Lab 12: GUI programming with Qt

Comp Sci 1585
Data Structures Lab:
Tools for Computer Scientists

Missouri S&T | Computer Science

Outline

Outline

Qt (Pronounced "cute")

- `https://www.qt.io/what-is-qt/`
- `https://showroom.qt.io/`
- `https://en.wikipedia.org/wiki/Qt_(software)`

Qt is used for developing multi-platform applications and graphical user interfaces (GUIs)

- So far, we've only created command-line applications
- GUIs can be nice at times, though
- Qt: cross-platform framework (works on Windows, Mac OS X, Linux, etc)
- Little to no underlying changes needed to port from one system to another
- Native OS capabilities and speed

Making GUIs with Qt 4

Besides making clickable programs, learning to program GUIs will give you several other skills with C++

- Event-based programming
- Working with a (very) large library
- Managing memory in more complicated programs

Getting Started

```
#include <QtGui>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QLabel hello("Hello World!");

    hello.resize(250, 150);
    hello.setWindowTitle("Simple example");
    hello.show();

    return app.exec();
}
```

Outline

Building Qt Applications

- Qt has its own preprocessor, the Meta Object Compiler (aka `moc`)
- `qmake-qt4` manages Qt projects and generates makefiles automatically
  - `$ qmake-qt4 -project` will make a project file (ends in .pro) that configures the makefile
  - `$ qmake-qt4` makes a makefile
- So, to build a Qt project:

```
$ qmake-qt4 -project; qmake-qt4; make
```

Qt Overview

- There is one, and only one, QApplication
- `qApp` is a global pointer to the QApplication
- Everything clickable is called a ‘widget’
- Widgets can hold other widgets
- A widget with no parent becomes a window

Outline

A Simple Notepad

```
#include<QApplication>
#include<QTextEdit>

int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit te;
  te.setWindowTitle("Not Vim");
  te.show();

  return app.exec();
}
```

Outline

Composite Objects

- Widgets can be added to another widget with the `addWidget()` function
- You can use a Layout to specify how the widgets are organized
- Memory Management: `addWidget()` takes a pointer and is responsible for cleaning up all its children

Layout Example

```
#include<QtGui>

int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit* te = new QTextEdit;
  QPushButton* quit = new QPushButton("&Quit");

  QVBoxLayout* layout = new QVBoxLayout;
  layout->addWidget(quit);
  layout->addWidget(te);

  QWidget window;
  window.setLayout(layout);

  window.show();

  return app.exec();
}
```

Outline

Making Buttons Do Things

- Qt is event-driven: QApplication monitors what the user does and sends events to widgets when something happens
- Signal: An event caused by a widget: button click, key press, etc.
- Slot: An action taken when a signal is sent
- Signals are connected to slots by using the `connect(...)` function
  e.g.
  ```
  connect(source-object, SIGNAL(signal_name()),
  destination-object, SLOT(slot_name()))
  ```
  connects signals to slots

Actually Quitting

```
#include<QtGui>
int main(int argc, char** argv)
{
  QApplication app(argc,argv);

  QTextEdit* te = new QTextEdit;
  QPushButton* quit = new QPushButton("&Quit");

  QObject::connect(quit, SIGNAL(clicked()),
      qApp, SLOT(quit()));

  QVBoxLayout* layout = new QVBoxLayout;
  layout->addWidget(quit);
  layout->addWidget(te);

  QWidget window;
  window.setLayout(layout);

  window.show();

  return app.exec();
}
```

Writing your own slot

- In order to make your own slots, you need to make a custom `QWidget` class
- In addition to public and private functions and members, QObjects have public and private slots
- A slot is just a function that gets called whenever a signal connected to it is sent

Example: `ask-quit`

Outline

Menus and Toolbars

- `QMainWindow` is a class for making standard applications with menus and toolbars
- `setCentralWidget()` sets the widget that fills the window
- `menuBar()` returns a pointer to the menubar, which you can use to add new menus
- `addToolbar()` creates a new toolbar
- To avoid repeating a lot of code, you can add a `QAction` to both a menu and a toolbar
- Then you can connect that one action to various slots

Example: `menus`

Outline

Sending Data Between Signals and Slots

- So far, we've used predefined signals
  e.g. `QPushButton::clicked()`
  e.g. `QAction::triggered()`
- `connect()` dictates which signals trigger which slots
  e.g. `openAction::triggered()` executes `Notepad::open()`
- Custom slots were responsible for gathering data
  - `Notepad::open()` promped user to select file
  - Grabbed filename
  - Tried to open; complained if it couldn't
  - Loaded file contents into `QTextEdit` instance
- Another approach: have signals send data to slots
- Solution: declare your own signals!

Sending Data Between Signals and Slots

- You can declare your own signals in the `signals:` section of your header files
- Then, custom slots can `emit` these signals: `emit signal-name();`
- You don't actually implement signals, just declare, emit, and connect them
- Signals can carry data, just add parameters
- Connect that signal to a slot that takes the same arguments
- The slot will be called with the data you use when you emit the signal

Example: `title`

Outline

Qt-creator

Qt makes an IDE for developing Qt applications:
`https://en.wikipedia.org/wiki/Qt_Creator`