Responsable pédagogique	AF	**AM**	OP	PM
Période	Sem1	Sem2	**Sem3**	Sem4
Volume horaire	Cours/TD		TP	
			6	

[CPP1]
C(pas)Complex !

Indicateur temporel (hors rédaction du compte-rendu) :

questions	1h	2h	3h	4h	5h	6h
1 .. 4						
5 .. 7						
8 .. 9						
10 .. 12						
13 .. 15						
16						

Documents à rendre : Compte-rendu contenant à minima les sources (avec en-tête standard) et les résultats obtenus.
Note : Pour ce TP, toutes les méthodes propres à la classe doivent être implémentées en *inline*.

Petit mémento relatif aux calculs sur les nombres complexes

$u = a + ib$
$v = a' + ib'$
$\overline{u} = a - ib$
$u + v = a + a' + i.(b + b')$
$u - v = a - a' + i.(b - b')$
$u * v = aa' - bb' + i.(ab' + ba')$
$u / v = (u * \overline{v}) / (v * \overline{v}) = [\ aa' + bb' + i.(ba' - ab')\ ] / (a'^2 + b'^2)$

Développement de CComplex v0.2

1. Écrire la base de la classe `CComplex` à partir des indications du cours Langage C++ part. 2, en commençant par la définition des attributs et accesseurs (slides 4 et 5).
2. Ajouter la méthode `asString()` et la surcharge de l'opérateur d'injection (slides 8 et 9).
3. Mettre en place les éléments statiques, les constructeurs et le destructeur (slides 14 et 15).
4. Valider l'ensemble obtenu par un petit programme de test pertinent : affichage de la version, création, initialisation et affichage d'une instance, affichage du nombre d'objets « vivants »...

Ajout de méthodes pour les opérations de base

5. Enrichir la classe avec les ressources relatives aux opérations d'addition (slides 17 et 18). Ajouter ensuite les méthodes permettant les autres opérations de base entre le complexe et un couple de valeurs réelle et imaginaire :

```
CComplex& sub(double re, double im = 0 ) ;      // soustraction
CComplex& mult(double re, double im = 0 ) ;     // multiplication
CComplex& div(double re, double im = 0 ) ;      // division
```

6. Implémenter une méthode retournant le conjugué du nombre complexe :

```
CComplex conj() const ;
```

7. Valider les méthodes avec le code de test suivant :

```
cout << CComplex::Version() << endl << endl ;
```

```
CComplex c0(-1, 2 ) ;
CComplex c1, c2 ;

c0.add(3, 2 ) ;
c1 = c0.conj() ;
c1.sub(1, -1 ) ;
c1.mult(0.5, -2 ) ;
c2 = c0 ;
c2.div(-2, 0.5 ) ;

cout << "c0 = " << c0 << endl ;
cout << "c1 = " << c1 << endl ;
cout << "c2 = " << c2 << endl ;
```

8. Surcharger les méthodes précédentes pour qu'elles acceptent directement un autre complexe en argument (*penser à réutiliser le code existant…*) :

```
CComplex& sub(const CComplex& c ) ;
CComplex& mult(const CComplex& c ) ;
CComplex& div(const CComplex& c ) ;
```

9. Valider les méthodes avec le code de test suivant :

```
CComplex c[3] ;   // tableau de CComplex

c[0].set(0.1, -3 ) ;
c[1].set(-1.5, 2 ) ;

c[0].add( c[1] ) ;
c[1].mult( c0 ) ;
c[2] = c2 ;
c[2].sub( c[1] ) ;
c[2].div( c1 ) ;

cout << "c3 = " << c[0] << endl ;
cout << "c4 = " << c[1] << endl ;
cout << "c5 = " << c[2] << endl ;
```

Ajout de surcharges d'opérateurs

10. Surcharger les opérateurs des 3 opérations de base afin de les adapter aux objets `CComplex` et qu'ils acceptent un deuxième opérande de type réel :

```
CComplex operator -(double v ) const ;
CComplex operator *(double v ) const ;
CComplex operator /(double v ) const ;
```

11. Ajouter les fonctions amies permettant d'utiliser les opérateurs avec un opérande gauche réel et l'opérande droit complexe :

```
friend CComplex operator -(double v, CComplex& c ) ;
friend CComplex operator *(double v, CComplex& c ) ;
friend CComplex operator /(double v, CComplex& c ) ;
```

12. Valider les méthodes avec le code de test suivant :

```
CComplex* c6 = new CComplex( 2 + c0 + 3 ) ;     // pointeur de CComplex
*c6 = ( *c6 - 1.2 ) / 2 ;
*c6 = 3.14 * *c6 ;

cout << "c6 = " << *c6 << endl ;
```

13. Surcharger maintenant l'opérateur **!** en tant que conjugué :

```
CComplex operator !() const ;
```

14. Surcharger les opérateurs des 3 opérations de base de manière à accepter un complexe en deuxième opérande (attention, aucun des opérandes de l'opération ne doit être modifié ; l'usage du constructeur de copie peut donc être nécessaire…) :

```
CComplex operator -(const CComplex& c ) const ;
CComplex operator *(const CComplex& c ) const ;
CComplex operator /(const CComplex& c ) const ;
```

15. Valider les méthodes avec le code de test suivant :

```
CComplex*   pc[2] ;     // tableau de pointeurs de CComplex

pc[0] = new CComplex( !c[2] + *c6 ) ;
*pc[0] = ( *pc[0] - c0 ) * c1 / c2 ;
cout << "c7 = " << *pc[0] << endl ;
```

16. Implémenter les méthodes nécessaires pour pouvoir écrire les lignes suivantes :

```
pc[1] = new CComplex( !c[2] ) ;

*pc[1] += *c6 ;
*pc[1] -= c0 ;
*pc[1] *= c1 ;
*pc[1] /= c2 ;
cout << "c8 = " << *pc[1] << endl ;

cout << "[ " << CComplex::Cpt() << " objet(s) ]" << endl ;
delete pc[0] ;
delete pc[1] ;
cout << "[ " << CComplex::Cpt() << " objet(s) ]" << endl ;
```

Moralité : *En écrivant une seule fois les algorithmes internes de calcul, il est possible d'étoffer l'interface d'usage des objets pour qu'ils soient utilisables de plusieurs manières !*

Annexe : *Pour vérifier les résultats affichés par le programme de test…*

	c0	c1	c2	c3	c4	c5	c6	c7	c8
re	+2.00	-5.50	-0.47	-1.40	-11.00	-1.35	+9.11	+15.20	+15.20
im	+4.00	-3.50	-2.12	-1.00	-2.00	+0.88	+6.28	-9.26	-9.26