Physics 45 Winter 2023: Laboratory Seven
Week of January 30 to February 3, 2023

We have been defining variables one at a time using statements like:

```c
int j,N;
double dt,t=0.,x,y,Msun,Mearth,r,vx,vy,G;
double E,Lz;
```

at the top of our codes. Suppose you wanted to do something like keep track of the homework scores of \( j = 1, 2, 3, \ldots, 200 \) students in a class using a C code. You could define 200 variables:

```c
double hw001,hw002,hw003,hw004,hw005,hw006,hw007,hw008,hw009,hw010;
double hw011,hw012,hw013,hw014,hw015,hw016,hw017,hw018,hw019,hw020;
double hw021,hw022,hw023,hw024,hw025,hw026,hw027,hw028,hw029,hw030;
[...]
double hw191,hw192,hw193,hw194,hw195,hw196,hw197,hw198,hw199,hw200;
```

That would be pretty tedious!

Even worse, suppose your grading rubric was that the final grade was determined by the average of the homework score and the quiz score. You would need to define 200 more quiz variables and 200 more average variables, and then compute for each student:

```c
ave001=(hw001+quiz001)/2.0;
ave002=(hw002+quiz002)/2.0;
ave003=(hw003+quiz003)/2.0;
[...]
ave200=(hw200+quiz200)/2.0;
```

That’s 200 lines of code!

**There has to be a better way!**

The answer is to define an “array”. You can think of an array as a trick to define 200 variables all at once. The line

```c
double hw[200],quiz[200],ave[200];
```

tells the computer that there are 200 variables with names

```c
hw[0],hw[1],hw[2], \ldots \ hw[199];
```

and similarly for quiz and ave.

**Important:** Note that C begins counting from zero, so the names begin \( \text{hw}[0] \) and end \( \text{hw}[199] \) rather than \( \text{hw}[1] \) and \( \text{hw}[200] \).
Example 1

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    int i,x[20];
    for (i=0; i<20; i=i+1)
    {
        x[i]=i*i;
    }
    for (i=1; i<20; i=i+1)
    {
        printf("%6i %10i\n",i,x[i]);
    }
    fprintf(stdout,"\n");
}
```

This code defines the array `x`, and tells the computer it has twenty elements. Then, in the first loop, the code fills the element `x[i]` with the square of `i`. The second loop prints out the values of `i` and `x[i]`. Try typing up this code, compiling it and running it for practice.

Example 2 The program below puts all the values of factorial into an array.

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    int j;
    long int temp,fact[20];
    temp=1;
    for (j=1; j<20; j=j+1)
    {
        temp=temp*j;
        fact[j]=temp;
    }
    for (j=1; j<20; j=j+1)
    {
        printf("\n%10d %20ld\n",j,fact[j]);
    }
    printf("\n");
}
```

Type it in, compile and run it, and see that it works. The code illustrates another useful aspect of arrays. They allow us to save a bunch of values for future use instead of recomputing them all the time.
**Important:** Up to now when we used 'printf' for an integer we used %i. It is also okay to use %d, and I have done that above. To print a long int, one uses %ld. Remember too that the ‘10’ in %10d tells the computer how much space to allocate for the number to be printed. This allows for prettier formatting. (Compare the output of the code above to one without the numbers to specify the space for printing.)

**Example 3:** As we have seen, to do molecular dynamics, you only need to keep track of the present value of the position and velocity and update them many times in a loop. But suppose for some reason you wanted to save the trajectory for all the time steps. Arrays let you do that. Type in this code, compile, and run it:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    double x[20000],v[20000],k,m,dt,a,t=0.;
    int i,N;
    printf("\nEnter k,m,dt,N ");
    scanf("%lf %lf %lf %i",&k,&m,&dt,&N);
    x[0]=5.0;
    v[0]=0.0;
    for (i=0; i<N+1; i=i+1)
    {
        x[i+1]=x[i]+v[i]*dt;
        a=-k*x[i+1]/m;
        v[i+1]=v[i]+a*dt;
        t=t+dt;
    }
    printf("\nEnter a time step at which you want to know");
    printf("\nthe position and velocity: ");
    scanf("%d",&i);
    printf("\nThe position at step %5i is %8.4lf",i,x[i]);
    printf("\nThe velocity at step %5i is %8.4lf
",i,v[i]);
}
```

As you can see, because the trajectory is stored in the array x[] you can go back and examine the position at any time step you want!
[HW4-1] Explain what the lines (“leapfrog algorithm”) 
\[ \begin{align*} 
    x[i+1] &= x[i] + v[i] \cdot dt; \\
    a &= -k \cdot x[i+1] / m; \\
    v[i+1] &= v[i] + a \cdot dt; 
\end{align*} \]
are doing, and contrast it with (“Euler algorithm”) 
\[ \begin{align*} 
    x[i+1] &= x[i] + v[i] \cdot dt; \\
    a &= -k \cdot x[i] / m; \\
    v[i+1] &= v[i] + a \cdot dt; 
\end{align*} \]

[HW4-2] Run the code with these values:

Enter k, m, dt, N  2.5 .001 10000
Enter a time step at which you want to know
the position and velocity:  785

What do you get for the position and velocity? Explain why those values make sense. What about the position and velocity at time step 7854?

[HW4-3] Make the change from leapfrog to Euler. What’s different about the position and velocity at time step 7854? Does this fit what we discussed in class?
Review: We have learned how to take derivatives and to do integrals on the computer. We also learned how to solve specific differential equation, Newton's second law:

\[ m \frac{d^2 x}{dt^2} = F(x, \frac{dx}{dt}) \]

For example, for a mass on a spring with friction \( F(x, \frac{dx}{dt}) = F(b, v) = -k x - b v. \)

Now we will learn how to solve partial differential equations(!!). Our first application will be to the diffusion equation. Arrays will be useful.

Review your code for taking a first derivative.

\[ \frac{df}{dx} \equiv \lim_{dx \to 0} \frac{f(x + dx) - f(x)}{dx} \]

```c
#include <stdio.h>
#include <math.h>
double myfunction();

int main(void)
{
    double x, dx, A, B, deriv;
    printf("Enter x and dx \n");
    scanf("%lf %lf", &x, &dx);
    A = myfunction(x);
    B = myfunction(x + dx);
    deriv = (B - A) / dx;
    printf(" df/dx= %lf \n", deriv);
}

double myfunction(double x)
{
    double fofx;
    fofx = x * x;
    return fofx;
}
```
[HW4-4] Write a similar code to take second derivatives. Use $f(x) = x^4$ as your function.

[HW4-5] What does your second derivative program give for $f(x) = x^4$ and $(x, dx) = (0.5, 0.1); (x, dx) = (0.5, 0.01); (x, dx) = (0.5, 0.001)$? Discuss.

[HW4-6] Modify your code to get the second derivative of the function $f(x) = e^x + 2/x$. What do you get for $(x, dx) = (1.0, 0.1)$ and for $(x, dx) = (1.0, 0.01)$ and for $(x, dx) = (1.0, 0.001)$? Discuss.

[HW4-7] Following the discussion in class, write a code to solve the diffusion equation in 1D. Write it for $N = 1000$ boxes (intervals in $x$). Use as an initial condition a density profile with all the particles in the central box, that is rho[500] = 1.0 and rho[j] = 0.0 for all $j \neq 500$. We will discuss it more and run it in the next Lab.