

## Exercice 6

27

Reprendre l'exercice 5, en utilisant des paramètres aux fonctions de calcul de distance.

Tout pixel d'une image est caractérisé par un couple de coordonnées (x,y). On peut donc calculer des distances entre pixels. Les distances les plus courantes sont (pour deux pixels P(xp,yp) et Q(xq,yq)):

- distance de Manathan :  $d1(P,Q) = |xp - xq| + |yp - yq|$
- distance euclidienne :  $d2(P,Q) = [(xp - xq)^2 + (yp - yq)^2]^{1/2}$
- distance de l'échiquier :  $dinf(P,Q) = \text{Max}(|xp - xq|, |yp - yq|)$

Ces distances sont reliées par la propriété :

$$dinf(P,Q) \leq d2(P,Q) \leq d1(P,Q)$$

Écrire un programme définissant les structures nécessaires définir les points et les fonctions permettant de calculer les distances de **Manathan** et **euclidienne** (echiquier en option ☺) + un petit programme *main()* utilisant ces fonctions.

PS: <math.h> fournit 2 fonctions intéressantes : abs() et sqrt() ☺

## Exercice 8

28

Boîte à outils 3D

Définir une structure permettant de manipuler les points  
3D : point3D

Définir une structure pour les vecteurs : vecteur

Écrire les fonctions/procédures suivantes (en utilisant les paramètres)

saisirPoint : permet de saisir les coordonnées d'un point par l'utilisateur

vect\_construc : construit un vecteur à partir de 2 points

vect\_norme : retourne la norme d'un vecteur

$$|\mathbf{V1}| = (x1.x1 + y1.y1 + z1.z1)^{1/2}$$

vect\_add : ajoute 2 vecteurs (résultat dans un 3<sup>ème</sup> vecteur)

vect\_mulScal : multiplication par un scalaire (modifie le vecteur en paramètre)

## Exercice 9 : tableaux

29

On déclare un tableau d'entiers A dimension 10

Deux sous-programmes, remplitA et impA, permettent de remplir et imprimer le tableau (sur la console).

Déclarer une référence vers l'élément 5 du tableau et incrémenter sa valeur

Déclarer un pointeur vers l'élément 7, incrémenter le pointeur et la valeur pointée

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

## Exercice 10 : tri par création

30

Le principe de ce tri est de créer un tableau vide de même taille que le tableau à trier, puis de chercher dans le tableau à trier le plus petit élément afin de le placer en première position du tableau nouvellement créé.

Ensuite, l'algorithme recherche successivement les éléments suivants afin de les placer, à la suite, dans le nouveau tableau. L'algorithme se termine lorsque tous les éléments du tableau ont été ajoutés.

5	3	1	2	6	4
---	---	---	---	---	---

1	?	?	?	?	?
1	2	?	?	?	?
1	2	3	?	?	?
1	2	3	4	?	?
1	2	3	4	5	?
1	2	3	4	5	6

1. Définir une structure permettant de simplifier cette méthode de tri (quels sont les éléments déjà triés ?)
2. Écrire une procédure permettant d'initialiser un tableau de ce type à partir d'une série d'entiers
3. Écrire la procédure tri\_creation retournant un tableau trié

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

# Exercices

et solutions

University of Geneva  
www.miralab.ch

## Exercice 6

32

Reprendre l'exercice 5, en utilisant des paramètres aux fonctions de calcul de distance.

Tout pixel d'une image est caractérisé par un couple de coordonnées  $(x,y)$ . On peut donc calculer des distances entre pixels. Les distances les plus courantes sont (pour deux pixels  $P(x_p,y_p)$  et  $Q(x_q,y_q)$ ):

- distance de Manathan :  $d1(P,Q) = |x_p - x_q| + |y_p - y_q|$
- distance euclidienne :  $d2(P,Q) = [(x_p - x_q)^2 + (y_p - y_q)^2]^{1/2}$
- distance de l'échiquier :  $dinf(P,Q) = \text{Max}(|x_p - x_q|, |y_p - y_q|)$

Ces distances sont reliées par la propriété :

$$dinf(P,Q) \leq d2(P,Q) \leq d1(P,Q)$$

Écrire un programme définissant les structures nécessaires définir les points et les fonctions permettant de calculer les distances de **Manathan** et **euclidienne** (echiquier en option ☺) + un petit programme *main()* utilisant ces fonctions.

PS: <math.h> fournit 2 fonctions intéressantes : `abs()` et `sqrt()` ☺

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

33

```

#include <iostream>
#include <math.h>

using namespace std;

typedef struct {
    float x,y;
} point2d;

float distManathan(point2d a, point2d b) {
    return (abs(a.x - b.x) + abs(a.y - b.y));
}

float distEuclidienne(point2d a, point2d b) {
    float dx = a.x - b.x;
    float dy = a.y - b.y;

    return ((float)sqrt(dx*dx + dy*dy));
}

float distEchiquier(point2d a, point2d b) {
    float dx = abs(a.x - b.x);
    float dy = abs(a.y - b.y);

    if (dx > dy)
        return dx;
    return dy;
}

int main() {
    point2d p,q;

    cout << "Saisir les coordonnees de p : ";
    cin >> p.x >> p.y;
    cout << "Saisir les coordonnees de q : ";
    cin >> q.x >> q.y;

    cout << endl << "Resultats : " << endl;

    cout << "distance de Manathan = " << distManathan(p, q) << endl;
    cout << "distance Euclidienne = " << distEuclidienne(p, q) << endl;
    cout << "distance de l'echiquier = " << distEchiquier(p, q) << endl;

    return 0;
}

```

34

## Exercice 8

### Boite à outils 3D

Définir une structure permettant de manipuler les points  
3D : point3D

Définir une structure pour les vecteurs : vecteur

Écrire les fonctions/procédures suivantes (en utilisant les paramètres)

saisirPoint : permet de saisir les coordonnées d'un point par l'utilisateur

vect\_construc : construit un vecteur à partir de 2 points

vect\_norme : retourne la norme d'un vecteur

$$|\mathbf{V1}| = (x1.x1 + y1.y1 + z1.z1)^{\frac{1}{2}}$$

vect\_add : ajoute 2 vecteurs (résultat dans un 3<sup>ème</sup> vecteur)

vect\_mulScal : multiplication par un scalaire (modifie le vecteur en paramètre)

## Ex8 : solution

35

```
#include <iostream>
#include <math.h>
using namespace std;

typedef struct {
    float x, y, z;
} point3D;

typedef struct {
    float dx, dy, dz;
    float norme;
} vecteur;

// Déclarations
point3D saisirPoint();
vecteur vect_construct(point3D P, point3D Q);
float vect_norme(vecteur V);
vecteur vect_add(vecteur V, vecteur W);
void vect_mulScal(vecteur &V, float scal);

// Définition
point3D saisirPoint() {
    point3D P;
    cin >> P.x >> P.y >> P.z;
    return (P);
}

vecteur vect_construct(point3D P, point3D Q) {
    vecteur V;

    V.dx = abs(P.x - Q.x);
    V.dy = abs(P.y - Q.y);
    V.dz = abs(P.z - Q.z);

    V.norme = vect_norme(V); // calcul de la norme

    return (V);
}

float vect_norme(vecteur V) {
    return ( sqrt ( (V.dx * V.dx) + (V.dy * V.dy) + (V.dz * V.dz) ) );
}

vecteur vect_add(vecteur V, vecteur W) {
    vecteur Res;
    Res.dx = V.dx + W.dx;
    Res.dy = V.dy + W.dy;
    Res.dz = V.dz + W.dz;
    Res.norme = vect_norme(Res);
    return (Res);
}

void vect_mulScal(vecteur &V, float scal) {
    V.dx *= scal;
    V.dy *= scal;
    V.dz *= scal;
    V.norme = vect_norme(V);
}

// Programme principal
void main() {
    point3D A, B;

    A = saisirPoint();
    B = saisirPoint();

    vecteur V1;
    V1 = vect_construct(A, B);
    vect_mulScal(V1, 3);
}
```

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

## Exercice 9 : tableaux

36

On déclare un tableau d'entiers A dimension 10

Deux sous-programmes, rempliA et impA, permettent de remplir et imprimer le tableau (sur la console).

Déclarer une référence vers l'élément 5 du tableau et incrémenter sa valeur

Déclarer un pointeur vers l'élément 7, incrémenter le pointeur et la valeur pointée

University of Geneva  
www.miralab.ch

LCI  
Introduction au langage C

37

```

#include <iostream>
using namespace std;

int A[9];

void rempliA() {
    for (int i=0; i<=9;i++) {
        A[i] = i;
    };
};

void impA() {
    for (int i=0; i<=9; i++) {
        cout << "A[" << i << "] = " << A[i] << "\n";
    };
    cout << "\n";
}

int main() {
    rempliA();

    impA();

    int &b = A[5];
    b++;
    impA();

    int * c= A+7;
    (*c)++;
    *c++, (*c)++;
    impA();

    cout << "b = " << b << " adr de b = " << &b <<
        "\n";
    cout << "c = " << c << " adr de c = " << &c << "
        val de *c = " << *c << "\n";
    return (0);
}

```