Université Pierre et Marie Curie **Master M1 "Mathématiques et Applications"**
Période 2 - 2013/2014 **Programmation en C/C++**

TP noté de C - Géométrie algorithmique : polygones et droites en dimension 2

— *Documents autorisés :* le polycopié de cours, les feuilles de TP et leurs corrigés, vos notes personnelles. Aucun livre extérieur n'est autorisé.

— L'usage d'Internet est **strictement** interdit pendant l'épreuve : tout accès à Internet détecté par le responsable entraînera automatiquement la note nulle.

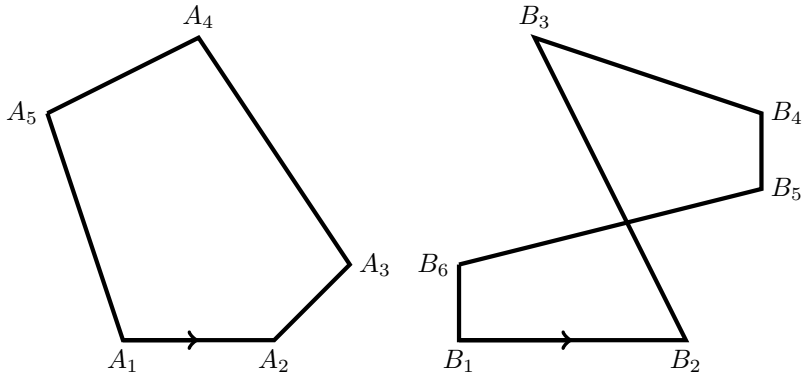
— *Durée :* 2 heures.

— *Les fichiers sources rendus à la fin de la séance doivent contenir en commentaire les noms, prénoms et numéros des étudiants du binôme.*

Nous souhaitons décrire efficacement des polygones et des droites en dimension 2 afin de calculer leurs points d'intersection éventuels.

Nous appelons ici *polygone* une suite ordonnée de n points *distincts* du plan $(A_k)_{1\leq k\leq n}$. Chaque point A_k a donc un sucesseur A_{k+1} (avec périodicité $n+1 \equiv 1$). Nous appelons *polygone marqué* un polygone avec un point spécifique choisi comme origine. Par défaut ici, tous nos polygones seront marqués par A_1. Un polygone est ainsi codée par une **liste chaînée qui revient sur son point initial et forme ainsi un cycle**.

Voici deux polygones marqués acceptables, avec les pointeurs de successeur de l'origine :



Nous souhaitons faire marcher le code donné en page 2.

Remarque importante : nous supposerons dans tous les programmes que les polygones dont nous cherchons le nombre d'intersections ne se coupent jamais sur un sommet, et qu'aucuns côtés ne se recouvrent partiellement. Cela compliquerait énormément les tests logiques et demanderait de traiter le cas d'un nombre infini d'intersections. De plus, la précision numérique offerte par les `double` ne le permet pas.

Exercice 1. Créer un fichier `polygone.c` avec vos noms, prénoms et numéros d'étudiants dans l'en-tête. Recopiez en faisant très attention la fonction `main()` donnée en deuxième page.

Fichier principal

```
// Nom1 Prenom1 numero1
// Nom2 Prenom2 numero2
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

Faites un dessin sur une feuille avec les deux polygones T et P après ajout du point de la ligne 14 et vérifier qu'ils se rencontrent en quatre points.

Fonction `main()` qui doit fonctionner

```
int main(void){
        Polygone T=create_triangle(1.,-2.,-3.,1.,4.,3.);

        double px[]={0.,2.,3.,1.,-1.};
        double py[]={0.,0.,1.,4.,3.};
        Polygone P=create_polygone(5,px,py);

        printf("Triangle T: ");
        affichage_polygone(T);
        printf("Pentagone P: ");
        affichage_polygone(P);

        P=ajouter_point(P,3,3.,2.);
        printf("Pentagone P apres ajout: ");
        affichage_polygone(P);

        printf("Intersections de T et P = %d\n",
                        intersection_polygone(T,P));

        T=detruire_Polygone(T);
        P=detruire_Polygone(P);
        return 0;
}
```

Le résultat après exécution **doit être** identique à :

```
Triangle T: Points du polygone:
        Le point 1 est situe en ( 1.000000 , -2.000000 )
        Le point 2 est situe en ( -3.000000 , 1.000000 )
        Le point 3 est situe en ( 4.000000 , 3.000000 )
---------
Pentagone P: Points du polygone:
        Le point 1 est situe en ( 0.000000 , 0.000000 )
        Le point 2 est situe en ( 2.000000 , 0.000000 )
        Le point 3 est situe en ( 3.000000 , 1.000000 )
        Le point 4 est situe en ( 1.000000 , 4.000000 )
        Le point 5 est situe en ( -1.000000 , 3.000000 )
---------
Pentagone P apres ajout: Points du polygone:
        Le point 1 est situe en ( 0.000000 , 0.000000 )
        Le point 2 est situe en ( 2.000000 , 0.000000 )
        Le point 3 est situe en ( 3.000000 , 1.000000 )
        Le point 4 est situe en ( 3.000000 , 2.000000 )
        Le point 5 est situe en ( 1.000000 , 4.000000 )
        Le point 6 est situe en ( -1.000000 , 3.000000 )
---------
Intersections de T et P = 4
```

Définition des différentes structures :

Structures

```
typedef struct point {
        double x;       double y;
} Point; // Description d'un point du plan

typedef struct polygonepoint {
        Point p;
        struct polygonepoint * suiv;
} Polygonepoint; // Definition d'un point d'un polygone

typedef Polygonepoint * Polygone; // Definition d'un polygone

typedef struct segment {
        Point A1;
        Point A2;
} Segment; // Definition d'un segment
```

Exercice 2. Écrire le code des deux fonctions :

Création de polygones

```
Polygone create_triangle(double x1,double y1,double x2,
                double y2,double x3,double y3);
Polygone create_polygone(unsigned int N, double * tx, double * ty);
```

La première fonction permet de créer un triangle de trois points dont les coordonnées forment les six arguments de la fonction. La seconde permet de créer un polygone à `N` sommets dont les abscisses (resp. les ordonnées) se lisent dans le tableau `tx` (resp. `ty`).

Exercice 3. Écrire une fonction

Fonction d'affichage

```
void affichage_polygone(const Polygone P)
```

qui permet d'afficher les paramètres d'un polygone **conformément** aux normes imposées en page 2.

Exercice 4. Écrire une fonction `Polygone detruire_Polygone(Polygone Q)` qui nettoie la mémoire et efface tout ce qui est relatif à un polygone donné et renvoie la valeur `NULL`.

Exercice 5. Écrire une fonction `Polygone ajouter_point(Polygone P, unsigned int p, double x0,double y0)` qui introduit un point supplémentaire avec les coordonnées (x_0, y_0) dans le polygone après le p-ème sommet (le premier pointé par `Polygone` étant le premier sommet). Cette fonction renvoie l'adresse du sommet-origine, qui ne change donc pas.

Exercice 6. Écrire les fonctions

Intersections

```
int intersection(Segment S1,Segment S2);
int intersection_polygone(Polygone P1, Polygone P2) ;
```

La première fonction teste si deux segments (dont on supposera qu'ils ne se recouvrent pas partiellement) se rencontrent ou non. Pour deux segments $[A_1A_2]$ et $[B_1B_2]$ avec les coordonnées $A_1 = (x_1, y_1)$, $A2 = (x_2, y_2)$, $B_1 = (u_1, v_1)$ et $B_2(u_2, v_2)$, on calcule les nombres

$$\Delta = (u_1 - u_2)(y_1 - y_2) - (x_1 - x_2)(v_1 - v_2) \qquad \begin{pmatrix} t \\ s \end{pmatrix} = \frac{1}{\Delta} \begin{pmatrix} v_2 - v_1 & u_1 - u_2 \\ y_2 - y_1 & x_1 - x_2 \end{pmatrix} \begin{pmatrix} u_2 - x_2 \\ v_2 - y_2 \end{pmatrix} \quad (1)$$

pour $\Delta \neq 0$. Les deux segments se rencontrent si et seulement si $\Delta \neq 0$ et $(s, t) \in [0, 1] \times [0, 1]$.

La deuxième fonction compte le nombre de points d'intersections entre toutes les arêtes de $P1$ et toutes les arêtes de $P2$ et doit utiliser la première fonction.

Exercice 7. Écrire la fonction `Point isobarycentre(const Polygone)` qui calcule le centre de gravité d'un polygone.

Écrire la fonction `double perimetre(const Polygone)` qui calcule le périmètre d'un polygone. Que vaut le périmètre du polygone P de la page 2 ? Vous indiquerez votre réponse en commentaire après votre fonction `main()`.

Exercice 8. Écrire une fonction `Polygone questionbonus(const Polygone P)` qui crée à partir d'un polygone P de sommets $A_1, \ldots, A_n$ un polygone de sommets $B_1, \ldots, B_n$ tel que B_i soit le milieu des points A_i et A_{i+1} (avec la convention que $A_{n+1} = A_1$).