Lab #12

Physics 91SI Spring 2012

Objective: In this lab, you will see multiple examples of code written in other languages and then will be tasked with deciphering uncommented code and translating it into Python code that should replicate the desired behavior. You will also be asked to describe the functionality of a program in another language, line-by-line, giving an abstract definition of what each line does.

As usual, log on to `corn` and clone over the starter repository:

```
hg clone /afs/ir.stanford.edu/class/physics91SI/src/lab12 lab12
```

This folder contains 3 subdirectories, titled Fortran, IDL, and ROOT. Each part of the lab will work with files in these subdirectories and in this order. Since the files are in subdirectories, please make sure to `hg init` in the lab12 folder and when you want to `hg add` some files, you can just do `hg add ./Fortran/*` to, for example, just add all of the files inside to Fortran subdirectory.

Remember to `hg commit` often to save your changes, and submit your code at the end of class.

Part 1: Translating Fortran Code

Fortran is one of the oldest modern programming languages and so there's a lot of legacy code written in it. Some people even still use Fortran occasionally, because it can sometimes out-perform C, but in general, C is the go-to fast language.

`cd` into the Fortran subdirectory and you'll see a few example programs in `.f` files. These programs are small and contain some level of documentation/explanation of what they do. Read through them and try to identify familiar structures, like variable declarations and assignments, if statements, loops, I/O, etc. The programs have also been compiled, so you can run them by typing `./hello` or `./convert , etc`. If you even want to edit them and then re-compile them, you can create an executable by typing `f77 -o hello hello.f`, and then you can run the program with `./hello`.

Once you've read the example code, open up and read `mystery.f`. Read through the code and try to decipher what it's doing and what the purpose of the program is. You can search online for specific Fortran language features that you're unsure of, but the example files should adequately introduce the structures you'll see in the mystery code.

Now write a program in Python (call it solved.py) that will replicate the same behavior of the Fortran program, ie it should get input the same way and it should have the same output for a given input. Hint: to get input from the user, you should use response = raw_input("your question/prompt here"). Run the Fortran-compiled executable and your Python code and see if they give the same result!

Part 2: Translating IDL Code

IDL is another very old language, and so there's a lot of legacy code written in it as well; mostly in astronomy. Once again, cd into the IDL subdirectory and read some of the example

programs in the `.pro` files.

IDL is a proprietary language, so you won't be able to run any of the code, but the comments in the example code should give good enough explanation as to the function of the code.

Once you've read the example code, open up and read `mystery.pro`. Read through the code and try to decipher what it's doing and what the purpose of the program is, just as you did with the Fortran code. Perhaps you may be more inclined this time to search for online documentation of some IDL functions, and you are certainly welcome to do so.

Now write a program in Python (call it solved.py) that will replicate the same behavior of the IDL program. Hint: to get input from the command-line, just as the `PRO` arguments, `a`, `b`, and `c` do, remember to use sys.argv. You won't be able to run the IDL version of the code, but here's a sample input and output:

```
IDL> mystery, "This is a weird program" , "I do not like IDL", "popscicle"
First answer is :       23t like IDLogram
Second answer is:       -2
```

Part 3: Deciphering ROOT Code

As was mentioned last class, ROOT shows up most often in particle and nuclear physics, and again, in those fields, you'll see a lot of old code written in ROOT. More unlike legacy code written in Fortran or IDL though, ROOT is usually used to write very large scale simulations or data analyses, and so you're less likely to take the time to translate code away from ROOT than you are with likely smaller programs written in Fortran or IDL. Instead, if you ever come across ROOT code, you'll probably have to learn how to use ROOT.

In the ROOT subdirectory, you'll find example.c , which is a multi-root finder program. If you're familiar with C++, you'll notice it's basically just the C++ language, with some ROOT-specific libraries `#include`-d in.

Read over the code first and then go through each line in the `exampleMultiRoot` function, and add a comment to each line, explaining in abstract/language-agnostic sense, what that line of code does. This should be pretty straightforward, but it's a good way to make sure you really understand what's going on in a new language.