université PARIS DIDEROT PARIS 7

INTRODUCTION À L'INFORMATIQUE ET À LA PROGRAMMATION (IF1)

TP6 : BOUCLES INDÉFINIES ET AU-DELÀ

1 Boucles indéfinies

► **Exercice 1** : Écrire une fonction `encore` qui demande à l'utilisateur « encore ? », et qui continue de lui poser la question tant que celui-ci lui répond « oui ».

► **Exercice 2** : La commande Unix `yes` affiche indéfiniment sur la console des lignes contenant le caractère `y`.
Écrire un programme Java `Yes.java` qui a le même comportement.
Ce programme peut être interrompu avec la commande `^c` (maintien de la touche `Control` avec la touche `c`).

► **Exercice 3** : Écrire une fonction qui lit des entiers jusqu'à ce que l'utilisateur entre 0, puis qui affiche la somme des entiers entrés par l'utilisateur.
Modifier votre fonction pour qu'elle affiche la somme et le produit des entiers entrés par l'utilisateur.

2 Des approximations numériques

► **Exercice 4** (Approximation de la racine carrée — méthode de Héron) : Étant donné un réel strictement positif a, on définit la suite réelle $(x_n)_{n\in\mathbf{N}}$ de la manière suivante :

$$\begin{aligned} x_0 &= a, \\ x_{i+1} &= \frac{x_i + \frac{a}{x_i}}{2} \qquad i \geq 0. \end{aligned}$$

Cette suite converge vers $\sqrt{a}$.
Écrire une fonction `mysqrt` qui prend en arguments un réel `a` et un entier `n` et qui retourne une valeur approchée de $\sqrt{\mathtt{a}}$ en utilisant l'approximation $x_{\mathtt{n}}$.
Vérifier le comportement de la fonction `mysqrt` en élevant au carré le résultat obtenu.

► **Exercice 5** (Approximation de e) : Un étudiant place 1€ dans une banque. Cette somme sera rémunérée au taux de 100%, l'étudiant se retrouvera donc en possession de 2€ au bout d'une année. Un deuxième étudiant choisit de placer son euro dans une banque lui offrant un taux de rémunération de 50% tous les six mois. Ce dernier se retrouvera en possession de 2,25€ à la fin de l'année.
Écrire une fonction `pecule` qui calcule ce que l'étudiant n, qui a placé son euro dans une banque lui offrant un taux de rémunération de $1/n$ toutes les $1/n$ années, possède à la fin de l'année.
Cette fonction calcule des valeurs approchées de e. Dire si la convergence paraît rapide ou pas.

► **Exercice 6** (Approximation de π) : On part de l'égalité suivante (qu'on ne vous demande pas de montrer) :

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

pour calculer une valeur approchée de π. Pour cela, on définit les suites $(S_N)_N$ et $(\pi_N)_N$ de la façon suivante :

$$S_N = \sum_{n=1}^{N} \frac{1}{n^2} = \frac{\pi_N^2}{6},$$

où la limite de $(\pi_N)_N$ est π. La différence $S_{N+1} - S_N$ tend vers 0 et c'est grâce à cette différence qu'on va évaluer si on est suffisamment proche de la valeur recherchée ou non, en utilisant le fait que

$$S_{N+1} - S_N = \frac{1}{6}\big(\pi_{N+1} - \pi_N\big)\big(\pi_{N+1} + \pi_N\big).$$

Donc pour N suffisamment grand

$$S_{N+1} - S_N > \pi_{N+1} - \pi_N.$$

Écrire une fonction `pi` qui prend en argument un réel représentant une erreur admissible pour la différence de deux éléments consécutifs de la suite $(\pi_N)_N$ et renvoie une valeur approchée de π.

3 Algorithme d'Euclide

► **Exercice 7** (Algorithme d'Euclide) : L'algorithme dit d'Euclide permet de calculer le pgcd de deux entiers strictement positifs α et β. Il manipule deux entiers strictement positifs $a \geq b$. Initialement, a vaut $\max(\alpha, \beta)$, et b vaut $\min(\alpha, \beta)$.
À chaque étape, on calcule le reste r de la division de a par b. Si ce reste est nul, alors l'algorithme termine, et le résultat est b. Sinon, a prend la valeur de b, puis b prend la valeur de r.
Écrire une fonction *non récursive* `pgcd` qui calcule le pgcd de ses deux arguments entiers à l'aide de l'algorithme d'Euclide.

4 Pour ceux et celles qui ont le temps

► **Exercice 8** (Approximation de π) : Chercher et implémenter d'autres méthodes d'approximation de π que celle proposée à l'exercice 6. Les comparer.

► **Exercice 9** (Ensembles de Julia) : Julia est un mathématicien qui a étudié l'itération de fonctions polynomiales et rationnelles au début du XXème siècle. Si f est une fonction, elle peut avoir différents comportements lorsqu'elle est appliquée plusieurs fois de suite. Si l'on part d'une valeur particulière x alors il faut considérer la suite de valeurs x, $f(x)$, $f(f(x))$, $f(f(f(x)))$, ... Cela revient à étudier la suite $(x_n)_{n\in\mathbb{N}}$ définie par $x_{n+1} = f(x_n)$. Ainsi, cette suite peut soit converger (c'est-à-dire se rapprocher de plus en plus d'une unique valeur), soit diverger.
Les ensembles de Julia sont définis à partir de la fonction f_c :

$$f_c : \mathbb{C} \to \mathbb{C} : z \mapsto f_c(z) = z^2 + c \text{ avec } c \in \mathbb{C}.$$

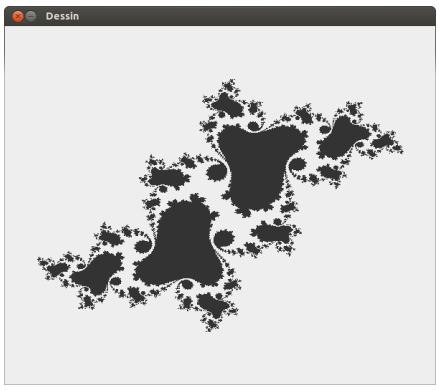
L'étude de ces ensembles revient à considérer la suite $(z_n)_{n\in\mathbb{N}} : z_{n+1} = f(z_n)$.
L'*ensemble de Julia rempli* au point c fixé est l'ensemble K_c des points de départ z_0 tels que (z_n) converge.

Le but de cet exercice est d'obtenir une représentation graphique d'ensembles de Julia remplis qui sont des ensembles fractals (exemples en figure 1). Ces ensembles sont constitués de points, c'est-à-dire de représentations de nombres complexes (partie réelle $\leftrightarrow$ abscisse, partie imaginaire $\leftrightarrow$ ordonnée).
Pour tracer ces ensembles, on peut par exemple utiliser la méthode du pixel. Elle consiste à tester tous les points du plan complexe, ce qui est impossible formellement, mais en fait réalisable sur un écran, car un point est représenté par un pixel et on se limite à une fenêtre finie qui contient un nombre fini de pixels. On itère N fois f_c et si son module dépasse une valeur critique (par exemple 2), il n'appartient pas à l'ensemble de Julia rempli. Sinon, on dessine ce point sur l'écran, et on passe au suivant. Pour résumer :

$$(\forall n \leq N, |z_n| \leq 2) \Leftrightarrow z \in K_c.$$

Télécharger sur didel le fichier `Complexe.jar` qui contient la classe `Complexe` (il y a un lien sur didel vers la documentation de cette classe). Cette classe sert à représenter des nombres complexes.
Récupérer également le fichier `Drawings.jar` du TP5.

(a) $c = -0.745 + 0.1i$



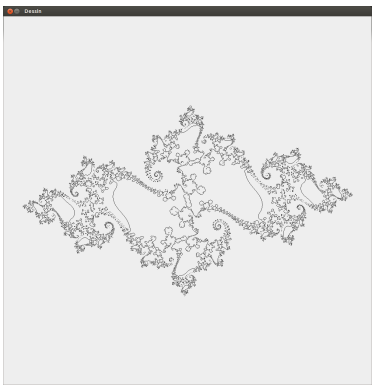
(b) $c = -0.15 + 0.65i$

FIGURE 1 – Exemples d'ensembles de Julia remplis (largeur = 600, hauteur = 500, échelle = 1/3).

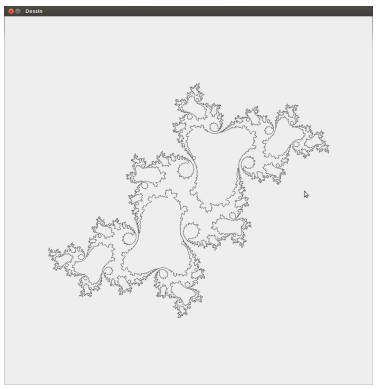
1. Écrire une fonction `fC` qui prend en arguments `z` et `c` et renvoie la valeur de $f_{\mathtt{c}}(\mathtt{z})$.
2. Écrire une fonction `converge` qui prend en arguments `z0` et `c` et teste si la suite $(\mathtt{z}_n)_{n\in\mathbb{N}}$ de premier terme `z0` converge (c'est-à-dire, dans ce cas précis, si au bout d'un certain nombre d'itérations, par exemple 100, son module n'a jamais dépassé 2).
3. Écrire une fonction `dessinerEnsembleDeJuliaRempli` qui prend en arguments :
   – la largeur et la hauteur de la fenêtre,
   – le point `c`,
   – l'échelle pour l'affichage,
   et affiche l'ensemble de Julia rempli correspondant (le point z décrivant la fenêtre d'affichage recentrée en $(0,0)$).
4. Écrire une fonction principale et tester le programme. Si la classe s'appelle `Julia`, on la compile avec la ligne de commande :

   ```
   javac -classpath Complexe.jar:Drawings.jar Julia.java
   ```

5. L'*ensemble de Julia* (non rempli) J_c est la frontière de K_c, c'est-à-dire les points qui sont au bord (exemples en figure 2). On peut dire par exemple qu'un point appartient à l'ensemble de Julia J_c s'il appartient à l'ensemble de Julia rempli K_c et qu'au moins un de ses voisins n'y appartient pas (on pourra considérer au choix 4 ou 8 voisins par point). Écrire une fonction `dessinerEnsembleDeJuliaNonRempli` qui prend en arguments :
   – la largeur et la hauteur de la fenêtre,
   – le point `c`,
   – l'échelle pour l'affichage,
   et affiche l'ensemble de Julia non rempli correspondant.



(a) $c = -0.745 + 0.1i$



(b) $c = -0.15 + 0.65i$

FIGURE 2 – Exemples d'ensembles de Julia non remplis (largeur = 600, hauteur = 500, échelle = 1/3).