Exercise:
Tune labs

Imagine you have a 3-dimensional box with NUM_OF_PARTS particles, that are randomly distributed within the box of size 1x1x1 (in arbitrary units).
Use the template files in this directory to write two programs, with two different data structure layouts (as shown in the lecture). There should be 5 source files (.c) and 5 header files (.h) and a Makefile.

Part 1:

You need to provide two subroutines, distance(...) and distcheck(...), for both types of data structures. These are already defined in distance.c and distcheck.c respectively, but you need to replace their dummy content with the actual computations.

In distance(...) you should calculate the distance of each particle to the point (0,0,0) and the total sum of all distances. In distcheck(...), you sum up only the distances, i.e. you should only access the distance part of your data. Print both results during the test phase to have a check for consistency.

You need also to provide a measure for the number of floating point operations in each subroutine, counted per particle. These numbers, DIST_FLOP and CHECK_FLOP, will be used in main.c to calculate the Mflop/s numbers. These are defined in the header files associated with the function.
Hint: For the sake of simplicity, assume that the sqrt() function counts 4 FLOPs.

The program takes two command line arguments: the number of loops and the number of particles. The first one has a default value of 1, the latter the compiled in default NUM_OF_PARTS (see data.h). With those two parameters, you can control the size (memory footprint) and runtime of your problem.
Both programs print out 5 values (on a single line):

1. Memory footprint in kB
2. Mflop/s of distance(...)
3. Mflop/s of distcheck(...)
4. Mflop/s of total program
5. Runtime in secs

You can check those values easily: For a given memory footprint, the Mflop/s numbers shouldn't vary much when varying the number of loops, and the runtime in secs should correspond approximately to the value you can obtain with the 'time program ...' command.

During the test phase, you can undefine the -DDATA_ANALYSIS in the Makefile, to get a more verbose output.

Part 2:

Run a series of experiments for both programs, e.g. for 1000 ... 400000 particles (choose the step size in a clever way) and 10000 loops.

Vary also the number of filling bytes (NBYTES) and re-run the series of experiments again (need to re-compile, unless you make this value also dynamic in your code). Plot the runtimes and Mflop/s numbers versus the memory footprint, as shown in the lecture notes. Under which conditions is one version faster, when the other one?

Optional: You have access to systems with different L2-cache sizes, i.e. systems based on AMD Opteron CPUs (access through linuxsh), and systems based on Intel Xeon 5550s (access through qrsh). Try to figure out the differences between those two CPU types by running a series of experiments on both machines. Can you tell the difference between those CPUs and their caches?

Part 3:

Note: Only possible after the analyzer has been introduced.

Use the Sun Studio analyzer to find out more about the program. Start with memory footprint values from part 2, that are below and above the level 2 cache size (how can you tell from the plots in part 2?), and do the following experiments.

a) Run a 'standard' experiment with the analyzer and look for the 'hot spots' in both programs

b) Use the hardware counter to compare the L2 D-cache references and L2 D-cache misses for both programs and different system sizes (memory footprints).

c) Repeat b) for the L3 cache.