Notes on command-line parameters

As you know, programs can read parameters from the command line. For instance, when you type

```
gedit string.c
```

you are running the program `gedit`, which is a C program just like the ones you are writing. It wakes up and spits out a window for you to type in, but how does it know that you want to edit the file `string.c`?

It turns out that command-line arguments get turned into parameters to `main` that work like any other function parameters. However, these arguments are *strings* – globs of text – that you haven't learned how to work with yet. However, it's pretty simple to turn them back into numbers.

Here's a simple C program:

```
#include <stdio.h>
#include <stdlib.h>

int main(int numparams, char **paramlist)
{
  if (numparams < 3) {printf("!Usage: ./string <N> <dt>\n"); exit(0);}
  int N=atoi(paramlist[1]);
  double dt=atof(paramlist[2]);
}
```

You run this program by doing something like `./string 100 1e-4`; this will store 100 in `N` and 1e-4 in `dt`. It looks a bit messy, but it's really not that bad. How does this work?

When `main` is run, the parameter `numparams` is set to the number of parameters specified on the command line, *plus one*. (The first parameter is always the name of the program; that's the plus-one.) So for this program, numparams should be 3: the name of the program, plus the values for `N` and `dt` specified on the command line.

The parameters themselves are stored in an array called (here) `paramlist`. The * character is another way to specify that the following thing is an array: the two asterisks mean that `paramlist` is an array of arrays. Why is that? Well, you have a list of parameters: that's one array. Then, each parameter is a string, which is a list of characters in itself, so each parameter is itself an array.

The first thing you should do when using this approach is to use an `if` statement to check to see that the user has typed in the right number of parameters, and then print a reminder and exit if not. Otherwise, you will wind up doing horrid things like driving off the end of arrays, resulting in unpleasantness and barfage when you try to run the code.

Then, you can use two new functions to convert these strings (holding a text representation of "1e-4") to numbers. This is done by the functions "atof" and "atoi", which convert strings to floating-point (`float` or `double`) and integer values, respectively. You need to include `stdlib.h` if you're going to use them. Note that `paramlist[0]` will hold the value `string` (the name of the program); the actual parameters start at `paramlist[1]`.