Examen final - LO01 A2016

2 H - Sans documents (sauf diagrammes de C) ni calculette
N'oubliez pas de commenter vos programmes. Les types des paramètres et des fonctions sont parfois volontairement omis dans l'énoncé. A vous de les définir.

Vous aurez besoin de 3 copies : une par exercice

---

1. Objet, récursivité, chaîne de caractères (6 points)

1.1/ Objet

Qu'est-ce qu'une instanciation en programmation orientée objet ? donnez un exemple en considérant que la classe *rectangle* existe.

1.2/ Occurrence d'un caractère

Ecrire la fonction récursive *int nombreDeChiffres(int nb)* qui calcule le nombre de chiffres composant le nombre nb (nb est positif).
Exemple : int nombreDeChiffres(2017) -> 4

1.3/ Les voyelles

1 - Soit le tableau char voyelles[]="aeiouy" contenant l'ensemble des voyelles de la langue française, écrire la fonction *int estUneVoyelle(char car, char *voyelles)* qui retourne 1 (vrai) si le caractère car est une voyelle ou 0 (faux) sinon.

2 - Ecrire la fonction *int calculNbVoyelles(char *phrase*) qui retourne le nombre de voyelles de la chaîne de caractères phrase.
Exemple : pour la chaîne de caractères "demain, des l'aube, a l'heure ou blanchit la campagne, je partirai.", la fonction retournera le nombre 24.

---

2. Agence de voyage (7 points) **Nouvelle copie**

Une agence de voyage stocke dans un fichier "clients.dat" l'ensemble des informations concernant ses clients, soit :

Adresse Client :
- numéro de rue : entier,
- nom de rue : chaîne de caractères,
- code postal : entier,
- ville : chaîne de caractères.

Client :
- nom : chaîne de caractères,
- prénom : chaîne de caractères,
- adresse : adresse client,
- année : entier correspondant à l'année de départ du voyage,
- numéro du séjour : entier,
- club : chaîne de caractères.

1- Définir la structure *unClient* permettant de décrire un client, en ayant au préalable défini la structure *uneAdresse*.
2- Ecrire la fonction int *rechercheUnVacancier(char *nom, char *prenom, unClient *unC)* qui recherchera dans le fichier *"clients.dat",* le client dont le nom et le prénom sont passés en paramètres et renverra la structure *unC* correspondant au client. Si la personne ne se trouve pas dans le fichier, la fonction renverra 0 (le nom et prénom contiendront alors la chaîne vide "") et 1 sinon.
3- Ecrire la fonction void *afficheUnVacancier(char *nom, char *prenom)* qui affichera le nom, la ville, le club souhaité de la personne passée en paramètre.

4- Pour des raisons de confidentialité, la CNIL demande d'anonymiser le champ nom des personnes dans le fichier "clients.dat" pour les séjours antérieurs à une certaine date. Ecrire la fonction *void anonymiseClients(int an)* qui réécrira dans le fichier "clients.dat" à la place du nom, une chaîne de caractères "" (vide) pour toutes les séjours antérieurs à l'année *an*.
5- Ecrire le programme principal qui affiche pour un client dont on demandera le nom et le prénom à l'utilisateur, le résultat de l'appel à la fonction *afficheUnVacancier()* avant et après anonymisation des voyages antérieurs à 2015 compris, si la personne a effectué son voyage avant 2015, le programme devrait afficher que la personne ne se trouve plus dans le fichier.

---

3. Un peu de géométrie (7 points) **Nouvelle copie**

Soit un polygone dans le plan décrit par une suite ordonnée de n sommets (p_0, p_1, p_2, p_3,… p_{n-1}). Chaque sommet est un point 2D décrit par ses coordonnées (x,y). Deux sommets consécutifs définissent une arête et il y a implicitement une arête entre les sommets p_{n-1} et p_0. Un polygone contiendra au maximum 50 sommets.

1- Définir le type *unPoint* basé sur une structure.
2- Définir le type *unPolygone* basé sur une structure qui contient le nombre de sommets et un tableau contenant la suite des sommets du polygone.

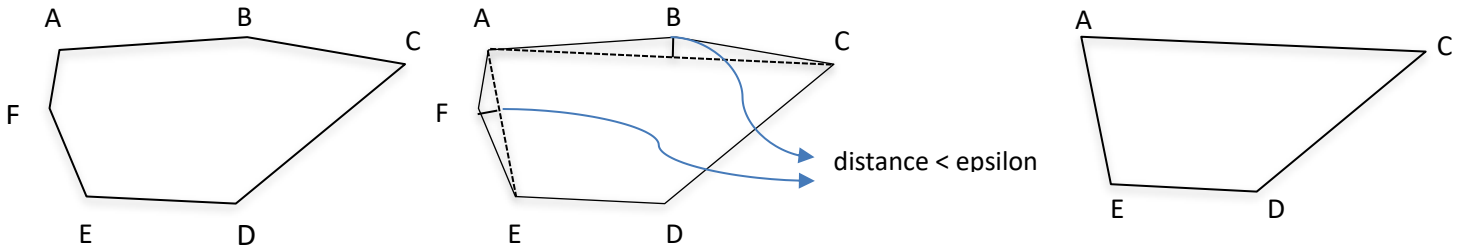
On suppose connue la fonction suivante : *void initPoint(unPoint *s)* qui effectue le dialogue utilisateur pour la saisie des coordonnées x et y d'un point passé en paramètre.

3- Ecrire la fonction *initPolygone(p)* à l'aide de la fonction *initPoint().*
4- Ecrire la fonction *enleveUnSommet(p,i)* qui enlève au polygone p le sommet d'indice i.

On suppose connue la fonction suivante : *float calculeDistance (unPoint s1, unPoint s2, unPoint a)* qui retourne la distance du point *a* au segment s*1s2*.

5- Ecrire la fonction *simplifiePolygone(unPolygone p, float epsilon)* qui réduit le nombre de sommets du polygone *p* en éliminant dans la liste des sommets, les points alignés à *epsilon* près.

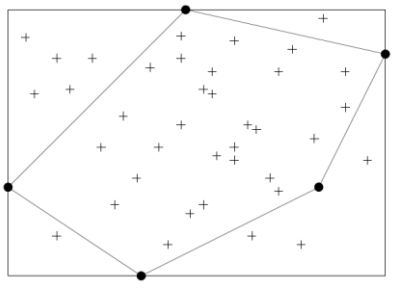
Par exemple le polygone p ci-dessous sera simplifié en retirant 2 sommets :



Une des façons d'approximer la surface d'un polygone est d'utiliser la méthode de Monte-Carlo. Le principe de cette méthode est de calculer une valeur numérique en utilisant des procédés aléatoires. Dans le cas du calcul d'une surface, il suffit de tirer au hasard des points qui sont à l'intérieur du plus petit rectangle contenant le polygone. La surface Aire du polygone pourra alors être approximée par la formule suivante :

Aire ≈ Surface du rectangle × Nb points dans le polygone / Nb points total

Par exemple, sur la figure ci-contre, en supposant que le rectangle fasse 3 cm de hauteur et 4, 25 cm de largeur, et qu'il y a 28 points sur 39 qui sont à l'intérieur du polygone, sa surface s peut être approximée par :



$$\text{Aire} \approx 3 \times 4{,}25 \times 28/39 = 9{,}39 \text{ cm}^2$$

On suppose connues les 2 fonctions suivantes :

- *float reelAleatoire (float borneMin, float borneMax)* qui permet d'obtenir un réel aléatoire entre une borne minimum et une borne maximum
- *int estAlInterieur(unPoint s, unPolygone p)* qui retourne 1 si *s* est à l'intérieur de *p* et *0* sinon.

6- Ecrire la fonction *calculeSurface(p, n)* qui calcule la surface du polygone *p* à l'aide de la méthode de Monte-Carlo avec un tirage de *n* points aléatoires.
7- Ecrire un programme *principal* qui initialise un polygone puis affiche les surfaces calculées avant et après simplification de ce polygone. Les valeurs epsilon et n seront demandées à l'utilisateur.