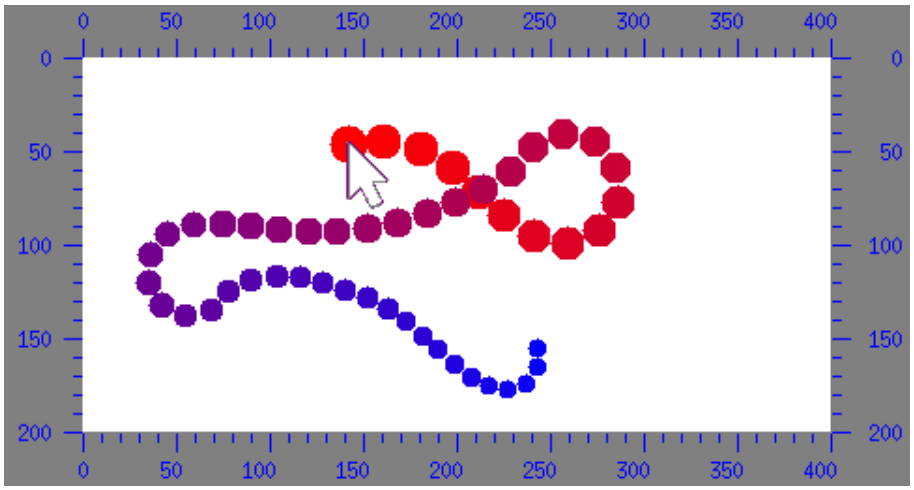
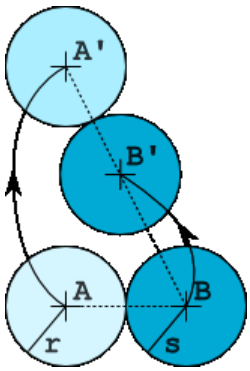
TD/TP de Programmation – Planche 2
Animation de Serpentin articulé

On souhaite animer un serpentin articulé constitué d'une série de disques contigüs. Sa tête forme le premier disque. On souhaite le dessiner et l'animer dans une fenêtre, avec sa tête suivant constamment le curseur de la souris. Le reste des disques suivent le mouvement. Lorsque qu'un disque se déplace, son successeur se déplace dans sa direction jusqu'à ce que les deux disques soient à nouveau contigüs, et le mouvement se propage ainsi de la tête vers la queue du serpentin. Plus formellement, si un disque de rayon r se déplace de A vers A', alors son successeur de rayon s se déplace de B vers B' tel que $B' = A' + (r + s) \times A'B/|A'B|$ avec $A'B = B - A'$.





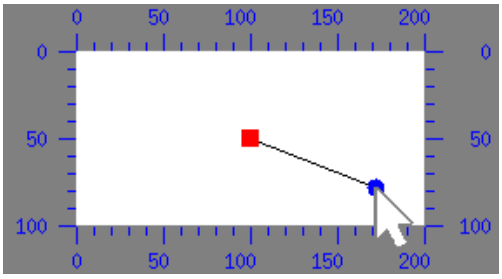
Structures de données

On utilisera les structures suivantes : `Coord` pour modéliser un point ou un vecteur dans le plan (par leurs coordonnées cartésiennes) ; `Joint` pour modéliser un disque d'articulation du serpentin (par son centre et son rayon) ; `Snake` pour modéliser un serpentin (par un tableau de disques de capacité fixe mais de longueur utile variable).

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

typedef struct Joint {
  Coord center;
  double radius;
} Joint;

#define MAX_JOINTS 50

typedef struct Snake {
  Joint joints [MAX_JOINTS];
  int nb_joints;
} Snake;
```

Librairie graphique `bx`

Pour les affichages, on utilisera la bibliothèque-jouet `bx` en modifiant le programme de démonstration ci-dessous qui affiche un carré rouge au centre d'une fenêtre, un disque bleu au niveau du curseur de la souris, et un segment noir entre les deux.




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

int main (void)
{
  bx_init (); /* A APPELER AVANT TOUTE AUTRE FONCTION BX */
  char win_title []= "demo";
  int win_x= 50, win_y= 50, canvas_width= 200, canvas_height= 100;
  bx_window win= bx_create_window (win_title, win_x, win_y, canvas_width, canvas_height);
  for (;;) {
    bx_mouse mouse= bx_read_mouse (win);
    int x1= canvas_width  / 2, x2= bx_mouse_x (mouse);
    int y1= canvas_height / 2, y2= bx_mouse_y (mouse);
    int r= 5, d= 2*r;
    bx_clear_canvas (win, bx_white());
    bx_set_color (bx_black()); bx_draw_line   (win, x1, y1, x2, y2);
    bool filled= true;
    bx_set_color (bx_red  ()); bx_draw_box    (win, x1-r, y1-r, d, d, filled);
    bx_set_color (bx_blue ()); bx_draw_circle (win, x2-r, y2-r, d, d, filled);
    int milliseconds_delay= 10;
    bx_show_canvas (win, milliseconds_delay);
  }
  bx_loop (); /* BOUCLE D'EVENEMENTS INFINIE */
  return 0;
}
```

Le librairie est initialisée avec la fonction `bx_init()`, puis, une fenêtre de type `bx_window` est créée avec la fonction `bx_create_window()`. Le canevas (surface dessinable) de cette fenêtre est raffraichi toutes les 10 milli-secondes dans une boucle active infinie par la fonction `bx_show_canvas()`. À chaque tour de boucle, l'état de la souris est lu avec la fonction `bx_read_mouse()` et sauvé dans une variable de type `bx_mouse`. On en extrait la position `(x2,y2)` par `bx_mouse_x()` et `bx_mouse_y()`. On efface le canevas avec `bx_clear_canvas()`. Un segment noir est ensuite dessiné avec `bx_draw_line()` reliant le centre `(x1,y1)` du canevas à `(x2,y2)`. Un carré rouge de coté 10 et centré en `(x1,y1)` est dessiné avec `bx_draw_box()`. Un cercle bleu de diamètre 10 et centré sur la souris est dessiné avec `bx_draw_circle()`. La couleur du crayon est changée avec `bx_set_color()` et on obtient des couleurs prédéfinies de type `bx_color` par les fonctions `bx_white()`, `bx_black()`, `bx_blue()`, `bx_red()`.

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
```

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)
```

1 Fonctions élementaires de calcul vectoriel

Écrire les fonctions suivantes : `Coord_FromXY()` qui fabrique et retourne un point ou un vecteur dans le plan à partir de leurs coordonnées cartésiennes ; `Coord_Sum()` et `Coord_Difference()` qui retournent respectivement la somme et la différence de deux vecteurs ; `Coord_DotProduct()` qui retourne le produit scalaire de deux vecteurs. En déduire `Coord_Length2()` qui retourne le carré de la longueur d'un vecteur, et `Coord_Length()` qui retourne la longueur d'un vecteur.

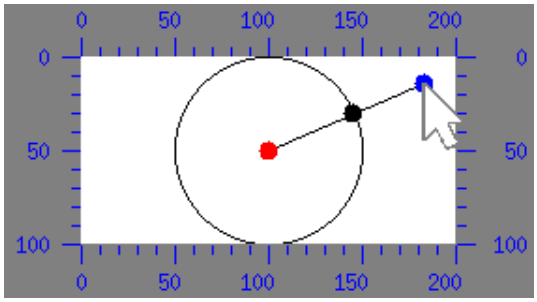
```
Coord Coord_FromXY (double x, double y);

Coord  Coord_Sum        (Coord p, Coord q); /* = p + q    */
Coord  Coord_Difference (Coord p, Coord q); /* = p - q    */
double Coord_DotProduct (Coord p, Coord q); /* = p . q    */
double Coord_Length2    (Coord p);          /* = |p|^2    */
double Coord_Length     (Coord p);          /* = |p|      */
```

Écrire `Coord_ScaledBy()` qui retourne un vecteur multiplié par le facteur d'echelle réél `lambda`. En déduire `Coord_Normalized()` qui retourne le vecteur normalisé d'un vecteur, ainsi que `Coord_MovedAtDistanceFrom()` qui retourne un point `p` après sa translation dans la direction d'un autre point `q` de telle sorte qu'il se place à une distance donnée de `q`.

```
Coord Coord_ScaledBy   (Coord p, double lambda);   /* = lambda * p    */
Coord Coord_Normalized (Coord p);                  /* = p / |p|       */

Coord Coord_MovedAtDistanceFrom (Coord p, double distance, Coord q);
/* = q + distance * qp / |qp|, avec qp= (p-q) */
```

2 Test des fonctions élémentaires de calcul vectoriel

Modifier le code de la démo pour tester les fonctions précédentes. Le programme affiche un cercle noir à la position `Coord_MovedAtDistanceFrom (cursor, 50, center)` où `cursor` est la position de la souris, `center` est le centre du canevas.




```
int MainTest_CoordDemo (void);
```

On pourra trouver utile d'écrire `Util_MouseCoord()` qui retourne la position de la souris sous la forme d'un type `Coord`, ainsi que `Util_DrawLine()` et `Util_DrawCircle()` qui dessinent respectivement un segment et un cercle à partir de coordonnées de type `Coord`, afin de ne jamais manipuler les abscisses et les ordonnées directement dans le programme principal.

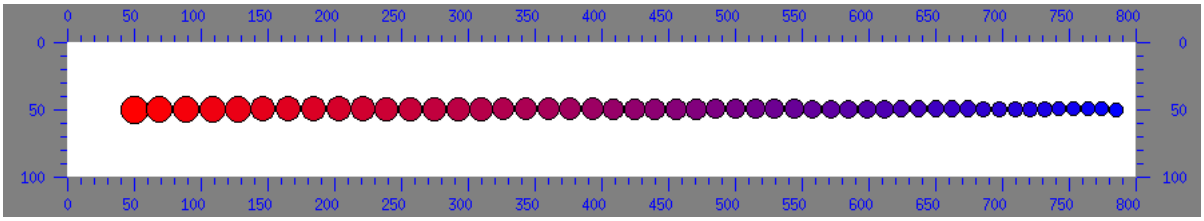
```
Coord Util_MouseCoord (bx_mouse mouse);
void Util_DrawLine   (bx_window win, Coord p, Coord q);
void Util_DrawCircle (bx_window win, Coord center, double radius, bool filled);
```

3 Initialisation du serpentin

Écrire la fonction `Joint_Init()` qui initialise un disque d'articulation de serpentin avec la position de son centre et son rayon. En déduire `Snake_Init()` qui initialise un serpentin avec `nb_joints` articulations, la tête étant centrée en `head_center` et de rayon `head_radius`, la queue étant de rayon `tail_radius`. Les disques intermédiaires ont des rayons obtenus par interpolation linéaire (ou *lerp*) de ces deux rayons. Tous les disques sont alignés horizontalement à droite de la tête.

```
double
Double_Lerp (double x0, double alpha, double x1)
{
  return x0 + alpha * (x1-x0);
}

void Joint_Init (Joint * j, Coord center, double radius);
void Snake_Init (Snake * s, int nb_joints, Coord head_center,
                 double head_radius, double tail_radius);
```

4 Affichage du serpentin



Écrire la fonction `Joint_Draw()` qui dessine un serpentin dans un fenêtre. En déduire `Snake_Draw()` qui dessine la totalité du serpentin. Tester l'initialisation et l'affichage du serpentin.

```
void Joint_Draw (Joint const * j, bx_window window);
void Snake_Draw (Snake const * s, bx_window window);

int MainTest_SnakeDrawingDemo (void);
```

5 Déplacement et animation du serpentin

Écrire `Joint_MoveTowards()` qui déplace un disque d'articulation `j` en direction d'un disque-cible `target` afin d'en assurer la contigüité. En déduire `Snake_MoveHeadTo()` qui déplace le disque de tête d'un serpentin en `head_center` et propage le mouvement sur les disques suivants jusqu'au disque de queue. Enfin, écrire le programme qui anime la tête du serpentin en fonction des mouvements de la souris.

```
void Joint_MoveTowards (Joint * j, Joint const * target);
void Snake_MoveHeadTo (Snake * s, Coord head_center);
int MainTest_SnakeAnimationDemo (void);

int main (void)
{
  /*
  return MainTest_BxDemo ();
  return MainTest_CoordDemo ();
  return MainTest_SnakeDrawingDemo ();
  */
  return MainTest_SnakeAnimationDemo ();
}
```