Recap from last time (Session 26):

- The Qt Designer is a convenient but not essential tool in building interfaces. You could code everything by hand.

However, unless the interface is quite small and likely to have a simple, static design, it will be much quicker and easier to use Designer to place widgets.

I encourage you to finish Session 26 on your own, including the OpenGL example at the end.

- What's missing in our Qt introduction?

  - Let's see the examples that come with the distribution.
  
  - Any sophisticated interface will have a menu bar and possibly a toolbar. We implement such things in Qt using a "MainWindow" widget.

  In Qt Designer, this is one of the options under 
  File \rightarrow New (cf. Dialog).

  Sending the menus up through Designer is definitely recommended.

  The plan originally was to have you build an application with menus and a toolbar that used the random walk program from Session 19.

  This would also illustrate how to use QPainter to plot points, etc.

  If we have time, we'll go through that in the last period.

  However, the first step was to rewrite the random walk code using classes, which leads us to today's main topic: a (brief) look at C++ in C++ for computational problems.

Also today: Voronoi and Monte Carlo (1st time); stop at noon for SEPs (questions on sheet)
Many (maybe most) of you have more experience than I do with C++ classes, since I've never taken a C++ course and have only really used the "C" aspects of C++. In practice, the "technical" aspects of not using classes is not a problem. The difficulty is deciding what classes to make. We'll try to make a representation of that. I think a topic worth exploring.

• Using classes is not essential in computational physics; we've gotten along just fine without them.
• But classes and their implementation of basic C++ characteristics has great potential, particularly for larger codes.

Unfortunately, I've not found the discussions in the literature on scientific C++ to be either clear or concise or accessible in a short period of time.

I've included two excerpts with discussions and examples in today's handouts:

• The Landau/Paz example is easy to read, but I don't think it is a particularly good example.
• The Bostock/Noackman excerpt is well worth reading but it takes some effort. They talk about storing and manipulating a mesh for finite element method programs as an example. Three possible implementations are compared and critiqued.

The critiques show why this is a difficult issue: deciding on what classes to make is critical but guidelines seem (to me) very sketchy.

We'll be less ambitious and start with a very simple example, creating a random walk.

You should first look at the old program to remember what we did.
Stability

- The old code is now called `random_walkogany.cpp`
  and just had a main function.
  - The number of steps is an input.
  - The output (to a file) is a set of coordinates
    representing those steps.
  - You can plot with gnuplot and get something like
    ![Diagram](image)
    (if you connect the points with lines).

- How would you implement this using classes?
  - What would you like the test program to look like?
    - create a random_walk object
    - get the number of points from the user
    - loop through the points
    - take a step
    - get the new coordinates
    - print the coordinates to a file

- Use this to decide on "methods." What variables
  should be "private" to the class? And so on.

- Compare your ideas to my implementation (see handouts).
  - Then try to extend it.

  - being able to easily and reliably extend your code is
    one of the main motivations.
  - avoid global variables that can be changed by users.