Travaux Pratiques de programmation n°5

Cours de programmation impérative

—Licence MPI L1 S2 - Info 121—

Structures de données

Cette séance de travaux pratiques est dédiée à l'utilisation des structures.

▸ **Exercice 1. (Fractale de Mandelbrot)**

Le but de l'exercice est d'afficher le célèbre ensemble fractale de Mandelbrot. Pour ce faire, on considère la suite de nombres complexes paramétrée par un complexe donné c, définie par

$$z_0 = 0 \qquad \text{et} \qquad z_{n+1} = z_n^2 + c\,.$$

L'ensemble de Mandelbrot est l'ensemble des points dont les coordonnées complexes c sont telles que la suite associée ne tend pas vers l'infini.

1. Le type `Complexe`, la fonction `creeComplexe`, la fonction `egaleComplexe` et la fonction `ajouterComplexe` vous sont donnés ci-dessous. Écrire la fonction `multiplierComplexe` qui prend en entrée deux complexes et renvoie leur produit. Attention, faites très attention à la formule : $(a_1 + ib_1) * (a_2 + i * b_2) = a_1 * a_2 - b_1 * b_2 + i(a_1 * b_2 + a_2 * b_1)$. En effet la moindre erreur risque de vous ralentir de une demi-heure (tests qui ne passent pas, mandelbrot tout bizarre)

```
struct Complexe {
    float re, im;
};

Complexe creeComplexe(float reel, float imaginaire){
    Complexe c;
    c.re = reel;
    c.im = imaginaire;
    return c;
}

bool egaleComplexe(Complexe c1, Complexe c2){
    return (c1.re==c2.re)&&(c1.im==c2.im);
}

Complexe ajouterComplexe(Complexe c1, Complexe c2){
    Complexe resultat;
    resultat.re = c1.re + c2.re;
    resultat.im = c1.im + c2.im;
    return resultat;
}
```

2. Proposer trois tests de la fonction `multiplierComplexe`, puis appeler la fonction de test dans le `main`. Indication : voici un exemple de test :

```
ASSERT(egaleComplexe(multiplierComplexe(creeComplexe(0, 0),
                                        creeComplexe(1, 1)),
                     creeComplexe(0, 0)));
```

3. Écrire une fonction qui calcule le module d'un nombre complexe (on pourra utiliser la bibliothèque `cmath` vue dans le TP précédent). La tester.
4. Écrire une fonction booléenne `znResteBorne` qui prend un complexe c, et retourne vrai si la suite z_n associée ne tend pas vers l'infini. On calculera la suite des termes z_n, jusqu'à ce que le module de z_n devienne plus grand que 1000 (auquel cas, on dira que la suite tend vers l'infini), ou 1000 itérations ont été faites (auquel cas, on dira que la suite reste bornée).

La procédure `mandelbrot` qui vous est donnée ci-dessous utilise la fonction `znResteBorne` que vous venez d'écrire pour afficher l'ensemble de Mandelbrot. Elle fait varier x entre -2 et 0.5 et y entre -1.5 et $+1.5$, et fait un affichage alphanumérique sur 80×80 caractères.

```
void mandelbrot(){
    const float xmin = -2;
    const float xmax = 0.5;
    const float ymin = -1.5;
    const float ymax = 1.5;
    const int resol = 79;
    for(int i=0; i<=resol; i++) {
        for(int j=0; j<=resol; j++) {
            if(znResteBorne(creeComplexe((resol-j)*xmin/resol+j*xmax/resol,
                                         (resol-i)*ymax/resol+i*ymin/resol))) {
                    cout << '#';
            }
            else {
                    cout << ' ';
            }
        }
        cout << endl;
    }
}
```

5. Exécuter `mandelbrot`, puis modifier cette procédure pour zoomer sur la figure, en faisant varier x et y sur des plus petits intervalles, autour de points pas trop loin de la frontière.

▸ Exercice 2. (Couples faisables)

1. Déclarer un type structure `Personne` pour représenter une personne avec seulement quatre informations : son nom, son prénom, son année de naissance, et son sexe. On pourra utiliser le type énuméré suivant :

   `enum genre {homme, femme};`

   Une fois ce type declaré, vous pouvez déclarer une variabe de type genre, et tester si cette variables est égale à femme ou homme.
2. Écrire une fonction `nouvellePersonne` qui prend en entrée les informations d'une personne et renvoie une `Personne`. On va utiliser un vecteur pour contenir une population de dix personnes. L'initialisation d'un vecteur de six entiers peut se faire comme ceci :

   `vector<int> vecteur = { 1, 2, 3, 5, 6, 7 };`

   Pour un vecteur de `Personne`, on procède de la même manière avec un appel un appel à la fonction `nouvellePersonne`. Initialiser un tableau `population` de dix personnes, née entre 1975 et 1990, directement dans le code.

3. Un couple de deux personnes est dit «faisable» si les deux personnes sont de sexes opposés, avec moins de sept ans de différences ; Écrire une procédure `afficherCoupleFaisable` qui prend en entrée un tableau de `Personne` et qui affiche les couples faisables. Utiliser deux boucles imbriquées, qui itèrent toutes les deux sur les éléments du tableau. On pourra d'abord écrire une fonction booléenne `estFaisable` qui teste si deux personnes données peuvent former un couple. Tester `afficherCoupleFaisable`.
4. On souhaite tenir compte du fait qu'environ 7% de la population est homosexuelle. Modifier le type structure, pour enregistrer si la personne est hétérosexuelle ou homosexuelle. Modifier la fonction `nouvellePersonne` et la procédure `afficherCoupleFaisable` pour en tenir compte (si on a pris soin de décomposer `afficherCoupleFaisable` comme suggéré à la question 3, seul le code de `estFaisable` est à modifier. Le code de `afficherCoupleFaisable` reste inchangé).
5. Donner un nom au type de population afin de simplifier la déclaration de type dans l'en-tête de `coupleFaisable`.

▸ **Exercice 3.** Réimplanter la fonction racine carrée du T.P. précédent, en utilisant la structure `Complexe` de l' exercice 1.