Notes on using arrays in C

In mathematics we often use indices to refer to an entire list of variables in one symbol. In linear algebra such lists are called "vectors"; the Cartesian vectors you are familiar withm, with three components, are just an example. We write particular elements within that list with subscripts; for instance, the tenth number within the collection x is called x_{10}.

We have been using this notation in class for a while; in the binary star system, our dynamical variables were $x_1, y_1, x_2, y_2, v_{x_1}, v_{y_1}, v_{x_2}$, and v_{y_2}. However, if we wanted to replicate that galaxy simulation from the beginning of class, we might have ten thousand stars, and sixty thousand dynamical variables:

- Ten thousand x's
- Ten thousand y's
- Ten thousand z's
- Ten thousand v_x's
- Ten thousand v_y's
- Ten thousand v_z's

Obviously we aren't going to write 120,000 lines of code whenever we want to do an RK2 update. What we need is a way to collect all ten thousand x variables into one C object and manipulate them at once.

You can do this; it's called an *array*. Since we can't write subscripts in C, we use square brackets.

You declare an array with N elements like this:

```
double x[N];
```

IMPORTANT: These N elements are called `x[0]` through `x[N-1]`. There is no `x[N]`; attempts to access it will fail.

Typically, we use integer counter variables (traditionally called `i,j,k...`) and a `for` loop to cycle through all the elements of an array. For instance, the following code sets all the elements of two arrays to zero:

```
double vx[N],vy[N];
int i;

for (i=0;i<N;i++)
  vx[i]=vy[i]=0; // yes, you can do this!
```

ALSO IMPORTANT: Arrays work differently with functions than other variables. When you pass another variable to a function, the function creates its own local copy of it, completely independent of the copy in the calling function. However, when you pass an array, only the location in memory is passed to the function. So, *if you manipulate an element of an array parameter within a function, it is changed outside the function as well.* You can pass *elements* of an array to a function without anything different happening; this new behavior happens only when you pass an entire array. For instance:

```
void f(int i, int j[]) // this is how you pass an array
{
  i=3;
  j[2]=10; // will be changed in the calling function as well!
}

int main(void)
{
  int x,y[10];
  x=1;
  y[0]=3;
  y[2]=5;

  f(x,y);   // this will change y[2], but leave x alone
  f(y[0],y); // this will also change y[2] (since it was passed as part of an array),
             // but leave y[0] alone (since it was passed as a single number)
}
```