مدرسة

Qt/c++

الدرس 1

تذكير بالمبادئ الأولية للسي++

في الدرس 1 إن شاء الله سنتطرق إلى مبادئ السي++ الأولية للمراجعة ثم ننتقل مباشرة للبرمجة بالواجهة و هو نظري فقط

الآن سنتحدث قليلا عن المكتبات بسي++
فالمكتبات بسي++ هي عبارة عن ملفات ذات صيغة .h تحتوي هذه المكتبات على أكواد كثيرة و عديدة مبرمجة بسي++ و توفر علينا هذه المكتبات عناء كتابة الأكواد من الصفر حيث يكفي استدعاء المكتبات و استعمال الأكواد المبرمجة بها و يتم استدعاء هذه المكتبات عبر:

#include <library>

حيث نعوض library بإسم المكتبة
و هذه المكتبات متواجدة داخل المترجم حيث يكفي تحميل المترجم و استدعاء المكتبات من خلاله
و أول أمر سنتعرف عليه هو cout و هو لطباعة جملة

```
cout<<"texte";
```

حيث نعوض texte بالنص الذي نود طباعته و هذا الأمر يوجد بمكتبة iostream ، إذن يجب أن نستدعي هذه المكتبة قبل إستعمال الأمر أي يجب وضع عبارة

```
#include <iostream>
```

لن نتحدث عن أول برنامج لنا بشاشة dos لأننا لن نحتاجه إلا قليلا لكن لا بأس بوضعه:

```
#include <iostream>//إستدعاء مكتبة iostream
using namespace std;//فضاء الأسماء
int main ()//السدالة الرئيسية
{//بداية الدالة الرئيسية
cout<<"hello world";//طباعة الجملة
getchar();//وقف الشاشة عند الوصول إلى هذا الأمر
}
```

```
using namespace std;
```

و مهمته الأساسي تتجلى في:
إذا كتبنا البرنامج بدون كتابة عبارة using namespace std;
سيكون هناك خلل في البرنامج و هو أننا نحتاج لكتابة

```
std::cout<<"hamza";
```

بدل

```
cout<<"hamza";
```

لكن بواسطة using namespace std; لا نحتاج لكتابة std في بداية كل أمر برمجي
كل ما يهمني في الكود هو الدالة الرئيسية كيفية تعريفها

```
int main(){....}
```

و أن كل أمر يجب أن ينتهي ب;
سننتقل إلى تعريف المتغيرات
المتغيرات هي أماكن تحجز في الذاكرة ram لتخزين فيها معلومات مؤقتة ، و تكون هذه المعلومات عبارة عن جمل أو أرقام أو كلمات

و نعرف المتغير في c++ على هذا الشكل:

type nom=vv;
type : type de variable نوع المتغير
nom : اسم المتغير مهما يكن اسمه ما عدى الكلمان المحجوزة
vv: أو قيمة المتغير يمكن أن تكون رقم أو حرف أو

بعض الكلمات المحجوزة و التي لا ينبغي أن تكون أسماء للمتغيرات

auto ,,,,, break ,,,,,, case ,,,,,, char ,,,,, const ,,,,, continue ,,,,, default ,,,,,,, do ,,,,,,,double ,,,,,,,, else,,,,,,,, enum ,,,,,,,,,,,, extern float,,,,,,,, for ,,,,,,, goto ,,,,,,,,, if ,,,,,,,,,,, int ,,,,,,,,, long,,,,,,, register return ,,,,,,,,short,,,,,,,,, signed ,,,,,,,,,,,sizeof,,,,,,,,,,,, static struct ,,,,,,,,switch ,,,,,,,,,,,,typedef,,,,,,,,,,,,,union ,,,,,,,,,unsigned,,,,,,,,, void volatile,,,,,,, while

بعض أنواع المتغيرات

هذا النوع يختص لتخزين الأعداد الصحيحة: int

1،2،3،4.....

هذا النوع مختص في تخزين جميع الأعداد الجذرية التي بها فاصلة: double

1.5 ، 6.7، 2.3

هذا النوع يستعمل لتخزين الحروف و نستعمله كجدول من الحروف لتخزين الكلمات : char

h,hamza,=,"

هذا النوع يستعمل لتخزين الأعداد الكبيرة : long

1232222111،155555444

الآن وصلنا للأمر cin هذا الأمر هو الأمر المخالف للأمر cout أي أنه أمر للإدخال
حيث نستعمل هذا الأمر مع المتغيرات فيقوم مستخدم البرنامج بإعطاء قيمة للمتغير عبر البرنامج المبرمج مثل

```
int a;
cin>>a;
```

حيث في هذا الجزء من البرنامج يتوقف هذا الأخير منتظرا أن يدخل المستخدم قيمة للمتغير a الذي هو من نوع أعداد صحيحة طبيعية
فإذا أدخل المستخدم قيمة مخالفة لعدد صحيح طبيعي لن يشتغل البرنامج بشكل جيد و إذا أدخل قيمة صحيحة طبيعية سيكمل البرنامج عمله بنجاح

ملاحظة: الأمر cin موجود بمكتبة iostream لذلك وجب استدعائها في الأول

include <iostream>

الحلقة التكرارية بكل بساطة هي أمر من أوامر السي++ يجعلك تكرر مرات محددة كودا معينا يعني أكرر مثلا كودا معينا 4 مرات أو 5 مرات أو 6 مرات حسب إرادتي . يعني الحلقة التكرارية تسهل عليك مأمورية البرمجة
تصاغ الحلقة التكرارية بشكل سهل و واضح:

```
for(int i=0;i<n;i++)
{
//code.........
}
```

حيث تكرر code عدد المرات من i إلى n

فإذا كان n=10 فأن

n-i=10-0=10

أي أن البرنامج سيكرر ما بالكود 10 مرات

حيث أن في المرة الأولى i=0 يكرر الكود ثم يقوم بزيادة i درجة واحدة فيصبح i=1 و هكذا إلى أن يصل i=n

يمكن استبدال i++ ب i+=2

حيث i تزيد بدرجتين فإذا كان i=0 و n=10 و i+=2 إن البرنامج سيكرر الكود 5 مرات فقط

بالنسبة للدوال فهي عبارة عن طريقة لاختصار كود كثير في سطر واحد لاستعمالها كل مرة و تكتب خارج main بينما تستعمل داخلها غالبا

```
type nom(parametre)
{
//ما بداخل الدالة
}
```
type : نوع الدالة عدد أو حرف أو ....

nom : إسم الدالة

parametre: المستخدم عن طريق دالة الإدخال أو تحدد داخل البرنامج متغيرات ترسلها الدالة يحددها

nom(parametre); و نستدعيها هكذا

و لنستعمل الدوال داخل main أو دالة أخرى حيث مثلا:

```
#include <iostream>

using namespace std;

void som(int a,int b)

{

cout<<a+b;

}

int main()

{

int v=5;

int g=4;

som(v,g);

}
```

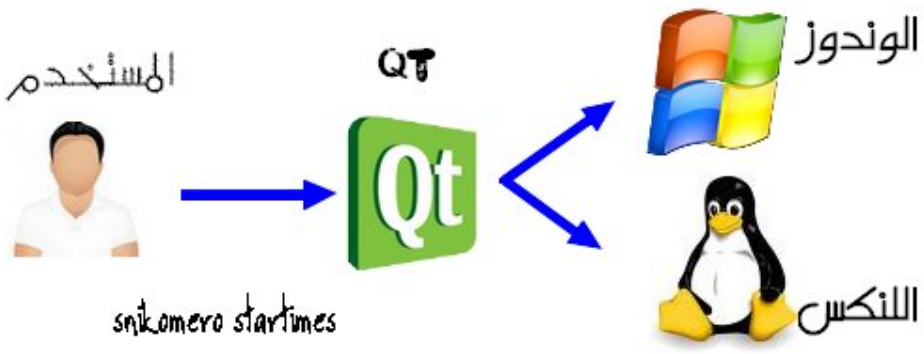
حيث أننا أنشئنا دالة لجمع عددين حيث أننا استدعينا دالة som() و وضعنا v و g بدل a و b

الدرس2

عالمنا نحو البرمجة بالواجهة Qtقد بدأ أول برنامج لنا ب Qt

ما هي Qt ؟

Qtهي عبارة عن مجموعة من مكتبات سي++ التي إجتمعت لتغلف دوالها على شكل فراموك حيث نستطيع من خلال هذه المكتبة الضخمة الخاصة تصميم الواجهة إعتمادا على دوال Qt و قوانين و سرعة c++ و بإعتبار هذه الخير لغة NATIVEفإن إستعمال Qt سيكون من أروع ما نقوم به ف كيوتيportable



ما قصة الزوج c++/Qt ؟

بإعتبار كيوتي مكتبة ضخمة فإنها لا تتعامل فقط مع السي++ بل أيضا مع الباسكال و الجافا و و لكن السي++ هي اللغة الأكثر إستعمالا لكيوتي لإعتبار هذه الأخيرة هي العماد الذي بنيت على اساسه Qt من طرفnokia

هل سنعمل بـ codeblock و dev و غيرها من المترجمات البسيطة ؟

لا لن نعمل بها لأنه c++/Qt لها مترجم خاص يسمى QtCreator لكن مع ذلك يوجد طرق لإستخدام المترجمات البسيطة لترجمة أكواد c++/Qt لكن ستجد صعوبة في عملية الترجمة لذلك من الأفضل تحميله

.....

سنحتاج لدروسنا Qt Creator و لتحميله:

لأصحاب الوندوز: هنا

لأصحاب اللينكس: هنا

ذكرنا أننا سنستعمل البرمجة الكائنية التوجه لكن ليس الآن فيماااا بعد سنستعمل اليوم فقط ملفmain.cpp

و لفعل ذلك نفتح Qt creator

2-creat project

3-other project

4-empty Qt project

5-نقوم بالضغط بالأيمن على الإطار الأبيض الفارغ على اليسار

6-add new

7-c++ source file

سيظهر لنا حقل للكتابة فارغ

أولا سنعرف أن Qt تسخدم كائنات في عملها مثلا في حياتنا اليومية لدينا الكرسي كائن و الطاولة كائن و و و فكيوتي تستخدم مجموعة من الكائنات سنستعمل كائن الزر و هو لإنشاء زر إسمهQPushButton

لنكتب ما يلي في الحقل

```
#include <QApplication>// تعريف المكتبة الرئيسية التي لابد منها في
#include <QPushButton>// تحتوي على الكائن الزر
int main(int argc, char *argv[])// تعريف الدالة
{// فتح الدالة الرئيسية
QApplication app(argc, argv);// تعريف تطبيق جديد للعمل عليه
QPushButton name("salam");// بكتابة تعريف الكائن و وضع
name.show();// الزر إظهار
return app.exec();// تفعيل التطبيق
}// غلق الدالة الرئيسية
```

كما نلاحظ يتم تعريف الكائن على هذا الشكل

```
typeobject nameobject(parametreobject);
```

حيث **typeobject** هي نوع الكائن في مثالنا السابق كان**QPushButton**

nameobjectهو إسم الكائن أسميناه نحن **name** يمكن أن نسميها ما نشاء

parametreobjectهي الباراميترات التي درسناها في الدوال يعني في **QPushButton** لدينا عذة باراميترات نختار ما نشاء منها ككتابة في الزر و هيالباراميتر الأول و أي نافذة ينتمي إليها و هي البارامتر الثاني نحن لم نقم بكتابة البارامتر الثاني لأننا نستخدم **QApplication** مباشرة و ليست نافذة أم ك **QWidget** التي سندرسها في ما بعد

و نتيجة الكود السابق هو نافذة بها زر فقط

كهذه النافذة

لكن ماذا بشأن النافذة الكبيرة بها الكثير من المكونات

هذا ما سنعرف فعله في الدرس القادم

الدرس 3

QWidget خصائصه ، دواله ، و الوراثة

widgetهو كل كائن له تصميم خارجي و وظائف يمتاز بها

كالأزرار عبارة عن widget خاصيتها هي عند الضغط عليها يتم تفعيل شيء ما

حقل النص عبارة عن widget يمتاز بالقدرة على الكتابة بداخل إلى غيرها

في الدرس السابق تعلمنا كيفية إظهار زر وحده لكن ماذا عن البرامج ذات النوافذ الكبيرة و الكائنات المتعددة و غيرها

لا لا تقلقوا فكل هذا متوفر في Qt فالنافذة هل كائن لا ينتمي لأي كائن آخر مثلا الزر ينتمي إلى النافذة لكن النافذة تنتمي إلى نفسها فقط

و النافذة في Qt هي الكائن QWidget مع مراعات الحروف الكبيرة و الصغيرة لأن سي++ حساس لمثل هذه الأشياء

يتم تعريفه هكذا

```
QWidget hamza;
```

و لتطبيق بعض الخصائص عليه هذه هي الصيغة

```
hamza.setFixedSize(100,100);
```

بصفة عامة

```
QWidget nomqwidget;

nomqwidget.setFixedSize(x,y);
```

setFixedSizeهي خاصية لحجم النافذة, هناك خصائص أخرى لكننا لا نحتاجها

+

نحتاج أيضا هذه العلامة "&" لكي نبين أن كائنا ما ينتمي إلى نافذة ما

ما رأيكم بتطبيق يزيل عنكم الغموض

```
#include <QtGui>

#include<QApplication>

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

QApplication app(argc, argv);


QWidget hamza;

hamza.setFixedSize(100,100);

QPushButton button("ici",&hamza);

hamza.show;

return app.exec();

}
```

كما تلاحظون في المثال وضعنا &hamza
يعني أن الزر ينتمي إلى hamza و بما أننا أظهرنا النافذة فسيظهر الزر بداخلها مباشرة لأن الزر ينتمي إلى النافذة

الخاصيةmove

```
#include <QtGui>

#include<QApplication>

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

QApplication app(argc, argv);


QWidget hamza;

hamza.setFixedSize(100,100);

QPushButton button("ici",&hamza);

button.move(50,50);

hamza.show;

return app.exec();

}
```

هذه الخاصية تسمح لنا يتحديد مكان الزر داخل النافذة

مالاا رأيكم الآن بالإنتقال إلى طور البرمجة الكائنية التوجه
ملف mafenetre.h مثلا
يكون دائما هكذا

```
#ifndef  DEF_NOMDECLASS

#define DEF_NOMDECLASS

#include <QApplication>

#include <QPushButton>

class MaFenetre: public QWidget
```

{

public :

MaFenetre();

private:

نعرف الكائنات هنا على هذا الشكل

QPushButton *hamza

};

#endif

ملف mafenetre.cpp

#include "mafenetre.h"

MaFenetre:: MaFenetre () : QWidget ()

{

هكذا بالنسبة لخصائص النافذة الرئيسية فتكتب مباشرة دون نقطة

setFixedSize(x,y);

ثم يتم إعادة تعريف الكائنات على هذا الشكل

hamza=new QPushButton("hamza",this);

this حيث توضع كلمة

بدل الإنتماء إلى نافذة معينة

الشكل أما بالنسبة لخصائص الكائنات فلا توضع نقط بل إشارة أخرى على هذا

hamza->move(x,y);

حيث نلاحظ وضع عارضة وأكبر قطعا بدل النقطة

}

ملف main.cpp

#include <QApplication>

```
#include "mafenetre.h"


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

MaFenetre fenetre;
fenetre.show();

return app.exec();
}
```

الدرس 4

الأحداث و التغيرات

أهم دروس Qt و التي نعتمد عليه كثيييرا

و هو درس يخص الأحداث

يعني ربط كل حدث بتغير خاص به

فالحدث مثلا هو الضغط على زر ما

أما التغير فهو نتيجة الضغط على الخدث و نرمز في Qt للحدث ب SIGNAL و للتغير بSLOT

و نربطهم من خلال دالة إسمها connect موجودة في كلاس QObject و نستدعيها في ملفMaFenetre.cpp

و التغييرات البسيطة هي;()quit

و;()aboutQt

و.....

فنجرب مثالا عن ذلك و هو إغلاق النافذة

أولا ملفMaFenetre.h

```
#ifndef  DEF_MAFENETRE

#define DEF_MAFENETRE

class MaFenetre :public QWidget

{

public :

MaFenetre();

private :

QPushButton *hamza;
```

```
}
```

ملف MaFenetre.cpp

```
#include <QApplication>

#include <QtGui>

MaFenetre::MaFenetre() : QWidget()

{

setFixedSize(100,100);

hamza=new QPushButton("quit",this);

hamza->move(50,50);

QObject::connect(hamza,SIGNAL(clicked()),qApp,SLOT(quit()));

}
```

ملف main.cpp

لا داعي لكتابته فهو سهل

بالنسبة للسطر المكتوب بالأحمر و الذي هو جديد فهي الدالة connect التي تربط الحدث بالتغير حيث تضم 4 بارامترات

الأول هو الكائن الذي يضم الحدث و هو الزر

أما الثاني فهو الحدث و يوضع دائما بين قويسن

SIGNAL(....);

البارامتر الثالث هو مصدر التغير

فالتغيرات الرسمية (quit,aboutQt,....) إذا أردنا إستدعائها فإننا سنكتب qApp في البارامتر الثالث

أما إذا أردنا أن نصنع تغير آخر فسنضع this

أما الرابع فهو التغير

.....

ملاحظة هامة ::::: أ لا يمكننا القيام بتغييرات غير الخروج و نافذة ''حول"Qt

بلى يمكننا ذلك حيث سنصنع تغييرات خاصة

أولا لفعل ذلك يجب التعريف بQ_OBJECT

في ملفMaFenetre.h

و كذلك التعريف بالتغير

مثال : عند الضغط على زر OK يتغير نص الزر إلىhamza

و لفعل ذلك لاحظوا ملف mafenetre.h الجديد

```
#ifndef  DEF_MAFENETRE

#define DEF_MAFENETRE

class MaFenetre : public QWidget

{

Q_OBJECT
```

```
public :

MaFenetre();

public slots:

void changetext();

private :

QPushButton *hamza;

}
```

كما تلاحظون يتم تعريف التغيير بواسطة دالة
الآن ملف**MaFenetre.cpp**

```
#include <QApplication>

#include <QtGui>

MaFenetre::MaFenetre() : QWidget()

{

setFixedSize(100,100);

hamza=new QPushButton("OK",this);

hamza.move(50,50);

QObject::connect(hamza,SIGNAL(clicked()),this,SLOT(changetext()));

}

void MaFenetre::changetext()

{

hamza->setText("hamza");

}
```

كما تلاحظون وضعنا **this**بدل**qApp**
و**()changetext**
بدل**()quit**
و كذلك أغلقنا**()MaFenetre**
و أعدنا تعريف الدالة **changetext** ووضعنا بداخلها أمرا بتغيير نص الزر
إذن عند الضغط على الزر يتم تغيير نص الزر

الدرس 5

نوافذ الحوار

في هذا الدرس سنتعرف على نوافذ الحوار

و الموجودة في كلاس المسمىQMessageBox

و تنقسم هذه النوافذ إلى نوافذ للمعلومات

نوافذ للتحذير نوافذ للمنع

نوافذ للسؤال

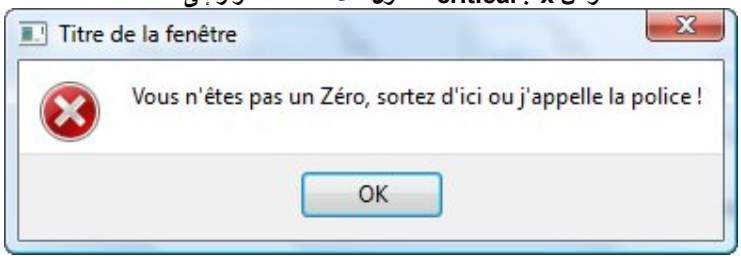
و التي تأخذ الشكل التالي

QMessageBox::x(QWidget *parent,QString *titre,QString *text,button);


نعوض x ب information فتتحول علامة نافذة الحوار إلى


نعوض x بwarning فتتحول علامة نافذة الحوار إلى

نعوض x بcritical فتتحول علامة نافذة الحوار إلى



مثل :

و كذلك نعوض QWidget *parent بthis

و نعوض QString *titre ب عنوان نافذة الحوار بين" "

و QString *text بنص الحوار بين" "

أما button فإذا لم نكتب شيء سيظهر زر واحد و هو OK

لكن ماذا إن عوضنا x ب questionهل سيكون الجواب ب OK لا سيكون الجواب بنعم أو لا لذلك نملأ البارامتر الأخير بـ QMessageBox::yes | QMessageBox::No

حيث يصبح على شكل

QMessageBox::question(this,"quetion","est-ce que tu es hamza ",QMessageBox::Yes | QMessageBox::No);

لكن كيف نستخدم نوافذ الحوار

سهلة::

عن طريق slot وsignal

الذان تعلمناهما في الدرس السابق

Mafenetre.h

```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include <QApplication>

#include<QtGui>

class MaFenetre : public QWidget
```

```
{
Q_OBJECT
public :
MaFenetre();
public slots :
void ouvrire();
private :
QPushButton *ok;
};
#endif
```

ملف mafenetre.cpp

```
#include <QtGui>
#include <QApplication>
MaFenetre::MaFenetre() : QWidget ()
{
setFixedSize(100,100);
ok=new QPushButton("ouvrire",this);
ok->move(50,50);
QObject::connect(ok,SIGNAL(clicked()),this,SLOT(ouvrire()));
}
void MaFenetre::ouvrire()
{
QMessageBox::information(this,"titre","je suis un programmeur");
}
```

أما main.cpp فهو سهل

هناك نوع آخر من نوافذ الحوار و هي نوافد الملفاتQFileDialogue
يمكن إستعمالها مثلا لحف ملف معين أو لإختيار ملف و القيام بنسخه أو ما شابه

```
QString fichier = QFileDialog::getOpenFileName(this, "Ouvrir un fichier", QString(), "Images (*.png *.gif *.jpg *.jpeg)");
```

و البارامترات سهلة و واضحة

```
QString fichier = QFileDialog::getOpenFileName(this, "Ouvrir un fichier", QString(), "Text(*.txt*.doc)");
```

أو يمكن حفظ ملف

```
QString fichier = QFileDialog::getSaveFileName(this, "sauvegarder un fichier", QString(), "Text(*.txt*.doc)");
```

لن نستعمل نوافذ الحوار
كثيرا لكن هي جد مهمة يمكن عن طريقها حفظ حقوق برنامجنا أو
التعامل مع الملفات أو الاستجوابات و التحذيرات

الدرس6
الكائنات الضرورية لبرامجنا

تعلمنا لحد الآن التعامل مع كائن واحد و هوQPushButton

لكن ماذا عن الكائنات الأخرى

:::::::

QLabel

هذا الكائن يستعمل لإظهار صور أو نصوص العادية تقريبا نفس دور cout كهذه النافذة

هو بدوره يرث الخصائص الرئيسية و هي move و text و setText و.....

إظهار نص

1-لإظهار نص يكفي التعريف بالبارامتر الخاص بQLabel

2-أو إستعمالsetText

دعنا نجرب الطريقتين

```
#ifndef DEF_MAFENETRE

#define  DEF_MAFENETRE
```

```
#include<QtGui>

#include<QApplication>

class MaFenetre : public QWidget

{

Q_OBJECT

public :

MaFenetre();

private:

QLabel *label;

};

#endif
```

```
#include<QtGui>

#include <QApplication>

#include "mafenetre.h"

MaFenetre::MaFenetre() : QWidget ()

{

setFixedSize(100,100);

label=new QLabel("salam",this);

label->setText("salamm");

label->move(50,50);

}
```

بعد أن أظهرنا نص ماذا عن معرفة محتوى ال QLabel و القيام بعمليات عليه كالمقارنة

نستعمل الدالةtext
حيث

```
QString hamza=label->text();
```

حيث hamza كائن من نوع QString و هو نوع خاص من النصوص يمكن تحويله إلى string ثم إلى char
حيث

```
const char *h=hamza.toStdString().c_str();
```

و كل ما هو مكتوب بالأحمر ينطبق على كل الكائنات التي تظهر النصوص مثل ال QPushButtonو QLineEdit و QLabel و QTextEdit

إظهار صورة

لإظهار صورة نستعمل الدالةsetPixmap

```
label=new QLabel(this);

label->setPixmap(QPixmap("image.png"));

label->move(50,50);
```

:::::::

QLineEdit

هو حقل نص عادي لإدخال النصوص بكل أنواعها

مثل

التعريف بالكائن

```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include<QtGui>


#include<QApplication>

class MaFenetre : public QWidget

{

Q_OBJECT

public :

MaFenetre();

private:
```

```
QLineEdit *hamza;

};

#endif
```

```
#include<QtGui>

#include <QApplication>

#include "mafenetre.h"

MaFenetre::MaFenetre() : QWidget ()

{

setFixedSize(100,100);

hamza=new QLineEdit(this);

}
```

إليكم هذه الدالة تقوم تحول كل ما يكتب إلى نمط كلمة السر أي إلى نقط
******

```
lineedit->setEchoMode(QLineEdit::Password);
```

الآن ما رأيكم في تجربة
القيام ببرنامج يقوم بعملية جمع سهلة

```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include<QtGui>
#include<QApplication>
class MaFenetre : public QWidget
{
Q_OBJECT
public :
MaFenetre();
public slots:
void ok();
private:
QLineEdit *m_champ1;
QLineEdit *m_champ2;
QLineEdit *m_champ3;
QPushButton *m_ok;
};
#endif
```

```
#include<QtGui>

#include "mafenetre.h"
#include <QApplication>
MaFenetre::MaFenetre() : QWidget ()
{
setFixedSize(200,115);
m_champ1=new QLineEdit("1",this);
m_champ2=new QLineEdit("2",this);
m_champ3=new QLineEdit("3",this);
m_ok=new QPushButton("ok",this);
m_champ1->move(2,10);
m_champ2->move(2,40);
m_champ3->move(2,70);
m_ok->move(2,90);
QObject::connect(m_ok,SIGNAL(clicked()),this,SLOT(ok()));
}
void MaFenetre::ok()
{
double a=m_champ1->text().toDouble();
double b=m_champ2->text().toDouble();
double c=a+b;
m_champ3->setText(QString::number(c));
}
```

ما هو صعب في الكود هو ما مكتوب بالأحمر

QString::number()

دائما نستعمل setText لتغيير نص عادي لكن لتغيير النص برقم يجب إستعمالQSting::number

champ1->text().toDouble()

بالنسبة للدالة text فهي معروفة لكي نعرف قيمة ال QLineEdit

لكن مادا عن().toDouble.

هذه الدالة تقوم بتحويل QString إلى Double القابل لإجراء عمليات رياضية عليه حيث جمنا محتوى الحقل 1 مع محتوى الحقل 2 و النتيجة وضعناها في الحقل 3 عبر setText

QTextEdit

هذا الكائن لإظهار نص كبير

مثل حقل النص الذي نكتب به ردود و مواضيع و مثل هذا الحقل

و لأخذ محتواه على شكل QString لا نستعمل text() كباقي الحقول بل نستعمل() toPlainText

```
QString contenu=textedit->toPlainText();
```

يعني الآن إذا كنتم تتقنون التعامل مع الملفات فيمكنكم أن تيتعملوا الدالة و تحولوا QString إلى char ثم تقومون بحفظ النص في ملف و أيضا تستطيعون فتح محتوى ملف نصي و إدخاله في QTextEdit عن طريق setText

و من باب المعرفة إليكم دورتي السابقة في التعامل مع الملفات عن طريق++c

رقم الدرس	عنوان الدرس	وصلة الموضوع
1	التعامل مع الملفات عبر القنوات (درس نظري)	هنا
2	ofstream و الكتابة في ملف	هنا
3	تطبيق 1 : تخزين المعلومات	هنا
4	ifstream و القراءة من ملف	هنا
5	تطبيق 2 : قراءة جمل من ملف نصي	هنا
6	getline و الحلول التي توفرها لك	هنا
7	الرايات و أهميتها	هنا
8	مشروح تخرجك من الدورة	هنا
9	تصحيح مشروع التخرج	قريبا

لحد الآن

QLabel

QLineEdit

QPushButton

وQTextEdit

تم شرحها و هي ما سنعتمد عليها الآن أما باقي الكائنات سيتم دراستها في حال إحتجنا لها في الدروس القادمة

ملاحظات جد هاااااااااااامة
في ملفmafenetre.h

من الضروري الحفاظ على ترتيب ifndef و define حيث

#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

و كذلك يجب أن نكتب public و ليس Public و أعتذر لأنني إرتكبت هذا الخطأ في الدروس السابقة و لم أقم بتجريب الأكواد ظناً مني أنها صحيحة 100%
و كذلك يجب الإنتباه إلى الأقواس حيث أننا في بعض الأحيان ننسى كتابة قوس أو قوسين
و كذلك الإنتباه إلى القوسين بعد الدوال و التأكد من صحة بارامترات هذه الدوال(show , move,.....)
الدرس القادم سوف نرتاح من عذاب move و سننتقل إلىles layouts

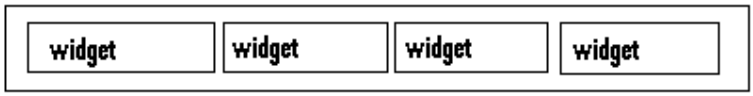
الدرس 7

MOVE الارتياح كم عذاب
QLayoutإستعمال

هذا الدرس هو خاص ب layout
و هي كائنات تنظم واجهة برنامجك

حيث أننا سيصعب علينا إستعمال move في البرامج الكثيرة الكائنات
و الكلاس المسؤول عنlayouts
QLayoutإسمه
ينقسم إلى
QHBoxLayout
QVBoxLayout
QFormLayout
QGridLayout

نبدأ ب layoutالأفقيةQHBoxLayout
لكن ما هو شكل هذا: layout
إن كائن بنظم الكائنات على هذا الشكل



QHBoxLayout snikomera

التعريف بكائن من نوع QHBoxLayout يتم في ملفmafenetre.cpp
على هذا الشكل

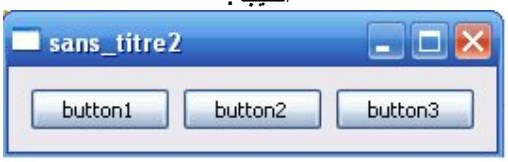
```
QHBoxLayout *hamza=new QHBoxLayout;
```

و يتم إضافة كائن إلى QHBoxLayout عن طريق الدالةaddWidget
فلنجرب مثال

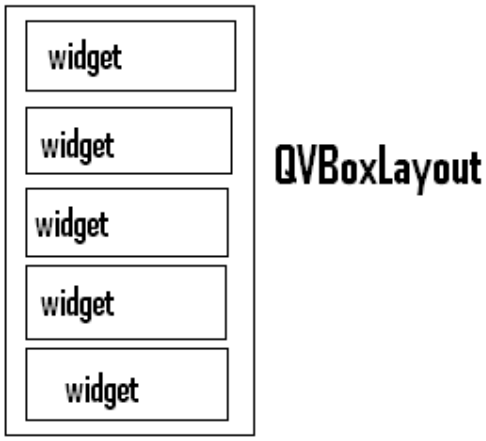
```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include <QtGui>

class MaFenetre : public QWidget

{

public :
```

```
MaFenetre();

private :

QPushButton *b1;

QPushButton *b2;

QPushButton *b3;

};

#endif


#include"mafenetre.h"

MaFenetre::MaFenetre() : QWidget()

{

b1=new QPushButton("button1");

b2=new QPushButton("button2");

b3=new QPushButton("button3");

QHBoxLayout *layout=new QHBoxLayout;

layout->addWidget(b1);// إضافة الزر الأول لـ layout

layout->addWidget(b2); إضافة الزر الثاني لـ// layout

layout->addWidget(b3); إضافة الزر الثالث لـ// layout

setLayout(layout);

هنا إدماج اللايوت داخل النافذة لكي يتفعل دوره//

}
```

الكود مفهوم لكن....
لماذا لم نضع this في البارامتر الثاني فيQPushButton
الجواب هو أنه لا داعي لذلك فالأزرار نسبناها إلى layout عن طريق addWidget و ال layoutنسبناه للنافذة عن طريق setLayout
إذن سيتوك الأزرار منسوبة إلى النافذة بشكل غير مباشر
النتيجة:



ننتقل الآن إلىQVBoxLayout
أي layout عمودي على هذا الشكل



التعريف عادي مثلQHBoxLayout

```
QVBoxLayout *hamza=new QVBoxLayout;
```

هي أيضا نستعمل فيهاaddWidget
مثال توضيحي

```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include <QtGui>

class MaFenetre : public QWidget

{

public :

MaFenetre();

private :

QPushButton *b1;

QPushButton *b2;

QPushButton *b3;

};

#endif


#include"mafenetre.h"

MaFenetre::MaFenetre() : QWidget()

{
```

```
b1=new QPushButton("button1");

b2=new QPushButton("button2");

b3=new QPushButton("button3");

QVBoxLayout *layout=new QVBoxLayout;

layout->addWidget(b1);//إضافة الزر الأول لـ layout

layout->addWidget(b2);//إضافة الزر الثاني لـ layout

layout->addWidget(b3);//إضافة الزر الثالث لـ layout

setLayout(layout);

//هنا إدماج اللايوت داخل النافذة لكي يتفعل دوره

}
```

النتيجة:

الآن ننتقل لنوع خاص من ال QLayout و هو QGridLayout
و هو مزيج بين ال QHBoxLayout و QVBoxLayout
و هي تشبه الإحداثيات بتربيعات كبيرة

0, 0	0, 1	0, 2	...
1, 0	1, 1	1, 2	...
2, 0	2, 1	2, 2	...
...	...	...	...

(صورة التربيعات مقتطفة من موقع زيرو)
حيث نضع لكل widget زوج يحدد مكانه في النافذة
تعريف QGridLayout

```
QGridLayout *hamza=new QGridLayout ;
```

نستعمل addWidget
مثال:

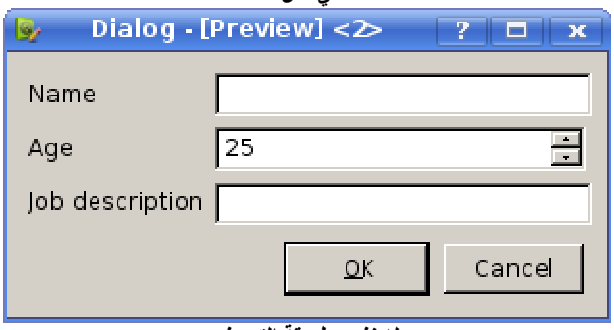
```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include <QtGui>

class MaFenetre : public QWidget

{

public :

MaFenetre();

private :

QPushButton *b1;

QPushButton *b2;

QPushButton *b3;

};

#endif


#include"mafenetre.h"

MaFenetre::MaFenetre() : QWidget()

{

b1=new QPushButton("button1");

b2=new QPushButton("button2");

b3=new QPushButton("button3");

QGridLayout *layout=new QGridLayout ;

layout->addWidget(b1,0,0);//إضافة الزر الأول لـ layout

layout->addWidget(b2,1,0);//إضافة الزر الثاني لـ layout

layout->addWidget(b3,1,1);//إضافة الزر الثالث لـ layout

setLayout(layout);
```

هنا إدماج اللايوت داخل النافذة لكي يتفعل دوره//

}

النتيجة:

الآن وصنا لآخر QLayout و هوQFormLayout
و هو يشبه تنظيم الformulaire
أي مثل



له نفس طريقة التعريف
لكن لا نستعمل addWidget
بل addRow و نستعمل فقط نوع واحد من widget و هوQLineEdit

```
#ifndef DEF_MAFENETRE

#define DEF_MAFENETRE

#include <QtGui>

class MaFenetre : public QWidget

{

public :

MaFenetre();

private :

QLineEdit *b1;

QLineEdit *b2;

QLineEdit *b3;

};
```

```
#endif


#include"mafenetre.h"

MaFenetre::MaFenetre() : QWidget()

{

b1=new QLineEdit ("");

b2=new QLineEdit ("");

b3=new QLineEdit ("");

QFormLayout *layout=new QFormLayout;

layout->addRow("nom",b1);//إضافة الزر الأول لـ layout

layout->addRow("prenom",b2);//إضافة الزر الثاني لـ layout

layout->addRow("class",b3);//إضافة الزر الثالث لـ layout

setLayout(layout);

//هنا إدماج اللايوت داخل النافذة لكي يتفعل دوره

}
```

النتيجة: