

TP 25 – Qt Widgets – Signaux – Slots

Exercice 0 Création d'un nouveau projet

- Créez un nouveau projet Qt (**Qt GUI Application** / **Qt Widgets Application**), dont la fenêtre principale hérite d'un **QWidget**.

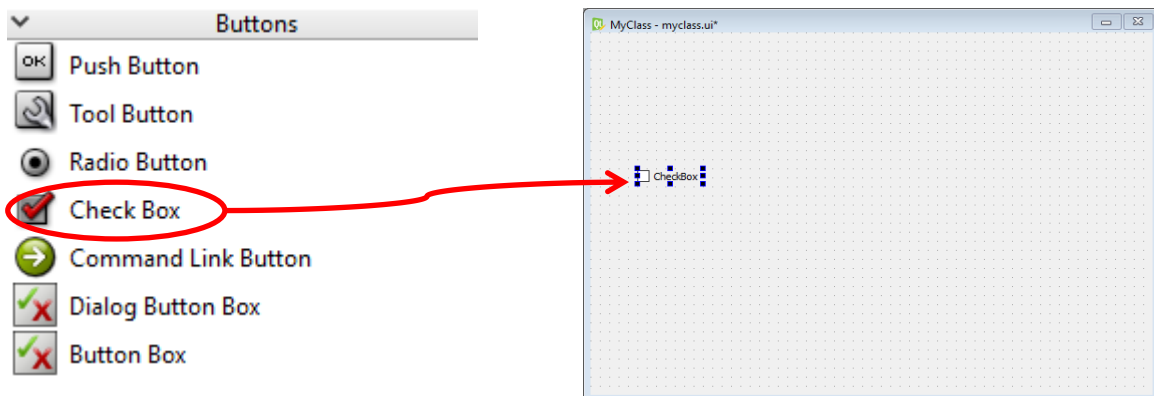
Par la suite, nous supposons que vous avez nommé la classe correspondant à la fenêtre principale « **MyClass** »

Il se peut que l'auto-complétion de Visual Studio ne fonctionne pas et que les noms de classes Qt soit soulignées en rouge. Pour résoudre ce bug, compilez le projet puis choisissez le menu **Projet / Relancer l'analyse de la solution**.

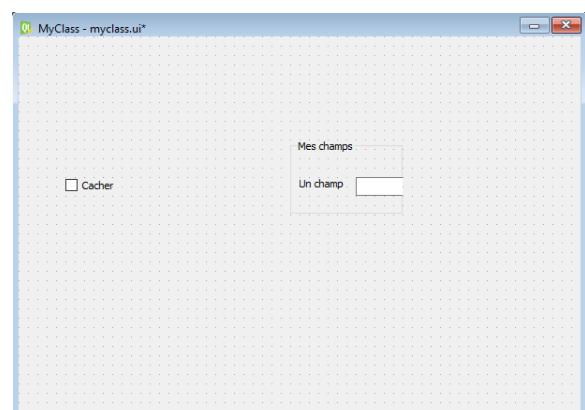
Exercice 1 Prise en main de Qt Designer

1.1 Design de l'interface

- Ouvrez le fichier **myclass.ui** avec **Qt Designer**
 - Normalement, un double clic fait le travail... Sinon, faites un clic droit sur son icône, **Ouvrir avec...**, puis choisissez **Qt Designer** dans la liste.
- Faites glisser sur la fenêtre d'édition un **Check Box** disponible dans la boîte de widget.



- Faites également glisser un **Group Box**, puis à l'intérieur de celui-ci un **Label** et un **Line Edit** de façon à reproduire la capture ci-contre :
 - Éditer également les textes des éléments en double cliquant dessus
 - CheckBox → Cacher
 - GroupBox → Mes champs
 - TextLabel → Un champ




Nous avons pour l'instant tous les contrôles nécessaires pour cette prise en main. Mais vous remarquez que l'interface n'est pas correctement organisée. Qt propose un mécanisme de mise en page automatique des fenêtres basé sur l'utilisation d'objets de type **QLayout**. Nous allons organiser l'espace de la fenêtre grâce à cela, directement depuis Qt Designer.

- ☐ Dans la barre d'outils supérieure, sont symbolisés différentes mises en page :



. Sélectionnez le fond de la fenêtre **MyClass**, puis

cliquez sur le bouton .

- ➡ Automatiquement, la case à cocher le groupe « Mes champs » se disposent l'un au-dessus de l'autre. Faites en sorte à la souris que la case à cocher se trouve au-dessus.

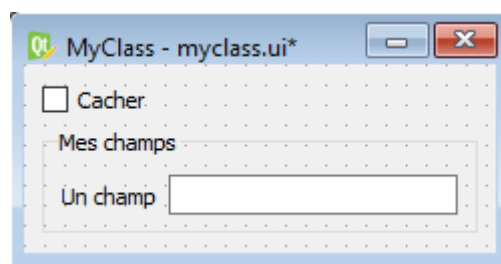
- ➡ Vous pouvez observer qu'en redimensionnant la fenêtre, la taille des contrôles est automatiquement adaptée.

- ☐ Sélectionnez maintenant le groupe « Mes champs » puis appliquez-lui une mise en page de

formulaire : .

- ➡ Le label et le champ de saisie de texte sont alors correctement disposés dans le groupe « Mes champs »


- ☐ Réduisez la fenêtre de façon à ce qu'elle ait les proportions suivantes :

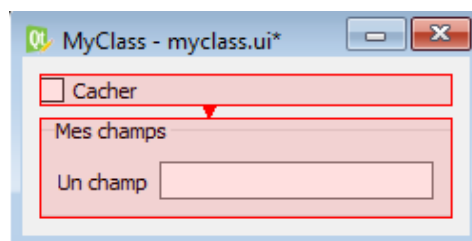


1.2 Connexion des signaux et slots

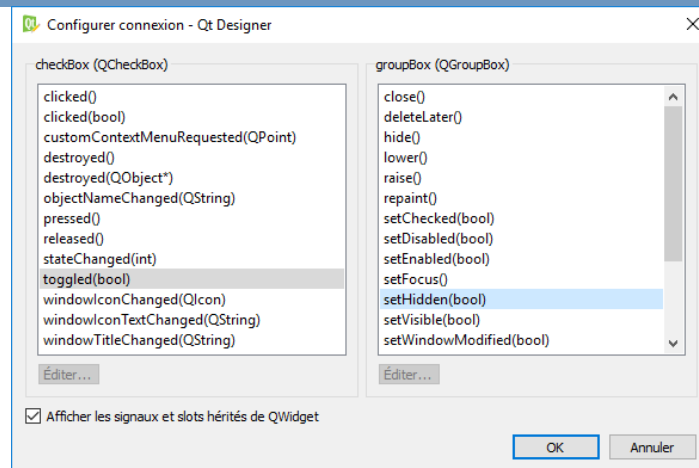
Pour les signaux et slots standards, il est possible de réaliser des connexions directement depuis Qt Designer. Pour cela, deux méthodes sont possibles :

1.2.1 Méthode à la souris

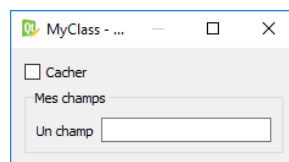
- ☐ Basculez en mode d'édition des signaux et slots en cliquant sur le bouton .
- ➡ Faites glisser une connexion entre la case à cocher et le groupe « Mes champs » :



- ➡ Dans la fenêtre qui s'ouvre, cochez la case « Afficher les signaux et slots hérités de QWidget » puis sélectionnez le signal **toggled(bool)** du checkbox et le slot **setHidden(bool)** du groupBox :



Vous pouvez prévisualiser votre interface en choisissant le menu **Formulaire / Prévisualisation...** ou **Qt Outils / Form Editor / Prévisualisation...**

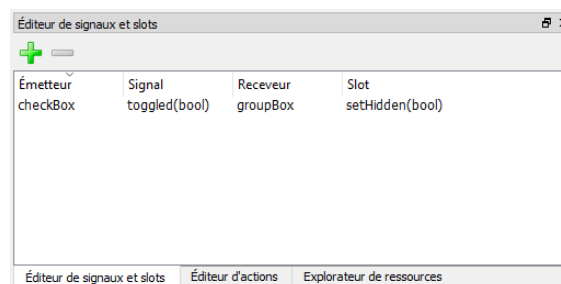


Prévisualisation sous Windows 10

- ➡ L'interface est semi-fonctionnelle. C'est-à-dire que seules les connexions de signaux et slots que vous avez définies à l'aide de Qt Designer sont fonctionnelles. Cochez la case « Cacher » : le groupe « Mes champs » doit disparaître.

1.2.2 Méthode à l'aide de l'éditeur de signaux et slots.

En bas à droite ou **Qt** en bas de la fenêtre de Qt Designer, se trouve l'éditeur de signaux et slots :





En utilisant cet outil, vous pouvez réaliser des connexions de façon parfois plus simple qu'à la souris, lorsque la fenêtre est encombrée. Remarquez que la connexion que vous venez d'établir à la souris se trouve répertoriée dans cette fenêtre.


1.3 Et le code dans tout ça ?



- ➡ Enregistrez votre fichier **myclass.ui**.
- ➡ Compilez votre solution
- ➡ Ouvrez le fichier **ui_myclass.h**
 - ➡ L'utilitaire de compilation UIC a généré pour vous ce fichier. Vous retrouvez dans le corps de la fonction **setupUi**, toute votre interface dessinée sous Qt Designer.

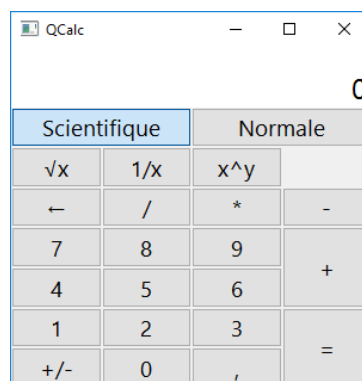
Exercice 2 Une calculette

-  En utilisant l'utilitaire Tortoise Git, clonez le dépôt [git@gitlab-lepuy.iut.uca.fr:TPs-QualiteDeDev/TP25.git](https://gitlab-lepuy.iut.uca.fr/TPs-QualiteDeDev/TP25.git).



 Le plus important dans les exercices 2, 3 et 4 est de tout comprendre. Aussi, il vous est fait confiance quant à votre force de travail. C'est pourquoi, la solution de l'exercice vous est donnée. Pour chacune des sous-parties suivantes (2.1, 2.2, etc..), la solution se trouve dans le dépôt Git, sous une étiquette distincte. Pour vous placer au niveau de la solution d'une partie, faites un clic droit sur le dépôt et choisissez **TortoiseGit / Basculer/Extraire...** et choisissez l'étiquette souhaitée dans le champ **Étiquette**.

-  L'objectif de ce TP est de se concentrer sur la création de l'interface utilisateur. L'aspect technique de la construction d'une expression mathématique est déjà implémenté. Il vous est cependant conseillé, à titre d'exemple propre de code, de lire et de comprendre le code du fichier **Operation.h**. Ce fichier a généré l'aide en ligne disponible dans le fichier **doc/html/index.html**. Ce code s'articule autour de la classe **CResult** (**doc/html/classbak_calc_1_1_c_result.html**) qui implémente le design pattern nommé « Composite » dont vous trouverez la définition sur Wikipédia : https://fr.wikipedia.org/wiki/Objet_composite.

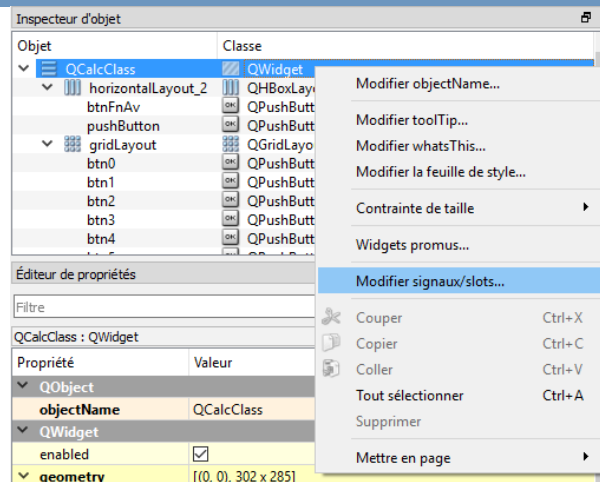
-  Ouvrez et compilez ce projet.
-  Vous obtenez une base de calculette non fonctionnelle :



2.1 Connexions dans l'interface

-  À l'aide de Qt Designer, connectez le signal **toggled(bool)** du bouton « Scientifique » aux slots **setVisible(bool)** des boutons « \sqrt{x} », « $1/x$ » et « x^y ».
-  Compilez exécutez, la première ligne de bouton doit s'afficher et se masquer en fonction du mode de la calculette

Il est possible également à l'aide de Qt Designer de réaliser des connexions entre des signaux et slots personnalisés de la classe de la fenêtre principale. Pour cela il faut indiquer à Qt Designer ces nouveaux signaux et slots. Cela se fait par l'action d'un clic droit sur l'objet dont les signaux et slots doivent être modifiés puis en choisissant « Modifier signaux/slots... ». La sélection de l'objet peut se faire à l'aide de « l'Inspecteur d'objet » :



- ☞ Réalisez alors les connexions telles que définies dans le tableau suivant :
- ➡ À titre d'exercice, n'en faites que certains.

Éditeur de signaux et slots			
Émetteur	Signal	Receveur	Slot
btnValid	clicked()	QCalcClass	btnValid()
btnSqrt	clicked()	QCalcClass	btnSqrt()
btnPow	clicked()	QCalcClass	btnPow()
btnPlus	clicked()	QCalcClass	btnPlus()
btnOp	clicked()	QCalcClass	btnOp()
btnMult	clicked()	QCalcClass	btnMult()
btnMinus	clicked()	QCalcClass	btnMinus()
btnInv	clicked()	QCalcClass	btnInv()
btnFnAv	toggled(bool)	btnPow	setVisible(bool)
btnFnAv	toggled(bool)	btnInv	setVisible(bool)
btnFnAv	toggled(bool)	btnSqrt	setVisible(bool)
btnFnAv	toggled(bool)	QCalcClass	setScientificMode(bool)
btnDot	clicked()	QCalcClass	btnDot()
btnDiv	clicked()	QCalcClass	btnDiv()
btnDel	clicked()	QCalcClass	btnDel()
btn9	clicked()	QCalcClass	btn9()
btn8	clicked()	QCalcClass	btn8()
btn7	clicked()	QCalcClass	btn7()
btn6	clicked()	QCalcClass	btn6()
btn5	clicked()	QCalcClass	btn5()
btn4	clicked()	QCalcClass	btn4()
btn3	clicked()	QCalcClass	btn3()
btn2	clicked()	QCalcClass	btn2()
btn1	clicked()	QCalcClass	btn1()
btn0	clicked()	QCalcClass	btn0()

2.2 Les slots côté code

Selon les connexions que vous avez réalisées dans la section précédente, chaque bouton est associé à un slot dans l'objet **QCalcClass**. Nous devons alors déclarer et définir chacun de ces slots dans les fichiers **qcalc.h** et **qcalc.cpp**. Leurs implémentations sont simples.

Les slots des boutons de chiffres et du bouton de virgule décimale doivent appeler la méthode **QCalc::AppendChar()** déjà implémentée, avec comme paramètre le caractère leur correspondant (*note* : la virgule décimale est en réalité un point « . »).

Les slots des opérateurs doivent appeler la méthode `QCalc::AppendOperator()` avec comme paramètre un pointeur intelligent vers l'objet les décrivant. Par exemple, l'opérateur « + » doit réaliser cet appel : `AppendOperator(std::make_shared<bakCalc::COpPlus<valType>>())`. Référez-vous à l'aide en ligne fournie dans le fichier `doc/html/namespacebak_calc.html` pour connaître l'ensemble des classes des opérateurs.

Le slot du bouton d'effacement (←) consiste à enlever un caractère du texte du contrôle `edtResultat`. Si le texte est vide, alors le texte devient « 0 ». Une implémentation possible est :

```
std::string str{ ui.edtResultat->text().toStdString() };
str.pop_back();
if (str.empty())
    str = "0";
ui.edtResultat->setText(str.c_str());
```

Le slot du bouton de validation (=) doit finaliser l'expression mathématique en appelant la méthode `AppendOperator(nullptr)`. Cela a pour effet d'ajouter la dernière valeur saisie sans opérateur supplémentaire. Ensuite, le slot doit afficher un « = » à la fin de l'expression dans la zone de texte `edtOperation` et le résultat de l'évaluation dans le contrôle `edtResultat`. Enfin, afin de pouvoir saisir une nouvelle expression, le slot doit réinitialiser l'expression mathématique en affectant `nullptr` à `m_pResult` puis passer à `true` le booléen `m_bJustValidated`. Une implémentation possible est :

```
AppendOperator(nullptr);


ui.edtOperation->setText((m_pResult->toString() + " =").c_str());
std::stringstream stream;
stream << std::setprecision(std::numeric_limits<long double>::digits10)
        << m_pResult->getVal();
ui.edtResultat->setText(stream.str().c_str());

m_pResult = nullptr;
m_bJustValidated = true;
```

Enfin, le slot `setScientificMode(bool)` doit réinitialiser la calculatrice en passant à `nullptr` l'expression mathématique, en effaçant le texte dans la zone `edtOperation` et en plaçant « 0 » dans le texte du résultat `edtResultat`. Ensuite, en fonction du mode, elle affecte un nouveau pointeur intelligent à la donnée membre `m_pOpConstr` de type `bakCalc::ExpressionConstructorSci<valType>` pour un mode scientifique ou `bakCalc::ExpressionConstructorStd<valType>` pour un mode standard. Voici une implémentation possible :

```
m_pResult = nullptr;
ui.edtOperation->setText("");
ui.edtResultat->setText("0");

if (bScientific)
    m_pOpConstr = std::make_shared<bakCalc::ExpressionConstructorSci<valType>>();
else
    m_pOpConstr = std::make_shared<bakCalc::ExpressionConstructorStd<valType>>();
```

 Créez donc ces slots puis compilez et testez, la calculatrice doit être fonctionnelle.

Exercice 3 Une calculette avec historique des commandes

Dans l'exercice précédent, vous avez fini d'implémenter la classe `QCalc` qui représente une calculette dont l'interface graphique est gérée par Qt. Concrètement, cela signifie que votre classe `QCalc` est une classe dérivée de `QWidget`. À ce titre elle peut être incorporée dans une interface graphique de plus haut niveau. C'est ce que nous allons réaliser dans cet exercice. Nous souhaitons incorporer notre calculette dans une interface qui affichera l'historique des opérations tapées.

Pour cela, vous allez par la suite créer un nouvel objet d'interface qui utilisera une instance de votre classe `QCalc`. Ainsi, le diagramme de classe de votre application sera le suivant :

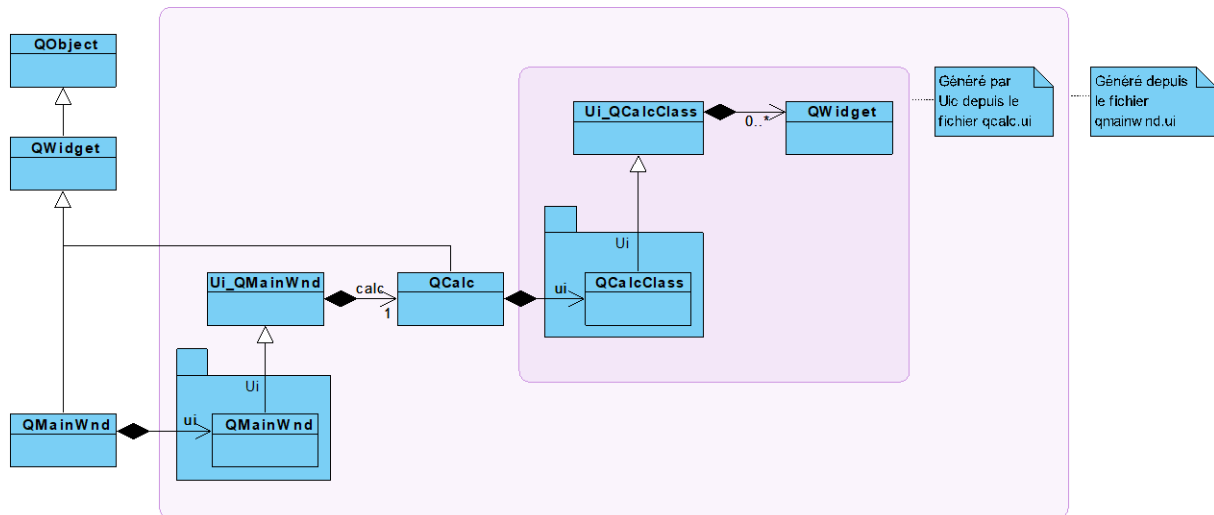


Diagramme de classe que vous obtiendrez à la fin de cet exercice

3.1 Ajout d'une nouvelle classe Qt GUI

- Ajoutez à votre projet une nouvelle classe Qt GUI en sélectionnant **Projet / Add Qt Class...** ou **Qt / Classe d'interface graphique Qt Designer** en faisant un clic droit sur le projet puis en choisissant **Add New...**
- Sélectionnez une classe de type **Qt GUI Class** ou **Qt / Classe d'interface graphique Qt Designer**
- Nommez-la `QMainWnd`
- Faites-la hériter de `QWidget`
- Validez l'assistant
 - ➔ Cela ajoute à votre projet les fichiers suivants :
 - (1) `qmainwnd.ui` : fichier de description de l'interface utilisateur éditables par Qt Designer
 - (2) `qmainwnd.h / .cpp` : déclaration et définition de votre nouvelle classe
- Modifiez le fichier `main.cpp` pour que votre application s'ouvre sur cette nouvelle interface, plutôt que sur la calculette.
- Compilez / exécutez. Vous devriez obtenir une fenêtre vide.

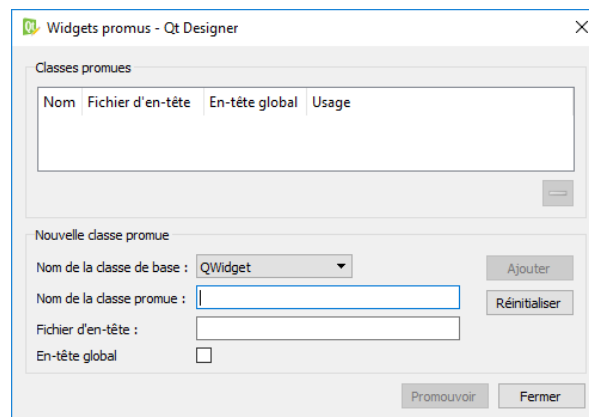
3.2 Incorporation de la calculette dans la nouvelle GUI

- Ouvrez le fichier `qmainwnd.ui` à l'aide de Qt Designer.

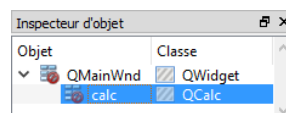
Nous devons insérer notre calculette sur cette fenêtre. Évidemment, Qt Designer ne connaît pas le widget `QCalc`, puisqu'il est de notre cru. Pour pouvoir l'insérer, il faut ajouter un simple

QWidget dans la fenêtre puis spécifier que ce **QWidget** est en réalité un objet de type **QCalc**. Pour faire cela, procédez comme suit :

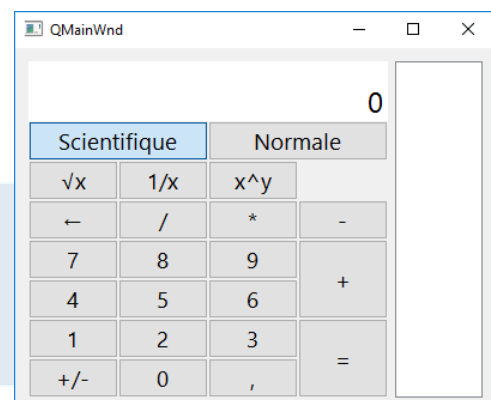
- Ajoutez un Widget dans la fenêtre en faisant glisser un **Containers** / **Widget**
 - ➔ Dans l'éditeur de propriétés, nommez cet objet « **calc** » de façon à ce que nous puissions le retrouver simplement plus tard dans le code.
- Faites un clic droit sur ce nouveau widget puis choisissez « **Promouvoir en...** ». La fenêtre suivante s'affiche :



- ➔ Dans le champ « nom de la classe promue », entrez « **QCalc** ». Vérifiez que le nom du fichier d'en-tête correspond bien au fichier de déclaration de la classe **QCalc**.
- ➔ Cliquez sur **Ajouter**
- ➔ Cliquez sur **Promouvoir**
 - ➔ Dorénavant, le widget que vous avez ajouté graphiquement sera en réalité un objet de type **QCalc**. Cela vous est confirmé par l'inspecteur d'objet :



- Spécifier un layout horizontal à la fenêtre principale en sélectionnant le fond puis en cliquant sur le bouton
- Ajoutez un **List Widget** (attention, pas un **List View**) sur la droite de la fenêtre. Cet objet affichera l'historique des opérations entrées.
 - ➔ Nommez cet objet **lsthisto**.
- Enregistrez votre fichier **qmainwnd.ui** puis compilez et exécutez votre solution.
 - ➔ Vous devez obtenir la fenêtre ci-contre.

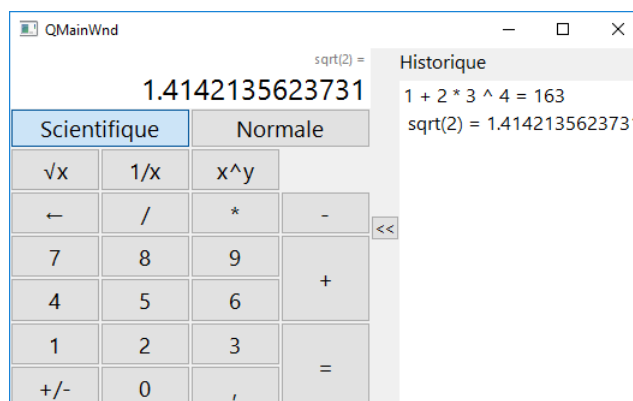


Vous pouvez améliorer cette interface en ajoutant un ou deux boutons pour afficher / masquer l'historique, régler les marges et espacements entre les zones, etc...

3.3 Ajout des opérations dans l'historique

Il faudrait qu'à chaque appuie sur le bouton de validation (« = »), une nouvelle ligne soit insérée dans l'historique. Pour cela, nous pouvons émettre un signal depuis l'objet de type **QCalc** lors de l'appui du bouton. Ce signal devrait être connecté à un slot de la fenêtre principale pour insérer l'opération dans l'historique. Nous allons donc réaliser cela :

- ☐ Modifiez la classe **QCalc** et sa méthode **QCalc::btnValid()** de façon à émettre un signal « **eval** » qui aura en paramètre une chaîne de caractères de la forme « **opération = résultat** »
- ☐ Ajoutez à la classe **QMainWnd** un slot public nommé **AppendToHist()**, prenant en paramètre une chaîne de caractères.
 - ➡ Ce slot devra ajouter au List Widget **lstHisto** un élément construit grâce à cette chaîne de caractères.
 - ➡ Utilisez la méthode **QListWidget::addItem()** sur l'objet **lstHisto** pour ajouter un nouvel élément à la liste.
- ☐ Dans le constructeur de la classe **QMainWnd**, établissez la connexion entre le signal **eval()** de l'objet **calc** de la GUI au slot **AppendToHist()** de l'objet courant.
- ☐ Compilez, exécutez
 - ➡ Vous devez obtenir l'application totalement fonctionnelle.



Exercice 4 Une interface « moderne »

Qt propose un mécanisme de mise en forme puissant, sur le même principe que les feuilles de style CSS (Cascading Style Sheets) des standards du web. Chaque Widget dispose d'une propriété **stylesheet** dans laquelle il est possible de définir son style. Le principe en cascade signifie que les propriétés de style définies pour un widget s'appliquent automatiquement à tous ces enfants (au sens de la relation parent-enfant du **QObject** sous-jacent). Ainsi, en affectant une feuille de style à l'objet principal de type **QMainWnd**, elle s'appliquera également à tous ces enfants, donc la calculatrice et l'historique. Référez-vous à l'aide en ligne de Qt pour le détail des styles applicables à chaque type de Widget : <http://doc.qt.io/qt-6/stylesheet.html>.

- ☐ À l'aide Qt Designer, définissez une feuille de style à l'objet principal dans le fichier **qmainwnd.ui**.
- ➡ Tentez d'obtenir le style ci-contre.

