TP 1 - Premiers programmes en C++

Objectifs : déclaration et implémentation d'une classe élémentaire, utilisation des flots **`cin`** et **`cout`**, utilisation de composants prédéfinis : type "complexe", utilisation du type **`bool`**.

Outils utilisés Dans cette série de TP, vous travaillerez avec une IDE ("*Integrated Development Environment*" qu'on peut traduire par "environnement de développement intégré"). Ceci vous facilitera les tâches d'éditions multi-fichiers et de compilation. Vous utiliserez celle qui vous sera indiquée par l'enseignant.

Introduction Pour ce premier TP, vous allez mettre en œuvre une classe simple, permettant de représenter, résoudre, et afficher une équation du 2nd degré :

$$ax^2 + bx + c = 0$$

Dans un premier temps, on se limitera au cas des réels, puis on ajoutera la résolution dans le plan complexe. Votre projet sera constitué de **3 fichiers** : **`main.cpp`**, **`equation.h`**, et **`equation.cpp`**

1 Résolution simple

1. En utilisant l'explorateur Windows (touche « Windows » + E), créer sur le disque T : un dossier **`Info3`** (SANS ESPACES). Dans ce dossier, créer un dossier **`TP1`**, et dans celui-ci, créer un dossier **`tp1A`**

2. Lancer l'IDE et créer un nouveau projet, de type "console", en lui donnant le nom **`tp1a`**. Un squelette pour une fonction **`main()`** est généré. Sauvegarder le fichier avec le nom **`main.cpp`**, puis ajouter au début de la fonction la ligne :

```
cout << "main() : debut\n";
```

3. Taper F9 pour lancer le *Build* du projet (compilation + édition de liens) et l'exécution : vous devez avoir une fenêtre type "console" qui s'affiche.

4. Créer deux nouveaux fichiers sources en les ajoutant au projet (bouton "Fichier Source"), et les sauvegarder en **`equation.h`** et **`equation.cpp`** (penser à mettre un commentaire en 1 re ligne pour les identifier !)

5. Dans le fichier **`equation.h`**, déclarer une classe **`Equation`**, permettant de gérer une équation du second degré. Elle sera pourvue de :

   — 6 attributs privés :
   - 3 **`double`** [1] pour mémoriser les coefficients : **`a`**, **`b`**, **`c`**
   - 2 **`double`** pour mémoriser les racines : **`r1`** et **`r2`**.
   - 1 entier **`nbRacinesReelles`**, qui ne pourra prendre que les valeurs 0, 1, ou 2.

   — 3 méthodes publiques ("interface" de la classe)
   - 1 constructeur à 3 arguments de type **`double`** :
     `Equation( double, double, double );`
     (les 3 arguments seront les 3 coefficients souhaités de l'équation)
   - 1 méthode **`void resoudre();`**
   - 1 méthode **`void afficher();`**

   Cette classe devra permettre d'écrire comme programme principal (fonction **`main()`** dans le fichier **`main.cpp`**) les lignes suivantes [2] :

```
#include "equation.h"
int main()
{
    Equation equ( 4, 5, 6 ); // creation Equation
    equ.afficher();          // affichage valeurs
}
```

6. Ecrire le code des trois méthodes de la classe dans le fichier **`equation.cpp`**. Quelques indications :
   - Penser à ajouter en tête de ce fichier **`#include "equation.h"`**
   - Le constructeur devra initialiser les attributs a, b, c avec les valeurs transmises, puis fera appel à la méthode **`resoudre()`**.
   - La méthode **`resoudre()`** commencera par initialiser **`nbRacinesReelles`** à 0, calculera le discriminant Δ, et, si positif ou nul, les racines, puis assignera une valeur à **`nbRacinesReelles`**.
   - La méthode **`afficher()`** fera un affichage dans le flot **`cout`**. Il faudra donc ajouterau début du fichier **`equation.cpp`** les lignes :

1. `double` : type réel à virgule flottante sur 64 bits
2. La fonction **`main()`** doit renvoyer un entier mais en pratique on peut s'en abstenir, la valeur 0 est renvoyée par défaut.

```
#include <iostream>
using namespace std;
```

Cette méthode devra afficher le texte suivant, suivant la valeur de **nbRacinesReelles**[3] :

```
L'Equation 4 x^2 + 5 x + 6 = 0  n'a pas de racines réelles
```

Ou :

```
L'Equation 1 x^2 + 2 x + 1 = 0  a une racine double
Racine double = -1
```

Ou :

```
L'Equation 1 x^2 + 4 x + 3 = 0  admet 2 racines
Racine 1 = -3 , Racine 2 = -1
```

Pour le calcul du discriminant, on fera appel à la fonction **sqrt()**, fonction de la bibliothèque standard C, déclarée dans le fichier d'en-tête **cmath**, et qu'il faudra inclure dans le fichier où elle sera utilisée.

7. Tester et vérifier le fonctionnement en recompilant le programme avec les valeurs (4,5,6), (1,2,1) et (1,4,3) pour a,b,c.

2 Saisie et rebouclage

On souhaite pouvoir disposer d'une méthode **bool Saisie()** permettant de saisir au clavier les coefficients. Après la saisie, cette méthode devra tester les trois valeurs saisies :

— si on a tapé trois zéros, renvoyer **false**
— sinon, cette méthode : (1) appellera la méthode **resoudre()**, puis : (2) renverra **true**.

1. Ajouter la déclaration et la définition de cette méthode.
2. Du coup, le constructeur à trois arguments devient superflu. **Ajouter** à la classe un deuxième constructeur sans arguments, qui se contentera de mettre **nbRacinesReelles** à 0 (Ne pas effacer le premier constructeur !).
3. Vérifiez le fonctionnement avec le programme suivant (dans le fichier **main.cpp**) :

3. et avec évidemment les coefficients réels saisis !

```
int main()
{
    Equation equ;
    equ.Saisie();
    equ.afficher();
}
```

4. Si ce code fonctionne, on peut aller plus loin : intégrez l'appel aux méthodes **Saisie()** et **afficher()** dans une boucle, dont on ne pourra sortir qu'en tapant trois zéros lors de la saisie. Attention, il ne faudra faire l'affichage des solutions QUE si l'utilisateur n'a pas saisi trois valeurs nulles.

3 Gestion des racines complexes

Si le discriminant Δ est négatif, on peut définir deux racines complexes, qui auront comme expression :

$$z_1 = \frac{-b - i\delta}{2\,a} \qquad z_2 = \frac{-b + i\delta}{2\,a} \qquad \text{avec} \qquad \delta = \sqrt{-\Delta}$$

On va modifier la classe de façon à ce qu'elle puisse traiter ce cas en utilisant le type complexe de la bibliothèque standard C++.

1. Avec l'explorateur Windows, créez dans le dossier TP1 un dossier **tp1_2**
2. Copier l'ensemble des fichiers du dossier **tp1_1** dans le fichier **tp1_2**
3. Dans le dossier **tp1_2**, renommez le fichier **tp1_1.cbp** en **tp1_2.cbp**
4. Ouvrir le projet **tp1_2.cbp**, et dans le dialogue "Options de projet" (bouton dans la barre d'outils), renommez le projet en **tp1_2**
5. Ajouter à la classe deux attributs **rc1** et **rc2** (racines complexes), qui seront de type **complex <double>**.
6. Modifier la méthode **resoudre()**, de façon qu'elle calcule soit **r1** et **r2**, soit **rc1** et **rc2**, selon le signe de Δ (voir en annexe l'utilisation du type **std::complex<>**).
7. Modifier la méthode **afficher()**, de façon à ce qu'elle donne les racines complexes si il n'y a pas de racines réelles.

8. Après validation par l'enseignant du fonctionnement, prenez du temps pour commenter la classe, puis copiez dans le bloc-notes les deux fichiers **equation.h** et **equation.cpp**, et imprimer (sur une seule page si possible).

```
//-------------------------------------------------
// Cette méthode fait ceci et cela...
void Equation::resoudre()
{
     nbRacinesReelles = 0;
...
}
//-------------------------------------------------
```

Annexe : Type parametré `complex` de la bibliothèque standard

La bibliothèque standard C++ fournit un certain nombre de classes qui permettent d'accélérer les développements en évitant d'avoir à "réinventer la roue" à chaque fois. En l'occurrence, il existe une classe **complex**, permettant de manipuler facilement des nombres complexes.

Ces classes utilisent un mécanisme particulier du C++ permettant de **paramétrer** ces classes en fonctions d'un type de donnée. Par exemple, on pourra travailler avec des complexes en simple précision (**float**), ou en double précision (**double**). La création d'un objet de ce type se fera avec la syntaxe suivante :

```
#include <complex> // en haut du fichier
...
// création d'un nombre complexe c1 en simple précision
     std::complex <float> c1;
// création d'un nombre complexe c2 en double précision
     std::complex <double> c2;
```

La mention **std::** devant le nom de la classe indique qu'on veut utiliser le "complex" de la bibliothèque standard, et pas une autre classe de même nom [4].

Ces classes sont dotées d'un constructeur, et on pourra assigner une valeur à un objet de ce type avec la syntaxe suivante :

```
     c1 = complex <float> (3,5);    // c1 = 3 + 5 i
ou : c1 = complex <float> ( a+ y, (b-2) *5 );
```

Les opérations arithmétiques usuelles (+ - * /) sont toutes valides avec ces classes, on pourra manipuler ces valeurs comme n'importe quelle variable simple. De plus, l'opérateur d'insertion dans un flux est également défini pour ces classes, ce qui fait qu'on peut afficher une valeur complexe très simplement :

```
     cout << "c1 = " << c1 << endl;
```

On peut accéder à la partie imaginaire et à la partie réelle avec des accesseurs, qui sont implémentés sous forme de fonctions :

```
     float a = real( c1 );          // c1 = a + i b
     float b = imag( c1 );
```

4. en effet, on pourrait aussi créer sa propre classe **complex** !