4/19/04

Monday 780.20 Class

Plan: Spend today on Qt practice and then let it rest for a while. We'll come back to explore Qt "Designer," which is a development environment for Qt applications, after you've had a chance to do an assignment using Qt.

On Wednesday we'll have a look at integral equations (and some associated physics).

* Please print out and read sections 18.0-18.3 of "Numerical Recipes in C." Links to these sections can be found on the 780 web page.

Today: The main goal is to see how to use a Qt interface to pass some input parameters to a computational code and then to display the results. (For Assignment #2, you'll develop an interface for the nonlinear pendulum simulation.)

The Session 13 sheet continues or extends what you did last time.

The "widgets" application is from the Qt examples. It is not incredibly well documented, but you can fairly easily search the code to find out how something is done (and try changing or extending it).

Make sure you at least try out the Hilbert matrix example.

Also a connection to discussion of object-oriented programming.
4/19/04

Some comments/observations on Qt:

Compiling and linking procedure

1. qmake-project
   makes a .pro file according to the name of the directory. Do not do this step for the "widgets" application. You may need to edit the .pro file to add additional info, such as libraries to link to (e.g., gsl), or the location of pixmaps/images your application will use.

2. qmake
   Create a makefile (called Makefile) based on the .pro file. This will have references to the specific version of Qt you're using, so you will often need to repeat this step if you switch machines (even if both are Linux).

3. make
   Compile and link the program according to Makefile. We don't need to name the makefile as we usually do, since we are putting only one application in a given subdirectory.

A widget is a window gadget. Technically, an object derived from QWidget. Buttons, labels, menus, text boxes, dialogs, ... are all widgets.

To use a widget class, you need to include the corresponding header file name. Almost always the class name in lowercase with _h appended.

   Application  →  application.h
   QLabel       →  qlabel.h

Signal and Slots

A widget emits a signal that something has happened without knowing what will be done with it.
4/19/64

A slot is a declared function, which doesn't need to know about what signals will connect to it. 

=> Strict separation

Basic setup:

class MyClass : public QObject

Q_OBJECT

Signals:
void somethingDone();

public slots:
void slotDoSomething();

private slots:
void slotDoSomethingInternal();

- Other buttons movements and clicks (not to do with Qt widgets) are not processed with signals and slots but with virtual methods.
- These are considered "low-level events."

Basic idea for simplest interface:

main.c (main.cpp) with QApplication definition, show command, and the exec command

include QWidgets/QtBase.h only one main widget with a bunch of subwidgets (spin boxes, text boxes, menus, etc.)

3) The computational code in the form of a subroutine or subroutines that are called either from a standard driver program or from the Qt interface. (Define class instead?)