Informatik III – Ergänzungen

von
Walter Krämer
überarbeitet von
Hans-Jürgen Buhl

Fachbereich Mathematik (7)
Institut für Angewandte Informatik
Bergische Universität – Gesamthochschule Wuppertal

Wuppertal 2000/2001, 2002

Inhaltsverzeichnis

Listings

Kapitel 5

QT

5.1 Graphische Benutzerschnittstelle

Zunächst ein paar wichtige Begriffe:

GUI: graphical user interface (die Schnittstelle des Programms zum Benutzer)

API: application program interface (Programmschnittstelle z.B. zu Betriebssystem, Graphiksubsystem, Datenbank, ...)

Widget: graphische Bedien- und Steuerelemente (z.B. Buttons, Rollbalken, Menü, ...)

Widget ist ein Kunstwort, das wahrscheinlich auf einer Zusammensetzung von *windows* und *gadget* (= Apparat, Vorrichtung, technische Spielerei) basiert.
Eine sehr gute Bibliothek zur Realisierung einer GUI stellt *QT* dar. Insbesondere auf Workstations existiert daneben die GUI-Bibliothek *Motif*.
QT ist

- Bibliothek zur Entwicklung graphischer Benutzeroberflächen
- frei verfügbar (nicht unter MS Windows)
- objektorientiert
- erweiterbar
- portabel (X11, MS Windows)
- sehr gut (auch online) dokumentiert (mit einem umfangreichen Tutorial)

- ausgestattet mit einem C++ GUI Toolkit

QT wird von Trolltech (z.B. unter http://www.trolltech.de) seit 1992 entwickelt. Die Bibliothek beinhaltet ca. 250 Klassen und ist derzeit in der Version 3.0.1 (vom 12.12.2001) erhältlich.

5.2 About Qt

Qt is a cross-platform C++ GUI application framework. It provides application developers with all the functionality needed to build state-of-the-art graphical user interfaces. Qt is fully object-oriented, easily extensible, and allows true component programming. Since its commercial introduction in early 1996, Qt has formed the basis of many thousands of successful applications worldwide. Qt is also the basis of the popular KDE Linux desktop environment, a standard component of all major Linux distributions. Qt is supported on the following platforms: MS/Windows - 95, 98, NT, and 2000 Unix/X11 - Linux, Sun Solaris, HP-UX, Digital Unix, IBM AIX, SGI IRIX and a wide range of others Embedded - Linux platforms with framebuffer support.

Qt is a product of Trolltech.

Qt is released in different editions:

Qt Enterprise Edition and Qt Professional Edition provide for commercial software development. They permits traditional commercial software distribution and includes free upgrades and Technical Support Service. For the latest prices, please see the Trolltech web site, Pricing and Availability page, or contact `sales@trolltech.com`. The Enterprise Edition offers extended modules over the Professional Edition.

Qt Free Edition is the Unix/X11 version of Qt available for development of Free and Open Source software only. It is provided free of charge under the terms of both the Q Public License and the GNU General Public License. The latest version is available for download.

Qt/Embedded Free Edition is the Embedded version of Qt available for development of Free software only. It is provided free of charge under the terms of the GNU General Public License.

5.3 Overview of Qt

You need Qt if you develop GUI software for the X Window System and/or Microsoft Windows. Qt brings you up to speed and helps you to create efficient software with a modern user interface.

The Qt Professional Edition is currently used worldwide, to produce successful commercial software running on multiple platforms. Our customers know they can rely on the professional technical support offered by Trolltech.

The Qt Free Edition has gained widespread acceptance as the leading C++ GUI toolkit for free software on Unix. As an example, it is the basis of the popular KDE desktop environment on Linux. This means that there are a lot of useful resources on the Internet about Qt programming, such as Qt contribution programs and an active Qt programmers' mailing list.

By using Qt, you ensure that your software will remain portable to all major operating systems. Qt is currently supported on Microsoft Windows 95/98, Microsoft Windows NT, Linux, Solaris, SunOS, HP-UX, Digital UNIX (OSF/1, Tru64), Irix, FreeBSD, BSD/OS, SCO, AIX and others. Qt is highly portable on Unix variants; it can run on IBM OS390 R2.5, and there are Professional Edition users running in on e.g. QNX.

Future-Safe

Investing in Qt is safe. In the unlikely event that Trolltech should become unable to maintain Qt, the last version is legally guaranteed to be released to the free software community, which then will be able to provide continuity.

The last few years, several toolkit vendors have discontinued maintenance of their products, leaving their customers in a difficult situation. The KDE Free Qt Foundation guarantees that Qt customers will never suffer in the same way.

Features

Here are some of the most important technical features of Qt:

Object Orientation
Qt has a modular design and a strong focus on reusable software com-

ponents. A widget does not need to know its context and communicates with the outside world through signals and slots. All Qt widgets can be specialized through inheritance.

Component Support

Qt provides a signals/slots concept that is a type-safe alternative to callbacks and at the same time allow objects to cooperate without any knowledge of each other. This makes Qt very suitable for true component programming.

Superior On-Line Documentation

Qt includes on-line reference documentation in heavily cross-referenced HTML, Unix man-page and Postscript formats. The reference documentation is also available on-line. For the beginner, a tutorial explains Qt programming step by step.

Easy to Customize

A common problem with other toolkits is that there will often be no widget which fits the requirements exactly, and creating a custom widget is black magic. For instance, the Motif manual discusses user-defined widgets starting on page 886. That's right, almost 900 pages into the book. That is why we made simple creation of widgets an absolute priority. With Qt it is painless to modify the behavior of existing widgets and create custom widgets.

Portability

Qt is a multi-platform GUI toolkit. It hides the problems of dealing with the differences between the underlying window systems (X Windows MS Windows) from the programmer. To achieve full portability of Qt-based programs, Qt also includes a set of classes which protect the programmer from OS-dependent details in file handling, time/date handling etc.

Internationalization

Qt provides integrated support for making localized applications, where all user interface texts are translated into the local language, based on message translation tables. Qt also has full support for Unicode 16-bit international character set.

Rich API

Qt offers the functionality you need to create professional applications. There are around 250 C++ classes in the Qt API. Most of these classes are GUI specific; however, Qt also provides template-based collections,

serialization, file and a general I/O device, directory management, date/time classes, regular expression parsing and more. Our philosophy is to provide these classes as optional functionality. We use them to implement the Qt internals, you can choose to not use them or to use the STL (Standard Template Library) or other toolkits instead.

Full Widget Set

The basic building block in Qt programming is the widget. Qt contains all the ready-to-use widgets you need to create professional-looking user interfaces for your software - everything from a push button, scroll bar or combo box, right up to a HTML text browser.

High-Performance Implementation

For libraries, efficiency matters much more than for applications. The Qt library is optimized for fast execution and conservative use of memory. Qt is able to perform many general tasks, for instance graphics rendering, much faster than platform-dependent code normally does. The X11 implementation of Qt is based directly on Xlib and does not depend on the infamously slow Motif toolkit.

GUI Emulation

Most GUI toolkits are based on a layered approach, i.e. the toolkit wraps C++ classes around native window system widgets. This architecture makes it difficult to inherit and customize widgets. In layered toolkits the GUI functionality often becomes the least common denominator of all underlying window systems. Qt emulates native window system widgets. This is a much more flexible and open technology. When we have not found the necessary functionality on a platform, we have implemented it as a part of Qt instead. That is how we can offer you a range of powerful functionality, like rotation of text, across all platforms. With Qt you get the best of both worlds.

Customizable Look and Feel

Qt supports „themes“, so Qt-based applications can change (even at runtime) between Motif look and feel, Windows look and feel, and any number of custom-made look and feel themes. This works independently of whether the application happens to be running under X Windows or Microsoft Windows.

Advanced Drawing Operations

Qt's drawing engine, the QPainter class, renders graphics on any paint device. Paint devices include widget, pixmap (off-screen image), picture

(meta-file) and printer (Postscript under Unix/X11). The exact same code is used to draw on all four types of devices. QPainter supports complex coordinate system transforms and can easily draw rotated text and pixmaps (on all platforms, of course).

2D/3D Graphics Rendering
For more advanced 2D/3D graphics, use of OpenGL together with Qt is supported. With the Qt OpenGL Extension, an OpenGL window can be created and used just as any Qt widget. For higher-level, object-oriented and more powerful 3D graphics than pure OpenGL, we recommend TechSoft America's HOOPS library. It is currently being integrated with Qt; see this press release for details.

5.4 Qt Platform Notes

Here are the platforms Qt is currently known to run on, with links to platforms-specific notes, including any known bugs or incompatibilities:

- AIX - 4.1 or later.
- BSDI/OS - 2.0 or later
- DG/UX
- FreeBSD - 2.1 or later
- HP-UX - 10.20 or later
- Irix - 6.x
- Linux
- NetBSD
- OS/2 (with XFree86)
- QNX
- SCO UNIX
- Solaris - 2.5.1 or later

- Tru64 (Digital UNIX) - 4.0 or later
- Windows 95
- Windows 98 and Me
- Windows NT (4.0 or later) and 2000
- XLib on Windows

Here are the compilers we develop or test Qt with, with links to compiler-specific notes, including any known bugs or incompatibilities:

- Borland C++ - Windows 95/98/NT/2000
- Comeau C++ - Linux
- Compaq C++ - Tru64 (Digital UNIX), Linux
- GCC / egcs - most platforms
- Hewlett-Packard aCC - HP-UX
- Hewlett-Packard CC - HP-UX
- KAI C++ - Linux
- MIPSpro - Irix
- Sun WorkShop / Forte Developer - Solaris
- Visual C++ - Windows 95/98/NT/2000
- IBM xlC - AIX

Since Qt works with both KAI C++ and SGI MIPSpro, we expect that it works with most other compilers based on EDG C++ as well.

If you have anything to add to this list or any of the platform- or compiler-specific pages, please write to `qt-bugs@trolltech.com`.

5.5 Dokumentation und erste Beispiele

Die Dokumentation von Qt finden Sie auf unseren LINUX-Rechnern unter

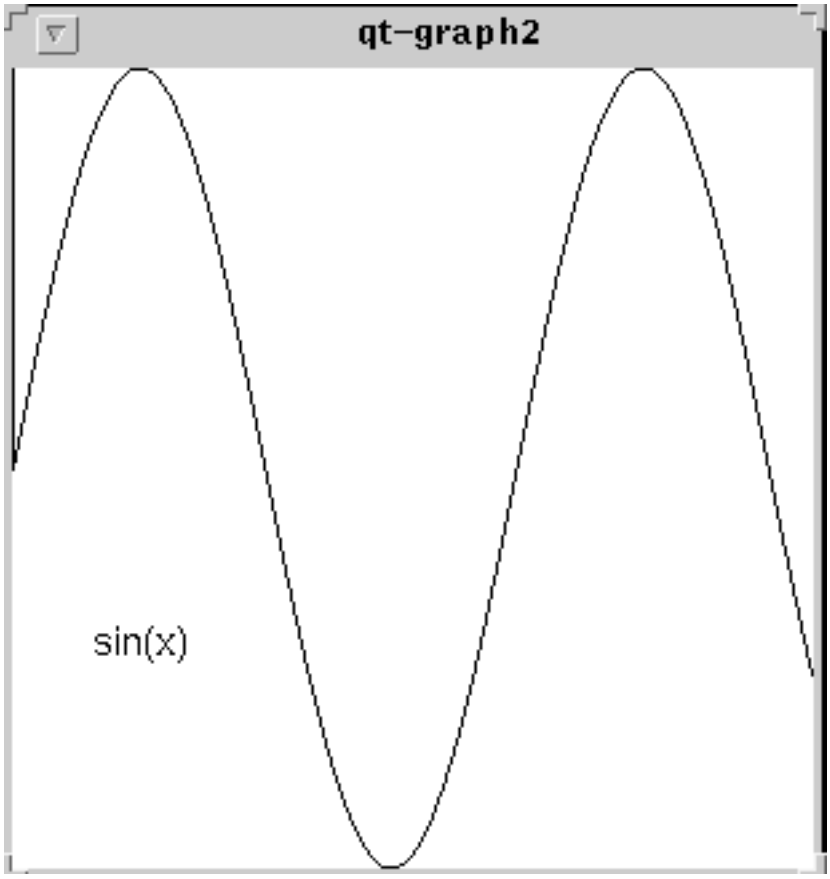
`/usr/share/doc/packages/qt/html/index.html`

beziehungsweise auf unseren Sun-Workstations unter

`/opt/local/Manuals/Libs/qt-2.2.0/index.html`.

Ein einfaches Programm zum Zeichnen des Sinus über zwei Perioden inklusive

Beschriftung sieht folgendermaßen aus:



Listing 5.1: Plot von sin(x)

```
#include <qapplication.h>
#include <qwidget.h>
#include <qpainter.h>
#include <cmath>
using namespace std;

//  g++ qt−graph2.cc −o qt−graph2 −lqt
//  mit Beschriftung

class Bild : public QWidget
{
public:
        void paintEvent( QPaintEvent * )
        {
                int hh = height()/2;
                int b  = width();
                double x, h(10.0/b);

                QPainter p( this );

                for(int i = 0; i < b; i++)
                {
```

```
                        x = i * h;
                        p.lineTo(i, (-sin(x)*hh)+hh);
                }

                QString s("sin(x)");
                p.drawText(30, 220, s);
        };
};



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

        Bild hello;

        hello.resize ( 300, 300 );
        hello.setPalette(QPalette(QColor(255, 255, 255)));

        a.setMainWidget( &hello );
        hello.show();

        return a.exec();
}
```

Eine zentrierte Ausgabe eines Textes in ein Textfenster

kann folgendermaßen realisiert werden:

Listing 5.2: zentrierte Textausgabe

```
#include <qapplication.h>
#include <qwidget.h>
#include <qbrush.h>
#include <qvbox.h>
#include <qhbox.h>
#include <qpushbutton.h>
#include <qtextview.h>

#include <string>

using namespace std;

//
//  g++ qt-graph3.cc -o qt-graph3 -lqt
//
//                  Text-Ausgabe

class TextFenster : public QVBox
{
protected:
```

```
        QTextView* view;
        QPushButton* bClose;

public:
        TextFenster(const QString inhalt,
                    QWidget* parent = 0, const char* name = 0)
       :  QVBox(parent, name){

        setMargin(5);
        view = new QTextView(this);
        QBrush paper;
        paper.setPixmap(QPixmap("marble.png"));
        view->setPaper(paper);
        view->setText(inhalt);
        view->setMinimumSize(450, 250);

        QHBox *buttons = new QHBox(this);
        buttons->setMargin(5);

        bClose = new QPushButton("&Close", buttons),
        connect(bClose, SIGNAL(clicked()), qApp, SLOT(quit()));

        }

};

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

        static QString Text(
                "<center>"
               "  <b><br>Der Trichter</b><br>"
                "<br>"
                "Zwei Trichter wandeln durch die Nacht.<br>"
                "Durch ihres Rumpfs verengten Schacht<br>"
                " fliesst  weisses Mondlicht<br>"
                " still  und heiter<br>"
                "auf ihren<br>"
                "Waldweg<br>"
                "u. s.<br>"
                "w.<br>"
                "</center>"
              );

        TextFenster win(Text , 0, 0);

        win. resize (450, 250);
        win.setCaption("Christian Morgenstern:");

        a.setMainWidget( &win );
        win.show();

        return a.exec();
}
```

5.6 Klassenhierarchie

...

- QRect
- QRegExp
- QRegion
- QScreenCursor
- QShared
  - QGLayoutIterator
- QSimpleRichText
- QSize
- QSizePolicy
- QString
  - QConstString
- Qt
  - QBrush
  - QCanvasItem
    - QCanvasPolygonalItem
      - QCanvasEllipse
      - QCanvasLine
      - QCanvasPolygon
      - QCanvasRectangle
    - QCanvasSprite
    - QCanvasText
  - QCustomMenuItem
  - QEvent
    - QChildEvent
    - QCloseEvent
    - QCustomEvent

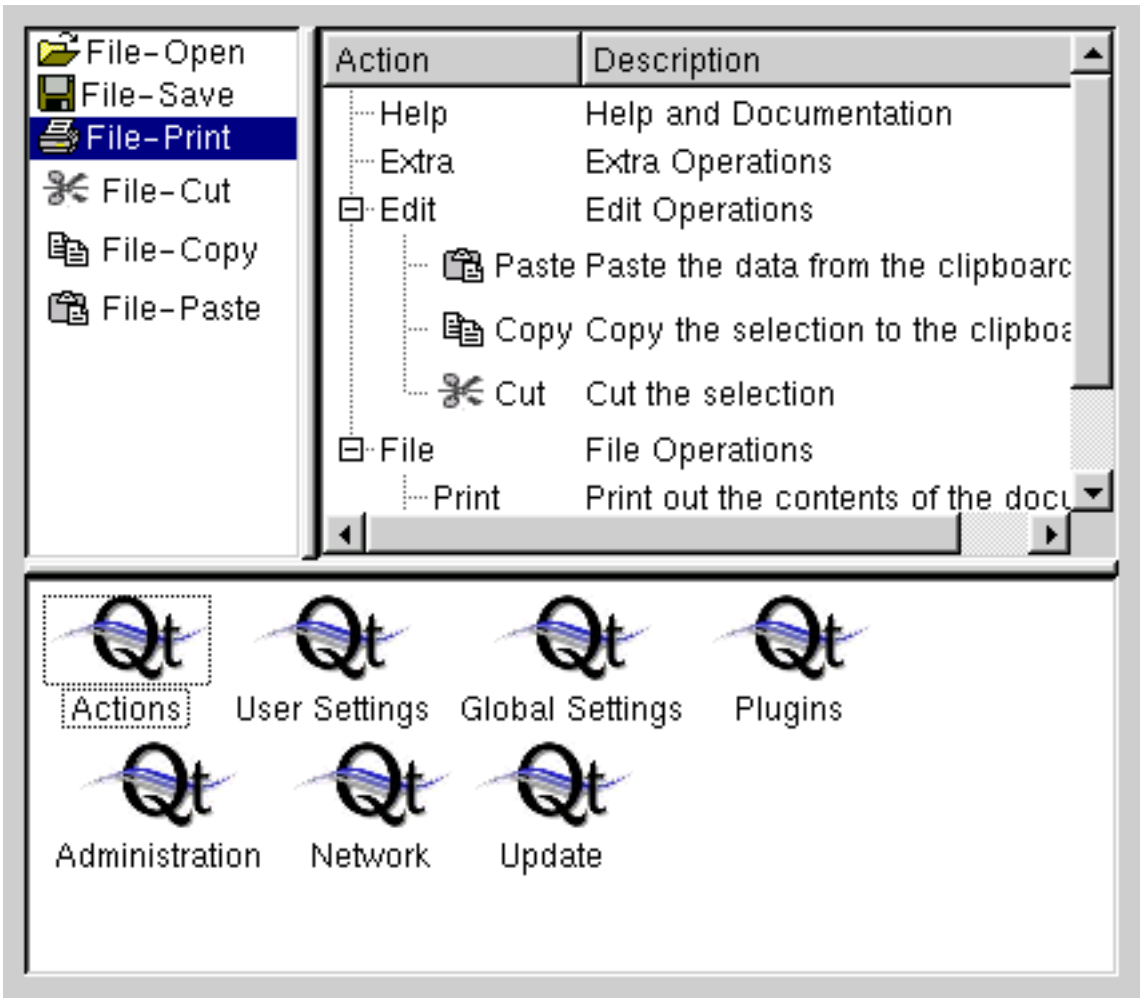
        * QDragLeaveEvent
        * QDropEvent
            · QDragMoveEvent
                1. QDragEnterEvent
        * QFocusEvent
        * QHideEvent
        * QKeyEvent
        * QMouseEvent
        * QMoveEvent
        * QPaintEvent
        * QResizeEvent
        * QShowEvent
        * QTimerEvent
        * QWheelEvent
    - QIconViewItem
    - QListViewItem
        * QCheckListItem
    - QMutex
    - QObject
        * QAccel
        * QAction
            · QActionGroup
        * QApplication
            · QXtApplication

...

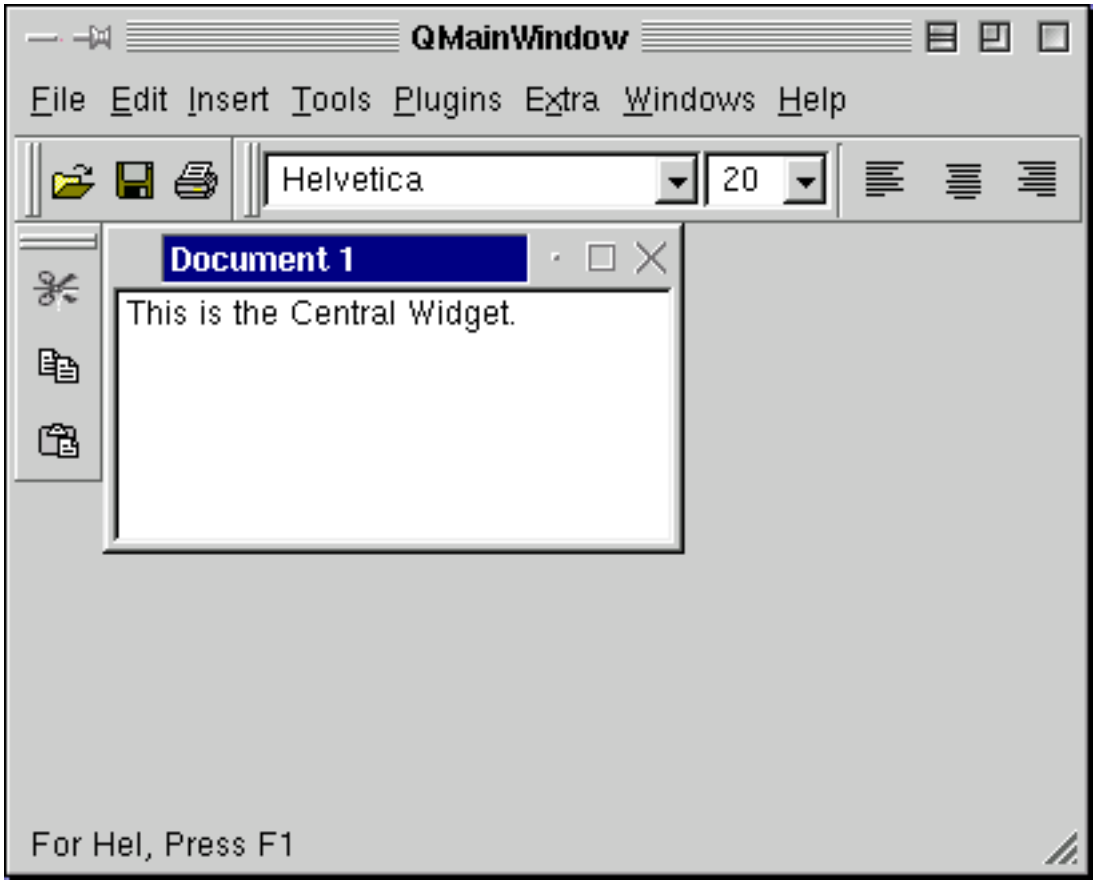
5.7 Pictures of Most Qt Widgets

Most of these widgets are pictured in either Motif or Windows style. All widgets are supported in both styles (plus other styles), but for clarity we've chosen to cut down on the number of repeated images.

Here you see three views seperated by QSplitters. On the lef/top you see a QListBox, at the right/top you see a QListView with a QHeader and two QScrollBars. And at the bottom there is a QIconView.

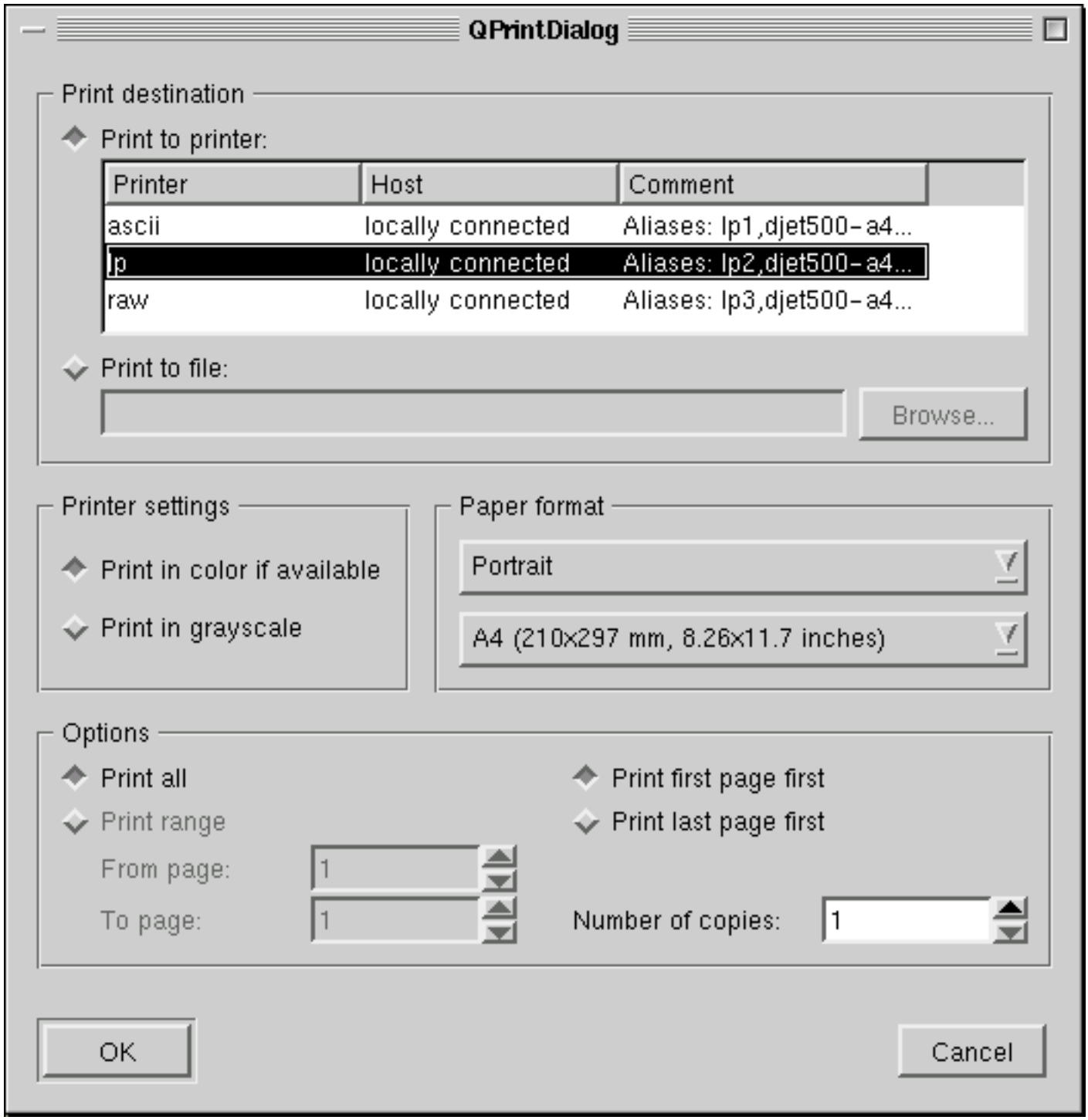


In this screenshot you see a QMainWindow which contains on the top a QMenuBar, below that and at the left some QToolBars with widgets like QToolButtons and QComboBoxes. The central widget is a QWorkspace which is used for MDI window management and which contains as MDI-Window a QMultiLineEdit here. At the bottom you see a QStatusBar and at the right/bottom edge a QSizeGrip.

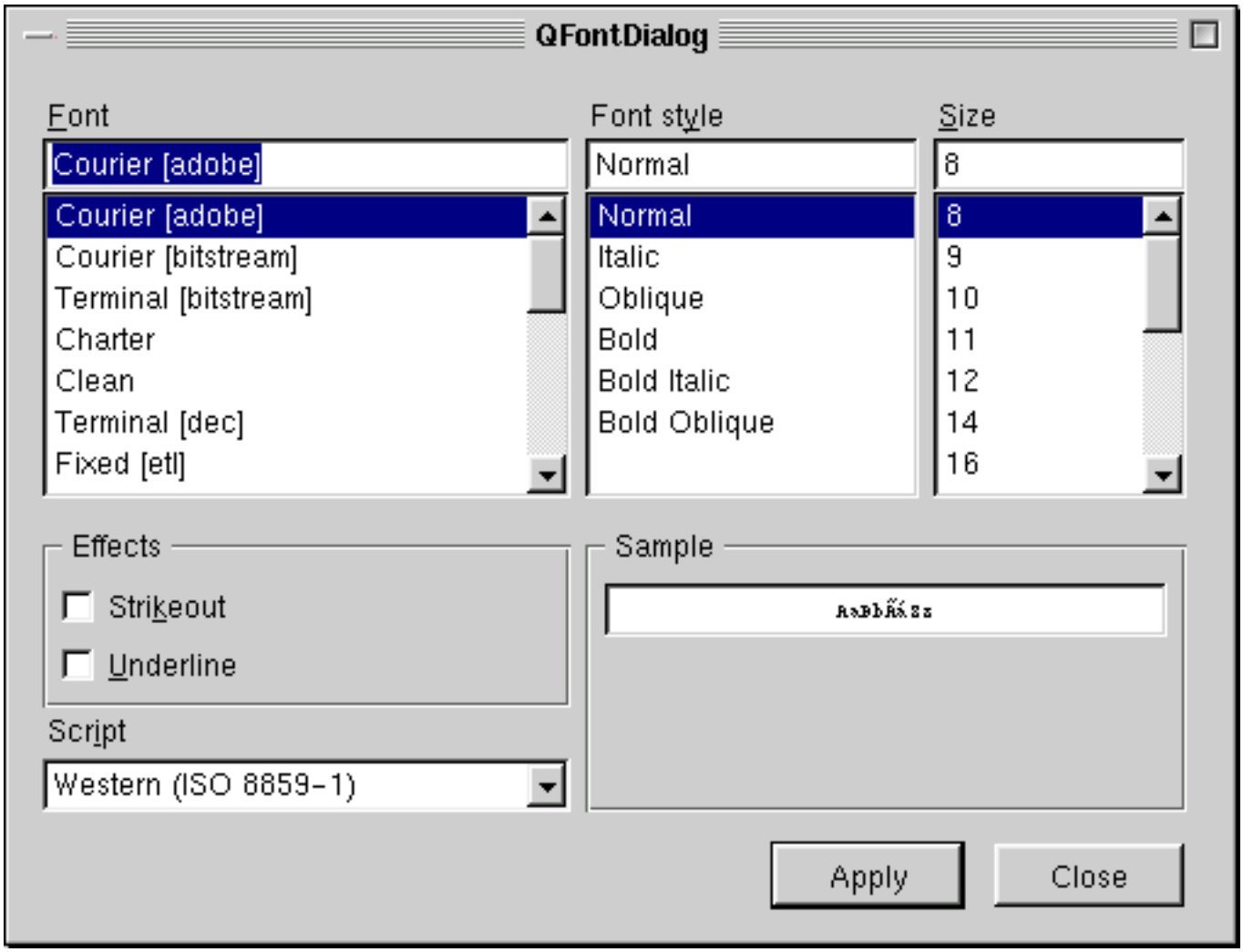


Here you see a QFileDialog. On the Windows Platform you can either use the QFileDialog or the native Windows Filedialog. For more information on that see the QFileDialog class documentation.

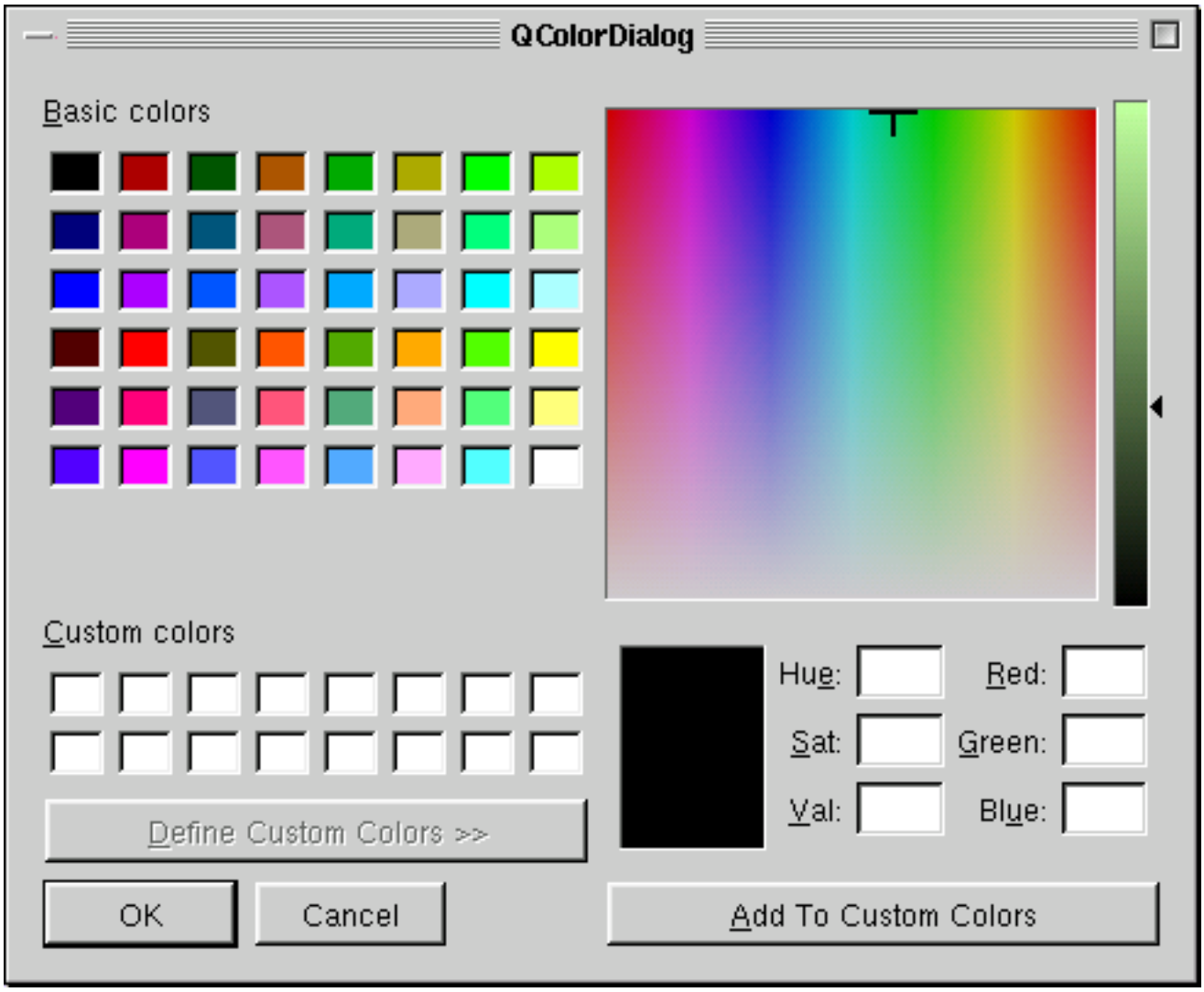
This is a QPrintDialog. On Windows this dialog is not supported, as the native Windows Print-Dialog is used there. Use QPrinter::setup() for portability instead of the QPrintDialog if you need to be platform independent.



In this screenshot you see a QFontDialog.

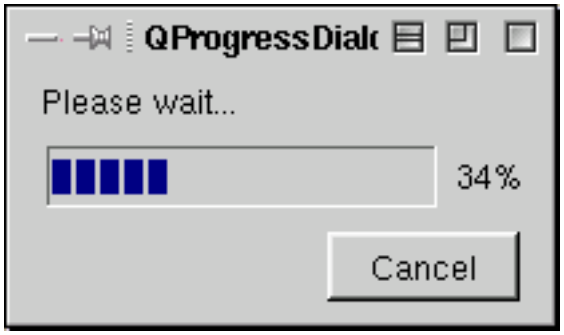


And in this picture a QColorDialog is shown.

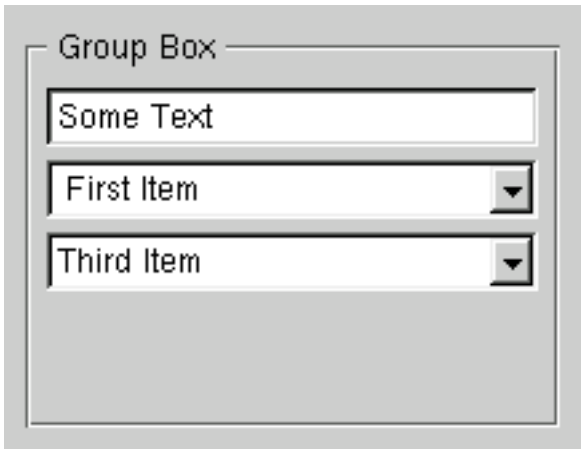


For displaying messages you will use the QMessageBox.

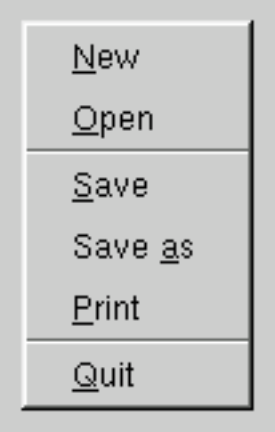
This image shows a QProgressDialog. The QProgressBar can be used as seperate widget too.



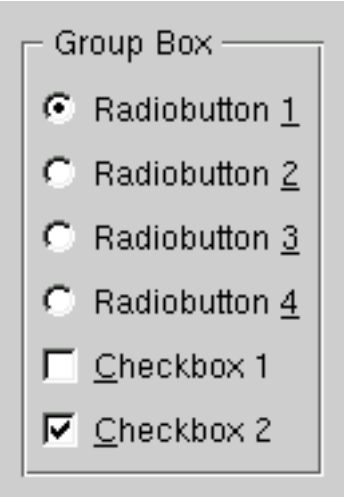
In the screenshot below you see a QGroupBox which contains a QLineEdit, a read-only QComboBox and a editable QComboBox.



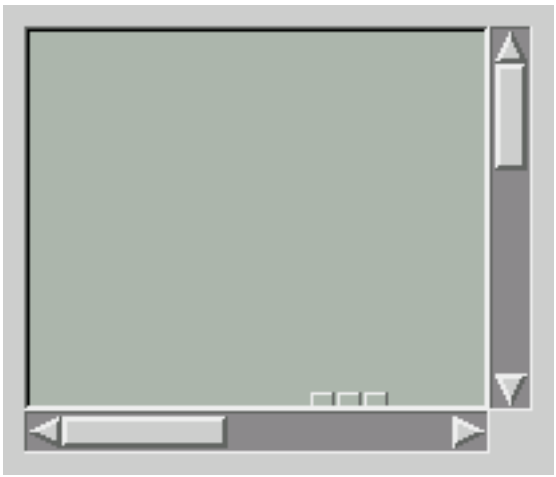
This screenshot shows a QPopupMenu.



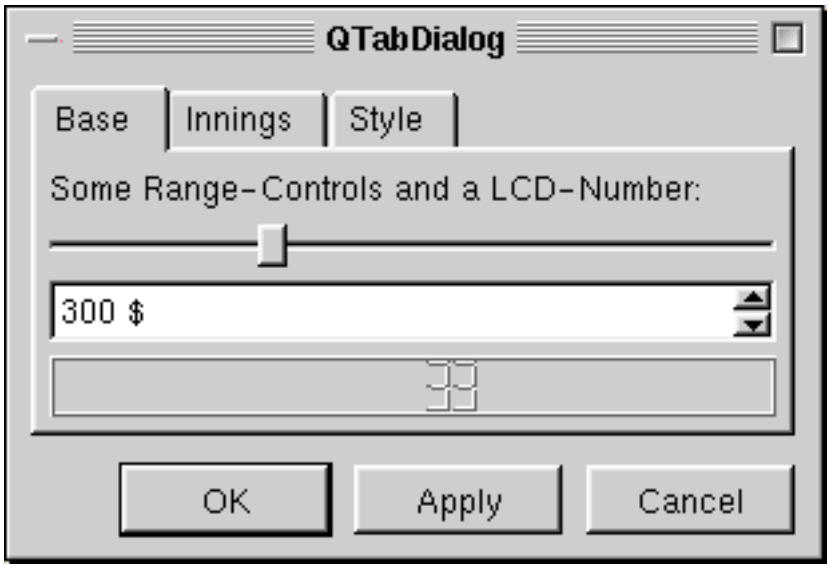
In the screenshot below you see a QButtonGroup containing four QRadio-Buttons and two QCheckBoxes



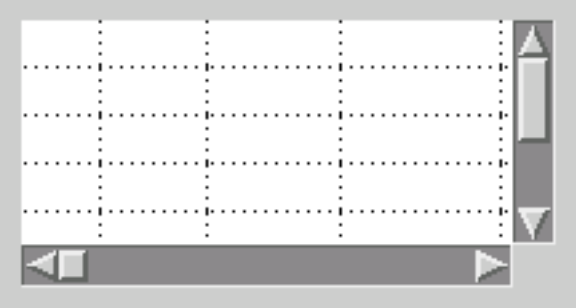
All views in the first screenshot are widgets derived from a QScrollView. But this class can also manage lots of child-widgets, like in this screenshot.



This screenshot shows a QTabDialog. The tabs (QTabBar) or the more convenient class QTabWidget, which combines a tab bar with the pages, can be used separately. In the visible page you see a QLabel, the range controls QSlider and QSpinBox and below a QLCDNumber. In the bottom row you can see some QPushButtons.



The screenshot below shows a QTableView.



In this screenshot you see the QTextBrowser displaying a HTML page. See also QTextView!

5.8 Beispiele aus der Online-Dokumentation

5.8.1 QMetaObject Class Reference

The QMetaObject class contains meta information about Qt objects. More...

#include <qmetaobject.h>

List of all member functions.

Public Members

- QMetaObject (const char * class_name, const char * superclass_name, QMetaData * slotdata, int n_slots, QMetaData * signal_data, int n_signals) (internal)
- QMetaObject (const char * class_name, const char * superclass_name, QMetaData * slot_data, int n_slots, QMetaData * signal_data, int n_signals, QMetaProperty * prop_data, int n_props, QMetaEnum * enum_data, int n_enums, QClassInfo * class_info, int n_info) (internal)
- virtual ~QMetaObject () (internal)
- const char* className () const
- const char* superClassName () const
- QMetaObject* superClass () const
- bool inherits (const char * clname) const
- int numSlots (bool super = FALSE) const
- int numSignals (bool super = FALSE) const
- QMetaData* slot (int index, bool super = FALSE) const (internal)
- QMetaData* signal (int index, bool super = FALSE) const (internal)
- QMetaData* slot (const char *, bool super = FALSE) const (internal)
- QMetaData* signal (const char *, bool super = FALSE) const (internal)
- QStrList slotNames (bool super = FALSE) const

- QStrList signalNames (bool super = FALSE) const
- int numClassInfo (bool super = FALSE) const
- QClassInfo* classInfo (int index, bool super = FALSE) const
- const char* classInfo (const char * name, bool super = FALSE) const
- const QMetaProperty* property (const char * name, bool super = FALSE) const
- QStrList propertyNames (bool super = FALSE) const
- void resolveProperty (QMetaProperty * prop) (internal)
- void set_slot_access (QMetaData::Access *) (internal)
- QMetaData::Access slot_access (int index, bool super = FALSE) (internal)

5.8.2 QSessionManager Class Reference

The QSessionManager class provides access to the session manager. More...

#include <qsessionmanager.h>

Inherits QObject.

List of all member functions.

Public Members

- QString sessionId () const
- bool allowsInteraction ()
- bool allowsErrorInteraction ()
- void release ()
- void cancel ()
- enum RestartHint RestartIfRunning, RestartAnyway, RestartImmediately, RestartNever
- void setRestartHint (RestartHint)

- RestartHint restartHint () const
- void setRestartCommand (const QStringList &)
- QStringList restartCommand () const
- void setDiscardCommand (const QStringList &)
- QStringList discardCommand () const
- void setProperty (const QString & name, const QString & value)
- void setProperty (const QString & name, const QStringList & value)
- bool isPhase2 () const
- void requestPhase2 ()

Detailed Description

The QSessionManager class provides access to the session manager. The session manager is responsible for session management, most importantly interruption and resumption. QSessionManager provides an interface between the application and the session manager, so that the program can work well with the session manager. In Qt, the session management requests for action are handled by the two virtual functions QApplication::commitData() and QApplication::saveState(). Both functions provide a reference to a session manager object as argument, thus allowing the application to communicate with the session manager. During a session management action, i.e. within one of the two mentioned functions, no user interaction is possible, unless the application got explicit permission from the session manager. You can ask for permission by calling allowsInteraction() or, if it's really urgent, allowsErrorInteraction(). Qt does not enforce this, but the session manager may. Perhaps.

5.9 Ein einfachstes Qt-Programm

Listing 5.3: Einfachstes Qt-Programm

```
//Einfachstes Qt−Programm, das nur einen beschrifteten Knopf zeigt

#include <qapplication.h> //Qt Header−Dateien
#include <qpushbutton.h>
using namespace std;

int main(int argc, char* argv[])
{
  QApplication Anwendung(argc, argv); //Ein Objekt der Klasse QApplication
                                      //braucht jedes GUI−Programm

  // Ein beschrifteter Knopf (noch ohne Funktion!):
  QPushButton *knopf = new QPushButton("Druecken nutzlos!", 0);
  knopf−>resize(150,40);    //ößGre (Breite, Hoehe) explizit vorgeben
  knopf−>show ();           //Sichtbar−machen im Haupt−Widget

  Anwendung.setMainWidget(knopf); //Jedes GUI−Programm hat ein Hauptfenster
  Anwendung.exec ();              //Start der Anwendung (Event−Schleife)
  return 0;
}
```

Nach Übersetzung mittels `g++ -o knopf knopf.cpp -lqt` und Ausführung des Binaries `knopf` erscheint:

Ein etwas komplizierteres Beispiel (zwei Knöpfe, einer mit Quit-Funktionalität über signal-slot-Mechanismus

signal: Dient dem Senden von Nachrichten

slot: Nachricht empfangen und agieren

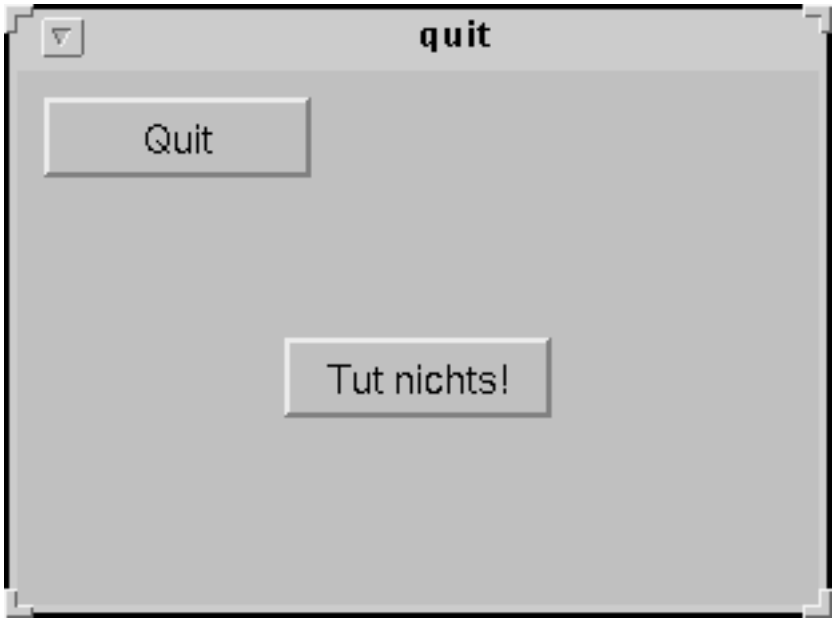
connect: Zuordnung von signal und slot

Objekte (z.B. Buttons) emittieren Nachrichten/Signale (z.B. „mit linker Maustaste angeklickt“), die für andere Objekte von Relevanz sein können. (Andere) Objekte können Nachrichten über einen sogenannten Slot empfangen und der Nachricht entsprechend eine Aktivität (in einer Memberfunktion genau beschrieben) ausführen.

Das folgende Programm besitzt einen funktionsfähigen Quit-Knopf und einen funktionslosen weiteren Knopf:

Listing 5.4: Qt-Programm mit aktivem Knopf

```
//Beispiel zur Verwendung der Qt-Bibliothek
//Einem Knopf wird eine Funktion gegeben: Klickt man den Knopf an,
//so beendet sich das Programm.

#include <qapplication.h>
#include <qpushbutton.h>
using namespace std;

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

  QWidget* hauptfenster = new QWidget();    // Objekt der Klasse QWidget
  hauptfenster->setGeometry(40,60,300,200); // aeusserer Rahmen

  // Der Knopf wird diesmal als Kind von hauptfenster erzeugt, d.h.
  //  der Knopf erscheint innerhalb des aeusseren Rahmens:
  QPushButton* knopf = new QPushButton("Quit", hauptfenster);
  knopf->setGeometry(10,10,100,30);
  // Linke obere Ecke, dann Breite, dann Hoehe

  // Verknuepfung des clicked-Signals des Knopfes mit der
  //  quit-Funktion des Programms.
  QObject::connect( knopf, SIGNAL(clicked()), &Anwendung, SLOT(quit()) );
  /*
    clicked() ist ein Signal (eine Memerfunktion) der Klasse
      QPushButton bzw. einer Oberklasse dieser Klasse
    quit() ist ein Slot der Klasse QApplication
    Verebungshierarchie: Qt --> QObject --> QApplication
      bzw. Qt --> QObject --> QWidget --> QButton -->QPushButton
    QObject::connect() ordnet dem Ereignis clicked() des ueber den Zeiger
    knopf ansprechbaren Objektes den Slot quit() des Objektes
    Anwendung zu.
    Die hier verwendeten Signals and Slots sind in Qt vordefiniert.
  */

  // Ein zweiter Knopf wird ebenfalls als Kind von hauptfenster erzeugt:
  QPushButton* knopf2 = new QPushButton("Tut nichts!", hauptfenster);
  knopf2->setGeometry(100,100,100,30);

  Anwendung.setMainWidget(hauptfenster);
  hauptfenster->show();
  Anwendung.exec(); // Anwendung starten
  return 0;
}
```


quit
Quit
Tut nichts!

5.10 Signal - Slot - Mechanismus in Qt

Signals und Slots dienen Objekten zur Kommunikation. Alle Klassen, die von `QObject` oder einer Unterklasse dieser Klasse (z.B. `QWidget`) abgeleitet werden, können `signal`s und `slot`s verwenden. Signale werden von Objekten emittiert, wenn diese ihren Zustand in einer Weise verändern, der für andere Objekte möglicherweise interessant sein könnte. Das aussendende Objekt weiß dabei nicht, ob es Objekte gibt, die diese Signale auch verarbeiten.

Slots dienen dem Empfang von Signalen. Sie sind gewöhnliche Memberfunktionen einer Klasse. Ein Slot weiß nicht, ob ihm irgendwelche Signale zugeordnet sind. Objekte wissen nichts von dem Kommunikationsmechanismus.

Man kann beliebig viele Signale einem einzigen Slot zuordnen, und ein Signal kann beliebig vielen Slots zugeordnet werden. Man kann sogar einem Signal direkt ein anderes Signal zuordnen (wann immer das erste Signal ausgesandt wird, wir dann auch das zweite emittiert).
Wird ein Signal emittiert, so werden unmittelbar danach alle mit ihm verbundenen Slots aufgerufen.

Signale werden automatisch vom MetaObjectCompiler (`moc`) generiert. Sie dürfen also nicht explizit implementiert werden (nur Deklaration der entsprechenden Memberfunktion).

Das Meta-Object-System in Qt ist verantwortlich für den signal/slot-Mechnismus zur Kommunikation zwischen Objekten.
Die `QMetaObject`-Klasse enthält Metainformation über Qt-Objekte. Alle Metainformation zu einer Klasse wird in einer einzigen Instanz der Klasse `QMetaObject` gehalten. Will man `signal`s und `slot`s selbst definieren, so muss auf das Quellprogramm zunächst der Präprozessor `moc` angewendet werden:

```
moc -o pname.moc pname.cpp
       erzeugte Datei  Quellprogramm
```

Die so erzeugte Datei muss nun mittels

#include "pname.moc"

in das Quellprogramm eingebunden werden (z.B. in der Zeile vor main()).

Klassen, die `signals` und `slots` definieren, müssen das Macro `Q_OBJECT` verwenden (s. Beispiel).

Metainformationen über benutzte Klassen anzeigen lassen:

Listing 5.5: Signal-Slot-Mechanismus: Metainformationen

```
// Zunaechst Aufruf vom moc:   moc -o sg.moc sg.cpp
// Danach uebersetzen mit:     g++ sg.cpp -lqt
#include<qobject.h>
#include<iostream.h>

// Eine Klasse, die Signals und Slots besitzen darf.
class MyClass : public QObject
{
  Q_OBJECT //Metainformation wird vom moc erzeugt
           //Q_OBJECT ist ein #define-Macro
           //Es werden z.B. Deklarationen weiterer Memberfunktionen eingefuegt
public:
  MyClass():val(1){ }; //Konstruktor
  void setValue(int k) { val= k; }
private:
 int val;
};

#include"sg.moc" //Einbinden der vom moc erzeugten Datei

int main()
{
  QObject *q= new QObject();
  cout << "*q: " << (*q).className() << endl; //Metainformation ausgeben

  //Ein Objekt der Klasse MyClass
  MyClass my;
  cout << "my: " << my.className() << endl; //Metainformation ausgeben
  //Die Memberfunktion className() wird vom moc automatisch in die
  //Klasse MyClass eingefuegt
}
```

Die Programmausgabe lautet dann:

```
*q: QObject
my: MyClass
```

Das Programm

Listing 5.6: Signal-Slot-Mechanismus: eine eigene Aktion

```
// Beispiel zum SIGNAL-SLOT-Mechanismus von Qt
// Achtung: Zunaechst  moc -o sig-slot2.moc sig-slot2.cpp  aufrufen!
   //                  g++ sig-slot2.cpp -lqt
#include<qobject.h>  //Qt Header-Datei
#include<iostream.h>
using namespace std;

//Eine Klasse, die Signals und Slots besitzt
class SSTest : public QObject //Klasse Signal-Slot-Test
{
 Q_OBJECT //moc-Praeprozessor wird benoetigt
public:
   SSTest(const int k=1) : wert(k) {}; //Konstruktor
   int getValue() const { return wert; } //Abfrage des Attributs wert
public slots:
   void setValue(int );     //Aenderung des Attributs wert
signals:
   //Dies Signal soll ausgesandt werden, wenn Attrib. wert geaendert wird
   void valueChanged(int);
private:
   int wert  ;              //Attribut wert
};

void SSTest::setValue(int w) //Def. der Memberfunktion setValue()
{
   //Das wert-Attribut wird nur dann modifiziert, wenn sich der aktuelle
   //  Parameterwert vom bisherigen Attributwert unterscheidet und auch
   //  nur dann wird tatsaechlich ein Signal emittiert!
   if ( w != wert )
   {
      cout << "Wertaenderung (alt: " << wert << " neu: " << w << ")" << endl;
      wert= w;
      emit valueChanged(w);  //Signal aussenden
   }
}

#include "sig-slot2.moc" //Ergebnisdatei des Praeprozessorlaufs einbinden
void main() {
   //Zwei Objekte der Klasse SSTest erzeugen
   SSTest* a = new SSTest();  //a->wert ist 1 (Default)
   SSTest* b = new SSTest(5); //b->wert ist 5

   //Das Signal des ersten wird mit dem Slot des zweiten Objekts verbunden
   QObject::connect(a, SIGNAL(valueChanged(int)), b, SLOT(setValue(int)));

   //Das Signal des zweiten wird mit dem Slot des ersten Objekts verbunden
   QObject::connect(b, SIGNAL(valueChanged(int)), a, SLOT(setValue(int)));

   cout << "a:" << a->getValue() << endl;
   cout << "b:" << b->getValue() << endl;

   a->setValue( 7 ); // a->wert wird auf 7 gesetzt, also geaendert. Durch
                     //   obige üVerknpfung wird auch b->wert auf 7 gesetzt!
   cout << "a:" << a->getValue() << endl; //Durch Ausgabe bestaetigen
   cout << "b:" << b->getValue() << endl;

   cout << "Programmausgabe zur Quellcodedatei: " << __FILE__ << endl;
}
```

liefert die Ausgabe:

```
a:1
b:5
Wertaenderung (alt: 1 neu: 7)
Wertaenderung (alt: 5 neu: 7)
a:7
b:7
Programmausgabe zur Quellcodedatei: sig-slot2.cpp
```

Der `moc`-Output sieht dabei folgendermaßen aus:

Listing 5.7: Signal-Slot-Mechanismus: Inhalt der moc-Datei

```
/****************************************************************************
** SSTest meta object code from reading C++ file 'sig-slot2.cpp'
**
** Created: Thu Jul 12 11:56:01 2001
**      by: The Qt MOC ($Id: qt/src/moc/moc.y   2.2.0   edited 2000-08-31 $)
**
** WARNING! All changes made in this file will be lost!
****************************************************************************/

#define Q_MOC_SSTest
#if !defined(Q_MOC_OUTPUT_REVISION)
#define Q_MOC_OUTPUT_REVISION 9
#elif Q_MOC_OUTPUT_REVISION != 9
#error "Moc format conflict - please regenerate all moc files"
#endif

#include <qmetaobject.h>
#include <qapplication.h>

#if defined(Q_SPARCWORKS_FUNCP_BUG)
#define Q_AMPERSAND
#else
#define Q_AMPERSAND &
#endif


const char *SSTest::className() const
{
    return "SSTest";
}

QMetaObject *SSTest::metaObj = 0;

void SSTest::initMetaObject()
{
    if ( metaObj )
        return;
    if ( qstrcmp(QObject::className(), "QObject") != 0 )
        badSuperclassWarning("SSTest","QObject");
    (void) staticMetaObject();
}

#ifndef QT_NO_TRANSLATION
```

```
QString SSTest::tr(const char* s)
{
    return qApp->translate( "SSTest", s, 0 );
}

QString SSTest::tr(const char* s, const char * c)
{
    return qApp->translate( "SSTest", s, c );
}

#endif // QT_NO_TRANSLATION

QMetaObject* SSTest::staticMetaObject()
{
    if ( metaObj )
        return metaObj;
    ( void) QObject::staticMetaObject();
#ifndef QT_NO_PROPERTIES
#endif // QT_NO_PROPERTIES
    typedef void(SSTest::*m1_t0)(int);
    m1_t0 v1_0 = Q_AMPERSAND SSTest::setValue;
    QMetaData *slot_tbl = QMetaObject::new_metadata(1);
    QMetaData::Access *slot_tbl_access = QMetaObject::new_metaaccess(1);
    slot_tbl [0]. name = "setValue(int)";
    slot_tbl [0]. ptr = (QMember)v1_0;
    slot_tbl_access [0] = QMetaData::Public;
    typedef void(SSTest::*m2_t0)(int);
    m2_t0 v2_0 = Q_AMPERSAND SSTest::valueChanged;
    QMetaData *signal_tbl = QMetaObject::new_metadata(1);
    signal_tbl [0]. name = "valueChanged(int)";
    signal_tbl [0]. ptr = (QMember)v2_0;
    metaObj = QMetaObject::new_metaobject(
        "SSTest", "QObject",
        slot_tbl , 1,
        signal_tbl , 1,
#ifndef QT_NO_PROPERTIES
        0, 0,
        0, 0,
#endif // QT_NO_PROPERTIES
        0, 0 );
    metaObj->set_slot_access( slot_tbl_access );
#ifndef QT_NO_PROPERTIES
#endif // QT_NO_PROPERTIES
    return metaObj;
}

// SIGNAL valueChanged
void SSTest::valueChanged( int t0 )
{
    activate_signal ( "valueChanged(int)", t0 );
}
```

Die Header-Datei, welche das Macro `Q_OBJECT` definiert, hat folgende Struktur:

Listing 5.8: Signal-Slot-Mechanismus: Struktur des Macros Q_OBJECT

```
/****************************************************************************
** $Id: qt/src/kernel/qobjectdefs.h   2.2.0   edited 2000-08-25 $
**
** Macros and definitions related to QObject
**
** Created : 930419
**
** Copyright (C) 1992-2000 Trolltech AS.  All rights reserved.
**
** This file is part of the kernel module of the Qt GUI Toolkit.
**
** This file may be distributed under the terms of the Q Public License ...
**
*****************************************************************************/

#ifndef QOBJECTDEFS_H
#define QOBJECTDEFS_H

#ifndef QT_H
#include "qglobal.h"
#endif // QT_H


// The following macros are our "extensions" to C++
// They are used, strictly speaking, only by the moc.

#ifndef QT_NO_TRANSLATION
#define QT_TR_FUNCTION static QString tr(const char*); \
                       static QString tr(const char*, const char*);
#else
#define QT_TR_FUNCTION // inherit the one from QObject
#endif

#ifdef QT_MOC_CPP
 #define slots                          slots
 #define signals                        signals
 #define Q_CLASSINFO( name, value ) Q_CLASSINFO( name, value )
 #define Q_PROPERTY( text )         Q_PROPERTY( text )
 #define Q_OVERRIDE( text )         Q_OVERRIDE( text )
 #define Q_ENUMS( x )               Q_ENUMS( x )
 #define Q_SETS( x )                Q_SETS( x )
 /* tmake ignore Q_OBJECT */
 #define Q_OBJECT                   Q_OBJECT
 /* tmake ignore Q_OBJECT */
 #define Q_OBJECT_FAKE              Q_OBJECT_FAKE

#else
 #define slots                                  // slots :   in class
 #define signals protected                      // signals : in class
 #define emit                                   // emit signal
 #define Q_CLASSINFO( name, value )             // class info
 #define Q_PROPERTY( text )                     // property
 #define Q_OVERRIDE( text )                     // override property
 #define Q_ENUMS( x )
 #define Q_SETS( x )
```

```
/* tmake ignore Q_OBJECT */
 #define Q_OBJECT                                        \
public:                                                  \
    QMetaObject *metaObject() const {                    \
        return staticMetaObject();                       \
    }                                                    \
    const char *className() const;                       \
    static QMetaObject* staticMetaObject();              \
    QT_TR_FUNCTION                                       \
protected:                                               \
    void initMetaObject();                               \
private:                                                 \
    static QMetaObject *metaObj;

/* tmake ignore Q_OBJECT */
#define Q_OBJECT_FAKE Q_OBJECT

#endif

// macro for naming members
#if defined(_OLD_CPP_)
#define METHOD(a)       "0""a"
#define SLOT(a)         "1""a"
#define SIGNAL(a)       "2""a"
#else
#define METHOD(a)       "0"#a
#define SLOT(a)         "1"#a
#define SIGNAL(a)       "2"#a
#endif

#define METHOD_CODE     0                       // member type codes
#define SLOT_CODE       1
#define SIGNAL_CODE     2


// Forward declarations so you don't have to include  files  you don't need

class QObject;
class QMetaObject;
class QSignal;
class QConnection;
class QEvent;
struct QMetaData;
class QConnectionList;
class QConnectionListIt;
class QSignalDict;
class QSignalDictIt;
class QObjectList;
class QObjectListIt;
class QMemberDict;


#endif // QOBJECTDEFS_H
```

Betrachten wir

```
#define Q_OBJECT
public:
    QMetaObject *metaObject() const {
        return staticMetaObject();
    }
    const char *className() const;
    static QMetaObject* staticMetaObject();
    QT_TR_FUNCTION
protected:
    void initMetaObject();
private:
    static QMetaObject *metaObj;
```

so wird klar, dass dies in den Rumpf der eigenen Klasse an der Stelle von `O_OBJECT` eingefügt wird.

Möchte man den Output des `moc` per Hand einfügen, so würde dies folgendermaßen aussehen:

Listing 5.9: Signal-Slot-Mechanismus: durch moc modifizierte Quelle

```
//Beispiel zum SIGNAL-SLOT-Mechanismus von Qt
//moc Aktivitaeten per Hand eingefuegt

#include<qobject.h>  //Qt Header-Datei
#include<iostream.h>
using namespace std;

//Eine Klasse, die Signals und Slots besitzt
class SSTest : public QObject //Klasse Signal-Slot-Test
{
 //Q_OBJECT //**********  moc-Praeprozessor wird benoetigt
public:
    QMetaObject *metaObject() const {
        return staticMetaObject();
    }
    const char *className() const;
    static QMetaObject* staticMetaObject();
    static QString tr(const char*);
    static QString tr(const char*, const char*);
protected:
    void initMetaObject();
private:
    static QMetaObject *metaObj;
 //**********************  Ende moc

public:
   SSTest(const int k=1) : wert(k) {};   //Konstruktor
   int getValue() const { return wert; } //Abfrage des Attributs wert
public slots   :              //Wort  slots   wird noch durch leeren String  ersetzt
   void setValue(int );     //Aenderung des Attributs wert
signals    :                  //Wort  signals   wird noch durch  protected  ersetzt
   //Dies Signal  soll  ausgesandt werden, wenn Attrib. wert geaendert wird
   void valueChanged(int);
private:
   int wert  ;               //Attribut wert
};
```

```
void SSTest::setValue(int w) //Def. der Memberfunktion setValue()
{
   //Das wert-Attribut wird nur dann modifiziert, wenn sich der aktuelle
   //  Parameterwert vom bisherigen Attributwert unterscheidet und auch
   //  nur dann wird tatsaechlich ein Signal emittiert!
   if ( w != wert )
   {
      cout << "Wertaenderung (alt: " << wert << " neu: " << w << ")" << endl;
      wert= w;
      emit valueChanged(w);  //Signal aussenden
   }
}

//#include "sig-slot2.moc"    //Ergebnisdatei des Praeprozessorlaufs einbinden
#define Q_MOC_SSTest
#if !defined(Q_MOC_OUTPUT_REVISION)
#define Q_MOC_OUTPUT_REVISION 9
#elif Q_MOC_OUTPUT_REVISION != 9
#error "Moc format conflict - please regenerate all moc files"
#endif

#include <qmetaobject.h>
#include <qapplication.h>

#if defined(Q_SPARCWORKS_FUNCP_BUG)
#define Q_AMPERSAND
#else
#define Q_AMPERSAND &
#endif

const char *SSTest::className() const
{
    return "SSTest";
}

QMetaObject *SSTest::metaObj = 0;

void SSTest::initMetaObject()
{
    if ( metaObj )
        return;
    if ( qstrcmp(QObject::className(), "QObject") != 0 )
        badSuperclassWarning("SSTest","QObject");
    (void) staticMetaObject();
}

#ifndef QT_NO_TRANSLATION

QString SSTest::tr(const char* s)
{
    return qApp->translate( "SSTest", s, 0 );
}

QString SSTest::tr(const char* s, const char * c)
{
    return qApp->translate( "SSTest", s, c );
}

#endif // QT_NO_TRANSLATION

QMetaObject* SSTest::staticMetaObject()
{
```

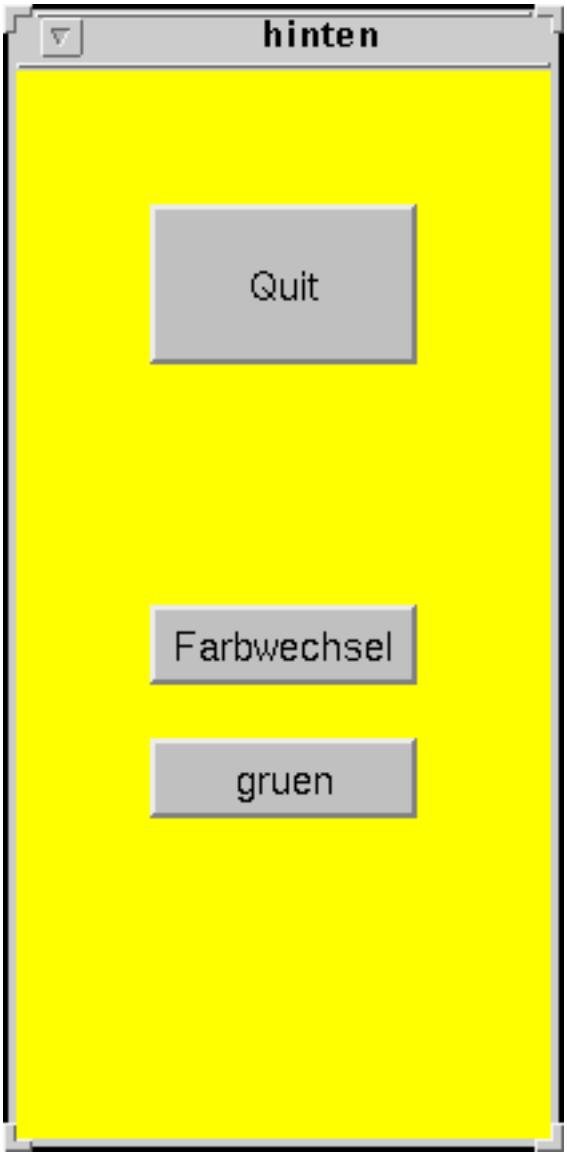
```
100     if ( metaObj )
            return metaObj;
        ( void) QObject::staticMetaObject();
#ifndef QT_NO_PROPERTIES
#endif // QT_NO_PROPERTIES
105     typedef void(SSTest::*m1_t0)(int);
        m1_t0 v1_0 = Q_AMPERSAND SSTest::setValue;
        QMetaData *slot_tbl = QMetaObject::new_metadata(1);
        QMetaData::Access *slot_tbl_access = QMetaObject::new_metaaccess(1);
        slot_tbl [0]. name = "setValue(int)";
110     slot_tbl [0]. ptr = (QMember)v1_0;
        slot_tbl_access [0] = QMetaData::Public;
        typedef void(SSTest::*m2_t0)(int);
        m2_t0 v2_0 = Q_AMPERSAND SSTest::valueChanged;
        QMetaData *signal_tbl = QMetaObject::new_metadata(1);
115     signal_tbl [0]. name = "valueChanged(int)";
        signal_tbl [0]. ptr = (QMember)v2_0;
        metaObj = QMetaObject::new_metaobject(
            "SSTest", "QObject",
            slot_tbl , 1,
120         signal_tbl , 1,
#ifndef QT_NO_PROPERTIES
            0, 0,
            0, 0,
#endif // QT_NO_PROPERTIES
125         0, 0 );
        metaObj->set_slot_access( slot_tbl_access );
#ifndef QT_NO_PROPERTIES
#endif // QT_NO_PROPERTIES
        return metaObj;
130 }

// SIGNAL valueChanged
void SSTest::valueChanged( int t0 )
{
135     activate_signal ( "valueChanged(int)", t0 );
}
//****************************** Ende #include sig-slot2.moc

void main() {
140     //Zwei Objekte der Klasse SSTest erzeugen
    SSTest* a = new SSTest();  //a->wert ist 1 (Default)
    SSTest* b = new SSTest(5); //b->wert ist 5

    //Das Signal des ersten wird mit dem Slot des zweiten Objekts verbunden
145 QObject::connect(a, SIGNAL(valueChanged(int)), b, SLOT(setValue(int)));

    //Das Signal des zweiten wird mit dem Slot des ersten Objekts verbunden
    QObject::connect(b, SIGNAL(valueChanged(int)), a, SLOT(setValue(int)));

150 cout << "a:" << a->getValue() << endl;
    cout << "b:" << b->getValue() << endl;

    a->setValue( 7 ); // a->wert wird auf 7 gesetzt, also geaendert. Durch
                      //   obige Verknuepfung wird auch b->wert auf 7 gesetzt!
155 cout << "a:" << a->getValue() << endl; //Durch Ausgabe bestaetigen
    cout << "b:" << b->getValue() << endl;

    cout << "Programmausgabe zur Quellcodedatei: " << __FILE__ << endl;
}
```

Möchte man die Hintergrundfarbe eines Fensters ändern, so kann dies folgendermaßen geschehen:

Listing 5.10: Hintergrundfarbe eines Fensters ändern

```
/*********************************************************/
/* Programmieren II fuer Maschinentechnik (C/C++), SS 2000 */
/*   W. Kraemer, W. Hofschuster, Fachbereich Mathematik   */
 /*          Bergische Universitaet GH Wuppertal          */
/*********************************************************/

// Hintergrundfarbe eines Fensters aendern;
// signal-slot-Mechanismus in Qt
//   signal: Nachricht senden
//   slot :  Nachricht empfangen und agieren
//   connect: Zuordnung von signal und slot

#include <qapplication.h>  // Qt Header-Dateien
#include <qpushbutton.h>

class meinFenster : public QWidget // Vererbung
{
  Q_OBJECT  // Praeprozessor moc muss verwendet werden!
public:
  meinFenster();
public slots: // wird vom Praeprozessor moc umgesetzt
  void aendereFarbe();
  void setzeAufGruen();
private:
  QPushButton *quit, *next, *gruen; // diverse (Aktions-)Knoepfe
};

meinFenster::meinFenster() // Konstruktor
{
  setBackgroundColor(yellow); // starte mit gelbem Hintergrund

  // Knopf zum Beenden des Programms
  quit = new QPushButton( "Quit", this );
  quit->setGeometry( 50, 50, 100, 60 );
  connect( quit, SIGNAL(clicked()), qApp, SLOT(quit()) );
  // qApp ist eine globale Variable!

  // Knopf zum zyklischen Aendern der Hintergrundfarbe
  next = new QPushButton( "Farbwechsel", this );
  next->setGeometry( 50, 200, 100, 30 );
  connect( next, SIGNAL(clicked()), this, SLOT(aendereFarbe()) );

  // Knopf fuer gruenen Hintergrund
  gruen = new QPushButton( "gruen", this );
  gruen->setGeometry( 50, 250, 100, 30 );
  connect( gruen, SIGNAL(clicked()), this, SLOT(setzeAufGruen()) );

}


void meinFenster::aendereFarbe() // Memberfunktion
{
   if ( backgroundColor()!=red )
      setBackgroundColor(red);
   else
      setBackgroundColor(black);
```

```
    update();
}

void meinFenster::setzeAufGruen() // Memberfunktion
{
    setBackgroundColor(green);
    update();
}

#include "hinten.moc"  // <==  Name, der vom Praeproz. erzeugten Datei

int main( int argc, char* argv[] )
{
    QApplication Anwendung( argc, argv ); // Objekt der Klasse QApplication

    meinFenster w;  // Objekt der Klasse meinFenster generieren
    w.resize(200,400);

    Anwendung.setMainWidget( &w ); // Hauptfenster der Anwendung ist w
    w.show();
    Anwendung.exec();  // Anwendung starten
    return 0;
}
```

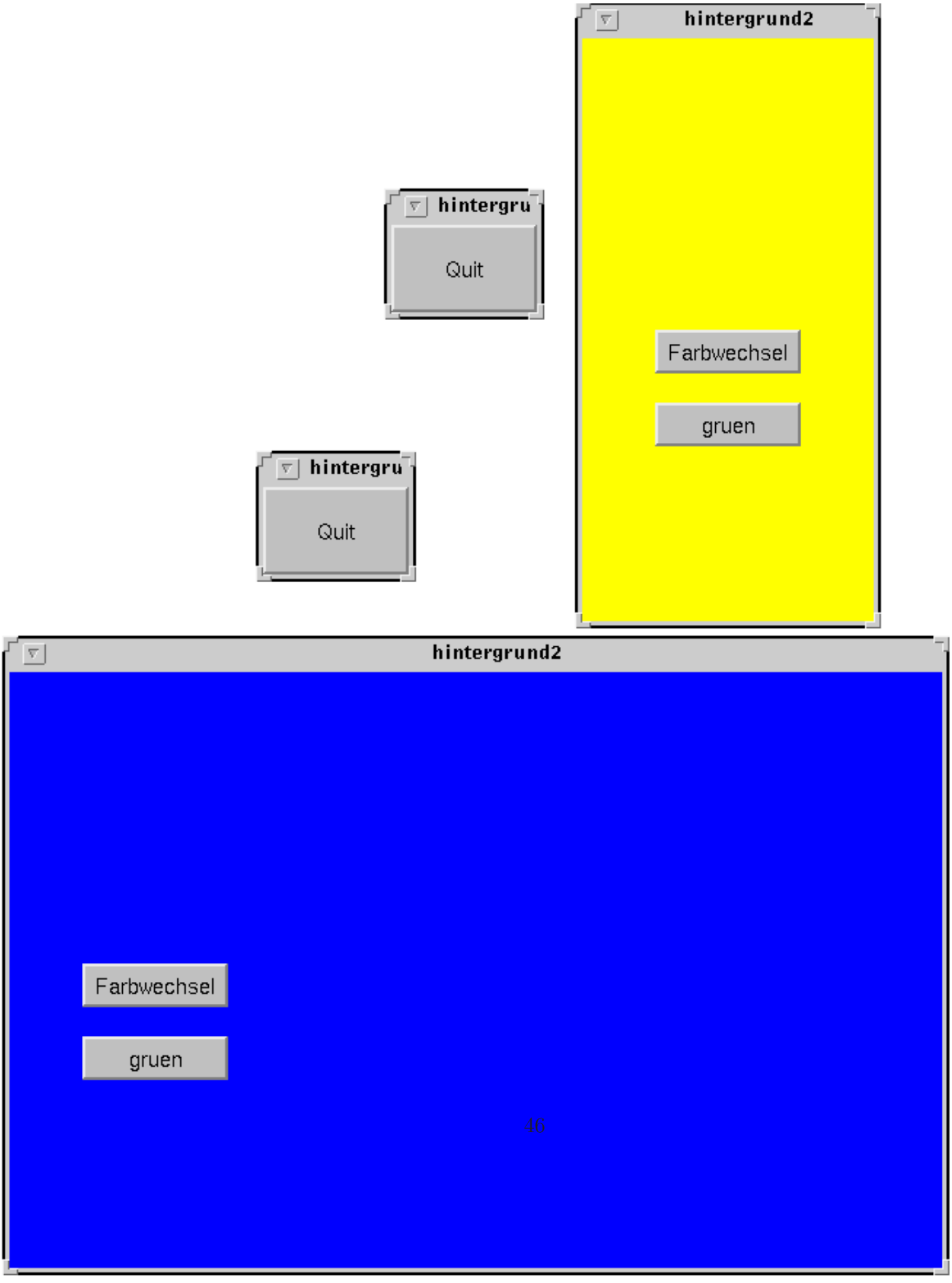

hinten
Quit
Farbwechsel
gruen

Ein wenig variabler geht das so:

Listing 5.11: Hintergrundfarbe eines Fensters ändern Vs. 2

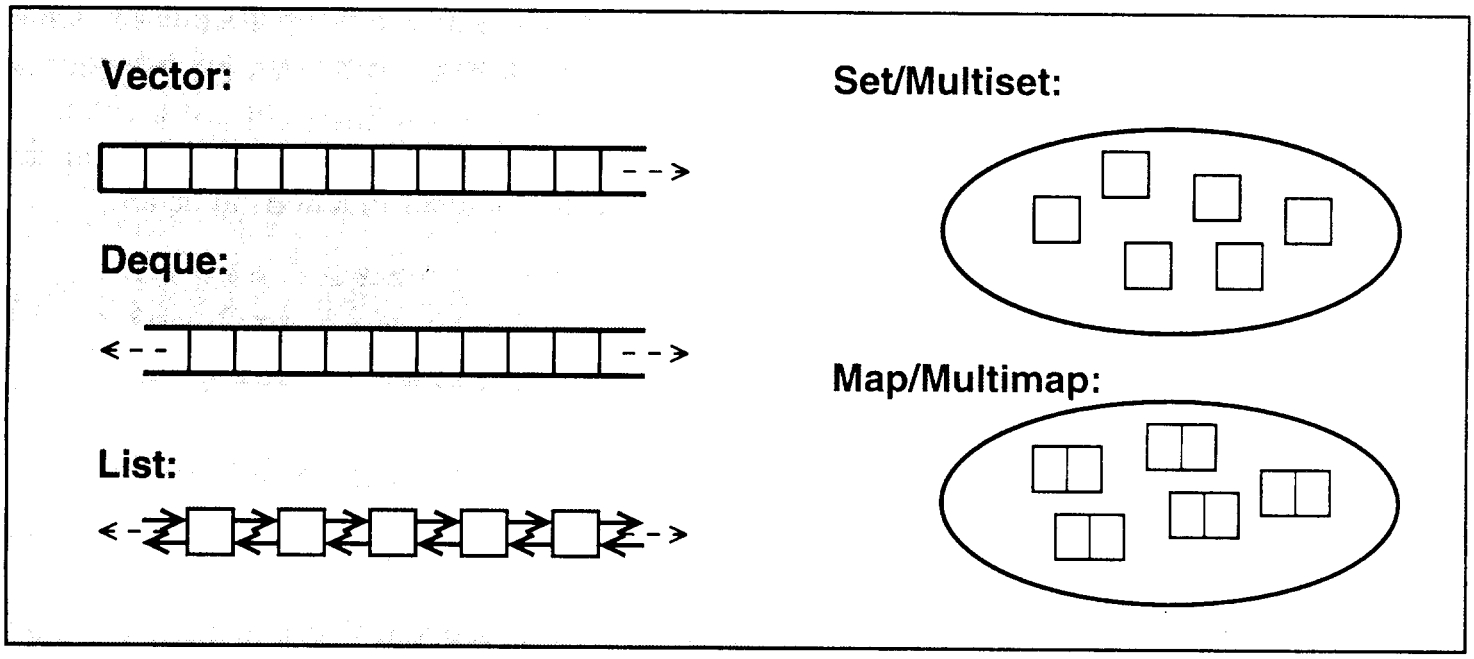
```
/*********************************************************/
/*   W. Kraemer, W. Hofschuster,  Fachbereich Mathematik   */
 /*            Bergische Universitaet  GH Wuppertal          */
/*********************************************************/
//Erinnerung:  moc -o progname.moc progname.cpp

// Hintergrundfarbe eines Fensters aendern;
// signal-slot-Mechanismus in Qt
//    signal : Nachricht senden
//    slot :   Nachricht empfangen und agieren
//    connect:  Zuordnung von signal und slot

#include <qapplication.h>  // Qt Header-Dateien
#include <qpushbutton.h>
#include <iostream>

class meinFenster : public QWidget // Vererbung
{
  Q_OBJECT  // steuert den Praeprozessor moc!
public:
  meinFenster(const QColor & c); // Konstruktor
public slots: // wird vom Praeprozessor moc umgesetzt
  void aendereFarbe();
  void setzeAufGruen();
private:
  QPushButton *quit, *next, *gruen; // diverse (Aktions-)Knoepfe
};

meinFenster::meinFenster(const QColor &c=yellow) // Konstruktor
{
  setBackgroundColor(c); // Defaulthintergrundfarbe ist gelb

  // Knopf zum Beenden des Programms
  quit = new QPushButton( "Quit", 0 );
  quit->setGeometry( 50, 50, 100, 60 );
  (*quit).show();
  connect( quit, SIGNAL(clicked()), this, SLOT(close()) );

  // Knopf zum zyklischen Aendern der Hintergrundfarbe
  next = new QPushButton( "Farbwechsel", this );
  next->setGeometry( 50, 200, 100, 30 );
  connect( next, SIGNAL(clicked()), this, SLOT(aendereFarbe()) );

  // Knopf fuer gruenen Hintergrund
  gruen = new QPushButton( "gruen", this );
  gruen->setGeometry( 50, 250, 100, 30 );
  connect( gruen, SIGNAL(clicked()), this, SLOT(setzeAufGruen()) );

}


void meinFenster::aendereFarbe() // Memberfunktion
{
   if ( backgroundColor()!=red )
      setBackgroundColor(red);
   else
      setBackgroundColor(black);
```

```
    update();
}

void meinFenster::setzeAufGruen() // Memberfunktion
{
    QColor my=green;
    setBackgroundColor(Qt::green);
    update();
}

#include"hintergrund2.moc"

using namespace std;

int main( int argc, char* argv[] )
{
     QApplication Anwendung( argc, argv ); // Objekt der Klasse QApplication

     meinFenster w1, w2(Qt::blue);  // Objekte der Klasse meinFenster generieren
     w1.resize(200,400);
     cout << " Programm! " << endl;
//     Anwendung.setMainWidget( &w1 ); // Hauptfenster der Anwendung ist w
     w1.show();
     w2.show();
     Anwendung.exec();  // Anwendung starten
     cout << "Programmende!" << endl;
    return 0;
}
```


hintergrund2
hintergru
Quit
Farbwechsel
gruen
hintergru
Quit
hintergrund2
Farbwechsel
gruen

Kapitel 6

Die Standard-Template-Library (STL)



Container: Strukturen wie `vector, list, deque, stack, queue,` die eine Ansammlung von Objekten (Elemente des **Containers**) verwalten. Die Namen für gleichartige Methoden sind dabei für die verschiedenen Container weitgehend gleich gewählt. Z.B.: `size(), begin(), end()` Die Container selbst sind als Template-Klassen realisiert, so dass sie sowohl Standarddatentypen aber auch Objekte von benutzerdefinierten Klassen aufnehmen können.

Iteratoren: Sie arbeiten wie Zeiger. Mit ihnen ist der Zugriff auf die Elemente von Containern möglich und auch dringend zu empfehlen. Sie sind üblicherweise als Objekte mit zeigerähnlichen Eigenschaften realisiert. Z.B. ist die Fortschaltung durch `Operator ++` möglich. Iteratoren werden von den jeweiligen Containern zur Verfügung gestellt, können jedoch auch selbst programmiert werden: vergleiche dazu

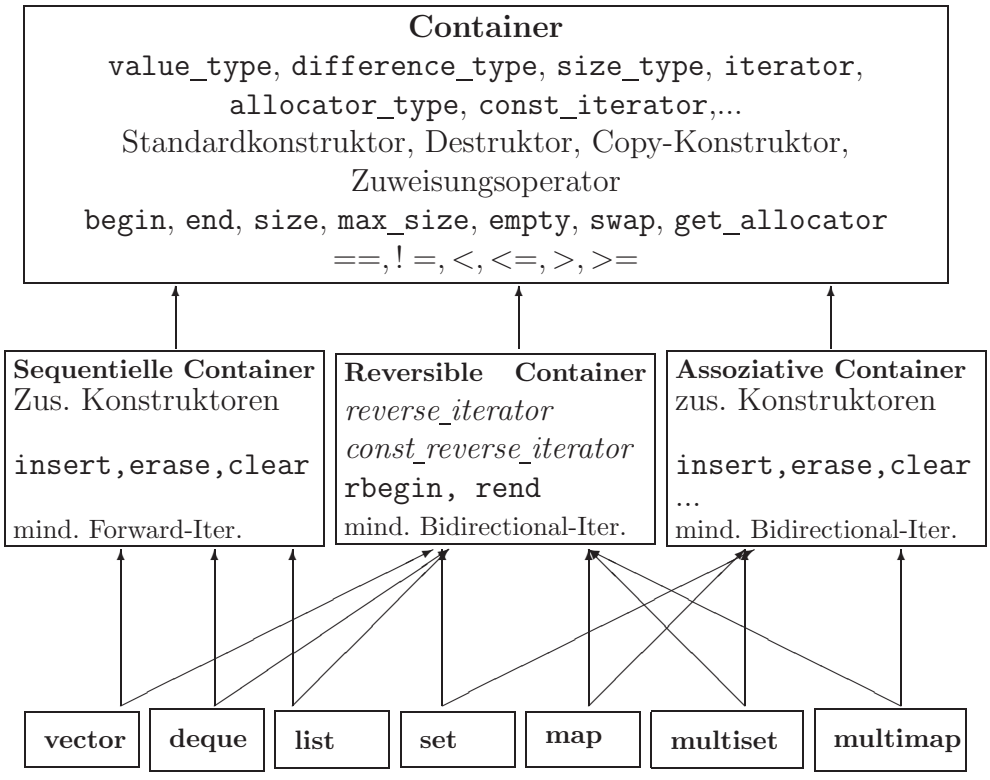
http://www.math.uni-wuppertal.de/~buhl/teach/exercises/Inf1-WS0001/script.pdf#section.1.5

sowie

http://www.math.uni-wuppertal.de/~buhl/teach/exercises/Inf1-WS0001/script.pdf#section.2.12.

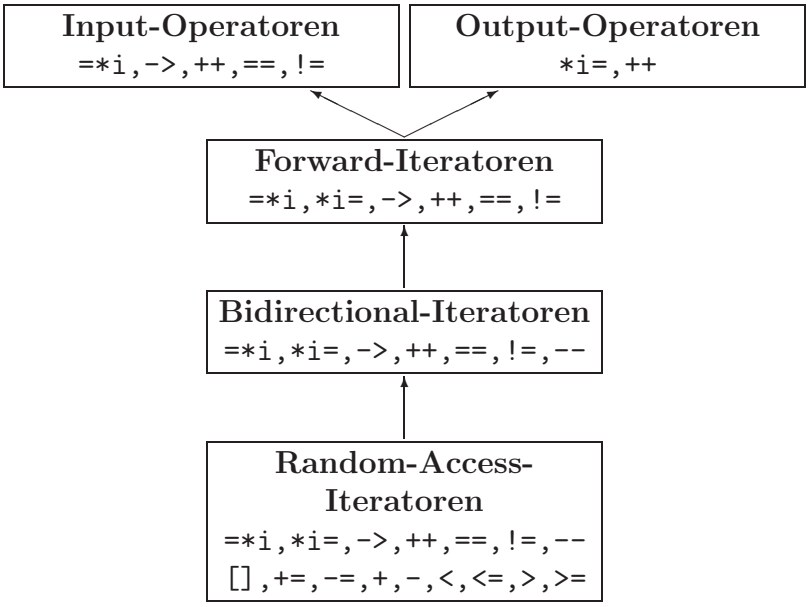
Algorithmen: Algorithmen sind globale Funktionen mit standardisierten Schnittstellen, die unter Verwendung von Iteratoren Container bzw. Containerelemente manipulieren. Man denke z.B. an einen Sortieralgorithmus, welcher auf einer Liste, einem Stack oder einem anderen Container arbeitet.

Die Container gliedern sich folgendermaßen:



	Vector	Deque	List	Set	Multiset	Map	Multimap
Typical internal data structure	Dynamic array	Array of arrays	Doubly linked list	Binary tree	Binary tree	Binary tree	Binary tree
Elements	Value	Value	Value	Value	Value	Key/value pair	Key/value pair
Duplicates allowed	Yes	Yes	Yes	No	Yes	Not for the key	Yes
Random access available	Yes	Yes	No	No	No	With key	No
Iterator category	Random access	Random access	Bidirectional	Bidirectional (element constant)	Bidirectional (element constant)	Bidirectional (key constant)	Bidirectional (key constant)
Search/find elements	Slow	Slow	Very slow	Fast	Fast	Fast for key	Fast for key
Inserting/removing of elements is fast	At the end	At the beginning and the end	Anywhere	—	—	—	—
Inserting/removing invalidates iterators, references, pointers	On reallocation	Always	Never	Never	Never	Never	Never
Frees memory for removed elements	Never	Sometimes	Always	Always	Always	Always	Always
Allows memory reservation	Yes	No	—	—	—	—	—
Transaction safe (success or no effect)	Push/pop at the end	Push/pop at the beginning and the end	All except `sort()` and assignments	All except multiple-element insertions	All except multiple-element insertions	All except multiple-element insertions	All except multiple-element insertions

Die Iteratoren lassen sich folgendermaßen einordnen:



6.1 Einführende Beispiele

Wir suchen in einem selbstdefinierten Container für `int`-Werte (einem einfachen C-Feld) nach einem einzulesenden `int`-Wert x.

6.1.1 Iteratoren selbst definieren

Es wird ein Algorithmus `find()` realisiert, der ähnlich wie der Algorithmus `find()` der STL funktioniert:

Listing 6.1: find() und Iteratoren (selbst implementiert)

```
#include <iostream>
using namespace std;

//neuer Typname IteratorType fuer Zeiger auf int
typedef int* IteratorType; //Namenskonvention der STL
//Prototyp des Algorithmus
IteratorType find( IteratorType begin, IteratorType end,
                   const int& value );
int main() {
   const int n=100;
   int MyContainer[n];  //Container anlegen

   IteratorType begin = MyContainer; //Zeiger (Iterator) auf Anfang

   //Position nach dem letzten Element
   IteratorType end = MyContainer + n;

   //Container mit geraden Zahlen fuellen
   for (int i=0; i<n; i++) MyContainer[i]= 2*i;

   int x = 0;
   while (x != -1)
   {
      cout << "Welche Zahl soll gesucht werden (Abbruch mit -1): ";
      cin >> x;
      if (x != -1)
     {
         IteratorType position = find(begin, end, x);
         if ( position != end)
             cout << "gefunden an Pos. " << (position - begin) << endl;
         else
            cout << x << " wurde nicht gefunden! " << endl;
     }
   } //while
} //main

//Implementierung
IteratorType find( IteratorType begin, IteratorType end,
                   const int& value )
{
   while(begin != end  &&  //Vergleich von Iteratoren (Zeigern)
      value != *begin )      //Dereferenzierung und Objektvergleich
    ++begin   ;              //Fortschaltung des Iterators
   return begin;
}
/*
```

```
Welche Zahl soll gesucht werden (Abbruch mit -1): 8
gefunden an Pos. 4
Welche Zahl soll gesucht werden (Abbruch mit -1): 3
3 wurde nicht gefunden!
Welche Zahl soll gesucht werden (Abbruch mit -1): -1
*/
```

Beachte: Die Realisierung von `find()` muss nichts über den Container wissen, ist also unabhängig vom Typ der im Container abgelegten Objekte. Es werden nur Zeiger (Iteratoren) benutzt, für die Operatoren

`++` zum Weiterschalten und `*` zum Dereferenzieren

definiert und die mittels `operator !=` vergleichbar sein müssen. Auch die Objekte im Container müssen mit `!=` vergleichbar sein.

6.1.2 find() als Templatefunktion

Hier wird der Algorithmus `find()` als Template realisiert:

Listing 6.2: find() als template

```
#include <iostream>
using namespace std;

//neuer Typname IteratorType fuer Zeiger auf int
typedef int* IteratorType; //Namenskonvention der STL

//Ersetzung des Prototyps fuer find() durch ein Template !!!
//  und Entfernung der bisherigen Definition  .          !!!
//Ansonsten wird am Programm nichts geaendert!          !!!


template<class IteratorType> //!!!
IteratorType find( IteratorType begin, IteratorType end,
                   const int& value )
/*
Es waere auch der folgende Kopf moeglich:
template<class T> T find(T begin, T end, const int& value)
*/
{
   while(begin != end  && value != *begin) ++begin;
   return begin;
}

int main() {
   const int n=100;
   int MyContainer[n];  //Container anlegen

   IteratorType begin = MyContainer; //Zeiger (Iterator) auf Anfang

   //Position nach dem letzten Element
   IteratorType end = MyContainer + n;

   //Container mit geraden Zahlen fuellen
```

```
    for (int i=0; i<n; i++) MyContainer[i]= 2*i;

    int x = 0;
    while (x != -1)
    {
        cout << "Welche Zahl soll gesucht werden (Abbruch mit -1): ";
        cin >> x;
        if (x != -1)
        {
            IteratorType position = find(begin, end, x);
            if (position != end)
                cout << "gefunden an Pos. " << (position - begin) << endl;
            else
                cout << x << " wurde nicht gefunden! " << endl;
        }
    } //while
} //main
```

6.2 Der list-Container der STL

Das nächste Beispiel zeigt einige Experimente mit dem `list`-Container der STL.

Listing 6.3: Benutzung des list-Containers

```
/*
Experimente mit dem list-Container der STL
*/

#include <iostream>
#include <list>
#include <algorithm>
#include<functional>
using namespace std;

template<class T> ostream& operator<<(ostream& os, const list<T>& l){
    if(l.empty())
        return os << "leer";
    for( list <T>::const_iterator i=l.begin(); i!=l.end(); i++) os <<*i<<" ";
    return os;
}

template <class T> void print(char c, list<T>& l)
{ cout << c << ": " << l << endl; }

int main() {
    list <int> a, b;  //Container anlegen

    for (int i=1; i<9; i++)  //Container fuellen
        if (i%2==1)
            a.push_front(i);
        else
            b.push_back(i);

    print('a',a); print('b',b);
    a.sort();
    cout << "a sortiert: " << a << endl;
```

```
    list <int>c(a); //c neu anlegen und wie a aufbauen
    print('c',c);
    a.merge(c);
    print('c',c);
    print('a',a);
    //const list <int> save(b);
    // list <int>::iterator cp= save.begin();
    //cout << "save: " << save << endl;
    b.reverse();
    print('b',b);
    a.remove(5);
    print('a',a);
    a.unique();
    print('a',a);
    c= a;
    print('c',c);
    c.clear();
    print('c',c);
    list <int>::iterator p= find(a.begin(), a.end(), 3);
    cout << "*p: " << *p << endl;
//  cout << "Position: " << distance( a.begin(), p ) << endl;
    a.insert(p, 13);
    print('a',a);
    cout << "*p: " << *p << endl;
    a.erase(++p);
    print('a',a);
    a.push_back(19);
    print('a',a);
    a.pop_front();
    print('a',a);
    a.sort();
    print('a',a);
    a.remove(13);
    print('a',a);
    c= a;
    print('c',c);
    c.splice(c.begin(), a);
    print('c',c);
    c.push_front(7);
    print('c',c);
    c.sort(); c.unique();
    print('c',c);
    print('a',a);
    c.splice(c.begin(), a);
    print('c',c);
} //main
/*
a: 7 5 3 1
b: 2 4 6 8
a sortiert : 1 3 5 7
c: 1 3 5 7
c: leer
a: 1 1 3 3 5 5 7 7
b: 8 6 4 2
a: 1 1 3 3 7 7
a: 1 3 7
c: 1 3 7
c: leer
*p: 3
a: 1 13 3 7
*p: 3
a: 1 13 3
```

```
a: 1 13 3 19
a: 13 3 19
a: 3 13 19
a: 3 19
c: 3 19
c: 3 19 3 19
c: 7 3 19 3 19
c: 3 7 19
a: leer
c: 3 7 19
*/
```

6.3 Eine selbstimplementierte einfach gelinkte Liste und STL find()

Die Suche in einem selbstdefinierten Container `list` nach einem einzulesenden `int`-Wert x.

Listing 6.4: find() der STL bei selbstgeschriebenen Containern

```
#include <iostream>
#include <algorithm>
#include <cassert>
#include <utility>

using namespace std;

template <class T> class list {  //selbstdef. rudimentaere Listenklasse
 public:
    list () : firstElem(0), count(0) {}

   void push_front(const T& datum) //neu erz. Elem. am Listenanfang einfuegen
   {
      firstElem= new ListElem(datum, firstElem);
     ++count;
   }

   struct ListElem { //geschachtelte Klasse
      T data; //Typ des Benutzers (template-Param.)
      ListElem* next;
      ListElem(const T& datum, ListElem* p) : data(datum), next(p) {}
   };

 private:
   ListElem* firstElem; //zeigt auf erstes Listenelement
   int count ;          //Anzahl der vorhandenen Listenelemente

 public:
   class iterator { //geschachtelte Klasse
    public:
      //von GNU-Compiler benoetigt (Typen sind dann oeffentlich):
      //typedef size_t  size_type;
      typedef ptrdiff_t  difference_type;
      typedef forward_iterator_tag iterator_category;
      typedef T value_type;
```

```
      typedef T* pointer;
      typedef T& reference;

      iterator(ListElem* init=0) : current(init){}
      T& operator*() { return current->data; } //Dereferenzierung
//    const T& operator*() const { return current->data; } //Deref.
      iterator& operator++() { //Praefix
        if (current) //noch nicht am Ende angekommen
          current= current->next;
        return *this;
      }
/*
      iterator operator++(int) { //Postfix
        iterator temp= *this;
       ++*this;
        return temp;
      }
*/
      bool operator!=(const iterator& x) const {
        return current != x.current;
      }

      int operator-(iterator it) {
        int count=0;
        while (it.current != current //noch nicht erreicht
          &&  it.current  != 0)      //noch nicht am Ende
        {
          ++count; //Erhoehen eines int-Wertes
          ++it;    //Fortschaltung eines Iterators
        }
        assert(current==it.current);
        //bei Ungleichheit ist *this von Iterator it mit ++ nicht erreichbar
        return count;
      }
    private:
      ListElem* current; //Zeiger auf aktuelles Element der Liste
  }; //iterator

  iterator begin() const { return iterator(firstElem); }
  iterator end()   const { return iterator(); }

}; //list


int main() {
  const int n=100;
  list<int> MyCont; // Container anlegen

  //Container mit geraden Zahlen fuellen, Reihenfolge beachten!
  for (int i=n; i>=0; i--) MyCont.push_front(2*i);

  int x = 0;
  while (x != -1)
  {
    cout << "Welche Zahl (Abbruch mit -1): ";
    cin >> x;
    if (x != -1)
    {
      list<int>::iterator position = find(MyCont.begin(),
                                          MyCont.end(), x);
      if ( position != MyCont.end() )
        cout << "gefunden an Pos. "
```

```
                << (   position - MyCont.begin()) << endl;
        else
            cout << x << " nicht gefunden! " << endl;
      }
   } //while
} //main
```

6.4 Exkurs typedef

Eine `typedef` in Klassen wird folgendermaßen angelegt und benutzt:

Listing 6.5: typedef in Klassen

```
#include <iostream>
using namespace std;

class A {
 public:
  typedef double real; //Typname real kann ausserhalb verwandt werden
};

int main() {
    A::real  x=1.3;
    cout << x << endl;
}
```

In `template`-Klassen erfolgt dies folgendermaßen:

Listing 6.6: typedef in Templates

```
#include <iostream>
using namespace std;

template <class T> class A {
 public:
   typedef T* zgr;
   class B {
    public:
      typedef long int longInt;
   };
};

int main() {
   double t=2.3;
   A<double>::zgr x(&t);
   cout << *x << endl;
   A<int>::B::longInt k= 7;
   cout << k << endl;
}
/*
2.3
7
*/
```

oder auch so:

Listing 6.7: typedef in Klasssen II

```
//typedef in Klassen

#include <iostream>
using namespace std;


int y;

struct A {
    typedef double XX;
    //double xx;
} a;

template <class T> void f(const T& t) {
    T::XX* y;
}

int main() {
    f(a);
}
```

6.5 Selbstimplementiertes find() benutzt mit einem STL-Container

Suche in einem Container für `int`-Werte nach einem einzulesenden `int`-Wert `x`.
Verwendet wird hier der STL-Container `vector`.

Listing 6.8: Selbstimplementiertes find() und STL-Container

```
#include <iostream>
#include <vector> //STL  !!!

using namespace std;

//neuer Typname IteratorType fuer Iteratoren auf Containerelemente
typedef vector<int>::iterator IteratorType; // !!!

//Algorithmus find() als  Template
template<class IteratorType>
IteratorType find( IteratorType begin, IteratorType end,
                   const int& value )
{
    while(begin != end  && value != *begin) ++begin;
    return begin;
}

int main() {
    const int n=100;
    vector<int> MyContainer(n);  // Container anlegen !!!

    //Container mit geraden Zahlen fuellen
    for (int i=0; i<n; i++) MyContainer[i]= 2*i;
```

```
25    int x = 0;
      while (x != -1)
      {
          cout << "Welche Zahl soll gesucht werden (Abbruch mit -1): ";
          cin >> x;
30        if (x != -1)
          {
              IteratorType position = find(MyContainer.begin(),        //!!!
                                           MyContainer.end(),   x); //!!!
              if (position != MyContainer.end   ())                    //!!!
35                cout << "gefunden an Pos. "                          //!!!
                     << (   position - MyContainer.begin()) << endl;   //!!!
              else
                 cout << x << " wurde nicht gefunden! " << endl;
          }
40    } //while
} //main
```

6.6 STL find() benutzt mit einem STL-Container

Suche in einem Container für `int`-Werte nach einem einzulesenden `int`-Wert x.
Verwendet wird auch hier der STL-Container `vector`.

Listing 6.9: STL find() benutzt mit einem STL-Container

```
#include <iostream>
#include <vector>
#include <algorithm> //!!!

5   using namespace std;

    //neuer Typname IteratorType fuer Zeiger (Iteratoren) auf Containerelem.
    typedef vector<int>::iterator IteratorType;

10  //Algorithmus find() als  Template entfaellt  !!!

    int main() {
       const int n=100;
       vector<int> MyContainer(n);  // Container anlegen
15
       //Container mit geraden Zahlen fuellen
       for (int i=0; i<n; i++) MyContainer[i]= 2*i;

       int x = 0;
20     while (x != -1)
       {
          cout << "Welche Zahl soll gesucht werden (Abbruch mit -1): ";
          cin >> x;
          if (x != -1)
25        {
              IteratorType position = find(MyContainer.begin(), //aus <algorithm>
```

```
                                        MyContainer.end(),   x);
        if (position != MyContainer.end())
            cout << "gefunden an Pos. "
                 << (  position - MyContainer.begin()) << endl;
        else
            cout << x << " wurde nicht gefunden! " << endl;
      }
    } //while
} //main
```

6.7 map

Der Container `map` ermöglicht den Zugriff und das Einfügen von Einträgen mittels Indexzugriff (`[]`):

Listing 6.10: Der map-Container

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

template <class S,class T> ostream& operator<<( ostream& os, map<S,T>& a ) {
   map<S,T>::const_iterator it=a.begin();
   while (it != a.end()) {
      os << (*it). first  << " " << (*it).second << endl;
     ++it;
   }
   cout << endl;
   return os;
}

int main() {
   map<double,string> m; //Schluessel sind Gleitkommazahlen
   m[1.3]= "eins.drei";
   m[4.2]= "vier.zwei";
   cout << m << endl;
   pair<double,string> x= *m.begin();
   m.erase(x. first ); //Erster Wert des Paares als Schluessel
   cout << m << endl;

   map<string,string> t; //Schluessel sind Zeichenketten
   t["Vogt"]=   "439-2010";
   t["Vogt"]+=  "  439-1312";
   t["Marx"]= "757338";
   cout << t << endl;

}
/*
1.3 eins.drei
4.2 vier.zwei

4.2 vier.zwei

Marx 757338
Vogt 439-2010  439-1312

*/
```

Im Folgenden geben wir einen Teil der Klassendefinition an, der die Elemente enthält, die zu den allgemeinen Containeranforderungen und zu den Anforderungen an reversible Container gehören:

Listing 6.11: Definition des map-Containers

```
template<class Kex, class T, class Compare=less<Key>,
    class Allocator=allocator<pair<const Key, T> > >
class map{
public:
    typedef pair<const Key, T> value_type;
    typedef Allocator allocotar_type;
    typedef implementationsspezifisch size_type;
    typedef implementationsspezifisch difference_type;
    typedef implementationsspezifisch iterator;
    typedef implementationsspezifisch const_iterator;
    typedef std:: reverse_iterator <iterator> reverese_iterator ;
    typedef std:: reverse_iterator <const_iterator> const_reverese_iterator ;
    typedef typename Allocator::reference reference;
    typedef typename Allocator::const_reference const_reference;
    typedef typename Allocator::pointer pointer;
    ...
};
```

6.8 Der Container set

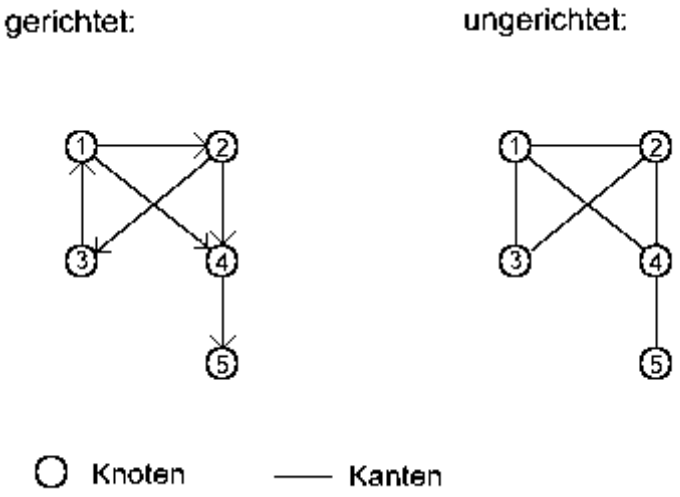
`set` ist ähnlich zu `map`, enthält aber nur Schlüssel und keine Werte. Den Operator `[]` gibt es nicht.

Listing 6.12: Der set-Container

```
#include <iostream>
#include <set>

using namespace std;

int main() {
   typedef set<int, greater<int> > SET;
   // greater ... liefert true, falls erstes Arg. > zweites Arg.
   SET s;
   s.insert(2);
   s.insert(1); //insert() hat den Ergebnistyp pair<iterator,bool>
   //Der Iterator zeigt auf den Eintrag zum Schluessel und als Wahrheitswert
   //wird true zurueckgeg., wenn Schluessel nicht vorhanden war, sonst false
   pair<SET::iterator,bool> pp= s.insert(1);
   s.erase(pp.first); //loesche Eintrag, auf den der Iterator zeigt
   if (s.insert(5).second)
      cout << "5 wurde eingefuegt\n";

   pair<SET::iterator, bool>p=s.insert(2);
   if (!p.second)
      cout << *p.first << " ist bereits vorhanden\n";
   cout << "s: " ;
   for (SET::iterator it = s.begin(); it!= s.end(); it++)
     cout << *it << "  " ;
}

/*
5 wurde eingefuegt
2 ist bereits vorhanden
s: 5  2  <== wird durch Funktor greater<int> bewirkt
*/
```

6.9 Graphen

Ein Graph besteht aus einer Menge von Knoten (vertices) und Kanten (edges); dabei unterscheidet man zwischen gerichteten und ungerichteten Graphen:



Graphen kann man auf verschiedene Weisen modellieren:

Adjazenzmatrix

Sei V={1,...,n} die Menge der Knoten und E die Menge der Kanten; wir wollen unter (i,j) die Kante zwischen den Knoten i und j verstehen, bzw. {i,j} wenn der Graph ungerichtet ist. Hiermit können wir nun zur Beschreibung des Graphen eine $n \times n$-Matrix $A = (a_{ij})_{i,j=1}^{n}$ benutzen, für die gilt:

$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

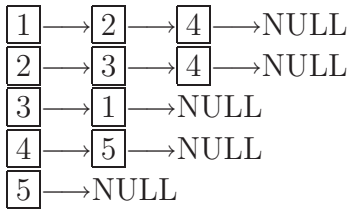
Der obige gerichtete Graph:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Beachte: Ungerichtete Graphen haben symmetrische Adjazenzmatrizen

Adjazenzliste

Eine weitere Möglichkeit sind Adjazenzlisten, in denen zu jedem Knoten eine Liste von Nachfolgerknoten existiert:



Zusätzlich könnten die Kanten auch noch mit sogenannten **Gewichten** bewertet sein.

Folgendes Beispiel zeigt eine (zu) einfache Datenstruktur für Adjazenzlisten eines Graphen:

Listing 6.13: Graphen

```
#include <iostream>
#include <list>
#include <map>

using namespace std;

class Graph {
 public:
   typedef map<int,list<int> >::iterator iterator;
   Graph(bool g) : gerichtet(g) {}
   void knoten(const int& k) {
      list <int> l;
      g[k]=l; //mit leerer Liste initialisieren
   }
   void kante(const int& k, const int& j) { //Kante von Knoten k zu Knoten j
      g[k].push_back(j);
      if (! gerichtet ) g[j].push_back(k); //Kante von Knoten j zum Knoten k
   }
    iterator begin(){ return g.begin(); }
    iterator end() { return g.end(); }
 private:
   bool gerichtet;
   map<int,list<int> > g;
};

void out(Graph& g) {
   cout << endl << "Knoten mit ihren Nachfolgern: ";
   Graph::iterator it= g.begin();
   while (it != g.end())
   {
       cout << endl << it->first << ": ";
       list <int>::iterator i= it->second.begin(); //Subliste durchlaufen
       while (i != it->second.end())
       {
           cout << *i << ", ";
         ++i;
```

```
        }
        ++it;
    }
    cout << endl;
}


int main() {
    Graph g(true); //gerichteter Graph
    g.knoten(1);
    g.kante(1,3);
    g.kante(1,5);
    g.knoten(2);
     list <int> l;
    out(g);
}
/*
Knoten mit ihren Nachfolgern:
 1: 3,  5,
2:
*/
```

Aufgabe:
Verbessere die Datenstruktur nach folgender Idee: Lege einen Vektor für die Knoten an und präge ihn so aus, dass er die Nachfolger und das Gewicht aufnehmen kann. Welche STL-Datenstrukturen wählen Sie?

6.10 Verwendung vorgegebener Templates für Funktionsobjekte

Funktionsobjekte sind

1. Objekte einer Klasse, für die `operator()` überladen ist
2. Pointer auf Funktionen

Listing 6.14: Funktionsobjekt

```
#include <iostream>
#include <functional> //Schablonen zur Generierung von Funktionsobjekten

using namespace std;

int main() {
    cout << less<int>()(2,3) << endl;
    cout << less<int>()(3,3) << endl;
    cout << less_equal<int>()(3,3) << endl;
    cout << bind2nd(less<int>(),20)(10) << endl;
    cout << bind2nd(less<double>(),3.5)(6.1) << endl;
}
/*
1
0
1
1
0
*/
```

Erläuterungen zum Beispielprogramm:

`bind2nd` ist eine Projektion: Sie macht aus einem zweistelligen Funktionsobjekt ein einstelliges, indem der zweite Operand an einen festen Wert gebunden wird. Mit der Projektion `bind1st` kann ein fester Wert an den ersten Operanden gebunden werden.

Die STL unterstützt ein- und zweistellige Funktionsobjekte.

Allgemeine Form:

```
result_type operator() (argument_type) bzw.
result_type operator() (first_argument_type, second_argument_type)
```

Einige STL-Funktionen wie z.B. `not1` und `not2` benutzen obige Typbezeichner. Funkionsobjekte müssen diese per `typedef` zur Verfügung stellen. Deshalb gibt es Basisklassen `unary_function` und `binary_function`.

Beispiel:

```
template <class Arg, class Resultat>
struct unary_function {
   typedef Arg argument_type;
   typedef Result result_type;
};
```

Da `less` ein zweistelliges Funktionsobjekt ist, wird es von `binary_function` abgeleitet:

```
template <class T>
struct less : public binary_function<T,T,bool> {
   bool operator()(const T& x, const T& y) const { return x < y; }
};
```

Es stehen dann automatisch die entsprechenden Typen (z.B. `firstargument_type`) zur Verfügung.

Auch selbstdefinierte Funktionsobjekte sollten entsprechend abgeleitet werden. Sie können dann mit Algorithmen der STL (z.B. mit `not1`) verwendet werden.

Beispiel: Ein Funktionsobjekt `Bereich`, das prüft, ob ein Wert innerhalb eines vorgebbaren Bereiches liegt.

```
template <class T>
class Bereich: public unary_function<T,bool> {
 public:
   Bereich(const T&u, const T& o): unten(u), oben(o) {}
   bool operator()(const T& x) const { return unten <= x && x <= oben; }
 private:
   const T unten, oben;
};
```

Nach Ausführung des folgenden Programmfragmentes hat `p` den Wert `f+5`.

```
int f []= {0,1, 4, 16, 25, 36};
int* z  find_if (f, f+10, Bereich<int>(20,30));
//es wird ein Iterator auf das erste Element in f mit Wert zwischen
//20 und 30 zurueckgegeben
```

In der STL heißen Funktionsobjekte, die andere Funktionsobjekte als Argument übernehmen und neue Funktionsobjekte daraus zusammensetzen, **Adapter**. Die STL kommt mit nur einem Satz zweistelliger Funktionsobjekte und den Adaptern zur Reduktion auf einstellige Funktionsobjekte aus.

Zur Definition von `bind2nd`:

```
template <class Operation>
class binder2nd
 : public unary_function<typename Operation::first_argument_type,
     typename Operation::result_type> {
 public:
```

```
    binder2nd(const Operation& o,
        const typename Operation::second_argument_type& v)
      :  op(o), value(v) {}
    typename Operation::result_type operator()
      ( const typename Operation::first_argument_type& x) const
      {  return op(x, value); }
  protected:
    Operation op;
    typename Operation::second_argument_type value;
};
```

Der Konstruktor initialisiert `op` mit `o` und `value` mit `v`. Der Funktionsaufrufoperator liefert `op(x,value)`.

```
template <class Operation, class T> inline
 binder2nd<Operation> bind2nd(const Operation& op, const T& x)
 { return binder2nd<Operation>(op, Operation::second_argument_type(x)); }
```

6.11 Arithmetische, logische und Vergleichsoperationen

Für alle arithmetischen und logischen Operationen sowie sämtliche Vergleichsoperationen von `C++` gibt es entsprechende Funktionsobjekte. Mit Hilfe des Standardkonstruktors können jeweils Objekte erzeugt werden, für die später der `operator()` aufrufbar ist. Beim Aufruf ist für einstellige Funktionobjekte ein Argument und für zweistellige sind zwei Argumente zu übergeben. Die folgende Tabelle listet in der ersten Spalte die Namen der Funktionsobjekte, in der zweiten die Anzahl der Argumente, in der dritten Spalte die ausgeführte Operation und in der vierten Spalte den Rückgabetyp auf.

Funktionsobjekt	**Argumente**	**Operation**	**Rückgabewert**
plus	2	x+y	T
minus	2	x-y	T
multiplies	2	x*y	T
divides	2	x/y	T
modulus	2	x%y	T
negate	1	-x	T
equal_to	2	x==y	`bool`
not_equal_to	2	x!=y	`bool`
greater	2	x>y	`bool`
greater_equal	2	x>=y	`bool`
less	2	x<y	`bool`
less_equal	2	x<=y	`bool`
logical_and	2	x&&y	`bool`
logical_or	2	x||y	`bool`
logical_not	1	!x	`bool`

Die in der Tabelle aufgeführten Funktionsobjekte sind Template-Klassen mit einem Typparameter. Bei zweistelligen Funktionsobjekten sind demnach beide Parameter vom selben Typ. Funktionsobjekte, die den Rückgabewert `bool` besitzen, werden in Anlehnung an die Aussagen der Prädikatenlogik auch als Prädikate (engl. `predicate`) bezeichnet. Ihr Aufrufoperator sollte als `const` Elementfunktion definiert werden.

Listing 6.15: Arithmetische, logische und Vergleichsoperationen

```
template<class T> struct plus : binary_function <T,T,T>{
    T operator()(constT& x, const T& y) const { return x+y;}
};

...

template<class T> struct less : binary_function <T,T,bool>{
    bool operator()(constT& x, const T& y) const { return x<y;}
};

...

template<class T> struct logical_and : binary_function <T,T,bool>{
    bool operator()(constT& x, const T& y) const { return x&&y;}
};
```

6.12 Funktionsobjekte – ein Negierer

Listing 6.16: Funktionsobjekte – ein Negierer

```
#include <iostream>
#include <algorithm>  //globale Algorithmen
#include <functional> //Funktionsobjekte wie less, greater, bind2nd

using namespace std;

int main() {
   int f []= {0,2,3,9};  //gewoehnliches C Array
   int n= sizeof f / sizeof *f;
   cout << "Anz. Feldelemente: " << n << endl;

   //suche erste ungerade Zahl (beachte: ungerade Zahl mod 2 = 1, also true):
   int* z= find_if(f, f+n, bind2nd(modulus<int>(),2));
   cout << *z << endl;
   //suche erste gerade Zahl:
   z= find_if( f, f+n, not1( bind2nd(modulus<int>(),2) ) );
   cout << *z << endl;
   //ausfuehrlicher:
   z= find_if( f, f+n, unary_negate<binder2nd<modulus<int> > >
      ( bind2nd(modulus<int>(),2)) );
   cout << *z << endl;
}
/*
Anz. Feldelemente: 4
3
0
0
*/
```

Es muss das Ergebnis des einstelligen Funktionsobjektes `bind2nd(modulus<int>(),2)` negiert werden. Dazu dient

```
template <class Predicate>
 class unary_negate
    : public unary_function<typename Predicate::argument_type,bool>{
  public:
     explicit  unary_negate(const Predicate& pred) : p(pred){}
    bool operator()(const typename Predicate::argument_type& x) const
   {  return !p(x); }
  protected:
    Predicate p;
 };
```

und schließlich zur Vereinfachung des Aufrufes (Template-Funktion, deren Ergebnis ein Funktionsobjekt ist)

```
template <class Predicate> inline
 unary_negate<Predicate> not1(const Predicate& pred)
 { return unary_negate<Predicate>(pred); }
```

Für zweistellige Funktionsobjekte gibt es `binary_negate` und `not2`.

6.13 Container, Iteratoren, Algorithmen und Funktionsobjekte

Listing 6.17: Algorithmen, Iteratoren, Funktionsobjekte und Container der STL

```
#include <iostream>
#include <vector>     //Container
#include <algorithm>  //globale Algorithmen wie find_if, copy
#include <functional> //Funktionsobjekte wie less, greater

using namespace std;

//Ausgabe eines vector-Containers
template <class T>
 void out(const vector<T>& v)
 { copy (v.begin(), v.end(), ostream_iterator<T>(cout, " "));
   cout << endl;
 }

int main() {
   int f[]= {0,1,-4,9}; //gewoehnliches C Array
   int n= sizeof f / sizeof *f;
   int* z= find(f, f+n, 5); //suche erstes Auftreten des Elements 5 in [f, f+n)
   if (z != f+n)
      cout << "5 nicht gefunden! " << endl;

   vector<int> v(f, f+n); //generiere vector-Container v und initialisiere
                          //diesen mit den im Array f abgespeicherten Werten
   vector<int>::iterator  //Typ des Iterators i
     i= find_if( v.begin(), v.end(), bind2nd(greater<int>(), 3) );
     //suche erstes Element in v dessen Wert > 3 ist und
     //gebe einen Iterator auf dieses Element zurueck
   if ( i != v.end() )
      cout << *i << " ist der erste Wert > 3 " << endl;
   i= find_if( v.begin(), v.end(), bind2nd(less<int>(), 0) );
   //gib Iterator auf erstes Element mit einem negativen Wert zurueck
   if ( i != v.end() )
      cout << *i << " ist der erste Wert < 0 " << endl;
   i= find(v.begin(), v.end(), 3);
   if ( i != v.end() )
      cout << "3 nicht gefunden! " << endl;

   sort(v.begin(), v.end());
   //aequivalent:  sort(v.begin(), v.end(), less<int>());
   //gib die im Container v gespeicherten Werte aus:
   out(v);
   sort(f, f+n, greater<int>()); //sortiere absteigend
   copy(f, f+n, ostream_iterator<int>(cout, " ")); //Ausgabe
   cout << endl;
   sort(v.begin(), v.end(), greater<int>()); //sortiere absteigend
   out(v); //Ausgabe
   sort(v.begin(), v.end(), not2(greater<int>()) ); //wieder aufsteigend
   out(v); //Ausgabe
}
/*
9 ist der erste Wert > 3
-4 ist der erste Wert < 0
-4 0 1 9
```

```
9 1 0 -4
9 1 0 -4
-4 0 1 9
*/
```

6.14 Die Algorithmen der STL

Zahlreiche Algorithmen sind in der STL vordefiniert. Ihr Standardeinsatz kann durch Funktionsobjekte modifiziert werden. Ein Algorithmus arbeitet i.a. nicht mit allen Iteratorkategorien zusammen. Die Zulässigkeit kann der Template-Parameterliste entnommen werden.

(Such-)Bereiche werden durch zwei Iteratoren abgegrenzt: [`first, last`), d.h. `first` gehört noch dazu, `last` hingegen nicht! Ist die Suche nicht erfolgreich, wird `last` zurückgegeben. Bereichesgrenzen müssen gültig sein (aus Effizienzgründen wird keine Überprüfung durchgeführt!).

Kategorien:

- nicht modifizierende Algorithmen: `find, for_each,` ...
- modifizierende Algorithmen: `copy, remove, replace, swap,` ...

Beispiel eines nichtmodifizierenden Algorithmus: `for_each`

```
template <class InputIterator, class Function>
 Function for_each(InputIterator  first , InputIterator  last , Function f);
```

Für jedes Element des Bereichs [`first, last`) wird die Funktion `f` aufgerufen. Typischer Rumpf für `for_each`:

```
  while (first!=last) f(*first++);
```

Der Rückgabewert von `f` wird hier jeweils ignoriert. f muss mit Argument des Typs `InputIterator::value_type` aufrufbar sein.

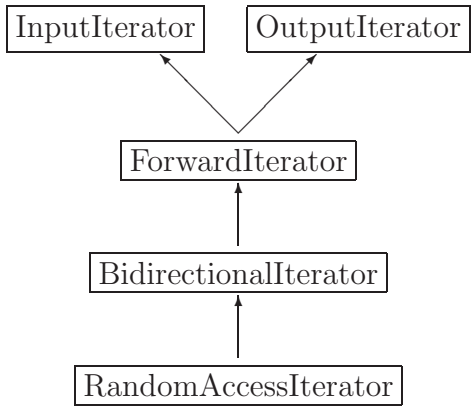
Beispiel: Die Summe von Feldelementen

Listing 6.18: Algorithmen der STL

```
#include <algorithm>
#include <functional>
#include <vector>
#include <iostream>

using namespace std;

template <class T> class Summe: unary_function<T,void>{
 public:
   Summe(const T& t=T()) : sum(t) {}
   void operator()(const T& t) { sum += t; }
   void out(void) const { cout << endl << sum << endl; }
 private:
   T sum;
};

void gerade_oder_ungerade(int i) { cout << ( (i%2==0)?"g ":"u " ); }

int main() {
  const int f []={5,2,3,7,6,4};
  for_each(f+1, f+5, gerade_oder_ungerade);
  Summe<int> s=for_each(f+2,f+4,Summe<int>());
  s.out();

  vector<int> v(f+0,f+6);
  for_each(v.begin(), v.end(), gerade_oder_ungerade);
  vector<int>::iterator i = v.begin();
  //Erg. von for_each ist ein Funktionsobj. fuer das Memberfunk. out ex.
  for_each(i+1,i+4,Summe<int>()).out();
}
/*
g u u g
10
u g u u g g
12
*/
```

Ein Algorithmus arbeitet im Allgemeinen nicht mit Iteratoren aller Iteratorklassen korrekt zusammen. Welche Iteratorkategorien unterstützt werden, ist anhand der Bezeichner in der Template-Parameterliste abzulesen. Dort wird jeweils die Iteratorkategorie mit den geringsten Anforderungen angegeben, für die der Algorithmus noch einsetzbar ist.

Übersicht zu den Iteratorklassen:



Wenn z.B. der Bezeichner ForwardIterator verwendet wird, heißt das, dass der Algorithmus mit Forward-, Bidirectional-, sowie Random-Access-Iteratoren korrekt arbeitet, aber nicht mit Input- und Output-Iteratoren.

6.14.1 Nichtmodifizierende Algorithmen

adjacent_find sucht innerhalb eines Bereichs nach aufeinander folgenden gleichen Elementen.

count und count_if zählt, wie oft ein bestimmtes Element in einem Bereich enthalten ist bzw. wie viele Elemente eine vorgegebene Bedingung erfüllen.

equal prüft, ob die Elemente zweier Bereiche paarweise gleich sind.

find und find_if sucht nach dem ersten Element, das einen bestimmten Wert besitzt bzw. eine vorgegebene Bedingung erfüllt.

find_end liefert das letzte Auftreten der Elemente eines Bereichs innerhalb eines anderen.

find_first_of liefert das erste Element eines Bereichs, das auch im Suchbereich vorhanden ist.

for_each führt eine Funktion für alle Elemente eines Bereichs aus.

mismatch liefert das erste Wertepaar, bei dem sich zwei Bereiche unterscheiden.

search liefert das erste Auftreten der Elemente eines Bereichs innerhalb eines anderen.

search_n liefert das erste Auftreten von n gleichen Elementen innerhalb eines Bereichs.

6.14.2 Modifizierende Algorithmen

copy kopiert die Elemente eines Bereichs vom ersten bis zum letzten Element in einen anderen Bereich.

copy_backward kopiert die Elemente eines Bereichs im Gegensatz zu copy vom letzten bis zum ersten Element in einen anderen Bereich.

fill und fill_n füllen einen Bereich mit Elementen eines bestimmten Werts.

...

6.14.3 Sortieren und ähnliche Operationen

nth_element verteilt die Elemente eines Bereichs, so dass die Elemente im vorderen Bereich kleiner oder gleich und die im hinteren größer oder gleich dem n-ten Element sind.

partial_sort berücksichtigt alle Elemente einer Bereichs bei der Sortierung, bringt aber nur die n kleinsten in eine sortierte Reihenfolge.

partial_sort_copy kopiert nur die n kleinsten Elemente eines Bereichs in einen anderen.

sort sortiert die Elemente eines Bereichs.

stable_sort sortiert die Elemente eines Bereichs, wobei die relative Reihenfolge gleicher Elemente erhalten bleibt.

6.14.4 Binäre Suchalgorithmen

binary_search prüft, ob ein gesuchtes Element in einem Bereich enthalten ist.

equal_range liefert ein Iteratorpaar entsprechend lower_bound und upper_bound.

lower_bound liefert die erste Position, an der ein Element in einen Bereich eingefügt werden kann, ohne die Sortierung der Elemente des Bereichs zu verletzen.

upper_bound liefert die letzte Position, an der ein Element in einen Bereich eingefügt werden kann, ohne die Sortierung der Elemente des Bereichs zu verletzen.

6.14.5 Mischalgorithmen

inplace_merge mischt die Elemente zweier aufeinander folgender, sortierter Teilbereiche, so dass ein sortierter Bereich ensteht.

merge kopiert die Elemente zweier sortierter Bereiche in einen dritten Bereich, dessen Elemente wieder sortiert sind.

6.14.6 Mengenalgorithmen für sortierte Bereiche

includes prüft, ob alle Elemente eines Bereichs in einem anderen enthalten sind, d.h. eine Teilmenge bilden.

set_difference bildet die Differenzmenge der Elemente zweier Bereiche, d.h. es werden die Elemente in den Ergebnisbereich aufgenommen, die im ersten aber nicht im zweiten Bereich enthalten sind.

set_intersection bildet die Schnittmenge der Elemente zweier Bereiche, d.h. es werden die Elemente in den Ergebnisbereich aufgenommen, die in beiden Bereichen enthalten sind.

set_symmetric_difference kopiert die Elemente zweier Bereiche, die nur in einem der beiden Bereiche enthalten sind, in einen Ergebnisbereich.

Anhang A

C-XSC (e*X*tended *S*cientific *C*omputing)

`C-XSC` ist eine `C++`-Klassenbibliothek für die Bereitstellung und Entwicklung numerischer Algorithmen mit automatischer Ergebnisverfikation.

Für die exakten Lösungskomponenten des Problems werden sichere (Gleitkomma-) Unter- und Oberschranken berechnet.

Beachte

- Gleitkommaalgorithmen können beliebig falsche Ergebnisse liefern!
- Die Verifikationsalgorithmen liefern entweder (enge) Einschließungen oder eine Meldung, dass solche nicht berechnet werden konnten

Beispiel A.1
Betrachten wir ein lineares Gleichungssystem

$$Ax = b$$

mit

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ b &= \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ x &= \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \end{aligned}$$

Gesucht sei x. Sei speziell: $b = e_1$ und

$$a_{11} = 64919421 \qquad a_{12} = -159018721$$
$$a_{21} = 4186952,5 \qquad a_{22} = -102558961$$

Wir berechnen mittels der Cramerschen Regel die Lösungen:

$$x_1 = \frac{a_{22}}{\det(A)} \qquad x_2 = \frac{-a_{21}}{\det(A)}$$

Auf dem Computer: $\widetilde{x_1} = 102558961.0$, $\widetilde{x_2} = 41869520.5$

Richtig wäre: $x_1 = 205117922$, $x_2 = 83739041$

Man sieht wie falsch Ergebnisse sein können, die der Computer mittels Gleitkommaarithmetik ermittelt.

Abhilfe:
Benötigt wird u.a. eine Mengenarithmetik $(M, \cdot)$, wobei M eine Menge und $\cdot$ eine der arithmetischen Operationen ist

$$\cdot : M \times M \quad \longrightarrow \quad M$$

z.B. $(M, \cdot) = (\mathbb{R}, +)$ oder $(M, \cdot) = (\mathcal{P}(M), \cdot_{\mathcal{P}})$ wobei

$$X \cdot_{\mathcal{P}} Y \quad := \quad \{x \cdot y |\ x \in X,\ y \in Y\}$$

Beachte:

1. $X \cdot_{\mathcal{P}} Y$ ist i.a. nicht auf dem Computer berechenbar (unendlich viele Operationen wären notwendig), wie dann?
2. Mengen sollen auf dem Computer einfach darstellbar sein.

Deshalb Intervallrechnung:
Einschränkung der zugelassenen Mengen auf Intervalle

$$x = [x] = [\underline{x}, \overline{x}] := \{x \in \mathbb{R} |\ \underline{x} \leq x \leq \overline{x}\}$$

Wir bezeichnen mit $\mathbb{IR}$ die Menge aller abgeschlossenen Intervalle.
Ist also $[a] \in \mathbb{IR}$, dann gibt es eine Darstellung

$$[a] = [\underline{a}, \overline{a}].$$

Definition A.2
Es sei $\emptyset \neq S \subset \mathbb{R}$, dann heißt

$$hull(S) = []S := [\inf(S), \sup(S)]$$

die Intervallhülle von S.

Beispiel A.3
Es sei

$$M := \left\{ \frac{1}{n} \in \mathbb{R} |\ n \in \mathbb{N} \right\}$$

dann folgt

$$[]M := [0, 1]$$

Bemerkung A.4
Es gilt (ohne Beweis)

$$[]S := \bigcap_{S \subset [a]} [a]$$

A.1 Operationen in IR

Definition A.5
Es seien $[a], [b] \in \text{IR}$ und $\circ \in \{+, -, \cdot, /\}$, dann definieren wir

$$\begin{aligned} [a] \circ [b] &:= []\{x \circ y |\ x \in [a],\ y \in [b]\} \\ &= []\{x \circ y |\ \underline{a} \leq x \leq \overline{a},\ \underline{b} \leq y \leq \overline{b}\} \end{aligned}$$

Satz A.6
Für $[a],\ [b] \in \mathbb{IR}$ und $\circ \in \{+, -, \cdot, /\}$ gilt

$$\begin{aligned} [a] \circ [b] &= []\{\underline{a} \circ \underline{b}, \underline{a} \circ \overline{b}, \overline{a} \circ \underline{b}, \overline{a} \circ \overline{b}\} =: []M \\ &= [\min(M), \max(M)] \end{aligned}$$

Es sind also nur vier reelle Operationen notwendig, um $[a] \circ [b]$ berechnen zu können!

Bemerkung A.7
In $\mathbb{IR}$ gilt $[a] \circ [b] = [a] \cdot_{\mathcal{P}} [b]$ (höherdimensional nicht mehr richtig!).

Beispiel A.8
Es ist

$$[\ 2{,}3\ ]+[\ 1{,}5\ ]=[3{,}8] \qquad [2{,}3] \cdot [1,5] = [2,15]$$
$$[\ 2{,}3\ ]-[\ 1{,}5\ ]=[\ -3{,}2\ ] \qquad [\ 2{,}3\ ]/[\ 1{,}5\ ] = \left[\frac{2}{5}, 3\right]$$

Die Intervalloperationen (mit Außenrundung) stellt `C-XSC` zur Verfügung.

Satz A.9
Ersetzt man in einem reellen Ausdruck alle Operationen durch reelle Intervalloperationen und interpretiert man reelle Größen als Punktintervalle, so ergibt sich ein Ergebnisintervall, welches den exakten Wert des Ausdrucks einschließt.

Ist $f(x)$ ein reellwertiger Ausdruck in x und ersetzt man dann jedes Auftreten von x durch einen Ausdruck der Form [x]=[$\underline{x}$,$\overline{x}$], so ergibt sich eine intervallmäßige Auswertung von f.
Es gilt dann: $f(x) \in f([\underline{x}, \overline{x}])$ für alle $x \in [x]$. Der Wertebereich $W_f[x]$ von f über $[x]$ wird durch das Ergebnis der intervallmäßigen Auswertung $f([x])$ eingeschlossen.

Leider gilt: Die (naive) intervallmäßige Auswertung kann zu großen Überschätzungen des Wertebereichs führen.

Beispiel A.10
Es sei $f(x) = \dfrac{1}{1 - x + x^2}$ für $x \in \left[0, \dfrac{1}{2}\right]$, dann ist $f\left(\left[0, \dfrac{1}{2}\right]\right)) = \left[\dfrac{4}{5}, 2\right]$ aber $W_f[0, \dfrac{1}{2}] = [1, \frac{4}{3}]$.

Man sieht an diesem Beispiel, dass bereits eine einzige intervallmäßige Auswertung reichen kann, um zu zeigen, dass eine Funktion im betrachteten Bereich keine Nullstelle haben kann.

Satz A.11 (Inklusionseigenschaft)
Es seien $[a], ..., [d] \in \mathbb{IR}$, $\circ \in \{+, -, \cdot, /\}$ und f eine reellwertige Auswertung

1. Sind $[a] \subset [b]$ und $[c] \subset [d]$, dann folgt $[a] \circ [c] \subset [b] \circ [d]$
   Ist $a \in [a]$ und $b \in [b]$, dann ist $a \circ b \in [a] \circ [b]$.

2. Ist $[a] \subset [b] \subset D_f$, so ist $W_f[a] \subset f[a] \subset f[b]$

A.2 Intervallrechnung auf dem Computer

Sei S die Menge der Gleitkommazahlen, dann beachte, dass aus $x, y \in S$ nicht unbedingt auch $x \circ y \in S$ folgt.
Die zu einem reellen Wert r nächstliegende Gleitkommazahl mit Wert kleiner gleich r bezeichnen wir mit ∇r $(\in S)$ und diejenige mit Wert größer gleich r mit Δr $(\in S)$ (∇ und Δ sind gerichtete Rundungen).
Wir bezeichnen nun mit

$\mathbb{I}S$:= die Menge alle abgeschlossenen reellen Intervalle deren Schranken in S sind

Definition A.12
Die Maschinenintervalloperationen $\circledcirc$ mit $\circ \in \{+, -, \cdot, /\}$ definieren wir wie folgt:
Seien $[x], [y] \in \mathbb{I}S$ und $[w] := [x] \circ [y] = [\underline{w}, \overline{w}]$, dann ist

$$\begin{aligned} \circledcirc : \mathbb{I}S \times \mathbb{I}S &\rightarrow \mathbb{I}S \\ [\mathrm{x}] \circledcirc [\mathrm{y}] &\mapsto [\nabla w, \Delta w] \supset [\underline{w}, \overline{w}] \end{aligned}$$

Auf der Maschine werden also nur die Einschließungen für die exakten Intervallverknüpfungen berechnet.

A.3 Intervallmäßige Ausdrucksauswertung auf der Maschine

Ersetze die reellen Operationen durch Maschinenoperationen, auftretende Konstanten durch einschließende Maschinenintervalle. Gibt es eine intervallmäßige Auswertung, so gilt $W_f \subset$ maschinenmäßige Auswertung von f.

Bemerkung A.13
Es können Standardfunktionen (sin, cos, exp, $\cdots$) miteinbezogen werden.
Im Mehrdimensionalen stelle man sich die Intervallvektoren als „achsenparallele Quader“ vor. In `C-XSC` sind entsprechende Arithmetiken (bestmöglich) realisiert.

A.4 Automatische Differentiation

Ist $u: D \longrightarrow \mathbb{R}$ eine auf D stetig differenzierbare Funktion, dann wollen wir unter

$$\texttt{U} \;=\; (u_0, u_1)$$

$u_0 = u(x)$ und $u_1 = u'(x)$ verstehen.

Beispiel A.14
Ist $u(x) = c$, so ist $\texttt{U} = (c, 0)$, ist $u(x) = x$, so $\texttt{U} = (x, 1)$.

Gehört $\texttt{U}=(u_0, u_1)$ zu $u(x)$ und $\texttt{V}=(v_0, v_1)$ zu $v(x)$ (an einer festen Stelle a), so gehört zu den Funktionen

1. $u(x) + v(x)$ das Tupel $\texttt{U+V}:=(u_0 + v_0, u_1 + v_1)$
2. $u(x) - v(x)$ das Tupel $\texttt{U-V}:=(u_0 - v_0, u_1 - v_1)$
3. $u(x) \cdot v(x)$ das Tupel $\texttt{U·V}:=(u_0 \cdot v_0, u_1 \cdot v_0 + v_1 \cdot u_0)$
4. $u(x)/v(x)$ das Tupel $\texttt{U/V}:=(u_0/v_0, (u_1 - \frac{u_0}{v_0} v_1)/v_1)$, falls $v_0 \neq 0, v_1 \neq 0$

Ist $f: D \longrightarrow \mathbb{R}$ ein Ausdruck aus Operationen $\circ \in \{+, -, \cdot, /\}$, so ergibt sich mit $x := (a, 1)$ das Paar

$$f(x) = f((a, 1)) = (f(a), f'(a)) \quad \forall\ a \in D$$

Beispiel A.15
Ist zum Beispiel $f(x) = \dfrac{x(4+x)}{3-x}$ gesucht sei f, f' in $[1, 2]$. Durch baumartige Auswertung erhält man hier als Ergebnis $\left(\left[\frac{5}{2}, 12 \right], \left[\frac{17}{4}, 20 \right] \right)$.

In diesem Fall stimmen das Ergebnis und $(W_f, W_{f'})$ sogar überein. Die Parameterarithmetik der automatischen Differentiation mit Intervalloperationen liefert die exakten Wertebereiche. In `C++` kann man im Gegensatz zu `Java` auch die Operatoren überladen und auf ähnliche Weise Taylorkoeffizienten berechnen (mittels Taylorarithmetik).

Bemerkung A.16
Verwendet man an Stelle der reellen Operationen Maschinenintervalloperationen, so erhält man simultan sichere Einschließungen für $f(a)$ und $f'(a)$ (bzw. von $W_f[\underline{x}, \overline{x}]$, $W_{f'}[\underline{x}, \overline{x}]$).

A.5 C-XSC

`C-XSC` stellt die folgenden Datentypen zur Verfügung:

```
real, interval, complex, cinterval
```

Ebenso sind auch Matrix/-Vektortypen vorimplementiert wie z.B.

```
rvector, ivector, cvector, icvector
```

Vordefiniert sind ebenfalls die arithmetischen Operationen mit maximaler Genauigkeit und elementare Funktionen ($\sin, \cos, \exp, \cdots$) insbesondere für Intervalle.
Ebenso steht ein Skalarprodukt und eine Bibliothek mir Problemlöseroutinen zur Verfügung.