Prof. Dr. Andreas Butz

Dipl.-Medieninf. Hendrik Richter
Dipl.-Medieninf. Raphael Wimmer

Computergrafik 1 Übung

C++ (Memory Management)

Debugging

Qt

Speicherverwaltung

Speicherverwaltung

Automatische Speicherverwaltung:

- Java/ .NET
- Garbage Collector
- gesteuert durch Programmkontrollfluss

Manuelle Speicherverwaltung:

- C++
- Anforderung von Speicherplatz aus dem Heap (Halde)
- Verwaltung von Speicherplatz
- Rückgabe
- Containerklassen der STL kümmern sich selber um Speicherverwaltung

http://www.c-plusplus.de/forum/viewtopic-var-t-is-159231.html

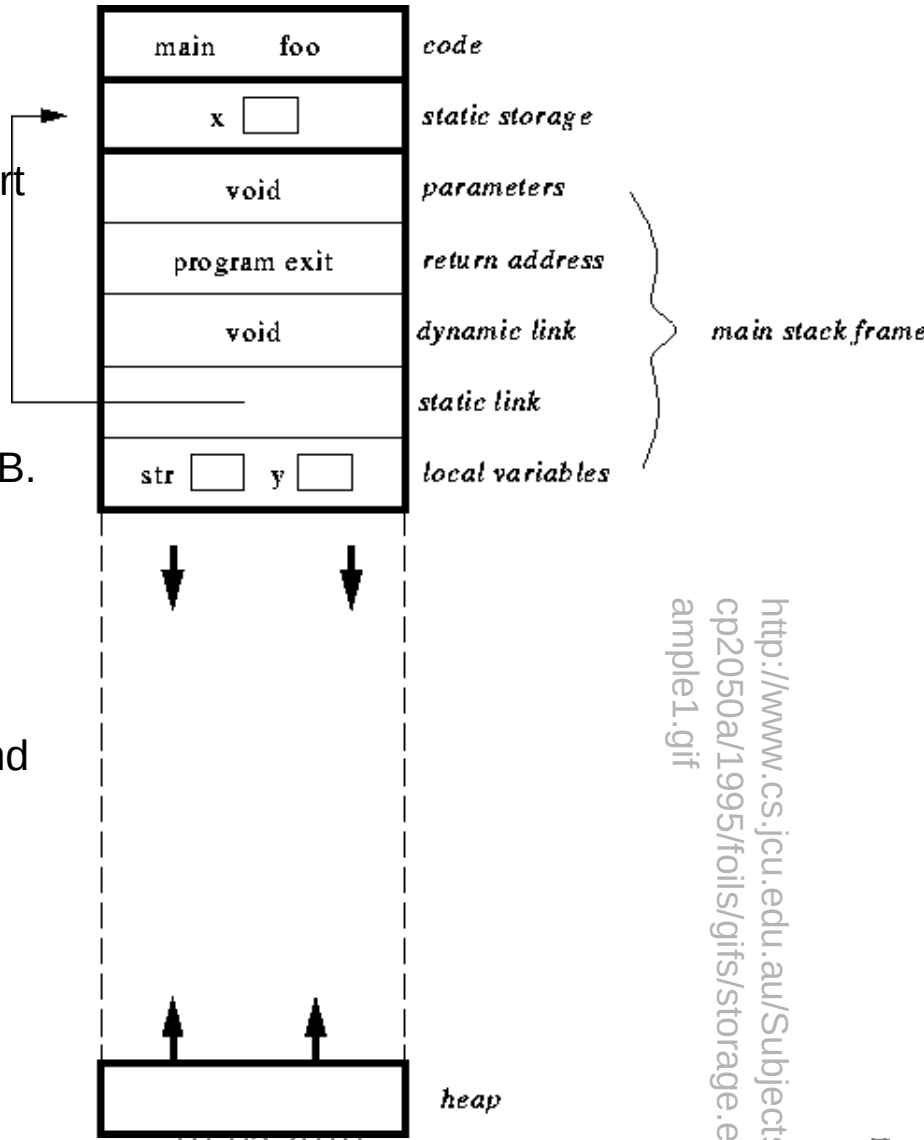
Stack vs. Heap

Stack/ Keller:

- enthält Variablen, die **VOR** der Laufzeit deklariert und initialisiert wurden
- LIFO
- enthält Wertetypen und Referenzen
- Speicher wird freigegeben sobald die dazugehörige Variable ihre Gültigkeit verliert (z.B. lokale Variablen innerhalb einer Methode)
- klein, schnell

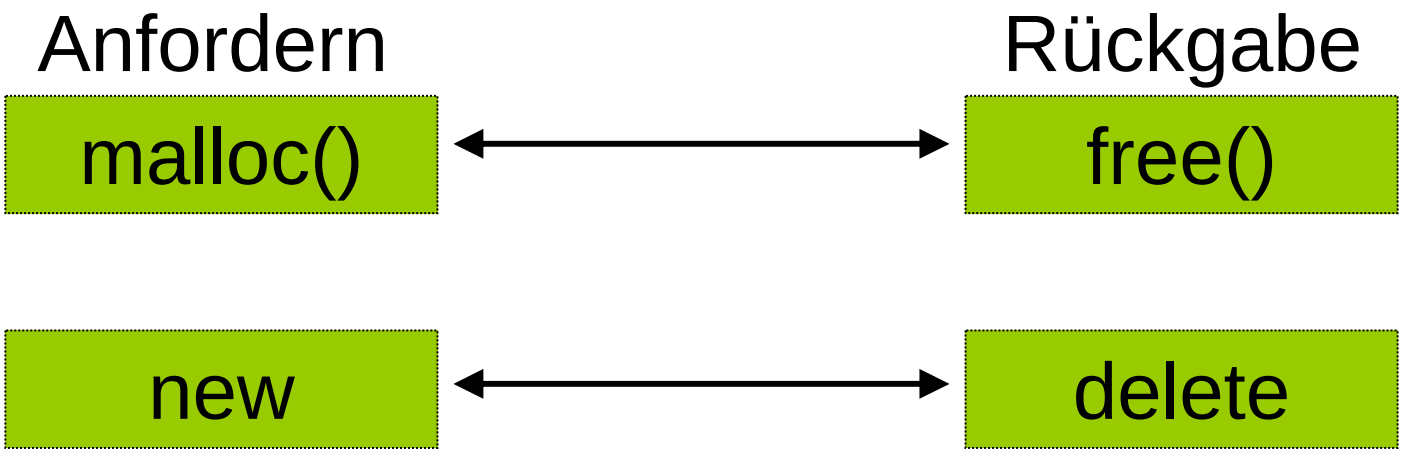
Heap/ Halde:

- enthält Variablen, die **ZUR** Laufzeit deklariert und initialisiert wurden
- dynamische Speicherzuteilung
- Objekte (Instanzen von Klassen)
- Speicher wird erst freigegeben, wenn `delete` aufgerufen wird
- groß



http://www.cs.jcu.edu.au/Subjects/cp2050a/1995/foils/gifs/storage.example1.gif

Speicherverwaltung auf dem Heap in C++



- Funktionspaare nicht mischen
- mögliche Fehler:
  - Speicherleck (keine Destruktion, zu wenig Heap-Speicher zurück)
  - unvorhersagbares Programmverhalten
- Konstruktoren füllen angeforderten Speicher mit Startwerten
- Destruktoren räumen Daten des Objekts wieder auf

malloc() und free()

Speicherzuweisung

Arrays (Felder):

- **Bisher:** Anlegen eines Arrays: `int werte[10];`
- **Aber:** Anlegen nicht ohne Weiteres dynamisch möglich!

Dynamische Speicherzuweisung:

- Zuweisung: `void* malloc(size_t size);`
  - Casten des Zeigers ist erlaubt, aber nicht nötig und zudem „Bad practice“!
- Speicherfreigabe: `void free(void* ptr);`
- `free` darf nur **einmal** auf jedem zuvor duch `malloc` (auch `calloc` oder `realloc`) **initialisierten** Zeiger aufgerufen werden!

Reserviert einen Speicherbereich mit mind. s Byte Größe und gibt Zeiger auf Beginn des Bereiches zurück. Der Inhalt des Bereiches ist nicht definiert.

Beispiel: Dynamische Zuweisung

```
// Zuweisung für ein Feld (Typ int) mit 10 Elementen
int *werte = malloc(10 * sizeof(int));
// Überprüfen ob der Speicher erfolgreich zugewiesen wurde
if (werte == NULL) {
   // Speicher konnte nicht zugewiesen werden
   // daher: Fehlerbehandlung!
   // free darf hier nicht aufgerufen werden
} else {
   // Speicher erfolgreich zugewiesen
   // Code zum bearbeiten des Zeigers
   …
   // Bearbeitung des Feldes erledigt
   // -> Freigabe des Speichers
   free(werte);
   werte = NULL;
   // Nun kann das Feld nicht mehr verwendet werden!
}
```

new & delete

Beispiel für new & delete

```
int main() {
  int * p = new int(3);
  int * q = p;
  delete q; // the same as delete p
  return 0; }
```

Unterschied zwischen malloc()/free() und new/delete

malloc()/ free()

- kümmern sich nicht um den Inhalt der verwalteten Speicherbereiche
- kein Objektkonzept bekannt (keine Werteinitialisierung)

new/ delete

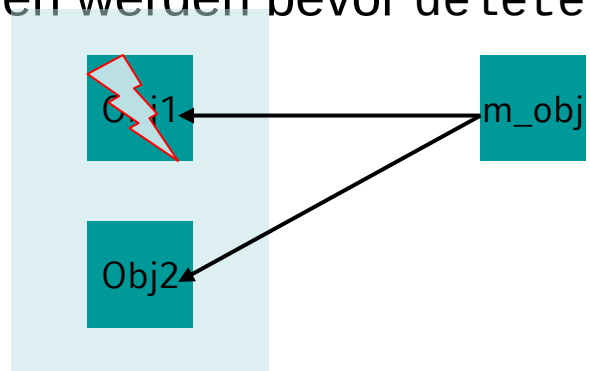
- Speicherbereiche werden als Objekte beliebiger Klassen aufgefasst
- Objekte werden mit Konstruktorwerten initialisiert
- Destruktoraufruf vor delete
- new und delete können vom Nutzer überschrieben werden (zur Vorgabe einer Zieladresse, Einbau von Debugging-Ausgaben)

http://www.forum.elektronikpraxis.vogel.de/showthread.php?p=761

http://www.c-plusplus.de/forum/viewtopic-var-t-is-159231.html

Speicherverwaltung - Pitfalls

- `new/delete` ist die Fehlerquelle Nummer 1 für C++ Anfänger – warum?
  - Heap-Overflow weil sehr viele Objekte angelegt und nicht mehr gelöscht wurden.
  - Aufruf von `delete` auf Objekten, die ihre Gültigkeit verloren haben.
- Zu jedem `new` gehört ein `delete`!
- Vorsicht mit Pointern
  - Pointer sollten immer initialisiert werden sonst ist `delete` ungültig.
  - Wenn für ein Objekt nur ein Pointer existiert darf dieser nicht umgebogen werden bevor `delete` aufgerufen wurde (Memory Leak)



Debugging

http://www.stevenbrown.ca/blog/files/2007/11/bug.png

Möglichkeiten in C++

1. **cout** Ausgabe aktueller Variablenwerte etc.
2. **Logging** z.B. std::ofstream logfile („filename“);
3. **assert(this)** stoppt Programmfluss, falls this == 0
4. **Exceptions** try/catch, erlaubt stabilen Zustand oder Ende
5. **return(value)** siehe main

http://www.gamedev.net/reference/articles/article1344.asp

ODER

Breakpoints setzen und aktuelle Belegung lokaler Variablen auslesen…

Qt

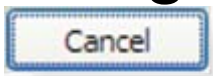
Qt

- Entwickelt von Qt Software (Nokia), ehemals Trolltech
- Prominente Anwendungen: KDE, Opera, Google Earth, Skype
- Eigentlich ein zusätzlicher Präprozessor, der C++-Code erzeugt
- Kostenlos und Open Source
- Verfügbar für X11, OSX, Windows, Embedded (PDA, Smartphone)
- SDK stellt Entwicklungsumgebung (QtCreator) und diverse Tools zur Verfügung

http://www.epyx-online.de/wp-content/2009/10/qt-logo.jpg

(Quelle: en.wikipedia.org/wiki/Qt_(toolkit)

Qt Widgets



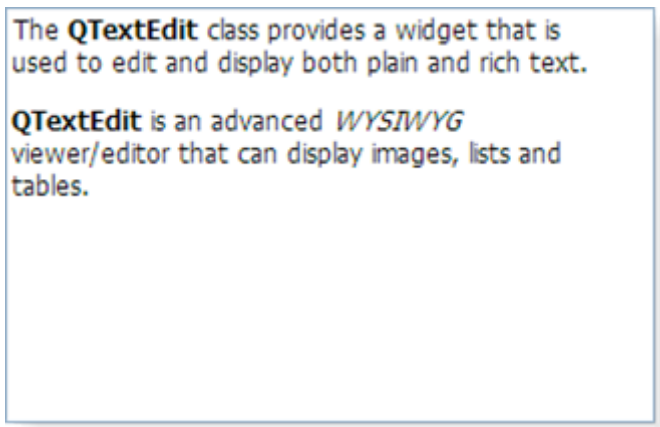
QPushButton

QSlider

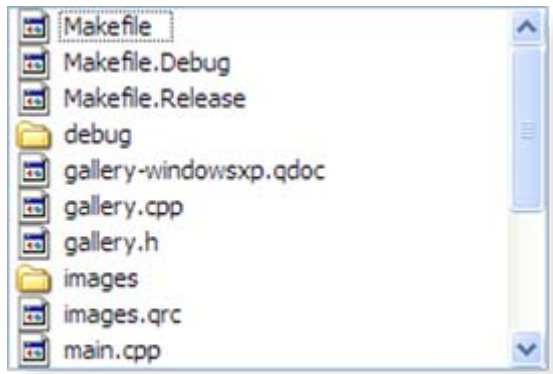
QFrame

QCheckBox

QLineEdit



QTextEdit

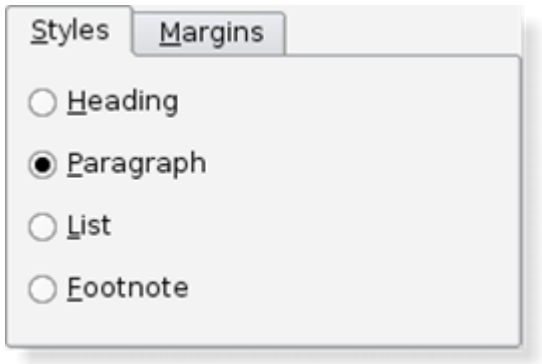


QListView

(Quelle: http://doc.trolltech.com/4.5/gallery-windowsxp.html)

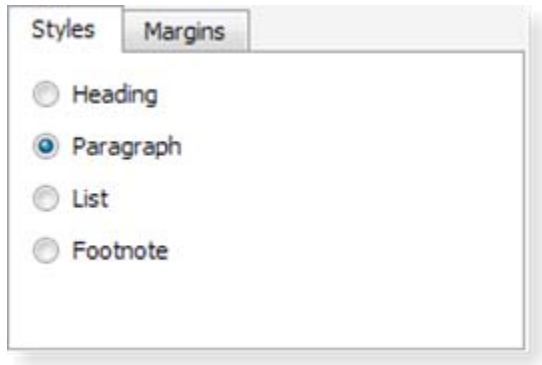
Qt Styles

- Qt bietet für die verfügbaren Widgets verschiedene Styles, die ein globales Look & Feel definieren
- Eigene Styles lassen sich durch Erben von QStyle erzeugen (Beispiel siehe http://doc.trolltech.com/4.3/widgets-styles.html)
- Styles werden durch Aufruf von QApplication::setStyle(Style s) gesetzt



QPlastiqueStyle

QMacStyle

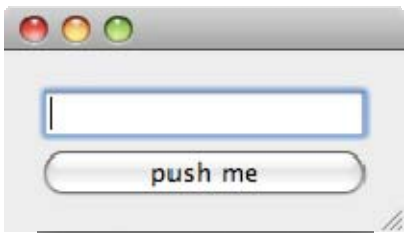


QWindowsVistaStyle

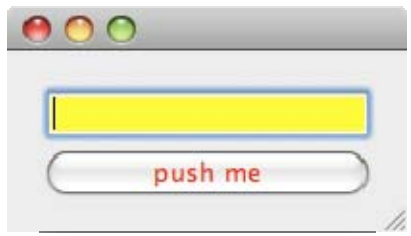
(Quelle: http://doc.trolltech.com/4.5/gallery.html)

Qt Style Sheets

- Style Sheets sind eine an CSS angelehnte Möglichkeit seine Applikation zu skinnen und damit weniger aufwändig als Styles

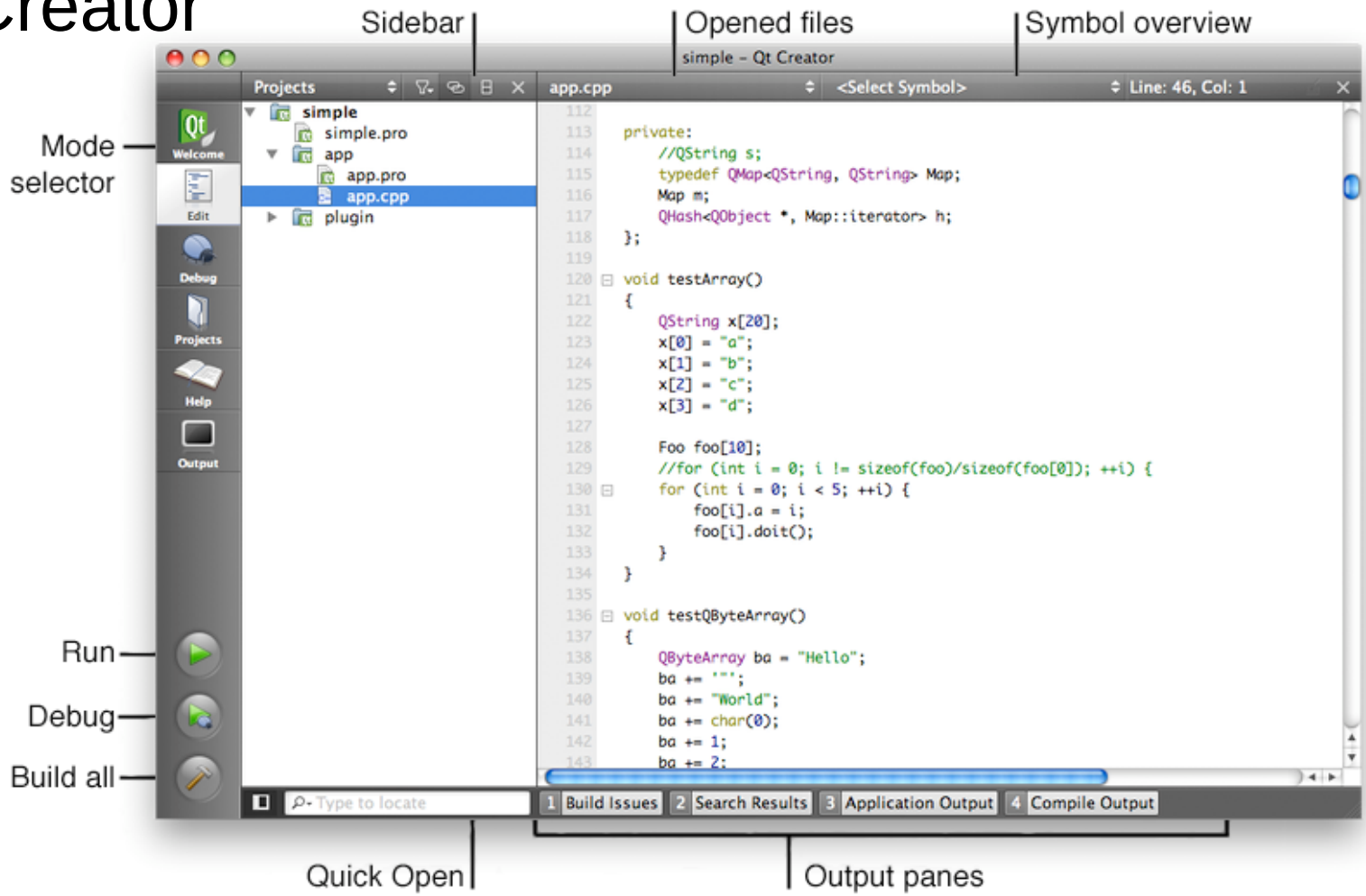



```
app.setStyleSheet(
        "QLineEdit { background: yellow }"
        "QPushButton { color: red }"
);
```



(Quelle: http://doc.trolltech.com/4.5/stylesheet.html)

Qt Creator



(Quelle: http://doc.trolltech.com/qtcreator-1.1/creator-quick-tour.html)

Hello Qt

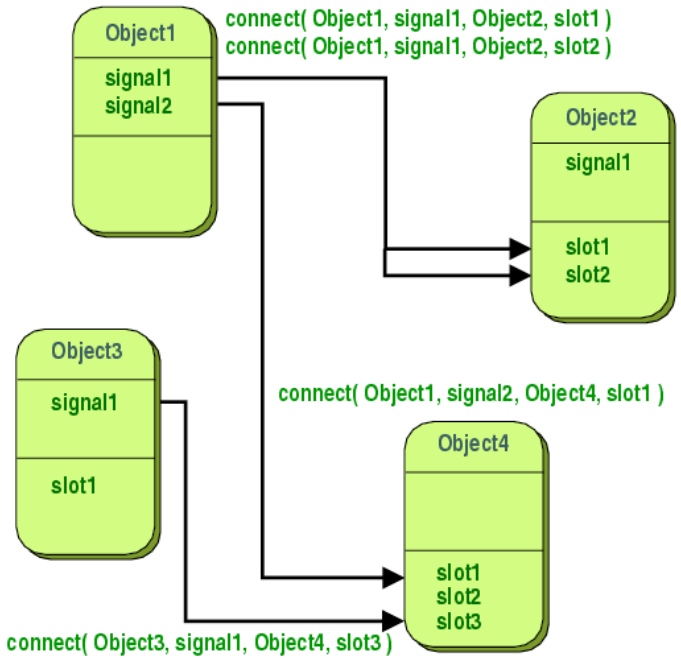
helloQt.cpp

```
#include <QApplication>
#include <QPushButton>

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

    QPushButton hello("Hello world!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

(Quelle: http://doc.trolltech.com/4.3/tutorial-t1.html)

Qt Signals and Slots

- In anderen grafischen Frameworks werden Callbacks benutzt um Kommunikation zwischen Komponenten zu ermöglichen.
- Qt bietet dazu den Signals and Slots Mechanismus an:



(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Signals and Slots

```h
#ifndef COUNTER_H
#define COUNTER_H

#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};

#endif // COUNTER_H
```

counter.h

```cpp
#include "counter.h"

void Counter::setValue(int value)
 {
     if (value != m_value) {
         m_value = value;
         emit valueChanged(value);
     }
 }
```

counter.cpp

(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Signals and Slots

- Signals und Slots werden über QObject::connect verbunden
- Nicht aufgefangene Signale werden ignoriert
- Verbindungen können über QObject::disconnect wieder gelöst werden

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);
```

main.cpp

(Quelle: http://doc.trolltech.com/4.5/signalsandslots.html)

Qt Meta-Object Compiler

- Qt bietet diverse Erweiterungen zu C++ an, aber arbeitet mit regulären C++-Compilern wie gcc
- Dazu wird der Programmcode zuerst durch den Qt's *moc* (Meta-Object Compiler) in regulären C++-Code umgewandelt
- Schlüsselwörter für moc sind:
  - `Q_OBJECT:`
    - Beginn eines Qt-Objekts, um z.B. `signal, slot, emit` zu verarbeiten
  - `Q_PROPERTY:`
    - Zur Definition von properties (siehe http://doc.trolltech.com/4.5/properties.html)
  - `Q_CLASSINFO:`
    - Für Metadaten

(Quelle: http://doc.trolltech.com/4.5/moc.html)

Qt Beispiel #2

main.cpp

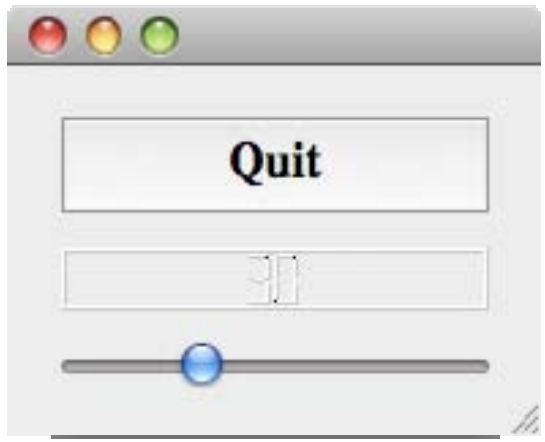
```
#include <QtGUI>

class MyWidget : public QWidget
{
public:
    MyWidget(QWidget *parent = 0);
};

MyWidget::MyWidget(QWidget *parent)
    : QWidget(parent)
{
    QPushButton *quit = new QPushButton(tr("Quit"));
    quit->setFont(QFont("Times", 18, QFont::Bold));

    QLCDNumber *lcd = new QLCDNumber(2);
    lcd->setSegmentStyle(QLCDNumber::Filled);

    QSlider *slider = new QSlider(Qt::Horizontal);
    slider->setRange(0, 99);
    slider->setValue(0);

    connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));
```

(Quelle: http://doc.trolltech.com/4.3/tutorial-t5.html)

Qt Beispiel #2

main.cpp (*ctd.*)

```
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(quit);
    layout->addWidget(lcd);
    layout->addWidget(slider);
    setLayout(layout);
}

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MyWidget widget;
    widget.show();
    return app.exec();
}
```



(Quelle: http://doc.trolltech.com/4.3/tutorial-t5.html)

Literatur

Weiterführende Literatur (Qt)

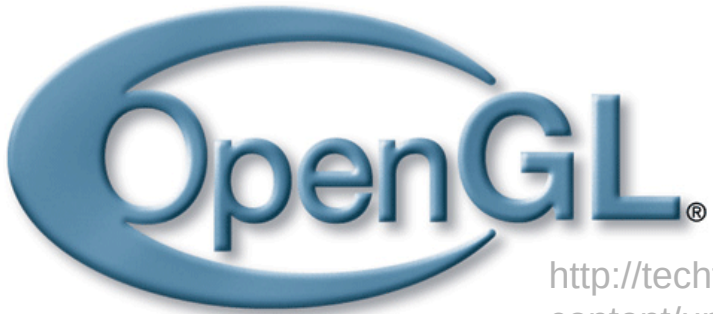
- Jasmin Blanchette, Mark Summerfield: “C++ GUI Programming with Qt 4”, ISBN-13: 978-0132354165
  Erste Edition kostenlos online: http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip

- http://doc.trolltech.com/4.5/
- http://www.qtsoftware.com/products/

Vielen Dank!

Nächstes Mal:

OpenGL



http://techtoggle.com/wp-content/uploads/2009/01/opengl-logo.gif