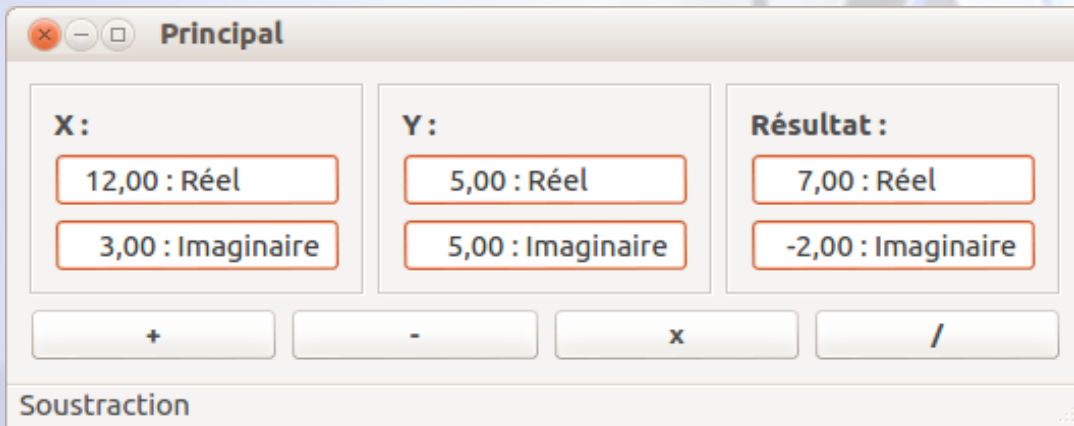




Ce projet permet de valider le cours concernant l'utilisation de fonctions afin de réduire la complexité au travers de la mise en œuvre du projet sur le calcul de nombres complexes.

Chaque nombre complexe est ici représenté par une **structure**.



## x PRINCIPAL.UI

The image shows the Qt Designer interface for the 'Principal.UI' widget. On the left, the widget is displayed with a grid layout containing input fields for real and imaginary parts of X and Y, and a result field. Below the widget is the Signal and Slot editor.

Émetteur	Signal	Receveur	Slot
xReel	valueChanged(double)	Principal	changerReelX(double)
xImaginaire	valueChanged(double)	Principal	changerImaginaireX(double)
yReel	valueChanged(double)	Principal	changerReelY(double)
yImaginaire	valueChanged(double)	Principal	changerImaginaireY(double)
boutonPlus	clicked()	Principal	choixPlus()
boutonMoins	clicked()	Principal	choixMoins()
boutonMultiplication	clicked()	Principal	choixMultiplier()
boutonDivision	clicked()	Principal	choixDiviser()

On the right, the Object Explorer shows the hierarchy of the widget:

- Principal (QMainWindow)
  - centralWidget (QWidget)
    - gridLayout (QGridLayout)
      - boutonDivision (QPushButton)
      - boutonMoins (QPushButton)
      - boutonMultiplication (QPushButton)
      - boutonPlus (QPushButton)
      - frame (QFrame)
        - label (QLabel)
        - xImaginaire (QDoubleSpinBox)
        - xReel (QDoubleSpinBox)
        - frame\_2 (QFrame)
          - label\_2 (QLabel)
          - yImaginaire (QDoubleSpinBox)
          - yReel (QDoubleSpinBox)
          - frame\_3 (QFrame)
            - label\_3 (QLabel)
            - resultatImaginaire (QDoubleSpinBox)
            - resultatReel (QDoubleSpinBox)
- barreEtat (QStatusBar)

At the bottom, the Properties panel shows the 'frame : QFrame' properties:

Propriété	Valeur
frameShape	Box
frameShadow	Sunken
lineWidth	1
midLineWidth	0

The bottom status bar shows: 1 Problèmes 2 Résultat de la recherche 3 Sortie de l'application 4 Sortie de compilation



```
x PRINCIPAL.H
#ifndef PRINCIPAL_H
#define PRINCIPAL_H

#include <QMainWindow>
#include "ui_principal.h"

//----- Déclarations personnelles -----
struct Complexe
{
    double reel, imaginaire;
};

Complexe addition(const Complexe &c1, const Complexe &c2);
Complexe soustraction(const Complexe &c1, const Complexe &c2);
Complexe multiplication(const Complexe &c1, const Complexe &c2);
Complexe division(const Complexe &c1, const Complexe &c2);

double module(const Complexe &c);
double argument(const Complexe &c);

enum Operation {Plus, Moins, Multiplier, Diviser};

//----- Déclaration de la classe représentant la fenêtre -----
class Principal : public QMainWindow, public Ui::Principal
{
    Q_OBJECT

public:
    Principal(QWidget *parent = 0);
private:
    Complexe x, y, resultat;
    Operation operation;
private:
    void calcul();
    void message();

private slots:
    void changerReelX(double valeur);
    void changerImaginaireX(double valeur);
    void changerReelY(double valeur);
    void changerImaginaireY(double valeur);
    void choixPlus();
    void choixMoins();
    void choixMultiplier();
    void choixDiviser();
};

#endif // PRINCIPAL_H
```

```
x PRINCIPAL.CPP
#include <math.h>
#include "principal.h"

Principal::Principal(QWidget *parent) : QMainWindow(parent)
{
    setupUi(this);
    x.reel = x.imaginaire = 0.0;
    y.reel = y.imaginaire = 0.0;
    operation = Plus;
    barreEtat->showMessage("Addition");
}
```





```

void Principal::changerReelX(double valeur)    { x.reel = valeur; calcul(); }
void Principal::changerImaginaireX(double valeur) { x.imaginaire = valeur; calcul(); }
void Principal::changerReelY(double valeur)    { y.reel = valeur; calcul(); }
void Principal::changerImaginaireY(double valeur) { y.imaginaire = valeur; calcul(); }

void Principal::choixPlus()                    { operation = Plus; calcul(); }
void Principal::choixMoins()                   { operation = Moins; calcul(); }
void Principal::choixMultiplier()              { operation = Multiplier; calcul(); }
void Principal::choixDiviser()                 { operation = Diviser; calcul(); }

void Principal::calcul()
{
    switch (operation)
    {
        case Plus : resultat = addition(x, y); barreEtat->showMessage("Addition"); break;
        case Moins : resultat = soustraction(x, y); barreEtat->showMessage("Soustraction"); break;
        case Multiplier : resultat = multiplication(x, y); barreEtat->showMessage("Multiplication"); break;
        case Diviser : resultat = division(x, y); barreEtat->showMessage("Division"); break;
    }
    resultatReel->setValue(resultat.reel);
    resultatImaginaire->setValue(resultat.imaginaire);
}

Complexe addition(const Complexe &c1, const Complexe &c2)
{
    Complexe c;
    c.reel = c1.reel + c2.reel;
    c.imaginaire = c1.imaginaire + c2.imaginaire;
    return c;
}

Complexe soustraction(const Complexe &c1, const Complexe &c2)
{
    Complexe c;
    c.reel = c1.reel - c2.reel;
    c.imaginaire = c1.imaginaire - c2.imaginaire;
    return c;
}

Complexe multiplication(const Complexe &c1, const Complexe &c2)
{
    Complexe c;
    double mod = module(c1) * module(c2);
    double arg = argument(c1) + argument(c2);
    c.reel = mod*cos(arg);
    c.imaginaire = mod*sin(arg);
    return c;
}

Complexe division(const Complexe &c1, const Complexe &c2)
{
    Complexe c;
    double mod = module(c1) / module(c2);
    double arg = argument(c1) - argument(c2);
    c.reel = mod*cos(arg);
    c.imaginaire = mod*sin(arg);
    return c;
}

double module(const Complexe &c) { return sqrt(c.reel*c.reel + c.imaginaire*c.imaginaire);}

double argument(const Complexe &c)
{
    if (c.imaginaire == 0.0) return 0.0;
    if (c.reel == 0.0) return c.imaginaire>0.0 ? M_PI_2 : -M_PI_2;
    return atan(c.imaginaire / c.reel);
}
    
```

