POLYTECH PARIS-SUD

Et3 - Tronc commun

Programmation C

TP6 - Structures de données

1 Objectifs :

- Savoir définir une structure dans le bon fichier
- Savoir déclarer une structure selon les deux manières possibles, en connaissant les différences et leurs avantages.
- Savoir gérer les structures imbriquées
- Savoir gérer les structures via des pointeurs

2 Rappel

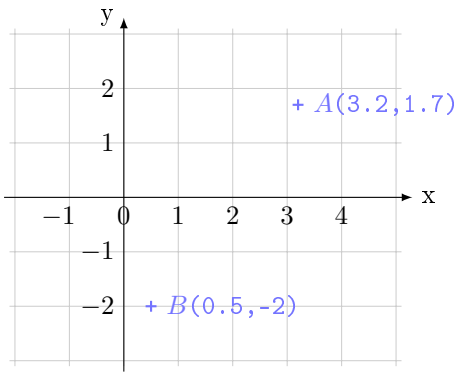
N'oubliez pas de répartir votre code dans plusieurs fichiers, en regroupant les fonctions de manière logique. Ici, nous vous proposons de séparer comme suit :

- main.c
- point.c, point.h
- segment.c, segment.h
- rect.c, rect.h

N'oubliez pas de faire les `include` dans l'ordre nécessaire et de correctement renseigner les fichiers *.h!

3 Exercice 1 : Des Points

Nous souhaitons définir une structure `point_t` contenant deux `double x` et `y`, les coordonnées d'un point, respectivement l'abscisse et l'ordonné, dans un plan muni d'un repère orthonormal. Exemple ici, le `A` de coordonnées `(3.2,1.7)` et le `B` de coordonnées `(0.5,-2)`.



1. Définissez cette structure `point_t`.
2. Pour créer une structure, deux possibilités s'offrent à nous :

— Faire une fonction `point_create` qui va créer un point en fonction des paramètres `x` et `y` que nous lui auront fournis. Il faut de plus ajouter une procédure `point_delete` pour libérer la mémoire.

— ou utiliser le raccourcis `point_t A = {3.2,1.7};`

Quel est l'avantage de l'un par rapport à l'autre ? N.B. : Utilisez ici la deuxième solution.

3. Nous voulons maintenant afficher les informations contenues dans un point. Écrivez la fonction `point_affiche(point_t const* p)`
4. Nous voulons maintenant calculer la distance entre deux points. La signature de la fonction devra être la suivante :
   `double point_distance(point_t const* p1, point_t const* p2)`.
   Rappel : $dist_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
   Testez avec des valeurs connues !
5. Nous souhaitons vérifier si deux points sont égaux. Votre fonction devra renvoyer 1 si oui, 0 sinon et prendra en paramètre deux `point_t const*`. (Deux points sont égaux si leurs x sont égaux ET leurs y).
6. Enfin, nous voulons appliquer une translation à un point p d'une amplitude dx en abscisse et dy en ordonnés. Écrivez la fonction `void point_translate(point_t* p, double dx, double dy)`

4 Exercice 2 : Puis des segments

Maintenant, nous allons créer des segments. Un `segment` sera défini par une structure contenant elle-même deux structures `point_t`.

1. Définissez cette structure `segment`.
2. Cette fois-ci, nous allez créer la fonction `segment* segment_create(point_t const* begin, point_t const* end)` pour tester les deux méthodes. N.B. : Vous remarquerez que nous avons ici fait un choix sur les paramètres, mais qu'on aurait pu prendre 4 coordonnées en entrée.
3. Chaque structure avec un `create` doit obligatoirement avoir la procédure `delete` correspondante.
   La signature de ovtre fonction devra être `void segment_delete(segment* seg)`.
4. Pour pouvoir vérifier les valeurs entrées dans un `segment`, définissez une procédure `void segment_affiche(segment const* seg)`.
   Remarque : Pensez à utiliser la fonction déjà faite pour afficher les points.
5. Maintenant, nous voulons calculer la longueur d'un segment passé en argument de la fonction. Le résultat de votre fonction sera stocké sous forme d'un `double`.
   Remarque : Pensez à utiliser la fonction déjà faite pour les points.
6. De la même manière que pour les points, nous voulons savoir si deux segments sont égaux. Écrivez la fonction `int segment_equal(segment const* seg1, segment const* seg2)`. Attention, ici nos segments ont un début et une fin et sont donc orientés.
   Remarque : Pensez à utiliser la fonction déjà faite pour les points.
7. Pour finir avec les segments, nous voulons définir si deux segments ont une intersection et si oui, renvoyer un `point_t*`. Pour se faire, nous vous proposons la méthodologie suivante pour votre fonction :

   — Tout d'abord calculer l'équation des droites (De la forme $y = a * x + b$). Si les coefficients directeurs ne sont pas égaux, cela signifie que les droites ne sont pas parallèles et donc qu'il y a une intersection.

   — Ensuite, il faut calculer le point d'intersection. (Défini par : $x_{inter} = (b' - b)/(a - a')$ et $y_{inter} = a * x_{inter} + b$)

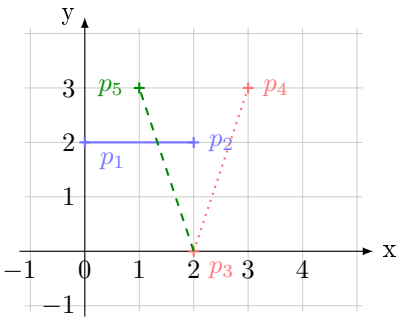
   — Enfin, comme l'illustre le schéma ci-dessous, nous travaillons sur des segments et non des droites. Par exemple, le segment vert et bleu ainsi que le vert et rouge sont sécants tandis que le rouge et bleu ne sont pas sécants alors que leur équation de droite l'est. Il faut donc vérifier que le point calculé est bien sur les deux segments : vérifier que le x_{inter} est bien compris entre le x_{begin} et le x_{end} de chaque segment. De même pour le y.
   N.B. : Cette dernière partie peut aussi se faire avec des produits scalaires.... A vous de voir !

   Remarque :

— Attention aux cas où le dénominateur peut être égal à 0 !

— Vous aurez probablement besoin de créer des sous-fonctions.

— Nous ne prenons pas en compte le cas où les deux segments sont sur la même droite, et peuvent donc avoir soit un point, soit un segment en commun.



5 Exercice 3 : Et enfin des rectangles !

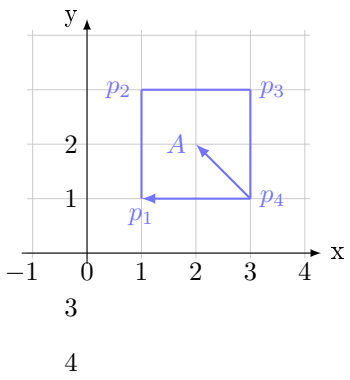
1. Nous définissons maintenant la structure `rect` qui est composée de 4 `point_t`. Les 4 points correspondent aux 4 sommets et sont stockés dans l'ordre horaire en commençant n'importe où.
   Note : Vous pouvez aussi choisir de stocker 2 segments, 1 segment et 2 points, etc. Mais chaque façon de faire a ses inconvénients.
2. Cette fois, nous sommes obligés de créer une fonction `rect_create` qui prendra 4 `point_t*` en paramètre. En effet, nous avons ici un invariant à vérifier : les points sont-ils dans le bon ordre ? Et est-ce vraiment un rectangle ? Les points sont-ils tous distincts ?
   Pour ce faire, il vous suffit de vérifier s'il y a au moins 3 angles droits.
   Indice : nous pouvons utiliser le produit scalaire qui nous dit deux choses :

$$\vec{AB}.\vec{AC} = x_{AB} * x_{AC} + y_{AB} * y_{AC} \tag{1}$$

$$\vec{AB}.\vec{AC} = |AB| * |AC| * \cos\theta \tag{2}$$

   Note : si $\cos\theta = 0$ alors $\theta = 90°$

   Remarque : nous vous conseillons de créer une sous-fonction `produit_scalaire(point_t const* A, point_t const* O, point_t const* B)` qui calcule la valeur $\cos\theta$ entre les vecteurs $\vec{OA}.\vec{OB}$.
3. N'oubliez pas la règle d'or : à chaque `create` son `delete` !
4. Nous souhaitons afficher les informations contenues dans un rectangle. Écrivez la fonction `void rect_affiche(rect const* rectangle)`
5. Nous voulons déterminer si deux rectangles sont égaux. Écrivez la fonction, sans oublier de pivoter les rectangles.
6. Calculez maintenant la surface d'un rectangle.
   Remarque : N'oubliez pas que vous avez déjà codé des fonctions !
7. Nous voulons vérifier qu'un point est compris ou non dans un rectangle. Ecrivez la fonction.
   Indices : Soit un rectangle défini par les 4 points p1, p2, p3 et p4 et un point A dans le plan, les 4 $\cos\widehat{Ap_4p_1}$, $\cos\widehat{Ap_1p_2}$, $\cos\widehat{Ap_2p_3}$ et $\cos\widehat{Ap_3p_4}$ sont compris entre entre 0 et 1 si et seulement si A est compris dans le rectangle. Pour les calculer, nous vous recommandons d'utiliser les formules des produits scalaires données précédemment.



8. Pour finir, nous voulons déterminer si deux rectangles sont bien disjoints : `int rect_disjoint(rect const* rectangle1, rect const* rectangle2)`.
   Veillez à bien vérifier tous les cas.