Costruzione di Interfacce
Lezione 16
Qt

cignoni@isti.cnr.it
http://vcg.isti.cnr.it/~cignoni

QT

- Cos'e'
- e' un toolkit per lo sviluppo multipiattaforma di applicazioni con una GUI complessa
- QT e' composto da una class library e da alcuni tool per lo sviluppo rapido di applicazioni
  - designer
  - assistant
  - qmake ecc.

QT risorse

- L'assistant ovviamente,
- I forum di QT
  - http://lists.trolltech.com/qt-interest/
- Seguiremo i tutorial mostrati nell'assistant.
- All'inizio senza l'ide del .net usando solo il compilatore.

tutorial 1

```
/****************************************************************
**
** Qt tutorial 1
**
****************************************************************/

#include <qapplication.h>
#include <qpushbutton.h>


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

    QPushButton hello( "Hello world!", 0 );
    hello.resize( 100, 30 );

    a.setMainWidget( &hello );
    hello.show();
    return a.exec();
}
```


QApplication

- manages the GUI application's control flow and main settings.
  - It contains the main event loop
  - It also handles the application's initialization and finalization
  - For any GUI application that uses Qt, there is precisely one QApplication object, no matter whether the application has 0, 1, 2 or more windows at any time.
  - parse the command line
  - it knows about the application's windows.
  - ***it must be created before any other objects related to the user interface are created***

QPushButton

- e' un widget (uno dei tanti)
  - A widget is a user interface object that can process user input and draw graphics
  - programmer can change many minor properties of a widget e.g. for a button
    - text
    - an image
- nel nostro caso e' il widget principale dell'app.
  - in pratica e' il widget che quando viene chiuso l'app termina.
  - non e' obbligatorio avere un widget principale

Per compilare

- Da command line
  1. qmake –project
  2. qmake
  3. nmake
- funzionano SOLO se avete settato tutto l'environment bene.
  - environment qt
    - le dir dei binari di qt nel path
    - QTDIR alla dir base di qt
    - QMAKESPEC = win32-msvc.net
  - environment .net
    - fate partire vcvars32.bat che si trova \Program Files\Microsoft Visual Studio .NET\Vc7\bin\

qmake

- qmake –project
- genera un file .pro che dice come fare un makefile…

```
#####################################################
# Automatically generated by qmake (1.06c) Mon Nov 3
#####################################################
TEMPLATE = app
CONFIG -= moc
INCLUDEPATH += .

# Input
SOURCES += hello.cpp
```

- Di default usa tutti i *.cpp, *.h, *.hpp, *.c, *ui, … che trova nella directory corrente

qmake

- qmake –makefile (o semplicemente qmake)
- genera un makefile a partire dal .pro che trova nella dir corrente e fatto per l'ambiente di compilazione specificato nella variabile d'ambiente qmakespec
  - win32-msvc.net
  - win32-borland
  - … guardate nella dir QTDIR/mkspecs

make

- nmake (che e' l'equivalente ms del make classico)
- interpreta il makefile e genera l'eseguibile
- warning ufficialmente ignorabili:

```
LINK : warning LNK4199: /DELAYLOAD:comdlg32.dll ignored; no imports found
from comdlg32.dll
   LINK : warning LNK4199: /DELAYLOAD:oleaut32.dll ignored; no imports found from
      oleaut32.dll
   LINK : warning LNK4199: /DELAYLOAD:winmm.dll ignored; no imports found from winmm.dll
   LINK : warning LNK4199: /DELAYLOAD:wsock32.dll ignored; no imports found from
      wsock32.dll
   LINK : warning LNK4199: /DELAYLOAD:winspool.dll ignored; no imports found from
      winspool.dll
```

- *cercate nel forum "LINK warning LNK4199 DELAYLOAD"*

Esperimenti

- Provate a togliere
  - il resize
    - qual'e' la grandezza che sceglie?
  - setmainwidget
    - cosa succede quando si chiude l'app?

Tutorial 2

```
/****************************************************************
**
** Qt tutorial 2
**
****************************************************************/

#include <qapplication.h>
#include <qpushbutton.h>
#include <qfont.h>


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

    QPushButton quit( "Quit", 0 );
    quit.resize( 75, 30 );
    quit.setFont( QFont( "Times", 18, QFont::Bold ) );

    QObject::connect( &quit, SIGNAL(clicked()), &a, SLOT(quit()) );

    a.setMainWidget( &quit );
    quit.show();
    return a.exec();
}
```

QFont

- Nulla di speciale, solo l'oggetto che per gestire font.
- istanziato al momento dell'uso per il bottone sfruttando il compilatore
- notate che ogni classe di qt ha il suo include…

Connect

```
QObject::connect(
        &quit, SIGNAL(clicked()),
        &a, SLOT(quit()) );
```

- Fa si che quando si prema il bottone l'applicazione termini
- E' il meccanismo centrale di tutto QT

connect

```
QObject::connect( &quit, SIGNAL(clicked()), &a, SLOT(quit()) );
```

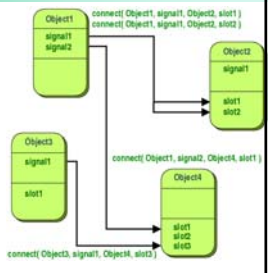
- it establishes a one-way connection between two Qt objects.
- Every Qt object can have both signals (to send messages) and slots (to receive messages).
- All widgets are Qt objects. They inherit QWidget which in turn inherits QObject.
- da un punto di vista c++

```
static bool QObject::connect (
    const QObject * sender, const char * signal,
    const QObject * sender, const char * member )
```

- nota: SIGNAL e SLOT sono macro che nascondono un meccanismo typesafe

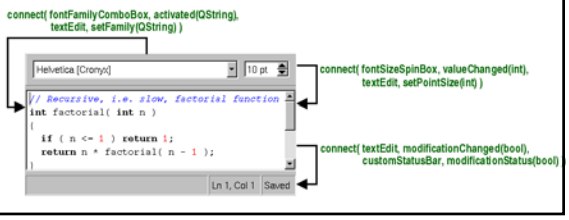
Signal and slots

- You can connect as many signals as you want to a single slot,
- A signal can be connected to many slots
- It is possible to connect a signal directly to another signal.
  - This will emit the second signal immediately whenever the first is emitted.



Signal and slots

- Signals are emitted by objects when they change their state in a way that may be interesting to the outside world.
- Slots can be used for receiving signals, but they are also normal member functions



Signal and slots

- The signals and slots mechanism is type safe
- the signature of a signal must match the signature of the receiving slot
- signals and slots can take any number of arguments of any type

Standard C++ vs QT style

```
class Foo
    {
    public:
        Foo();
        int value() const
                { return val; }
        void setValue( int );
    private:
        int val;
    };
```

```
class Foo : public QObject
    {
      Q_OBJECT
    public:
      Foo();
      int value() const
              { return val; }
    public slots:
      void setValue( int );
    signals:
      void valueChanged( int );
    private:
      int val;
    };
```


MOC

- Nota quello che si scrive non e' piu' C++, ma qualcosa di piu'.
- Occorre un particolare preprocessor chiamato moc (Meta Object Compiler) che trasforma il tutto in C++ standard
- tutto cio' e' nascosto da qmake o dall'integrazione di qt con .net

Tutorial 3

```
/****************************************************************
**
** Qt tutorial 3
**
****************************************************************/
#include <qapplication.h>
#include <qpushbutton.h>
#include <qfont.h>
#include <qvbox.h>

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

    QVBox box;
    box.resize( 200, 120 );

    QPushButton quit( "Quit", &box );
    quit.setFont( QFont( "Times", 18, QFont::Bold ) );

    QObject::connect( &quit, SIGNAL(clicked()), &a, SLOT(quit()) );

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

    return a.exec();
}
```


Contenitori

- Si aggiunge un contenitore di widget

```
QVBox box;
```

- All its child widgets will be placed vertically
- riferito nel costruttore del pushbutton

```
QVBox box;
box.resize( 200, 120 );
QPushButton quit( "Quit", &box );
```

- e' il contenitore il main widget
- e' del contenitore che si fa show
- provate a cambiare contenitore (QHBox) e ad aggiungere altri pulsanti

Tutorial 4

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qfont.h>

class MyWidget : public QWidget
{
public:
    MyWidget( QWidget *parent=0, const char *name=0 );
};

MyWidget::MyWidget( QWidget *parent, const char *name )     : QWidget( parent, name )
{
    setMinimumSize( 200, 120 );
    setMaximumSize( 200, 120 );

    QPushButton *quit = new QPushButton( "Quit", this, "quit" );
    quit->setGeometry( 62, 40, 75, 30 );
    quit->setFont( QFont( "Times", 18, QFont::Bold ) );

    connect( quit, SIGNAL(clicked()), qApp, SLOT(quit()) );
}

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

    MyWidget w;
    w.setGeometry( 100, 100, 200, 120 );
    a.setMainWidget( &w );
    w.show();
    return a.exec();
}
```


Subclassing

- Si nasconde il bottone ed il suo comportamento dentro una classe
- tutto fatto nel costruttore di MyWidget
- il bottone e' una variabile temporanea
  - qt delete all the qt objects figli di un altro widget.
- nel nostro caso il pushbutton quit viene deallocato quando sparisce myWidget
- qApp alias all'applicazione per evitare variabili globali dell'utente.

Tutorial 6 (1)

```
/****************************************************************
**
** Qt tutorial 6
**
****************************************************************/

#include <qapplication.h>
#include <qpushbutton.h>
#include <qslider.h>
#include <qlcdnumber.h>
#include <qfont.h>
#include <qvbox.h>
#include <qgrid.h>

class LCDRange : public QVBox
{
public:
    LCDRange( QWidget *parent=0, const char *name=0 );
};

LCDRange::LCDRange( QWidget *parent, const char *name )
        : QVBox( parent, name )
{
    QLCDNumber *lcd  = new QLCDNumber( 2, this, "lcd"  );
    QSlider * slider = new QSlider( Horizontal, this, "slider" );
    slider->setRange( 0, 99 );
    slider->setValue( 0 );
    connect( slider, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)) );
}
```


Tutorial 6 (2)

```
class MyWidget : public QVBox
{
public:
    MyWidget( QWidget *parent=0, const char *name=0 );
};


MyWidget::MyWidget( QWidget *parent, const char *name )
    : QVBox( parent, name )
{
    QPushButton *quit = new QPushButton( "Quit", this, "quit" );
    quit->setFont( QFont( "Times", 18, QFont::Bold ) );

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

    QGrid *grid = new QGrid( 4, this );

    for( int r = 0 ; r < 4 ; r++ )
        for( int c = 0 ; c < 4 ; c++ )
            (void)new LCDRange( grid );
}

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

    MyWidget w;
    a.setMainWidget( &w );
    w.show();
    return a.exec();
}
```


LCDRange

- Nuovo widget fatto da un numero e uno slider
  - derivato da QVBox
  - slider e numero connessi da
  - `connect( slider, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)) );`
  - Non ha un' api
    - dall'esterno non posso sapere nulla!
  - Il resto del codice e' un altro widget che fatto da tanti lcdrange.

Tutorial 7

- Strutturiamo il codice
  - **lcdrange.h** contains the LCDRange class definition.
  - **lcdrange.cpp** contains the LCDRange implementation.
  - **main.cpp** contains MyWidget and main.
- Facciamo diventare LCDrange una classe con signals e slot

lcdrange.h

```
/****************************************************************
**
** Definition of LCDRange class, Qt tutorial 7
**
****************************************************************/
#ifndef LCDRANGE_H
#define LCDRANGE_H

#include <qvbox.h>

class QSlider;

class LCDRange : public QVBox
{
    Q_OBJECT
public:
    LCDRange( QWidget *parent=0, const char *name=0 );

    int value() const;

public slots:
    void setValue( int );

signals:
    void valueChanged( int );

private:
    QSlider *slider;
};

#endif // LCDRANGE_H
```


lcdrange.h

- LCDrange ha uno stato e un'api
- la macro Q_OBJECT
- le keyword non c++ utilizzate prima di alcuni membri della classe
  - tradotto in c++ dal MOC
  - signals
    - seguono le regole di accesso delle protected
    - l'implementazione non c'e'!
    - il codice e' qualcosa che riguarda qt non la nostra classe;
    - la nostra classe deve solo dire QUANDO emette segnali
  - slots
    - sono normali funzioni membro della classe
    - e' la nostra classe a sapere cosa fare con il valore che gli arriva.

lcdrange.cpp

```
/*****************************************************************
**
** Implementation of LCDRange class, Qt tutorial 7
**
*****************************************************************/

#include "lcdrange.h"

#include <qslider.h>
#include <qlcdnumber.h>

LCDRange::LCDRange( QWidget *parent, const char *name )  : QVBox( parent, name )
{
    QLCDNumber *lcd  = new QLCDNumber( 2, this, "lcd"  );
    slider = new QSlider( Horizontal, this, "slider" );
    slider->setRange( 0, 99 );
    slider->setValue( 0 );
    connect( slider, SIGNAL(valueChanged(int)),
             lcd, SLOT(display(int)) );
    connect( slider, SIGNAL(valueChanged(int)),
             SIGNAL(valueChanged(int)) );
}

int  LCDRange::value() const {     return slider->value();   }
void LCDRange::setValue( int value )  {     slider->setValue( value ); }
```


lcdrange.cpp

- nel costruttore due connect
  - la prima per legare assieme slider e numero come nel tutorial precedente
  - la seconda per dire quando la nostra classe emette un segnale
    - quando si muove lo slider anche la nostra classe emette un segnale (connect tra due signal)
  - setValue(int) e' il nostro slot,
    - semplicemente setta il valore in arrivo nello slider (che poi fara' una signal, che aggiorna il numero)
  - Entrambe le connect partono dallo stesso segnale
    - NESSUNA garanzia sull'ordine di esecuzione

main.cpp (1)

```
#include <qapplication.h>
#include <qpushbutton.h>
#include <qlcdnumber.h>
#include <qfont.h>
#include <qvbox.h>
#include <qgrid.h>
#include "lcdrange.h"

class MyWidget : public QVBox
{ public: MyWidget( QWidget *parent=0, const char *name=0 );};

MyWidget::MyWidget( QWidget *parent, const char *name ) : QVBox( parent, name )
{
    QPushButton *quit = new QPushButton( "Quit", this, "quit" );
    quit->setFont( QFont( "Times", 18, QFont::Bold ) );
    connect( quit, SIGNAL(clicked()), qApp, SLOT(quit()) );
    QGrid *grid = new QGrid( 4, this );
    LCDRange *previous = 0;
    for( int r = 0 ; r < 4 ; r++ ) {
        for( int c = 0 ; c < 4 ; c++ ) {
            LCDRange* lr = new LCDRange( grid );
            if ( previous )
                connect( lr, SIGNAL(valueChanged(int)),
                         previous, SLOT(setValue(int)) );
            previous = lr;
        }
    }
}
```


main.cpp (2)

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

    MyWidget w;
    a.setMainWidget( &w );
    w.show();
    return a.exec();
}
```


main.cpp

- notare il concatenamento in cascata dei segnali

```
if ( previous )
        connect( lr, SIGNAL(valueChanged(int)),
                 previous, SLOT(setValue(int)) );
```

- muovendo uno slider cambiano tutti quelli precedenti…

Tutorial 8

- La prima classe che si ridisegna, 5 file
  - lcdrange.h, lcdrange.cpp
  - cannon.h, cannon.cpp
  - main.cpp
- lcdrange.h e lcdrange.cpp quasi uguali
  - aggiunto setRange

```
void LCDRange::setRange( int minVal, int maxVal )
    {
        if ( minVal < 0 || maxVal > 99 || minVal > maxVal ) {
          qWarning( "LCDRange::setRange(%d,%d)\n"
                    "\tRange must be 0..99\n"
                    "\tand minVal must not be greater than maxVal",
                    minVal, maxVal );
          return;
        }
        slider->setRange( minVal, maxVal );
    }
```

Tutorial 8

- notare l'uso di qWarning
  - stampa messaggi di errore e di warning su stderr (o al debugger)
  - simili qDebug e qFatal
  - l'output puo essere gestito in maniera diretta
    - qInstallMsgHandler

Cannon.h

```
#ifndef CANNON_H
#define CANNON_H
#include <qwidget.h>

class CannonField : public QWidget
{
    Q_OBJECT
public:
    CannonField( QWidget *parent=0, const char *name=0 );

    int angle() const { return ang; }
    QSizePolicy sizePolicy() const;

public slots:
    void setAngle( int degrees );

signals:
    void angleChanged( int );

protected:
    void paintEvent( QPaintEvent * );

private:
    int ang;
};

#endif // CANNON_H
```


Cannon.h

- la classe deve rappresentare un pezzo di interfaccia dove un cannone 2d spara.
- l'unico dato e' l'angolo del cannon con signal e slot circa la sua modifica
- gestione del paint event
- protected:
- void paintEvent(QPaintEvent *);

cannon.cpp (1)

```
#include "cannon.h"
#include <qpainter.h>


CannonField::CannonField( QWidget *parent, const char *name )
        : QWidget( parent, name )
{
    ang = 45;
    setPalette( QPalette( QColor( 250, 250, 200) ) );
}


void CannonField::setAngle( int degrees )
{
    if ( degrees < 5 )        degrees = 5;
    if ( degrees > 70 )        degrees = 70;
    if ( ang == degrees )        return;
    ang = degrees;
    repaint();
    emit angleChanged( ang );
}
```


cannon.cpp (2)

```
void CannonField::paintEvent( QPaintEvent * )
{
    QString s = "Angle = " + QString::number( ang );
    QPainter p( this );
    p.drawText( 200, 200, s );
}


QSizePolicy CannonField::sizePolicy() const
{
   return QSizePolicy( QSizePolicy::Expanding,QSizePolicy::Expanding );
}
```


cannon.cpp

- la setAngle emette un segnale
- notare la keyword non c++
  - emit angleChanged(ang)
- il repaint e' fatto tramite la classe Qpainter
  - The painter provides highly optimized functions to do most of the drawing GUI programs require.
  - QPainter can draw everything from simple lines to complex shapes like pies and aligned text and pixmaps
  - The typical use of a painter is:
    - Construct a painter.
    - Set a pen, a brush etc.
    - Draw.
    - Destroy the painter.

Riferimenti

- nell'assistant
  - tutorial #1
  - Signal and slots
  - moc
  - qmake
- il codice prendetelo direttamente dall'assistant