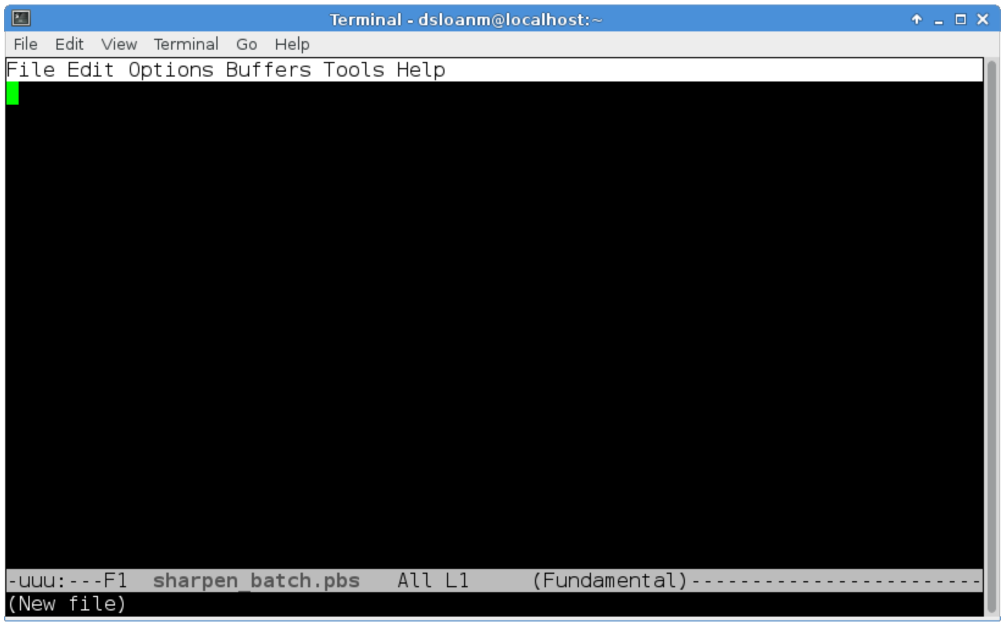
8.3 Using the Emacs text editor

As you do not have access to a windowing environment when using ARCHER, Emacs will be used in *in-terminal* mode. In this mode you can edit the file as usual but you must use keyboard shortcuts to run operations such as “save file” (remember, there are no menus that can be accessed using a mouse).

Start Emacs with the *emacs* command and the name of the file you wish to create. For example:

```
guestXX@archer:~> emacs Makefile
```

The terminal will change to show that you are now inside the Emacs text editor:


Terminal - dsloanm@localhost:~
File Edit View Terminal Go Help
File Edit Options Buffers Tools Help

```
-uuu:---F1  sharpen_batch.pbs   All L1     (Fundamental)------------------------
(New file)
```

Typing will insert text as you would expect and backspace will delete text. You use special key sequences (involving the Ctrl and Alt buttons) to save files, exit Emacs and so on.

Files can be saved using the sequence “Ctrl-x Ctrl-s” (usually abbreviated in Emacs documentation to “C-x C-s”). You should see the following briefly appear in the line at the bottom of the window (the minibuffer in Emacs-speak):

```
Wrote ./Makefile
```

To exit Emacs and return to the command line use the sequence “C-x C-c”. If you have changes in the file that have not yet been saved Emacs will prompt you (in the minibuffer) to ask if you want to save the changes or not.

Although you could edit files on your local machine using whichever windowed text editor you prefer it is useful to know enough to use an in-terminal editor as there will be times where you want to perform a quick edit that does not justify the hassle of editing and re-uploading.

8.4 Useful commands for examining files

There are a couple of commands that are useful for displaying the contents of plain text files on the command line that you can use to examine the contents of a file without having to open in in Emacs (if you want to edit a file then you will need to use Emacs). The commands are *cat* and *less*. *cat* simply prints the contents of the file to the terminal window and returns to the command line. For example:

```
guestXX@archer:~> cat build
rm -f percolate percolate.o  uni.o uni.mod *~ core
ftn -c uni.f90
ftn -c percolate.f90
ftn -o percolate percolate.o uni.o
```

This is fine for small files where the text fits in a single terminal window. For longer files you can use the *less* command. *less* gives you the ability to scroll up and down in the specified file. For example:

```
guestXX@archer:~> less percolate.f90
```

Once in *less* you can use the spacebar to scroll down and 'u' to scroll up. When you have finished examining the file you can use 'q' to exit *less* and return to the command line.