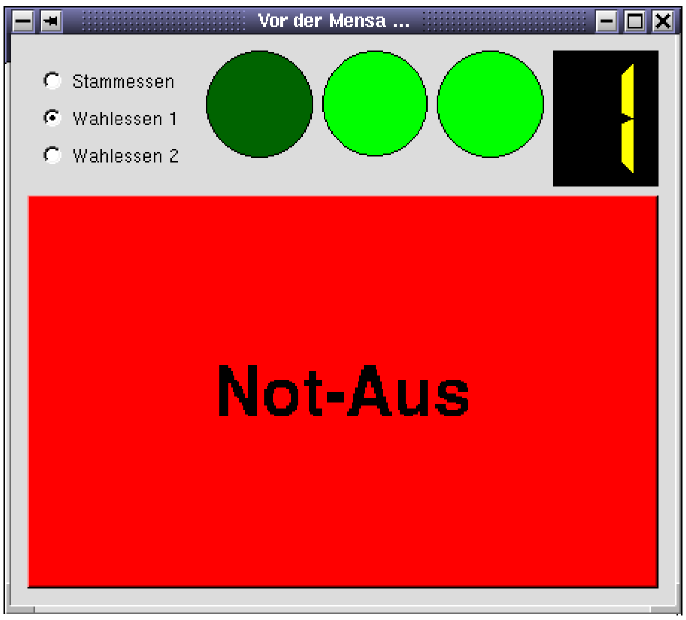
Objektorientierte Programmierung mit C++ (WS 2001/2002)

Abgabetermin: 31. Januar 2002

Beispiellösung

12 „Volkshochschule“ :-) (3+3 Punkte)

Jetzt wird’s mal richtig schön praktisch. Bauen Sie mit dem Qt-Designer nach folgendem Bild einen Qt-Dialog mit Funktionalität und OHNE Programmieren nach:



Was soll passieren: Die Anordnung soll genauso wie im Bild sein. Das Widget soll sich vergrößern lassen und dabei soll nur der „Not-Aus“-Button vertikal wachsen. Die anderen wachsen nur horizontal. Bei den Radiobuttons soll maximal einer gleichzeitig aktiv sein. Ist der oberste aktiv, sollen alle 3 LEDs leuchten, beim mittleren nur die rechten beiden LEDs und beim untersten Radiobutton nur die rechte LED. (Außer nach Drücken von „Not-Aus“.) In der LCD-Anzeige soll die Nummer des aktiven Radiobuttons angezeigt werden (von 0 bis 2). Außerdem sollen beim Drücken von „Not-Aus“ alle LEDs ausgehen. (Sie bekommen das Binary *main* zur Veranschaulichung der gewünschten Funktionalität.)

So, das war's aber noch nicht ganz! ;-) Jetzt wird's nämlich erst spannend. Eine LED ist ein primitives von mir selbst geschriebenes QWidget, das Sie im Quellcode bekommen (*led.h* und *led.cc*). Das LED-Widget sollen Sie zunächst in den Designer einbinden und dann für Ihr Widget – wie beschrieben – verwenden. Zum Lösen dieser Aufgabe dürfen Sie keine Zeile Quellcode selbst schreiben. (Die Hauptdatei *main.cc* ist nämlich ebenfalls gegeben.) Lediglich das Makefile müssen Sie in Anlehnung an das aus der Vorlesung noch entwerfen.

Lösung:

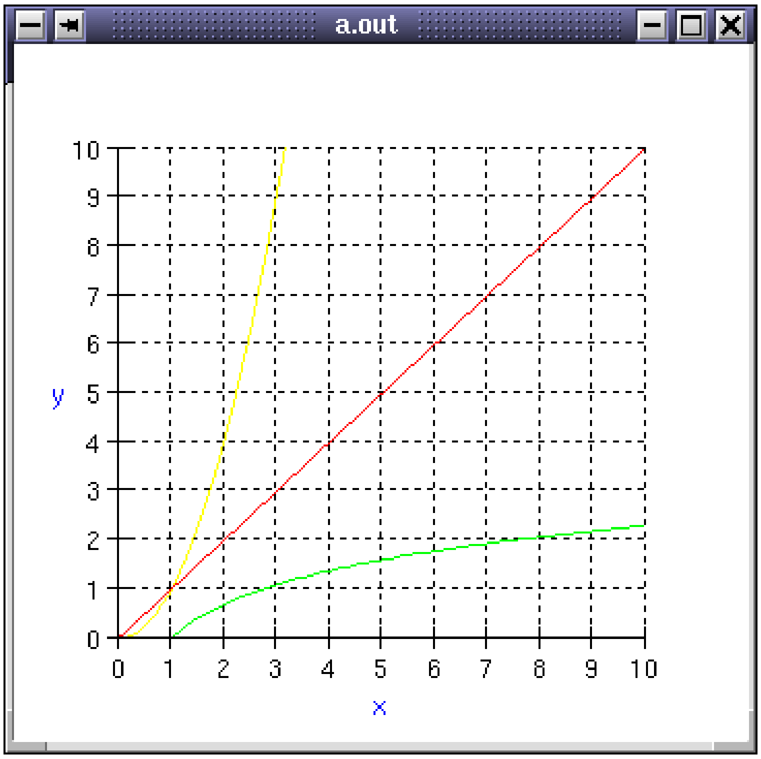
Die Datei *edit.ui* wurde mit dem Designer erstellt und kann via ftp heruntergeladen werden.

makefile

```
QTDIR=/usr/local/qt-2.3.1
CFLAGS=-I$(QTDIR)/include
CXX=g++ $(CFLAGS) $(CPPFLAGS)

main:   main.o MyWidget.o moc_MyWidget.o moc_led.o led.o
        g++ -L$(QTDIR)/lib -lqt -o main main.o MyWidget.o moc_MyWidget.o moc_led.o led.o

main.o: MyWidget.h

MyWidget.h:     edit.ui
        $(QTDIR)/bin/uic edit.ui -o MyWidget.h

MyWidget.cc:    MyWidget.h
        $(QTDIR)/bin/uic -impl MyWidget.h edit.ui -o MyWidget.cc

moc_MyWidget.cc:        MyWidget.cc
        $(QTDIR)/bin/moc MyWidget.h -o moc_MyWidget.cc

led.o:  led.h

moc_led.cc:     led.h
        $(QTDIR)/bin/moc led.h -o moc_led.cc
```

13 Funktionsplot (7+7 Punkte)

Nun haben Sie ja schon was für's Berufsleben wieder kennen gelernt – nämlich Qt als Beispiel für eine umfangreiche Klassenbibliothek, die sogar in der Praxis verwendet wird. Wow – und das an der Uni! :-) Und jetzt lernen Sie schon wieder was: In der Praxis schaut man auch darauf, dass Software wiederverwendbar ist. Bei Qt sind die QWidgets die Programme, welche in jeder Qt-Anwendung wieder verwendet werden. Schrieben Sie nun auch ein wiederverwendbares Programm, einen Funktionsplotter. Dieser soll ein QWidget sein und etwa folgendes Aussehen besitzen:



Natürlich soll die Größe des Funktionsplots von der Größe des Widgets abhängen. Wie jedes QWidget soll auch ihr Widget einige Anpassungsmöglichkeiten (durch Methodenaufrufe) besitzen. So soll der Definitions- und Wertebereich einstellbar sein. Außerdem soll man angeben können, ob ein Gitter dargestellt wird, oder nicht. Des weiteren muss es natürlich möglich sein, die darzustellenden Funktionen von aussen einzustellen. (Die Anzahl der darstellbaren Funktionen sei beliebig.) Für jede Funktion, die zur Darstellung angegeben wird, soll es auch möglich sein, eine eigene Farbe zu spezifizieren. Außerdem soll die Darstellung von jeder Koordinatenachse jeweils auch logarithmisch möglich sein (unabhängig voneinander).
Schreiben Sie also eine Klasse `FunctionPlot`, die von `QWidget` abgeleitet ist, und implementieren Sie in dieser Klasse – unter Verwendung von `QPainter` – das Zeichnen von Funktionen wie beschrieben. Schreiben Sie außerdem eine kleine Qt-Anwendung (`main.cc`) zum Test ihres Widgets.

Lösung:

Makefile

```
QTDIR = /usr/local/qt-2.3.1
CFLAGS = -I$(QTDIR)/include -L$(QTDIR)/lib
CXX = g++ $(CFLAGS)
MOC = $(QTDIR)/bin/moc

main:           main.o plot.o moc_plot.o
                $(CXX) -lqt -o main main.o plot.o moc_plot.o

main.o:         main.cc plot.h

plot.o:         plot.cc plot.h
```

```
moc_plot.o:     moc_plot.cc

moc_plot.cc:    plot.h
                $(MOC) plot.h -o moc_plot.cc
```

main.cc

```
#include <qapplication.h>
#include "plot.h"

// zwei Beispielfunktionen zum Plotten
double f(double x) {
    return x * x;
}
double g(double x) {
    return x;
}

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

    FunctionPlot w;
    w.addFunction(log, QPen(Qt::green, 2));     // built-in-Funktion log
    w.addFunction(f, QPen(Qt::yellow));         // eigene Funktion f
    w.addFunction(g);                           // eigene Funktion g mit default-QPen
    w.setLogDomain(true);                       // doppelt ...
    w.setLogRange(true);                        // logarithmische Anzeige
    w.show();

    app.setMainWidget(&w);
    return app.exec();
}
```

plot.h

```
#ifndef __PLOT_H

#include <qapplication.h>
#include <qpainter.h>
#include <qpixmap.h>
#include <qpaintdevice.h>
#include <qwidget.h>
#include <qmap.h>
#include <math.h>

// Typvereinbarung fuer einen Funktionszeiger
typedef double (*Function)(double);

// QWidget zum plotten von Funktionen
class FunctionPlot : public QWidget {
```

```
    Q_OBJECT

public:
    FunctionPlot(QWidget *father=0, const char *name=0, WFlags f=0):
     QWidget(father, name, f), axisColor(Qt::black), gridColor1(QColor(150,150,150)),
     gridColor2(QColor(200,200,200)), labelColor(Qt::blue) {
        setMinimumSize(200, 200);
        setBackgroundColor(Qt::white);
        bx = by = 5;
        xmin = ymin = 0;
        xmax = ymax = 10;
        drawGrid = true;
        logDomain = false;
        logRange = false;
    }
    // weitere Funktion plotten lassen
    void addFunction(Function fun, const QPen &p = QPen(Qt::red)) {
        funs.insert(fun, p);
        update();
    }
    QColor getAxisColor() const {              // Farbe der Koordinatenachsen
        return axisColor;
    }
    void setAxisColor(const QColor &c) {
        axisColor = c;
        update();
    }
    QColor getGridColor1() const {             // Gitterfarbe 1 (grobe Unterteilung)
        return gridColor1;
    }
    void setGridColor1(const QColor &c) {
        gridColor1 = c;
        update();
    }
    QColor getGridColor2() const {             // Gitterfarbe 2 (feine Unterteilung)
        return gridColor2;
    }
    void setGridColor2(const QColor &c) {
        gridColor2 = c;
        update();
    }
    QColor getLabelColor() const {             // Farbe des Achsennamens
        return labelColor;
    }
    void setLabelColor(const QColor &c) {
        labelColor = c;
        update();
    }
```

```
int getXmin() const {                       // Definitionsbereich
    return xmin;
}
int getXmax() const {
    return xmax;
}
void setDomain(int xmin, int xmax) {
    if (xmin < xmax && (xmin != this->xmin || xmax != this->xmax)) {
        this->xmin = xmin;
        this->xmax = xmax;
        update();
    }
}
int getYmin() const {                       // Wertebereich
    return ymin;
}
int getYmax() const {
    return ymax;
}
void setRange(int ymin, int ymax) {
    if (ymin < ymax && (ymin != this->ymin || ymax != this->ymax)) {
        this->ymin = ymin;
        this->ymax = ymax;
        update();
    }
}
bool getLogDomain() {                       // log. Darstellung ...
    return logDomain;
}
bool getLogRange() {
    return logRange;
}
bool getDrawGrid() {                        // Gitterlinien zeichnen
    return drawGrid;
}
int getBorderX() {                          // Randbreite in x-Richtung
    return bx;
}
void setBorderX(int val) {
    if (val >= 0 && val != bx) {
        bx = val;
        update();
    }
}
int getBorderY() {                          // Randbreite in y-Richtung
    return by;
}
void setBorderY(int val) {
    if (val >= 0 && val != by) {
```

```
            by = val;
            update();
        }
    }
public slots:
    void setLogRange(bool logRange) {
        if (logRange != this->logRange) {
            this->logRange = logRange;
            update();
        }
    }
    void setLogDomain(bool logDomain) {
        if (logDomain != this->logDomain) {
            this->logDomain = logDomain;
            update();
        }
    }
    void setDrawGrid(bool drawGrid) {
        if (drawGrid != this->drawGrid) {
            this->drawGrid = drawGrid;
            update();
        }
    }
protected:
    void paintEvent(QPaintEvent*);      // Event erzeugt von QT bei Aenderungen
                                        // am Fenster (Groesse, Minimieren ...)
    void paintFunction(Function fun, QPainter *p, const QPen &p);
    void paintCoordinateSystem(QPainter *paint);
    void paintCoordinates(QPainter *paint);
    void paintAxisLabels(QPainter *paint, int maxw);
    void paintGrid(QPainter *paint);
private:
    QMap<Function,QPen> funs;           // Map mit den Funktionen

    bool logDomain;                     // log. Darstellung der x-Achse
    bool logRange;                      // log. Darstellung der y-Achse

    bool drawGrid;                      // Gitter zeichnen?
    QColor axisColor;                   // Achsenfarbe
    QColor gridColor1, gridColor2;      // Gitterfarben
    QColor labelColor;                  // Farbe der Achsenbezeichnungen

    int bx, by;         // Rand in x- und y-Richtung
    int xmin, xmax;     // Definitionsbereich
    int ymin, ymax;     // Wertebereich

    double dx, dy;      // Anzahl der Pixel pro Einheit (akt. Zeichenvorgang)
    int ox, oy;         // Koordinaten des Ursprungs (akt. Zeichenvorgang)
    int deltax, deltay; // Differenz zwischen Maximum und Minimum
```

```
};

#endif
```

plot.cc

```
#include "plot.h"

// Funktion plotten
void FunctionPlot::paintFunction(Function fun, QPainter *paint, const QPen &functionPen) {
    paint->setPen(functionPen);

    double x0, y0, log10 = log(10.0);

    int y;
    bool lastSet = false;               // moveTo() wurde schon ausgefuehrt
    bool lastInside;                    // letzter Punkt war im Plot
    for (int x = 0; x <= deltax * dx; x++) {
        x0 = (double) x / dx + xmin;

        if (logDomain)                  // evtl. log-Darstellung der x-Achse
            x0 = exp(x0 * log10);       // also: x0 = 10^x0

        y0 = (*fun)(x0);

        if (logRange)                   // evtl. log-Darstellung der y-Achse
            y0 = log(y0) / log10;

        y0 = (y0 - ymin) * dy;

        y = (int) y0;

        if (lastSet && (lastInside || y0 <= deltay * dy))
            paint->lineTo(ox+x, oy-y);
        else if (y0 > y + 1.1)          // problematischer Integer-Ueberlauf ;-) (daher der Aufwand)
            lastSet = false;            // => dann nichts Plotten und keinen Punkt setzen
        else {
            paint->moveTo(ox+x, oy-y);
            lastSet = true;
        }
        lastInside = (y0 <= deltay * dy);
    }
}

// Koordinatenachsen plotten
void FunctionPlot::paintCoordinateSystem(QPainter *paint) {
    paint->setPen(axisColor);

    // Koordinatenachsen zeichnen
    paint->drawLine(ox, oy, (int) (ox + deltax * dx), oy);
```

```
    paint->drawLine(ox, oy, ox, (int) (oy - deltay * dy));

    // Striche auf den Koordinateachsen und Werte an diesen zeichnen
    for (int x = 0; x <= deltax; x++) {
        int x0 = (int) (ox + x * dx);
        paint->drawLine(x0, oy - 5, x0, oy + 5);
    }
    for (int y = 0; y <= deltay; y++) {
        int y0 = (int) (oy - y * dy);
        paint->drawLine(ox - 5, y0, ox + 5, y0);
    }
}

// Zahlen an den Koordinatenachsen plotten
void FunctionPlot::paintCoordinates(QPainter *paint) {
    paint->setPen(axisColor);

    QFontMetrics fm = paint->fontMetrics();
    int h = fm.height();
    int a = fm.ascent();
    int d = fm.descent();
    int w;

    // Werte an den Koordinatenachsen diesen zeichnen
    for (int x = 0; x <= deltax; x++) {
        int x0 = (int) (ox + x * dx);
        w = fm.width(QString::number(xmin + x));
        paint->drawText(x0 - w / 2, oy + (a + 3 + 5), QString::number(xmin + x));
    }

    for (int y = 0; y <= deltay; y++) {
        int y0 = (int) (oy - y * dy);
        w = fm.width(QString::number(ymin + y));
        paint->drawText(ox - (w + 3 + 5), y0 + (a - d) / 2, QString::number(ymin + y));
    }
}

// Beschriftungen der Koordinatenachsen plotten
void FunctionPlot::paintAxisLabels(QPainter *paint, int maxw) {
    QFontMetrics fm = paint->fontMetrics();
    int h = fm.height();
    int a = fm.ascent();
    int d = fm.descent();
    int w;

    paint->setPen(labelColor);

    QString xAchse("x");
    if (logDomain)
```

```
        xAchse = "log10(x)";

    w = fm.width(xAchse);
    paint->drawText(ox+(int)((double)deltax/2.0*dx)-w/2, oy-(-a-3-h-3-5), xAchse);

    QString yAchse("y");
    if (logRange)
        yAchse = "log10(y)";

    w = fm.width(yAchse);
    QPixmap pix(h, w);
    pix.fill(backgroundColor());
    QPainter p(&pix);
    p.setPen(labelColor);
    p.rotate(-90);
    p.drawText(-w, a, yAchse);
    p.rotate(90);
    p.end();
    paint->drawPixmap(ox-h-3-maxw-3-5, (int) (oy-((double)deltay/2.0*dy+w/2)), pix);
}

// Gitterlinien plotten
void FunctionPlot::paintGrid(QPainter *paint) {
    /* 1.) Gitterlinien bei den Abschnitten der Koordinaten-
     *     achsen zeichenen
     */

    paint->setPen(gridColor1);

    for (int x = 0; x <= deltax; x++)
        paint->drawLine(ox+x*dx, oy, ox+x*dx, oy-deltay*dy);

    for (int y = 0; y <= deltay; y++)
        paint->drawLine(ox, oy-y*dy, ox+deltax*dx, oy-y*dy);

    /* 2.) aequidistante Gitterlinien zwischen den Linien
     *     aus 1.) zeichnen
     */

    paint->setPen(gridColor2);

    if (!logDomain) {
        // Anzahl der Linien abhaengig von der Groesse ausrechnen
        int n = 1;
        while (n < 10 && 1.0/(double)(n+1) * dx >= 5.0)
            n++;
        // und Linien zeichnen
        for (int x = 0; x < deltax; x++)
            for (int i = 1; i < n; i++)
```

```
                paint->drawLine(ox+(x+(double)i/n)*dx, oy-deltay*dy, ox+(x+(double)i/n)*dx, oy);
    }
    else {
        // Anzahl der Linien abhaengig von der Groesse ausrechnen
        int n = 1;
        while (n < 9 && (1.0 - log(1+(double)n/(n+1)*9) / log(10)) * dx >= 3.0)
            n++;
        // und Linien zeichnen
        for (int x = 0; x < deltax; x++)
            for (int i = 1; i < n; i++) {
                double v = log(1+(double)i/n*9) / log(10);
                paint->drawLine(ox+(x+v)*dx, oy-deltay*dy, ox+(x+v)*dx, oy);
            }
    }

    if (!logRange) {
        // Anzahl der Linien abhaengig von der Groesse ausrechnen
        int n = 1;
        while (n < 10 && 1.0/(double)(n+1) * dy >= 5.0)
            n++;
        // und Linien zeichnen
        for (int y = 0; y < deltay; y++)
            for (int i = 1; i < n; i++)
                paint->drawLine(ox, oy-(y+(double)i/n)*dy, ox+deltax*dx, oy-(y+(double)i/n)*dy);
    }
    else {
        // Anzahl der Linien abhaengig von der Groesse ausrechnen
        int n = 1;
        while (n < 9 && (1.0 - log(1+(double)n/(n+1)*9) / log(10)) * dy >= 3.0)
            n++;
        // und Linien zeichnen
        for (int y = 0; y < deltay; y++)
            for (int i = 1; i < n; i++) {
                double v = log(1+(double)i/n*9) / log(10);
                paint->drawLine(ox, oy-(y+v)*dy, ox+deltax*dx, oy-(y+v)*dy);
            }
    }
}

void FunctionPlot::paintEvent(QPaintEvent *e) {
    QPainter paint(this);

    // Anzahl der x-/y-Achsenabschnitte berechnen
    deltax = xmax - xmin;
    deltay = ymax - ymin;

    // Ausmasse des aktuellen Fonts holen
    QFontMetrics fm = paint.fontMetrics();
    int h = fm.height();
```

```
    int a = fm.ascent();
    int d = fm.descent();

    // maximale Breite einer y-Achsenbeschriftung holen
    int maxw = -1;
    for (int y = 0; y <= deltay; y++) {
        int y0 = (int) (oy - y * dy);
        int w = fm.width(QString::number(ymin + y));
        maxw = (maxw < w ? w : maxw);
    }

    // halbe Breite der letzten x-Achsenbeschriftung
    int maxx = fm.width(QString::number(xmax)) / 2;

    // notwendiger Platz fuer x- und y-Achsenbeschriftung
    int ax = 5 + 3 + maxw + 3 + h;
    int ay = 5 + 3 + h + 3 + h;

    // Anzahl der Pixel pro Achsenabschnitt berechnen
    dx = (double) (width() - 2 * bx - ax - maxx) / (double) deltax;
    dy = (double) (height() - 2 * by - ay - h/2) / (double) deltay;

    // Koordinaten des Ursprungs des Koordinatensystems berechnen
    ox = bx + ax;
    oy = height() - by - ay;

    // evtl. Gitter zeichnen
    if (drawGrid)
        paintGrid(&paint);

    // Koordinatensystem zeichenen
    paintCoordinateSystem(&paint);

    // Koordinaten zeichenen
    paintCoordinates(&paint);

    // Achsenbeschriftungen zeichnen
    paintAxisLabels(&paint, maxw);

    // Clipping-Rechteck setzen
    QRect r(ox, (int) (oy - deltay * dy), (int) (deltax * dx), (int) (deltay * dy));
    paint.setClipRect(paint.xForm(r));

    // ... und alle Funktionen zeichnen
    for (QMap<Function,QPen>::Iterator i = funs.begin(); i != funs.end(); ++i)
        paintFunction(i.key(), &paint, i.data());       // i.key() ist Ftk.pointer und i.data() der Pen

    paint.end();
}
```