

Examen de Info C

Durée : 1h
Aucun document autorisé
Mobiles interdits

On représente les coordonnées d'un point p du plan cartésien par deux réels (**double**) x et y tels que $p = (x, y)$.

- 1. À l'aide de **typedef**, utilisez une structure pour déclarer le type **Point** formé des deux coordonnées réelles x et y .

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

- 2. Écrivez la fonction `distance` qui renvoie la distance entre deux **Point**. La fonction admet en paramètre deux pointeurs sur **Point**.

```
/**  
 * Antécédent : p1, p2 deux pointeurs sur Point  
 * Rôle : renvoie la distance entre 2 de points du plan p1 et p2  
 */  
double distance(const Point *p1, const Point *p2) {  
    return sqrt((p2->x-p1->x)*(p2->x-p1->x)+(p2->y-p1->y)*(p2->y-p1->y));  
}
```

- 3. Écrivez la fonction `égal` qui renvoie 1 si deux **Point** sont égaux, et 0 sinon. La fonction admet en paramètre deux pointeurs sur **Point**.

```
/**  
 * Antécédent : p1, p2 deux pointeurs sur Point  
 * Rôle : renvoie 1 si deux de points p1 et p2 sont égaux et 0 sinon  
 */  
bool égal(const Point *p1, const Point *p2) {  
    return p1->x==p2->x && p1->y==p2->y;  
}
```

autre possibilité (mais préférence à la précédente) :

```
bool égal(const Point *p1, const Point *p2) {  
    return distance(p1, p2)==0.0;  
}
```

- 4. On veut maintenant définir des triangles. À l'aide de **typedef**, déclarez le type **Triangle** représenté par un tableau de 3 pointeurs sur **Point**.

```
typedef Point *Triangle[3];
```

- 5. Écrivez la fonction `périmètre` qui renvoie le périmètre d'un **Triangle** t passé en paramètre. Vous vérifiez que les 3 sommets du triangle sont distincts. Vous pouvez utiliser la notation de tableau.

```
/**  
 * Rôle : teste si les 3 points p1, p2, et p3 sont alignés  
 */  
bool sontAlignés(const Point *p1, const Point *p2, const Point *p3) {  
    return (p2->y-p1->y)*(p3->x-p2->x) == (p3->y-p2->y)*(p2->x-p1->x);  
}  
  
/**  
 * Antécédent : les 3 points du triangle t sont distincts  
 * Rôle : renvoie le périmètre du triangle t  
 */  
double périmètre(const Triangle t) {  
    assert(!égal(t[0], t[1]) && !égal(t[0], t[2]) && !égal(t[1], t[2]));  
    // les 3 points du triangle sont distincts  
    assert(!sontAlignés(t[0], t[1], t[2])); // voir DS1 du S5  
    // les 3 points du triangle ne sont pas alignés  
    return distance(t[0],t[1]) + distance(t[1],t[2]) + distance(t[2],t[0]);  
}
```

- 6. En utilisant uniquement la notation pointeur, écrivez la fonction `nbOccurrences` qui renvoie le nombre d'occurrences d'un caractère présent c dans une chaîne de caractères s . La chaîne de caractères et le caractère sont passés en paramètre de la fonction.

```
/**  
 * Rôle : renvoie le nombre d'occurrences de c dans s  
 */  
int nbOccurrences(const char *s, const char c) {  
    int nbc=0;  
    while (*s)  
        if (*s++==c)  
            // nouvelle occurrence c dans s  
            nbc++;  
    //  
    return nbc;  
}
```

-
- 7. En utilisant uniquement la notation pointeur, écrivez un **programme** qui écrit sur la sortie standard le nombre d'occurrences d'un caractère dans des chaînes de caractères. Le programme prend ses données à partir des paramètres programmes. Le caractère à rechercher est le premier paramètre programme. Vous vérifierez la validité des paramètres programme. Ci-dessous, des exemples d'exécution du programme :

```
> ./nbocc x
Usage : ./nbocc char string...

> ./nbocc al toto bal
al n'est pas un caractère unique.

> ./nbocc e hello demain elle clair
1 occurrence(s) de e dans hello
1 occurrence(s) de e dans demain
2 occurrence(s) de e dans elle
0 occurrence(s) de e dans clair
```

```
/**
 * Rôle : teste si la chaîne s est formée d'un seul caractère
 */
bool estUnCar(const char *s) {
    return *(s+1)=='\0';
}

int main(int argc, char *argv[]) {
    // vérifier le nombre minimal de paramètres programmes
    if (argc<=2) {
        fprintf(stderr, "Usage : %s char string...\n", *argv);
        exit(EXIT_FAILURE);
    }
    // le nombre de paramètres programmes est valide =>
    // vérifier que le 1er paramètre est un caractère unique
    char *car = **argv;
    if (!estUnCar(car)) {
        fprintf(stderr, "%s n'est pas un caractère\n", car);
        return EXIT_FAILURE;
    }
    // les paramètres sont valides
    argc--;
    // on a sauté le nom du programme => parcourir tous les autres paramètres
    while (--argc>0)
        printf("%d occurrence(s) de %c dans %s\n", nbOccurrences(*argv, *car), *car, **argv);
    //
    return EXIT_SUCCESS;
}
```

.....