LIFAMI – TD : Nombres complexes et transformations du plan

Objectifs : Définition d'un nombre complexe
Transformations du plan : translation, homothétie, rotation

En mathématiques, les nombres complexes forment une extension de l'ensemble des nombres réels. Un nombre complexe z se présente en général sous forme algébrique comme une somme.

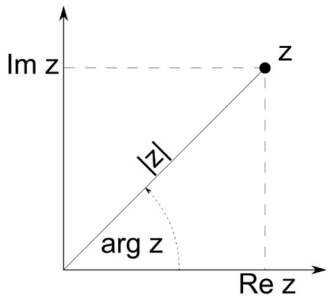
- z = x + iy, forme algébrique

où x et y sont des nombres réels quelconques et où i (l'unité imaginaire) est un nombre particulier tel que $i^2 = -1$. Le réel *x* est appelé partie réelle de *z* et se note Re(z), le réel *y* est sa partie imaginaire et se note Im(z).

Pour tout couple de réels (x,y) différent du couple (0,0), il existe un réel positif r et une famille d'angles déterminés à un multiple de 2π près tels que a = r cos(θ) et b = r sin(θ). Tout nombre complexe non nul peut donc s'écrire sous une forme trigonométrique :

- z = r (cos(θ) + i sin(θ)) avec r > 0, forme polaire
- $z = re^{i\theta}$, forme exponentielle

Le réel positif r est appelé le module du complexe z et est noté |z|. Le réel θ est appelé un argument du complexe z et est noté arg(z).



Dans un plan complexe $\mathcal{P}$ muni d'un repère orthonormé $(O; \vec{u}, \vec{v})$, l'image d'un nombre complexe $z = x + iy$ est le point *M* de coordonnées (*x*, *y*), son image vectorielle est le vecteur $\overrightarrow{OM}$. Le nombre *z* est appelé affixe du point *M* ou du vecteur $\overrightarrow{OM}$.

Un vecteur V du plan peut donc être représenté en coordonnées cartésiennes par V(x,y) ou par le nombre complexe x + iy. Un nombre complexe peut à la fois représenter un vecteur ou un point P qui n'est autre que le vecteur OP, avec O l'origine.

Les nombres complexes sont super cools !

1. Déclarez en C/C++ une structure *Complex* comportant deux champs nommés x et y.

2. Ecrivez en C/C++ les fonctions suivantes.

```
Fonction make_complex ( x, y : Reel) → Complex
Fonction make_complex_exp ( r, theta : Reel) → Complex
```

3. Soit le point P et le vecteur V représentés par les complexes p et v suivants :
   - p = x + i.y
   - v = dx + i.dy

   L'addition de ces deux complexes p+v = x+dx + i.(y+dy) correspond à la translation du point P par le vecteur V.

   En C++ il est possible d'écrire des fonctions particulières entre deux structures pour pouvoir utiliser les symboles +, - et *. Ces fonctions sont dites opérateurs d'addition, de soustraction et de multiplication. Ecrivez en C++ les opérateurs suivants.

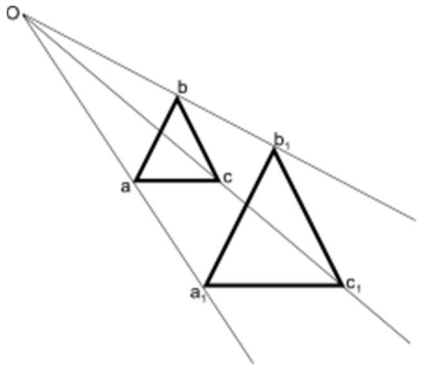
```
Fonction operator+( a,b : Complex) → Complex  // Algo
Complex operator+(Complex a, Complex b) { …   // C++

Fonction operator-( a,b : Complex) → Complex   // Algo
Complex operator-(Complex a, Complex b) { …   // C++
```

   On pourra alors écrire ceci en C++ :

```
Complex a = make_complex(1, 1);
Complex b = make_complex_exp( 1, M_PI/2) ;
Complex c;
c = a+b;
```

4. Soient le point *P* et λ un réel, l'image P' du produit $\lambda.p$ est définie par la relation OM'=λ.OP. Cette multiplication d'un nombre complexe par un scalaire s'interprète géométriquement comme une homothétie de centre *O* et de rapport λ sur le plan complexe.



   Soit le triangle ayant pour points A(1,-1), B(0,1), C(-1,-1). Multipliez ces 3 points par le scalaire $\lambda=2$ (Faites le calcul). Dessinez les deux triangles. Et si $\lambda=0.5$ ou $\lambda=0$?

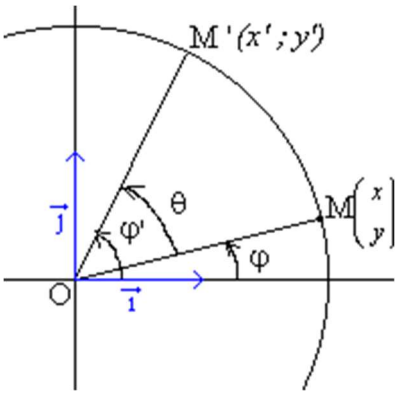
5. Un changement d'échelle (*Scale* en anglais) se fait par rapport à un centre C de coordonnée (Cx,Cy). Ecrivez les deux fonctions suivantes :

   `Complex operator*(float lambda, Complex b) // C++`
   qui multiplie un scalaire à un complexe.

   `Complex scale(Complex p, float cx, float cy, float lambda)// C++`
   qui effectue l'homothétie de centre (cx,cy) d'un facteur lambda. Pour cela il faut placer le point/complexe *p* dans le repère ayant pour origine (cx,cy), faire l'homothétie donc la multiplication, puis replacer le point dans le repère O.

6. Soit M un point représenté par le complexe p et *z* un complexe de norme un et d'argument θ (donc $z=e^{i\theta}$), l'image *M'* du produit *p.z* est définie par les relations :
   - OM' = OM
   - l'angle (OM,OM') = θ

   La multiplication d'un nombre complexe quelconque par un autre complexe de module un et d'angle θ s'interprète géométriquement comme une rotation de centre O (l'origine) et d'angle θ.



7. Soit le triangle ayant pour points A(1,-1), B(0,1), C(-1,-1). Multipliez ces 3 points par le complexe $r= e^{i\theta}$ *avec* $\theta= \pi /2$. Dessinez les deux triangles.

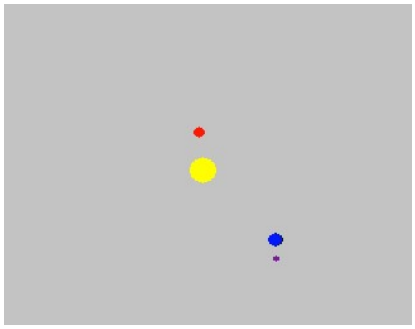
8. Ecrivez les deux fonctions suivantes.

```
Complex operator*(Complex a, Complex b)          // C++
```
qui multiplie deux complexes ensemble.

```
Complex rotate(Complex p, float cx, float cy, float theta_deg) // C++
```
qui effectue la rotation de centre (cx,cy) d'un angle theta_deg. Pour cela il faut placer le point/complexe p dans le repère ayant pour origine (cx,cy), faire la rotation donc la multiplication, puis replacer le point dans le repère O.

Saturne, ça tourne !

9. Nous allons afficher un système solaire simplifié comportant 3 planètes en plus du soleil : mars, la terre et la lune. Chaque planète est représentée par un nombre complexe représentant sa position.



Une représentation du système solaire :
le soleil en jaune, mars en rouge, la terre en bleu et la lune en gris.

Définissez la structure *SolarSystem* et écrivez les 3 fonctions suivantes.

```
Procédure init(ss : donnée-résultat SolarSystem)
```
qui initialise la position du soleil au centre de la fenêtre et les 3 planètes alignées.

```
Procédure draw(ss : donnée SoloarSystem)
```
qui affiche avec 4 couleurs différentes les 4 astres.

```
Procédure update(ss : donnée-résultat SolarSystem)
```
qui met à jour la position des 3 planètes de façon à ce que mars, la terre et la lune tournent autour du soleil et la lune tourne autour de la terre. Pour cela utilisez la procédure *rotate*.

Les polygones sont des petis lyonnais polis ! (ahahah …)

Un polygone à N cotés comporte N sommets qui seront représentés par un point/*Complex*.

10. Définissez la structure *Polygon* comportant un entier indiquant le nombre de sommets et un tableau de *Complex* de taille MAX.

11. Ecrivez la procédure qui ajoute un sommet au polygone *p*.
```
Proc polygon_add(p : donnée-résultat Polygon ; px, py : Réel)
```

12. Ecrivez la procédure qui affiche le polygone en utilisant la fonction *line*.
```
Proc polygon_draw(p : donnée Polygon)
```

13. Ecrivez le sous-programme qui calcule le centre de gravité d'un polygone.

14. Ecrivez le sous-programme qui déplace un polygone de (dx,dy).

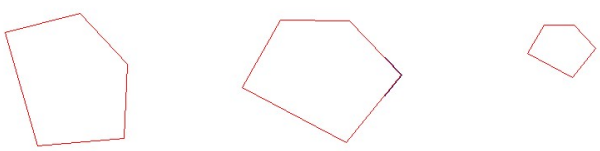
15. Ecrivez la procédure qui applique l'homothétie de centre (cx,cy) et de facteur *lambda* au polygone *p*.

```
Proc polygon_scale(p : donnée-résultat Polygon ; cx,cy : Réel ;
lambda : Réel)
```

16. Ecrivez la procédure qui fait tourner le polygone *p* d'un angle *theta* par rapport au centre (cx,cy).

```
Proc polygon_rotate(p : donnée-résultat Polygon ; cx,cy : Réel ;
theta : Réel)
```

17. Ecrivez les sous-programmes qui modifient le polygone *p* de manière à en faire le symétrique par rapport à X et Y. Il faut le recentrer au milieu de la fenêtre



Rotation et changement d'échelle d'un polygone avec des nombres complexes