Université de la Rochelle
Licence d'Informatique - IHM (INFO132601)

TP 1 - *Introduction à la programmation Qt*

© B. Besserer, R. Péteri

Année universitaire 2016-2017

Ce premier TP sous Qt peut s'executer sous **linux**, **Mac OS** ou bien sous **Windows**. Dans tous les cas, il faut installer la bibliothèque Qt (`http://www.qt.io/`). Nous vous proposons de travailler sous linux.

A chaque fois que vous rencontrerez la coche suivante : ☑, appeler l'enseignant pour qu'il puisse valider de votre travail. Il est aussi très fortement conseillé de terminer pour la semaine suivante le TP si vous n'avez pu le faire durant la séance.

1 Programmation Orienté Objet et IHM : Qt

Pour vous faire la main et vous approprier les ordres de compilation pour construire les projets sous Qt, commençons par le traditionnel "hello world !", que nous completerons par 2 widgets supplémentaires.

1. Avec votre éditeur favori, saisissez le code suivant (et enregistrez sous le nom main.cpp) :

```
// main.cpp
#include <QApplication>
#include <QWidget>
#include <QLabel>

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

    QWidget window;
    QLabel* message = new QLabel("Hello, World!", &window);
    window.show();

    return app.exec();
  }
```

Remarquez la notion d'héritage (message est *fils* de window, et le fait d'afficher window affiche les *fils*. A la sortie du programme, window étant une variable locale, elle est détruite ainsi que les *fils*). Pour compiler / linker cette application, effectuez les opérations suivantes (dans un interpréteur de commande, en vous plaçant dans le répertoire qui contient `main.cpp`).

```
qmake-qt4 -project    # (qmake pour qt4) creates a .pro file saying, SOURCES += main.cpp
```

Ouvrez le fichier .pro ainsi généré pour vous faire une idée de contenu de celui-ci. Ensuite enchaînez les deux opérations suivantes :

```
qmake-qt4        #    ( qmake pour qt4) uses the .pro file to create a Makefile
make              # builds the application
```

Regardez le contenu de votre répertoire. Vous y trouverez eventuellement des fichiers moc_* (un par fichier définissant des slots), et les fichiers binaires (*.o et exécutable). Lancez l'application.
Si cette étape fonctionne, nous allons modifier ce code en ajoutant un gestionnaire de géométrie et deux widgets suplemmentaires :

```
#include <QLCDNumber>
#include <QLayout>
#include <QDial>
QDial *dial = new QDial();
QLCDNumber *lcd = new QLCDNumber(2); // nombre de digits
```

Le gestionnaire de géometrie sera du type QVBoxLayout. Il faut créer l'objet :

```
QVBoxLayout *layout = new QVBoxLayout();
```

puis ajouter au gestionnaire les élements *fils* :

```
layout->addWidget(message);
layout->addWidget(lcd);
...
```

Enfin affecter ce gestionnaire de géometrie à la fenêtre de l'application :

```
window.setLayout(layout);
```

A remarquer que dans ce cas, il n'est pas nécéssaire de créer l'objet message en passant la fenêtre parent en paramêtre. La hierarchie des widgets est assurée via le gestionnaire de géométrie.
Si l'application s'execute, on va connectez le signal du QDial `(valueChanged(int))` au slot du LCDNumber `(display(int))`. A remarquer qu'il faut utiliser la syntaxe `QObject::connect()`. Par contre, lorsque vous définissez votre propre classe Qt, vous ajouter une macro `Q_OBJECT` dans votre définition de classe, qui hérite alors de `QObject`. En conséquence, dans les méthodes de la classe, vous pouvez directement invoquer la methode `connect()`.
The Q_OBJECT macro is expanded by the preprocessor to declare several member functions that are implemented by the moc


1 Testez pour verifier si l'affichage reflète l'action sur le bouton rotatif.

```
#include <qapplication.h>


#include <QLCDNumber>
#include <QLabel>
#include <QVBoxLayout>  // ou bien <QLayout.h>
#include <QWidget>
#include <QDial>


int main(int argc, char* argv[])
{
// Create an application object.
QApplication app(argc, argv);

QWidget window;
QLCDNumber *lcd =new QLCDNumber(2);
QLabel *message = new QLabel("Hello");
QDial *dial = new QDial();
QVBoxLayout *layout = new QVBoxLayout();
layout->addWidget(message);
layout->addWidget(lcd);
layout->addWidget(dial);
window.setLayout(layout);

QObject::connect(dial,SIGNAL(valueChanged(int)),lcd,SLOT(display(int)));
window.show();

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

2. **Utilisation de Qt Creator**
Nous vous proposons maintenant de travailler en utilisant Qt Creator, la plateforme de développement Qt.
Qt Creator est un environnement de développement intégré C++ multiplateforme. Il intègre directement dans l'interface un débogueur, un outil de création d'interfaces graphiques ainsi que la documentation Qt, très complète (et donc à utiliser intensivement !). L'éditeur de texte intégré permet

l'autocomplétion ainsi que la coloration syntaxique.
Qt Creator utilise sous Linux le compilateur *gcc*.

(a) Effacez tous les fichiers *.o, .pro, *.moc, makefile de votre application précédente (bref, ne gardez que les *h et *cpp !).

(b) Lancez Qt Creator, et créez un nouveau projet (vide). En cliquant à droite, ajoutez au projet les fichiers $*.cpp$ et $*.h$ créés à l'exercice précédent.

(c) Enchainez la suite des commandes pour créer votre executable, puis lancer-le (le tout depuis Qt Creator evidement !)

(d) Découvrez l'aide intégrée de Qt Creator en tapant différentes classes de widgets Qt (`QPushButton`, `QLCDNumber` ...)

Remarque : dorénavant, en cliquant sur le fichier .pro, Qt Creator se lancera et ouvrira le projet associé.

2 Traiter explicitement les évènements X11 avec Qt

On utilisera dorénavant Qt Creator.

— Créer tout d'abord un nouveau projet Qt de type `Application Qt avec widgets`, avec une classe `MainWindow` ayant pour classe parent `QMainWindow` (suivre l'assistant). Exécuter le programme pour vérifier que la fenêtre principale s'affiche bien.

— Comme vu en TD2, Qt utilise sous Linux les mécanismes sous-jacents de X11 pour la gestion de l'IHM graphique. On peut récupérer les événements transmis à la fenêtre de l'application construite avec Qt. Pour cela, on dérive une classe de la classe de base `QApplication`, et l'on implémente la méthode `bool x11EventFilter (XEvent *e)`, ce qui donne dans le fichier `main.cpp` :

```
#include "mainwindow.h"
#include <QApplication>
#include<QDebug>
#include<X11/Xlib.h>

class XApplication: public QApplication
{
public:
XApplication (int & argc, char **argv): QApplication (argc , argv) { }
protected:
bool x11EventFilter (XEvent *e)
{
qDebug() << "X11 Event: " << e->type;
return QApplication::x11EventFilter(e); // appel de la méthode de la classe de base
}
};
```

En vous servant du TD, modifier votre programme :

1. pour afficher sur la sortie standard les coordonnées de la souris lors dans clic dans la fenêtre principale.
2. pour sortir du programme lorsque la touche `CONTROL` est appuyée **pendant** un clic souris (méthode : `qApp.Exit()`)

 2

Solution :

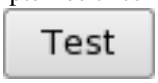
main.cpp

```
#include "mainwindow.h"
#include <QApplication>
#include<QDebug>
#include<X11/Xlib.h>

class XApplication: public QApplication

{
```

```
public:

    XApplication (int & argc, char **argv): QApplication (argc , argv) { }

protected:
    bool x11EventFilter (XEvent *e)

    {
        // qDebug() << "Event type:" << e->type;

        switch(e->type)
        {
         case ButtonPress:
            qDebug() << "MousePress Event! " << e->type;
            // Question 1:
            qDebug() << "Position du clic souris (x= " << e->xbutton.x <<
            ",y=" << e->xbutton.y<<")";

            // Question 2:
            if(e->xbutton.state&ControlMask)
            {
                qDebug() << "Exit!";
                qApp->exit();
            }
            break;

        default:
            break;
        }
        return QApplication::x11EventFilter(e);
    }
};


int main(int argc, char *argv[])
{
    XApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

3 Comptage de clics avec Qt

On souhaite créer un bouton de type PushButton disposant d'une fonctionnalité particulière : celle de compter les clics fait sur ce bouton depuis le lancement de l'application.



1. Commencez par écrire une fichier ButtonCount.h et un fichier ButtonCount.cpp qui implemente une classe ButtonCount, dérivée de QPushButton, et disposant d'une variable membre m_Count.
   On redefiniera le constructeur :
   `QPushButton ( const QString & text, QWidget * parent = 0 )`, en y ajoutant la mise à zero de la variable.

   ButtonCount.h

   ```
   #ifndef BUTTONCOUNT_H
   #define BUTTONCOUNT_H

   #include <QObject>
   #include <QPushButton>

   class ButtonCount : public QPushButton
   {
   Q_OBJECT
   public:
   ButtonCount( const QString & text, QWidget *parent = 0 );
   protected:
   ```

```
int m_Count;
};
#endif
```

ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
}
```

2. Maintenant, définissons un SLOT nommée `Increment()` (c'est à dire une méthode) permettant d'incrémenter la variable de comptage à chaque clic. Le classe QPushButton possède déjà un signal `void clicked()`, hérité de QAbstractButton. Afin d'exploiter ce signal, l'objet ButtonCount va connecter ce signal à son propre `slot Increment()`. La connection sera faite dans le constructeur. Ainsi, à chaque clic, la variable `m_Count` doit s'incrementer.

   Completer votre fichier ButtonCount.h et le fichier ButtonCount.cpp pour déclarer le slot, ecrire la methode correspondante et modifier le constructeur pour assurer la connection signal → slot.

ButtonCount.h

```
   #ifndef BUTTONCOUNT_H
    #define BUTTONCOUNT_H

#include <QObject>
#include <QPushButton>

class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
public slots:
void Increment(void);
};
#endif
```

ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
// connection du signal (existant pour la classe QPushButton) a notre slot
connect(this, SIGNAL(clicked()), this, SLOT(Increment()));
}

// un slot s'écrit comme une simple methode
void ButtonCount::Increment()
{
m_Count++;
}
```

3. Incrementer une variable sans pouvoir la consulter, ce n'est pas très interessant. Qt dispose de nombreux widgets permettant d'afficher des informations, comme par exemple le widget QLCDNumber.

The QLCDNumber widget displays a number with LCD-like digits. It can display a number in just about any size. It can display decimal, hexadecimal, octal or binary numbers. It is easy to connect to data sources using the display() slot, which is overloaded to take any of five argument types.
Parmis les types acceptés par le slot display(), nous avons :

```
void display ( const QString & s )
void display ( double num )
void display ( int num )
```

On peut donc envoyer vers cet objet une chaine de caractères, un nombre en virgule flottante ainsi qu'un nombre entier.

Revenons à notre objet ButtonCount. Nous allons ajouter à cet objet un signal. Par souci de cohérence avec d'autres contrôles de Qt, nous allons nommer notre signal valueChanged(int). A chaque increment de la variable m_Count, un signal valueChanged(int) sera émis, avec la valeur courante de la variable m_Count. Modifez votre code.

ButtonCount.h

```
#ifndef BUTTONCOUNT_H
#define BUTTONCOUNT_H

#include <QObject>
#include <QPushButton>

class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
signals:
  void valueChanged(int);
public slots:
void Increment(void);

};
#endif
```

ButtonCount.cpp

```
#include <QtGui>
#include "ButtonCount.h"

ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
connect(this, SIGNAL(clicked()), this, SLOT(Increment()));
}

void ButtonCount::Increment()
{
m_Count++;
// la ligne suivante assure un feedback sonore
QApplication::beep();
// emission du signal
emit(valueChanged(m_Count));
}
```

4. Soit les lignes suivantes dans un programme principal :

```
ButtonCount *button_c = new ButtonCount("Je compte mes clics");
  QLCDNumber *lcd = new QLCDNumber(2);  // ici on indique le nombre de digits
  lcd->setSegmentStyle(QLCDNumber::Filled);  // options d'affichage
```

Ajoutez la ligne permettant à l'afficheur LCD d'afficher le nombre de clics effectués sur le bouton.

```
connect(button_c, SIGNAL(valueChanged(int)),lcd, SLOT(display(int)));
```

☑ 3

5. Dans votre programme principal, placez votre bouton et votre affichage LCD dans un gestionnaire de géométrie (empilement vertical). Testez.

6. Adapter votre programme pour qu'un clic sur la widget LCD mettre à 0 le compteur.

☑ 4

7. On va utiliser le débugger intégré de Qt (qui est en fait une encapsulation de *gdb*). Placez un point d'arrêt permettant de visualiser la valeur de la variable incrémentée par chaque clic sur le bouton (Fênêtre d'observateurs). Executez votre programme en mode *Debugging*.


5

8. En s'inspirant du TD, modifiez votre programme pour qu'il affiche une pop-up d'information chaque fois que la fenêtre principale est redimensionnée

```
##Fichier Main.cpp
#include<QApplication>
#include<QWidget>
#include<QLabel>
#include<QDial>
#include<QLayout>
#include<QLCDNumber>
#include "ButtonCount.h"

int main(int argc, char* argv[])
{
  QApplication app(argc,argv);
  QWidget window;
  ButtonCount* bc=new ButtonCount("Coucou",&window);
  QLCDNumber* lcd=new QLCDNumber(2);
  QLabel * message=new QLabel("Hello,World",&window);
  //QDial* dial=new QDial();

  QVBoxLayout* layout=new QVBoxLayout();
  layout->addWidget(message);
  layout->addWidget(lcd);
  layout->addWidget(bc);
  //layout->addWidget(dial);
  window.setLayout(layout);

  //QObject::connect(dial,SIGNAL(valueChanged(int)),lcd,SLOT(display(int)));
  QObject::connect(bc, SIGNAL(valueChanged(int)),lcd, SLOT(display(int)));
  window.show();
  return app.exec();
}



##Fichier ButtonCount.cpp
#include <QtGui>
#include "ButtonCount.h"
ButtonCount::ButtonCount(const QString & text, QWidget * parent) : QPushButton(text,parent)
{
m_Count = 0;
connect(this, SIGNAL(clicked()), this, SLOT(Increment()));
}
void ButtonCount::Increment()
{
m_Count++;
// la ligne suivante assure un feedback sonore
QApplication::beep();
// emission du signal
emit(valueChanged(m_Count));
}


##Fichier ButtonCount.h
#ifndef BUTTONCOUNT_H
#define BUTTONCOUNT_H
#include <QObject>
#include <QPushButton>
class ButtonCount : public QPushButton
{
Q_OBJECT
public:
ButtonCount( const QString & text, QWidget *parent = 0 );
protected:
int m_Count;
signals:
void valueChanged(int);
public slots:
void Increment(void);
};
#endif

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


// Fichier MyLCD.h
#ifndef MYLCD_H
#define MYLCD_H

#endif // MYLCD_H
#include <QLCDNumber>


class MyLCD : public QLCDNumber
{
Q_OBJECT

public:
```

```
MyLCD( int, QWidget *);

slots
void MyLCD::mousePressEvent(QMouseEvent *);

signals:
void reset(int );

};



// Fichier MyLCD.cpp
#include "MyLCD.h"

MyLCD::MyLCD( int value=0, QWidget * parent = 0): QLCDNumber(value,parent)
{
    printf("Constructeur de MyLCD \n");
connect(this,SIGNAL(reset(int)),this,SLOT(display(int)));
}

void MyLCD::mousePressEvent(QMouseEvent *event)
{
printf("MyLCD pressé! \n");
emit reset(0);
}




// Fichier main.cpp
#include <QApplication>
#include <QFont>
#include <QLCDNumber>
#include <QPushButton>
#include <QSlider>
#include <QVBoxLayout>
#include <QWidget>
#include<math.h>
#include "MyLCD.h"



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

MyWidget::MyWidget(QWidget *parent)
    : QWidget(parent)
{
    QPushButton *quit = new QPushButton(tr("Quit"));
    quit->setFont(QFont("Times", 18, QFont::Bold));
    MyLCD *lcd = new MyLCD(2,NULL);

    lcd->setSegmentStyle(QLCDNumber::Filled);
  //  lcd->mousePressEvent();
    QSlider *slider = new QSlider(Qt::Horizontal);
    slider->setRange(0, 99);
    connect(quit, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(slider, SIGNAL(valueChanged(int)),
            lcd, SLOT(display(int)));
    connect(lcd, SIGNAL(reset(int)),slider,SLOT(setValue(int)));

      srand ( time(NULL) );
      int r =rand()%99;
    slider->setValue(r);

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