Pile, Mandelbrot

1 Quelques expériences sur la pile et les pointeurs

1.1 Exercice 2

On considère le code suivant :

```c
#include <stdio.h>

void f(int n, int *nmax) {
    printf("Début de l'appel de f(%d, _)\n", n); // debut
    printf("n     = %d\n", n);
    printf("&n    = %p\n", (void *)&n);
    printf("nmax  = %p\n", (void *)nmax);
    printf("*nmax = %d\n", *nmax);
    printf("&nmax = %p\n", (void *)&nmax);
    if (n < *nmax) f(n + 1, nmax);
    printf("Fin de l'appel de f(%d, _)\n", n);  // fin
}

int main() {
    int n = 2;
    f(0, &n);
    return 0;
}
```

1. En oubliant les `printf` situés ailleurs qu'aux lignes `debut` et `fin`, prévoir l'affichage produit par la fonction.
2. Parmi les autres lignes provoquant un affichage, lesquelles :
   (a) affichent toujours la même chose ?
   (b) affichent des choses différentes, mais que l'on peut prévoir parfaitement en regardant le code ?
   (c) affichent des choses différentes et partiellement imprévisibles ?
3. Prévoir le plus complètement possible l'affichage produit par le programme.
4. Exécuter ce programme (dans le terminal et éventuellement avec *C Tutor*) et observer l'affichage produit.
5. Quelle est la taille (en octets) du bloc d'activation de `f` ?

1.2 Exercice 3

1. Écrire une fonction `incremente` qui prend en entrée un pointeur vers un entier et incrémente la valeur de cet entier de 1 (cette fonction ne renverra rien).
2. Dans tous les exemples suivants, on souhaite qu'un appel `f(px, py)` incrémente celle des valeurs pointées par `px` et `py` qui est la plus petite (ou celle pointée par `px` en cas d'égalité). Par exemple :

```c
#include <stdio.h>

int main() {
    int x = 4;
    int y = 3;
    printf("x = %d, y = %d\n", x, y);
    f(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    f(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

doit donner l'affichage suivant :

```
x = 4, y = 3
x = 4, y = 4
x = 5, y = 4
```

Dans chaque cas, dire si :
— il y a un problème de type (et si oui, où) ;
— les fonctions ont bien le comportement attendu (uniquement dans les cas où il n'y a pas de problème de type). Si ce n'est pas le cas, on expliquera d'où vient le problème.

(a) Code 1

```c
int plus_petit(int x, int y) {
    if (x <= y) return x;
    return y;
}

void f(int *px, int *py){
    incremente(plus_petit(*px, *py));
}
```

(b) Code 2

```c
int *plus_petit(int x, int y) {
    if (x <= y) return &x;
    return &y;
}

void f(int *px, int *py) {
    incremente(plus_petit(*px, *py));
}
```

(c) Code 3

```c
int *plus_petit(int *x, int *y){
    if (*x <= *y) return x;
    return y;
}

void f(int *px, int *py){
    incremente(plus_petit(px, py));
}
```

(d) Code 4

```c
int *plus_petit(int *x, int *y){
    int a = *x;
    int b = *y;
    if (a <= b) return &a;
    return &b;
}

void f(int *px, int *py){
    incremente(plus_petit(px, py));
}
```

2 Fractale de Mandelbrot

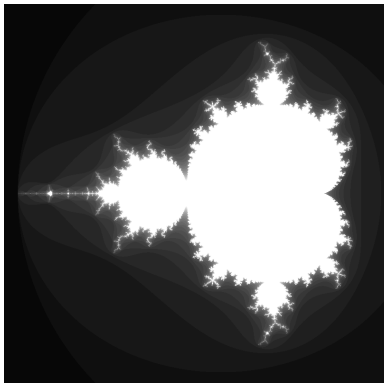

FIGURE 1 – L'ensemble de Mandelbrot.

Pour tout $c \in \mathbb{C}$, on considère la fonction

$$\begin{array}{rccc} f_c : & \mathbb{C} & \longrightarrow & \mathbb{C} \\ & z & \longmapsto & z^2 + c. \end{array}$$

À partir de cette fonction, on définit la suite $(z_n(c))$ par

$$\begin{cases} z_0(c) \coloneqq 0 \\ \forall n \in \mathbb{N}, \quad z_{n+1}(c) \coloneqq f_c(z_n(c)) \end{cases}$$

L'ensemble de MANDELBROT est l'ensemble des complexes c tels que $(u_n(c))_n$ est bornée. Dans la suite, on note $\mathcal{M}$ cet ensemble.

2.1 Mathématiques

Cette question est purement mathématique : il peut être pertinent de la traiter plus tard, chez vous, et d'admettre le résultat de la dernière question pour la suite.

1. Soit $c \in \mathbb{C}$ et $n \in \mathbb{N}$. On note $z_n = z_n(c)$ et l'on suppose $|z_n| > 2$ et $|z_n| > |c|$.
   (a) Montrer que pour $m \geqslant 0$, on a $|z_{n+m}| \geqslant |c| + 2^m (|z_n| - |c|)$.
   (b) En déduire que $c \notin \mathcal{M}$.
2. Montrer que si $|c| > 2$, alors $|z_1| > |c|$.
3. Montrer que $c \in \mathcal{M} \iff (\forall n \in \mathbb{N}, \quad |z_n| \leqslant 2)$.

On définit désormais

$$\mathcal{M}_N = \{c \in \mathbb{C} \mid \forall n \leqslant N, \quad |z_n(c)| \leqslant 2\}$$

D'après ce qui précède, on a :
— $\forall N \in \mathbb{N}, \quad \mathcal{M} \subset \mathcal{M}_N$
— $\mathcal{M} = \bigcap_{N \in \mathbb{N}} \mathcal{M}_N$

Pour N suffisamment grand, $\mathcal{M}_N$ sera une bonne approximation de $\mathcal{M}$. Pour déterminer si $c \in \mathcal{M}$, on pourra donc se fixer un entier `itermax` puis calculer les valeurs successives de $z_n(c)$.
— Dès que l'une de ces valeurs a un module strictement supérieur à 2, on sait que $c \notin \mathcal{M}$.
— Si toutes les valeurs jusqu'à $n =$ `itermax` ont un module inférieur ou égal à 2, alors $c \in \mathcal{M}_{\texttt{itermax}}$ et l'on considère que $c \in \mathcal{M}$ (ce qui n'est pas forcément vrai).

2.2 Code

1. Écrire une fonction `int divergence(double xc, double yc, int itermax)`. Cette fonction prend en entrée un complexe c (parties réelle et imaginaire) et un entier `itermax` et doit renvoyer :
   — le plus petit $n \leqslant$ `itermax` tel que $c \notin \mathcal{M}_n$, s'il en existe un.
   — `itermax + 1` sinon.
2. On définit deux constantes `ROWS` et `COLS` ainsi qu'un tableau bidimensionnel global :

```c
#include <stdio.h>
#include <stdlib.h>

#define ROWS 800
#define COLS 800

int arr[ROWS][COLS];
```

— `arr[0][0]` correspondra au pixel en haut à gauche de l'image ;
— `arr[ROWS - 1][0]` au pixel en bas à gauche ;
— `arr[0][COLS - 1]` au pixel en haut à droite ;
— `arr[ROWS - 1][COLS - 1]` au pixel en bas à droite.

On souhaite que l'image corresponde dans le plan complexe aux points $z = x + iy$ tels que $x_{min} \leqslant x \leqslant x_{max}$ et $y_{min} \leqslant y \leqslant y_{max}$. Écrire deux fonctions

```c
double re(int j, double xmin, double xmax);

double im(int i, double ymin, double ymax);
```

prenant en entrée un numéro de ligne ou de colonne dans le tableau `arr` et renvoyant la partie réelle ou imaginaire du complexe correspondant à ce pixel. On souhaite bien sûr que $x_{min} + iy_{min}$ soit en bas à gauche de l'image et $x_{max} + iy_{max}$ en haut à droite.

3. Écrire une fonction

```c
void fill_tab(double xmin, double xmax,
              double ymin, double ymax,
              int itermax);
```

qui remplit le tableau global `arr` avec les valeurs renvoyées par la fonction `divergence` pour les différents pixels.

4. On suppose dans cette question que la fonction `fill_tab` a déjà été appelée, et donc que le tableau `arr` contient les valeurs de `divergence` pour les différents pixels.
   (a) Écrire une fonction `void print_pixel_bw(int i, int j, int itermax)` qui affiche `255 255 255 \n` ou `0 0 0 \n` selon que le pixel (i, j) de l'image appartient ou non à $\mathcal{M}_{\texttt{itermax}}$.
   (b) Écrire une fonction `void print_tab(int itermax)` qui affiche le fichier PPM correspondant à l'image calculée. On pourra reprendre le code écrit dans un TP précédent.
   (c) Écrire la fonction `main` de manière à obtenir le comportement suivant :
   — si le programme est appelé sans argument en ligne de commande, il utilise les valeurs $x_{min} = y_{min} = -2$, $x_{max} = y_{max} = 2$ et $itermax = 20$;
   — s'il est appelé avec un argument, cet argument est utilisé pour $itermax$;
   — s'il est appelé avec cinq arguments, le premier est utilisé pour $itermax$ et les autres pour $x_{min}, x_{max}, y_{min}$ et y_{max} (dans l'ordre) ;
   — s'il est appelé avec un autre nombre d'arguments, il termine sans rien faire en affichant un message d'erreur.
   Dans tous les cas (sauf le dernier), le programme doit afficher le contenu du fichier PPM décrit dans les questions précédentes sur la sortie standard. On utilisera la fonction `atoi` pour convertir une chaîne de caractères en entier et `atof` pour convertir en flottant.
5. Modifier le code pour obtenir un affichage en niveaux de gris. On affichera une couleur d'autant plus sombre que le point c est rapidement sorti du disque de rayon 2 (centré en l'origine).

Remarque

⇨ Pour obtenir un dessin vraiment joli, il faut définir une notion d'itération partielle pour éviter les aplats de couleur, puis utiliser une palette cyclique et interpoler.

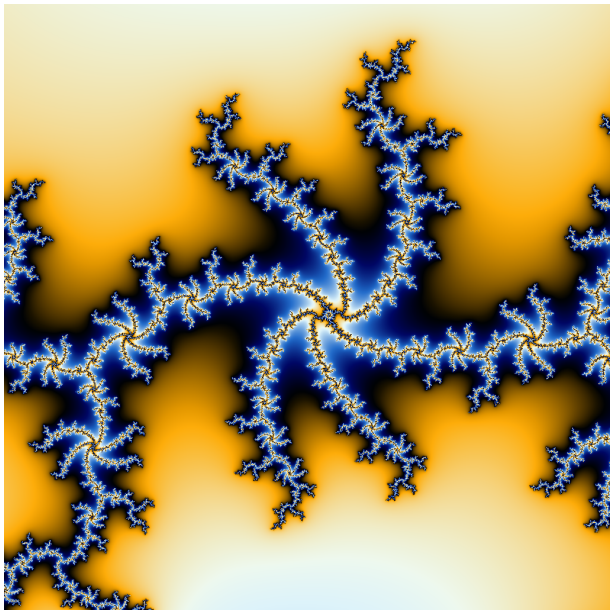

FIGURE 2 – Zoom sur la partie $0.00172 \leqslant x \leqslant 0.00184$, $-0.82258 \leqslant y \leqslant -0.82246$.

3 Pour chercher

3.1 Maxima par plage

On considère un tableau $t = [t_0, \ldots, t_{n-1}]$ et un entier $h \geqslant 1$. Pour $0 \leqslant i \leqslant n - h$, on définit :

$$m_h(i) \coloneqq \max(t_i, \ldots, t_{i+h-1})$$

1. Écrire un algorithme (simple) permettant de calculer tous les $m_h(i)$.
2. Déterminer la complexité de cet algorithme.
3. Trouver un algorithme permettant de trouver tous les $m_h(i)$ en temps $\mathcal{O}(n)$.
4. Écrire un programme en C implémentant cet algorithme.