"GNU's Not Unix": Basics Of Computing

Kun-Chun Lee
(January 23, 2002)

It is quite an experience to see that your simulation reproduces the Maxwell distribution.

-Itamar Borukhov

I. INTRODUCTION

This article will serve as sort of like a computing journal/diary for the class. I will add more things into it every week, covering the things that you should learn in your free time for the following week in preparation for the simulation projects. The plan for the Molecular Dynamics simulation project, at least at the moment, is to study a particular phenomenon. You, of course, are responsible for figuring it out and write up on it. I will provide you with the skeleton of the MD, maybe with a few 'unintentional' bugs. And you are responsible for writing the analysis programs yourself when Andrea starts talking about liquids.

II. BASICS

A. Tools

The key to learn programming is to know where to get informations, and the stuffs you need. So, here are a few recommended sites that you may want to visit in your free time.

1. For general downloads, I recommend the following sites. However, if you are downloading things at home, remember to have the anti-virus scanners up and ready.

   (a) **www.gnu.org**
   This is sort of like the home of all free (legal) softwares out there. On this site, you can find things like compiler, development environments, operating systems (linux), etc. The things available are not restricted only to unix-like systems, you can find things for PC or MAC's here too.

   (b) **www.tucows.com** and **www.downloads.com**
   Again these are also sites for free stuff although alot of the listed stuffs on these sites comes with different license than those at GNU.

   At 'downloads.com', you can find something called 'cygwin', which is basically a set of tools so that you can have a shell environment on Window that's almost like unix, plus all the utilities and compilers, include fortran I believe. Believe me this program is a must have. The version you can download is the 1.31 version and it is free as far as I know, check the license. I have the 2nd version beta of cygwin, which I really like compare to the 1st because it is much faster; unfortunately, the final stable version is not free.

   Another program I like is something called 'sc' which you can get from 'tucows.com'. It's a terminal based spreadsheet program. I like it because it has a 'vi' (the standard unix text editor) like feel to it.

   And, of course, I also like to recommend to you 'vim', the improved 'vi', as the text editor of choice. I like it because the person who introduced me to unix used it, and he used it because the person who taught him used it, so on and so on. My colleagues in the office like something called 'emacs' in particular its GUI reincarnate 'xemacs'. So, don't say I am biased.

   (c) **www.gnuplot.org** and **www.octave.org**
   I told you guys a little about 'gnuplot' last quarter although I don't think any you actually used it. I highly recommend learning how to use 'gnuplot'. It can make your life so much easier, especially if you are trapped in the pits of 'Excel'.

   'Octave' is basically a 'mathlab' colone. I only used it a few times because I like Mathematica better. And I mainly use it for matrix multiplication.

2. For standard simulation related sites. Here are two of them. It is more important that you know they exists than that you read whatever they have.

(a) **www.dl.ac.uk/CCP/CCP5/librar.html**

This is the source of alot of simulation subroutines. This may come in handy if you decided to become hardcore in the field of simulation. Make sure to check the licenses, some of them comes with some restrictions. This site also talks about a book called 'Computer Simulation of Liquids', which the library has a couple of copies, if they have not been stolen. When I tried to check one out 2 years ago, they ended up telling me that it's missing. So, good luck.

(b) **www.ulib.org/webRoot/Books/Numerical_Recipes/bookfpdf.html**
www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf.html

These are the on-line versions of the famous 'Numerical Recipes In C (Fortran)'. It contains not only the subroutines for numerical work, but also discussion about how they work. However, don't just assume that what they described in the text is exactly what the subroutine does. They usually have the right principles, but the algorithm may turn out to be slightly different, maybe due to efficiency reasoning.

If you know of something computing related that may be useful in general, please tell me so that I can take a look.

One thing I have not listed is visualization softwares. I will give you one that's strictly for 2D systems, once I finished working it out, no gurantee though, in which case I will come up with a standard recommendation. At the moment, I know a few that's good for 3D system, but not 2D. If you know something good, please tell me.

B. Linux, Unix, and DOS

You should skip this part if you already know Unix.

There's really no easy way to learn Linux/Unix system except by actually using it. Although if you have some DOS background, it should come in handy. For the purpose of the course, I recommend you to try learning Linux/Unix as soon as possible. However, if you can get everything to work on the operating system that you feel most comfortable with then stick with it. As a start, you need to have a fortran compiler, which I have suggested one for window (I don't know too much about Mac and other computers.)

The way I learned Linux/Unix is buy a book and read through the first 70% of it. If this works for you, I recommend the book by Sobell called *A Practical Guide To The Unix System*. Unfortunately, our library has surprisingly few books on the basics of Linux/Unix.

Another way is to learn it on the job. What you do is to look up the stuff you need as you work and pray that your efficiency will grow exponentially with time, this method should work for people with DOS background, in theory. If you do decide on this, here are a few commands that may help you.

1. **help**

   Used to ask for assistance from someone.

2. **apropos** *keyword*

   This command lists everything in the man page associated with *keyword* in summary form.

3. **man** *command*

   This command shows you the manual for the command *command*.

Another great source of help on linux is internet and the usegroups, there's always something on something.

C. Machines

The main advantage with Linux/Unix is that you can either use silicon, or the O_2's in the computer room to run the program. They are quite reasonable for the things we need. I will check to see if Andrea can get everyone signed up on the O_2's.

In principle, you should be able to run the program on any system as long as you have access to a fortran compiler. And, in principle, each simulation run should not run for more than an hour although I have not really tested this. If it does take very long to run, tell me.

III. THE FORTRAN LANGUAGE

I have decided to stick with *fortran 77* as the choice of programming language for this class. You are welcomed to use any language as you like, as long as you can read fortran. You can come ask me for help on *fortran 77*. As a side remark, you are not here to learn how to be a programmer, just to know how, so if you run into trouble, don't hesitate to ask for help.

I will help you get started with basics of the commands. But, you should try to read up more on it if possible. I remember finding an on-line fortran manual. But, I have not being able to find it again. I will ask around. As a practice, you should try to have an easy access to a book on fortran.

A. Before You Start

Basically all you need to program are text editor and compiler atthough there are a few tools that can make your life easier. The idea of programming, or more precisely, computing, is just to write a few commands called codes to manipulate a few variable then output.

You store the codes in a text file in 'ascii' ('txt') format, not 'doc', 'xl', or '%#$'. For fortran, It is custom to store the codes in a file with a '.f' extension, just so that you can recognize it.

Compiling the code means translating it to a machine readable language (it's called the punch card in the old days). The detail is not important. The command to do it is 'f77'. Suppose you have a file called 'example.f', and you want to compile it, and you want the translated codes to be called 'executable', you type

```
f77 -O3 -o executable example.f
```

The '-o' after f77 is a flag that tells the compiler which file to store the executable codes (a.k.a. binary codes). There are other flags available for other purposes and they are all optional including '-o'; if '-o' is not specified the output goes to a file called 'a.out', this default name is due to historical reason. The one we will particularly be interested is '-O3', which is also used in the example. This flag tells the compiler to optimize the looping process. I used to think there's not much difference between unoptimized code and optimized code but after trying it out myself, I am convinced. To run the resulting code on most Linux clones you type './executable', and, on your first try, you can bet you will see messages that 'segmentation fault'. I will talk about these errors later.

B. Spacing

One of the thing I don't like about fortran is that every executable line needs to start on the 7th column, because fortran uses the first 6 for other purposes like numbering the line and commenting the lines. Let me show you with a few examples.

```
C            111           1
123456     123456     123456
```

First example tells the compiler that the line is being commented, and therefore, don't actually process it. You can use this line to add in description of what is going on in the program. On some compilers, it is essential that this column is only fill with C, or c, or empty, otherwise, the compiler would produce error.

Second example identifies the line with a number '111'. This is useful suppose you want to jump from different lines in the program. This is actually a very bad feature for a programming language because it makes the program hard to read.

Third example tells the compiler that this line is a continuation of the previous line because the compiler is not designed to read pass the 60th column. (I still need to check on this number though.)

Becareful of spacing, they are typically the major source of error. But, they are easy to pick up because a good compiler will tell you which line is violating the rule.

All my program examples from now on will be listed with these spacing criteria in mind.

C. Variables and Data Type

A variable is any unique name you can come up with that does not conflict with the language. In the machine, a variable is associated with a memory space. What you can do with a variable is basically to assign a number to this memory space, which you can access by using the variable.

Before you can actually use this space, you need to assign space, which you do by *declaring the variable*. It is important that you do this first before everything else, because once the compiler sees the first command it will assume that you have finished declaring variables. So this step usually is the first few lines of your program.

What you do is preface the name you want with a data type, like

```
integer name
```

In this example I declared 'name' as a data type 'integer', which can only store integers. The data type we will be working with are 'integer', 'double precision', and 'character', which is used to store text data type. To declared a multi-character, also called a string, variable you have to specified the number of characters, which you can do by the using declaraction 'character*number' where number is the maximum number of characters in the string. The following example declares a variable called name which can store upto 10 characters.

```
characters*10 name
```

Most programming language are very insistant about declaring data type. Fortran has a command which you can use to turn off, if you will, this restriction. I don't recommend it for beginners, so don't try to look it up. The reason I bring it up is because I used it in the skeleton MD, which means I will need to explain it eventually.

The procedure to assign a value to a variable is quite simple. Here are a few examples.

```
number = 10
name = 'Lee'
age =1.5
```

D. Arrays

An array is a variable which contains a list of the same thing. Like any variable, you need to declare it first (this is a must for an array) to use it. The command to declare it is

```
integer numbers(10)
character*20 names(10)
```

The second example declares 10 strings of length 20.

To assign or store information associated with an array, we need to know where in the array we need to access as idenfied by an integer. Suppose we have the 10 'names' array as declared earlier, and we want to stored my name into position 3, then, we type

```
names(3) = 'kunchun'
```

Suppose we really want to store it in position 2, we can type

```
names(2) = names(3)
```

The important thing to notice is that we can treat each indexed array variable as a simple element variable, so 'names(3)' is a variable which can only store one element.

E. Hello World And I/O

Typically at this stage, people begin writing they first program. To keep up with tradition, we will write, compile, and run a simple program that outputs the characters 'hello world'. (You will be surprised to see that virtually all programming books start with this example, including those that discusse graphical stuffs; how to make print hello world in your picture.) So, let's start.

The codes for a simple hello program is list below, I have made a more complicated to show the things we have learned so far.

```
C      THIS LINE IS USED TO COMMENT
C      PROGRAM: hello
C      SOURCE: hello.f
C      COMMAND TO COMPILE: f77 -o hello hello.f
C      COMMAND TO RUN: ./hello
C-----------------------------------------------------------------------
C      DECLARING VARIABLES
       character*80 hello
C      FINISHED DECLARING

C      ASSIGN VARIABLE
       hello='hello world'

C      PRINT VARIABLE
       print*, hello

C      ENDING THE PROGRAM
       end
```

F. Computer Logic

G. Subroutine And Function

H. Make It With Make

I. Debugging

J. Misc.

Now, let's discuss the features I used in the skeleton MD.