```
vector < vector <float> > our_matrix {{some_val1, some_val4, some_val7},
                                      {some_val2, some_val5, some_val8},
                                      {some_val3, some_val6, some_val9}};
```

Note that the values in the above should be floats given how the inner vector is defined; if they are doubles, you will likely get a compilation error due to loss of precision.

Modulo (%) in C++ and Negative Numbers

If you type `-1 % 5` into either Python script or Google, you should get `4` as the answer. However, if you do the same in C++, you'll get `1`.

Why? Well, it isn't actually *modulo* in C++, but the remainder! Check out **this Stack Overflow post** if you want to read about why this is.

As you may remember, Sebastian used the *modulo* operator when writing his code in Python for localization. If you don't account for the difference between these implementations in Python vs. C++, you may end up with a **segmentation fault** when you try to call an index outside of the size of your vector!

If you're stuck on how to deal with this difference between Python and C++, take a look at the top answers **here** for some useful tips.

Segmentation Faults

One potential cause of these is concerning modulo as described above, but there is also another potential cause.

Depending on your implementation, after you've coded `blur()`, running the out file from `tests.cpp` may produce a segmentation fault when the tests reach `move()` (i.e. `normalize()`, `blur()` and `initialize_beliefs()` will show their test results, and then the fault occurs). Typically this should resolve itself once `newGrid` within the `move()` function takes shape.

Additional Simulation (*optional*)

While all you need to pass the project is to pass all tests in `tests.cpp`, you can also visualize a simulation of your localizer with `simulate.cpp`. In order to do so, the first step is to uncomment all the lines at the bottom of the file around the `main()` function