Physics 40: Laboratory One

Tuesday, March 31, 2020

Note: The general information below can be found on the course website.
`http://scalettar.physics.ucdavis.edu/p40/p40_S2019.html`

General Information:

Class meets 12:10 pm - 3:00 pm Tuesdays and Thursdays in Physics 285.

Additional problem sessions Wednesdays, noon - 2pm; Physics 285. Attendance optional.

Grading:

Problem sets, each due on Friday, 5:00 pm: 60%.

Weekly quizzes, Thursdays, start of class, 20%.

Final Project, due Friday June 7, 5:00 pm: 20%.

Text: None required. However, it is very useful to have a book on computational physics, and guides to C and python programming. I suggest: "Numerical Methods for Physics", Alejandro Garcia, which is available in both C++ and Python versions ($19 and $16 respectively on Amazon): http://www.algarcia.org/nummeth/nummeth.html. These cover both the programming and tools in one swell foop, and hence are probably your best bet as a single, efficient, source. Longer books on the languages themselves are "Teach yourself C++" by Al Stevens, "Think Python" by Allen B. Downey. Everything you need to know will be provided in class, so it is up to you to decide if background reading helps you learn better.

Course Goals/Outline: The goal of this course is to familiarize you with the basic tools and algorithms for using a computer to do physics problems (and "scientific computing" more generally). Concerning tools, we will learn both C and python. We will, however, not attempt a formal presentation of these languages. Instead we will learn by example, writing codes from the most simple "hello world" to, ultimately, more complex ones to implement molecular dynamics, solve Laplace's equation, diagonalize matrices, etc. Similarly, we will not essay a formal presentation of numerical analysis. Instead we will learn the material through applying tools and concepts to some specific problems.

Required Background: My experience is that there is a *very* broad range of student backgrounds in computational physics courses. Therefore, I will not assume you know *anything* about coding or computational methods. I will provide supplementary problems to those students who do have a deep background, so they can push forward as well.

Required Hardware and Software: You will be doing your work on your own laptops, but you will be able to compile and run your programs anywhere, eg the Physics Department Computer Room (Physics 106). You will need some sort of editor to type programs in, and a C compiler (and python, later in the quarter). You should have received several emails about installing the compiler prior to the first class. If you do not have a compiler installed, we will try to help you do so in class today.

Today's Goal: Write and compile nine very simple C programs.

[0] Introduction.

[1] Discussion of programs.

[2] Fire up your computer and open up an editing window.

[3] Type in the following 'hello world' C program. Then compile and run it.

```
#include <stdio.h>
int main()
{
    printf("Hello, world\n");
    return 0;
}
```

[4] Type in the following C program which adds two numbers. Then compile and run it.

```
#include <stdio.h>
#include <math.h>
int main()
{
    double x,y,z;
    printf("\nEnter x\n");
    scanf("%lf",&x);
    printf("\nEnter y\n");
    scanf("%lf",&y);
    z=x+y;
    printf("\nx+y=%12.8lf",z);
    printf("\nx+y=%lf\n",z);
    return 0;
}
```

[5] Type in the following C program. Compile and run it. What does it do?

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    int i,j,k;
    printf("\nEnter i\n");
    scanf("%i",&i);
    printf("\nEnter j\n");
    scanf("%i",&j);
    k=i%j;
    printf("%8i\n",k);
    return 0;
}
```

[6] Type in the following C program. Compile and run it.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
int j;
for(j=0; j<20; j=j+2)
{
printf("\n%i",j);
}
printf("\n");
return 0;
}
```

[7] Type in the following C program. Compile and run it.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
    float x,y;
    printf("\n\aEnter x");
    printf("\n");
    scanf("%f",&x);
    printf("\nEnter y");
    printf("\n");
    scanf("%f",&y);
    if (x==y) {
    printf("\nthe numbers are equal");
    }
    if (x!=y) {
    printf("\nthe numbers are not equal");
    }
    printf("\n");
    return 0;
}
```

[8] Type in the following C program. Compile and run it.

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   float x,y;
   printf("\n\aEnter x\n");
   scanf("%f",&x);
   printf("\nEnter y\n");
   scanf("%f",&y);
   if (x < y && x>0 ) { printf("\n x<y and x>0\n");}
   else { printf("\n either x was not less than y, or x was not positive\n");}
   return 0;
}
```

[9] Type in the following C program. Compile and run it. What does it do?

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   int j;
   j=43;
   do
   {
      j=j-1;
      printf("\n%i",j);
   } while(j>0);
   return 0;
}
```

When you have finished these programs, call me or the TA over. We will look at a few of them briefly and ask questions. Then you may leave.

The first part of Problem Set One (due Friday April 5, 5:00 pm) involves you writing two programs 'on your own'

[PS1-1] Modify the program in [6] to print out the first 20 multiples of 7.

[PS1-2] Write a program to read in a number and decide if it is odd or even. Hint: Using the function in [5] is one good way to go. You will also need ingredients from [7] or [8].

Optional Goal 1: The Collatz Conjecture

A fascinating thing about mathematics (and science/engineering in general) is that some remarkably simply problems are unsolved. One of these is the "Collatz Conjecture":

Start with a positive integer, and then repeatedly execute the following algorithm. If the integer is even, divide it by two. If it is odd, multiply by three and add one. Stop when you get to the number one.

For example, beginning with "7" we get the sequence:

$$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to \mathbf{1}$$

The Collatz Conjecture asserts that this process always ends at one no matter what positive integer you start with.

- Write a program to test the Collatz Conjecture for a single (positive) integer starting point.
- Write a program to test the Collatz Conjecture for the first N positive integers.
- If you make N large enough, at some point your program will run into trouble. What sorts of problems might you anticipate? How can you fix them?

Optional Goal 2: The Locker Problem

To do this problem you will need to know about arrays. (We will cover this in class in the next few weeks.)

At the local high school there is a long row of 1000 lockers, one for each student.

The lockers start with their doors wide open, and, one-by-one, each of the 1000 students passes by the lockers. Each student changes the state of every locker she examines. That is, if she finds a locker open then she closes it, and if she finds it closed, she opens it.

Student 1 looks at all the lockers: 1, 2, 3, 4, 5, 6, ...
Student 2 looks only at lockers: 2, 4, 6, 8, 10, 12, ... (multiples of 2)
Student 3 looks only at lockers: 3, 6, 9, 12, 15, 18, ... (multiples of 3)
Student 4 looks only at lockers: 4, 8, 12, 16,20, 24, ... (multiples of 4)
and so forth

Question: How many lockers are open at the end of the process? Which ones?

- Write a C program to solve this problem. You will probably want to use an array to store the 'state' of the locker (open or closed). Do $N = 1000$.

- You'll probably recognize the pattern of closed locker numbers. Can you prove your answer?