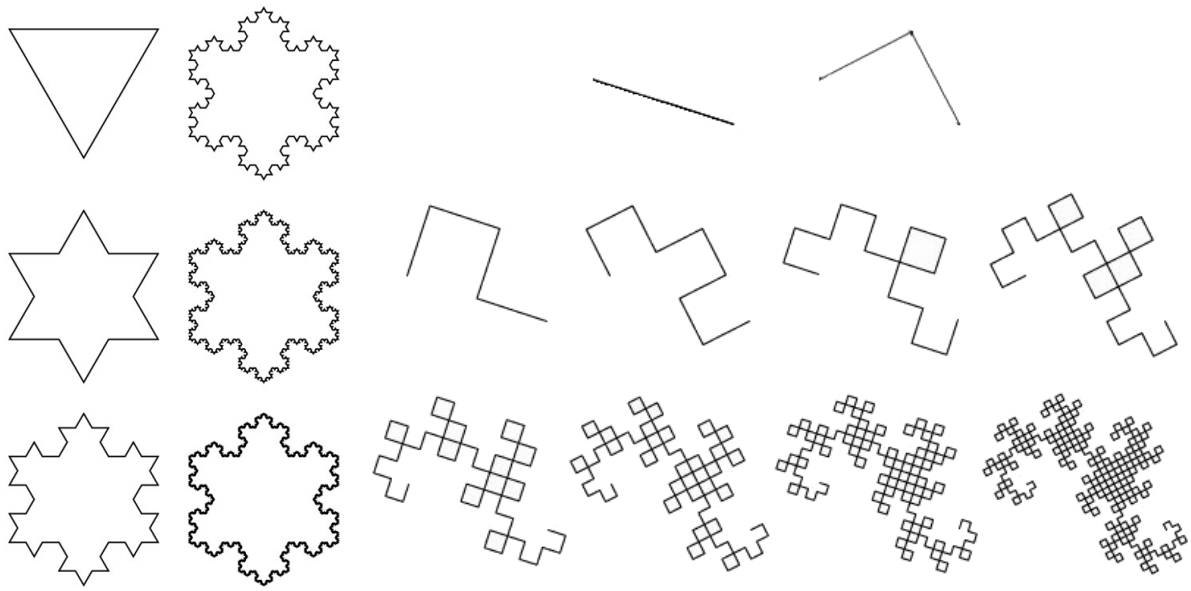
TD/TP de Programmation – Planche 4
Affichage de fractales par fonctions récursives

On souhaite afficher quelques fractales classiques telles que le flocon de Koch et le dragon de Heighway ci-dessous. Ces fractales sont obtenues en brisant un segment de droite en un motif constitué de plusieurs segments, et en ré-appliquant récursivement le processus sur les segments obtenus.



Structure de Donnée

On reprend la structure `Coord` déjà vue dans le TD/TP du serpentin, et qui modélise un point ou un vecteur dans le plan. Il représentera également un nombre complexe dans cette planche.

```
typedef struct Coord {
  double x, y;
} Coord;
```

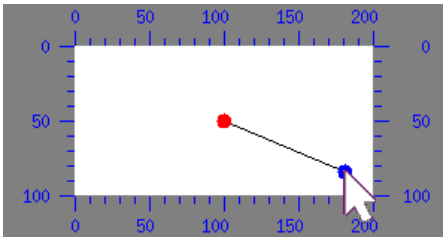
On rappelle les fonctions élémentaires de calcul vectoriel qui avaient été codées à cette occasion :

```
Coord Coord_FromXY    (double x, double y);

Coord Coord_Sum        (Coord p, Coord q);
Coord Coord_Difference (Coord p, Coord q);

double Coord_DotProduct (Coord p, Coord q);
double Coord_Length2    (Coord p);
double Coord_Length     (Coord p);

Coord Coord_ScaledBy   (Coord p, double lambda);
Coord Coord_Normalized (Coord p);
```

On rappelle également les fonctions utilitaires `bx_read_mouse()`, `bx_draw_line()` et `bx_draw_circle()` réexprimées avec la structure `Coord` afin d'éviter des manipuler directement les abscisses et les ordonnées :

```
Coord Util_MouseCoord (bx_mouse mouse);
void Util_DrawLine   (bx_window window, Coord p, Coord q);
void Util_DrawCircle (bx_window window, Coord center, double radius, int filled);
```

Librairie `bx`



Voici la démo déjà présentée dans le TD/TP du serpentin, réexprimée avec `Coord`. Ce programme affiche un disque rouge centré dans le canvenas de la fenêtre, un disque bleu centré sur les coordonnées de la souris, et un segment noir reliant les deux disques.

```
#include <bx.h>  /* UNIQUE HEADER A INCLURE POUR UTILISER BX */

int main (void)
{
  bx_init (); // A APPELER AVANT TOUTE AUTRE FONCTION BX
  char win_title []= "demo";
  Coord dim= Coord_FromXY (200, 100);
  bx_window win= bx_create_window (win_title, 0, 0, dim.x, dim.y);
  for (;;)  {
    Coord center= Coord_ScaledBy (dim, 0.5);
    bx_mouse mouse= bx_read_mouse (win);
    Coord cursor= Util_MouseCoord (mouse);
    bx_clear_canvas (win, bx_white());
    bx_set_color (bx_black()); Util_DrawLine   (win, center, cursor);
    int radius= 5;
    bool filled= true;
    bx_set_color (bx_red  ()); Util_DrawCircle (win, center, radius, filled);
    bx_set_color (bx_blue ()); Util_DrawCircle (win, cursor, radius, filled);
    int milliseconds_delay= 10;
    bx_show_canvas (win, milliseconds_delay);
  }
  bx_loop (); // BOUCLE D'EVENEMENT INFINIE
  return 0;
}
```

Rappel : Pour compiler et exécuter un programme avec `bx` sur les machines en salle de TP, il faut que le fichier de configuration `.bashrc` situé à la racine de votre compte contienne la ligne :

```
source ~barbanchon/BX/BX.CONF
```

Rappel : Si votre source C et votre exécutable s'appellent respectivement `demo.c` et `demo`, alors la commande de compilation à utiliser sera :

```
clang -W -Wall -std=c99 -pedantic    demo.c -o demo    $(bx-config)
```

Une documentation se trouve sur le Web à l'adresse :

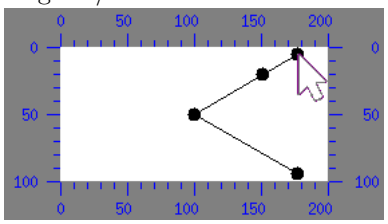
```
http://www.dil.univ-mrs.fr/~regis/bx/html
```

1 Rotation et interpolation linéaire

Écrire les fonctions suivantes : `Coord_Lerp()` qui calcule l'interpolation linéaire de deux points `p` et `q` pour un coefficient `alpha` (le résultat vaut `p` si `alpha=0.0` et vaut `q` si `alpha=1.0`) ; `Coord_FromAngle()` qui renvoie le point associé au nombre complexe $e^{i\theta}$; `Coord_FromPolar()` qui retourne le point associé au nombre complexe $\rho e^{i\theta}$; `Coord_ComplexMul()` qui retourne le produit de deux nombres complexes `p` et `q`. En déduire `Coord_Rotate()` qui retourne `p` transformé par la rotation d'angle `theta` autour de l'origine, ainsi que `Coord_RotateAround()` qui effectue la même rotation autour d'un point `center`.

```
Coord Coord_Lerp (Coord p, double alpha, Coord q);

Coord Coord_FromAngle (double theta);
Coord Coord_FromPolar (double rho, double theta);

Coord Coord_ComplexProduct  (Coord p, Coord q);
Coord Coord_RotatedBy       (Coord p, double theta);
Coord Coord_RotatedByAround (Coord p, double theta, Coord center);
```

Tester ces fonctions en modifiant le programme de démo `bx`. Par exemple, `MainTest_LerpAndRotationDemo()` peut afficher un point au deux-tiers du segment `(center,cursor)` ainsi qu'un deuxième segment `(center,rot)` où `rot` est obtenu par la rotation d'angle $\pi/3$ de `cursor` autour de `center`.




```
int MainTest_LerpAndRotationDemo (void);
int main (void) { return MainTest_LerpAndRotationDemo (); }
```

2 Flocon de Koch

Le flocon de Koch de niveau 0 est un triangle équilatéral (3 segments). Le flocon de niveau 1 est une étoile à 6 branches (12 segments). Pour obtenir un flocon de niveau $n+1$ à partir d'un flocon de niveau n, il suffit de fracturer chaque segment `(p,q)` en 4 segments `(p,a)`, `(a,c)`, `(c,b)`, `(b,q)`, tel que `(a,b)` forme le tiers central de `(p,q)` et `c` est la rotation d'angle $2\pi/3$ de `q` autour de `b`.

Écrire dans un premier temps la fonction `Util_DrawKochStep()` qui affiche la fracturation de niveau 1 d'un segment `(p,q)` en 4 segments. Puis modifier cette fonction pour obtenir la fonction récursive `Util_DrawKochLine()` qui affiche la fracturation de niveau `level` d'un segment `(p,q)` (la fracturation de niveau 0 est le segment `(p,q)` lui-même). En déduire la fonction `Util_DrawKochFlake()` qui affiche un flocon de niveau `level` centré en `center` et où `corner1` est l'un des 3 coins de son triangle de niveau 0.

```
void Util_DrawKochStep  (bx_window window, Coord p, Coord q);
void Util_DrawKochLine  (bx_window window, Coord p, Coord q, int level);
void Util_DrawKochFlake (bx_window window, Coord center, Coord corner1, int level);
```

Tester ces 3 fonctions en modifiant le programme de démo `bx`. Prendre le centre de la fenêtre pour `p` ou `center`, ainsi que le curseur de la souris pour `q` ou `corner1`. On peut utiliser la boucle pour faire varier le niveau de récursion de façon cyclique et ainsi obtenir un effet d'animation.

```
int MainTest_KochStepDemo  (void);
int MainTest_KochLineDemo  (void);
int MainTest_KochFlakeDemo (void);

int main (void) { return MainTest_KochFlakeDemo (); }
```

3 Dragon de Heighway

Le dragon de Heighway de niveau 0 est formé d'un unique segment `(p,q)`. Le dragon de niveau $n+1$ est obtenu à partir du dragon de niveau n en fragmentant chaque segment `(p,q)` en 2 segments `(p,t)` et `(q,t)` de sorte que `(p,t,q)` soit un triangle isocèle rectangle en `t`. Autrement dit, si `m` est le milieu de `(p,q)`, alors `t` est obtenu par la rotation d'angle $\pi/2$ de `q` autour de `m`. Attention, les segments sont ici orientés : `(q,t)` est le segment de sens opposé à `(t,q)`.

Écrire la fonction `Util_DrawDragonStep()` qui affiche la fracturation de niveau 1 d'un segment `(p,q)` en deux segments. Modifier celle-ci afin d'obtenir la fonction récursive `Util_DrawDragon()` qui affiche un dragon de niveau `level`.

```
void Util_DrawDragonStep (bx_window window, Coord p, Coord q);
void Util_DrawDragon     (bx_window window, Coord p, Coord q, int level);
```

Tester ces 2 fonctions en modifiant le programme de démo `bx`. Prendre le centre de la fenêtre pour `p`, ainsi que le curseur de la souris pour `q`. Là encore, on peut utiliser la boucle pour faire varier le niveau de récursion de façon cyclique et ainsi obtenir un effet d'animation.

```
int MainTest_DragonStepDemo (void);
int MainTest_DragonDemo     (void);

int main (void) { return MainTest_DragonDemo (); }
```

4 Dragon de Heighway à angle paramétrable

On obtient des résultats très variés si l'on choisit un autre angle que $\pi/2$. Les deux segments issus d'une même fracturation ne sont alors plus de longueur égale, et il est alors souhaitable d'aller plus profondément dans les récursions sur les segments longs. On peut par exemple continuer à fracturer un segment tant que sa longueur est supérieure à 5 pixels et que le niveau de récursion n'a pas excédé 64.

Écrire `Util_DrawDragon2()` qui dessine un dragon avec un angle paramétrable. Pour la tester, on peut prendre pour angle celui que forme le segment `(center,cursor)` avec l'axe horizontal.

```
void Util_DrawDragon2 (bx_window window, Coord p, Coord q, double angle);
int MainTest_Dragon2Demo (void);

int main (void) { return MainTest_Dragon2Demo (); }
```

On pourra trouver utile d'écrire les fonctions qui calculent la longueur d'un segment `(p,q)` et l'angle que forme le segment `(0,p)` avec l'axe horizontal :

```

double Coord_Distance2 (Coord p, Coord q);
double Coord_Distance  (Coord p, Coord q);
double Coord_Angle     (Coord p);          /* cf  <math.h>, atan2() */
```