Jean Perrin
Faculté des Sciences
Université d'Artois

C avancé
TD n° 1

UNIVERSITÉ D'ARTOIS
Licence 3 – semestre 1
2022 – 2023

Les tableaux et les pointeurs

Dans ce TD vous allez encore utiliser votre format de représentation d'image préféré (à part pour le dernier ;)), c'est-à-dire le format `ppm`. Pour rappel, le codage `ppm` qui permet de stocker des images graphiques. Plus précisément, vous utiliserez le format P3 qui se décrit ainsi :

```
P3 # Le P3 signifie que les couleurs sont en ASCII, et qu'elles sont en RGB.
3 2 # Par 3 colonnes et 2 lignes :
255 # Ayant 255 pour valeur maximum :
255   0   0     0  255   0       0   0  255
255 255   0   255 255 255       0   0    0
```

Le format limite le nombre de pixel par ligne, afin de vous simplifiez la vie, écrivez un pixel par ligne.

Exercice 1 : Le Mathématicien anglais Michael Barnsley a décrit comment, à partir d'un point, créer des figures en forme de fougères en répétant un grand nombre de fois une transformation simple sur ce point. Les règles sont les suivantes :

- Le premier point est O est (0, 0) ;
- À partir de ce point O, chaque point est obtenu à partir de son prédécesseur en appliquant une des quatre transformations suivantes :
  - f_1 avec une probabilité de 0,01 ;
  - f_2 avec une probabilité de 0,85 ;
  - f_3 avec une probabilité de 0,07 ;
  - f_4 avec une probabilité de 0,07.

  Les transformations f_i sont associées à une relation matricielle de la forme

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = M_i \times \begin{pmatrix} x \\ y \end{pmatrix} + V_i$$

  avec :

  - pour f_1 :
$$M_1 = \begin{pmatrix} 0 & 0 \\ 0 & 0.16 \end{pmatrix} \text{ et } V_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
  - pour f_2 :
$$M_2 = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix} \text{ et } V_2 = \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$
  - pour f_3 :
$$M_3 = \begin{pmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{pmatrix} \text{ et } V_3 = \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$
  - pour f_4 :
$$M_4 = \begin{pmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix} \text{ et } V_4 = \begin{pmatrix} 0 \\ 0.44 \end{pmatrix}$$

L'objectif de cet exercice est de générer avec cette procédure un nombre conséquent de points et de les afficher en utilisant le format `ppm`.
Attention : le domaine de définition n'est pas forcément celui de la résolution de l'image que vous allez générer. Plus précisément pour tout couple (x,y) généré via cette procédure nous avons $(x,y) \in [-3,3] \times [0,10]$. Vous devrez donc faire en sorte que votre fougère s'intègre complètement dans l'image et qu'elle soit centrée comme dans la figure 1.

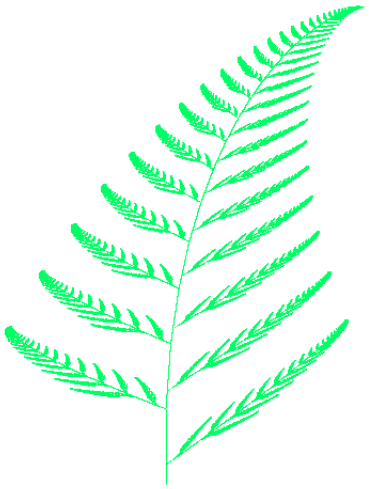


Figure 1: Fougère de Barnsley

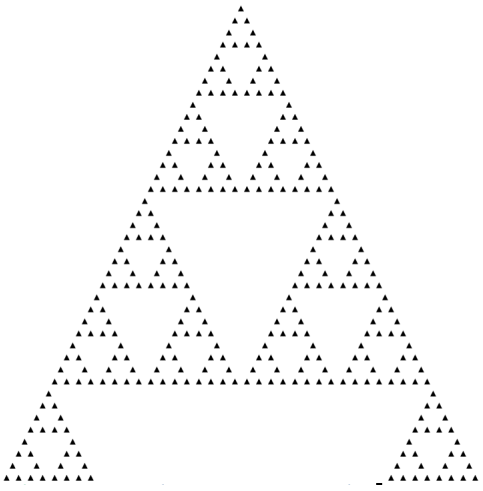
Exercice 2 : Le but de cet exercice est de réaliser un programme qui affiche un triangle de Pascal.

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
  1   7   21  35  35  21  7   1
```

La construction de ce triangle de Pascal ce réalise de la manière suivante :

- Nous partons de 1 à la première ligne, par convention c'est la ligne zéro (n = 0)
- Pour avoir un terme de la ligne suivante, nous prenons le terme juste au-dessus, et nous lui additionnons celui qui est juste avant, (0 si il n'y a rien).

Si nous inscrivons le triangle de Pascal dans un triangle, la réunion des cellules contenant des termes impairs est un triangle de Sierpiński. La seconde partie de l'exercice consiste alors à générer cette figure en mode texte de la manière suivante (pour afficher un triangle avec `printf` vous pouvez utiliser le caractère unicode `25B2`):



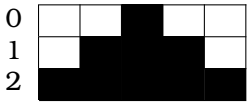
Exercice 3 : Le monde du vivant est d'une fantastique complexité qui à la base n'est fait que d'un nombre assez limité d'éléments chimiques, qui interagissent selon des lois relativement simples et bien connues. C'est aussi le cas de ce qu'on appelle les automates cellulaires élémentaires.

Pour réaliser un automate cellulaire élémentaire, nous allons considérer un tableau de "Booléens". Chaque case peut être soit fausse, soit vrai. Au démarrage, nous allons supposer qu'une seule case est vraie. Dans la suite on va représenter notre tableau de Booléen avec un tableau de case noir ou blanche pour les cas où la valeur du tableau est vraie ou fausse. Par exemple, on a $\{0,0,1,0,0\}$ qui donne visuellement :

Ensuite nous allons faire évoluer notre système, en définissant des règles qui vont conditionner comment la couleur des cases change à chaque étape. Pour faire au plus simple, nous allons supposer que la nouvelle couleur d'une case dépend de sa couleur actuelle et de celle de ses voisines. Voici une règle de ce genre :

- si une case est noire, elle reste noire ;
- si elle est blanche, elle devient noire si elle possède au moins une voisine noire.

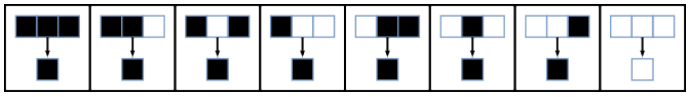
Si on applique cette règle à notre situation de départ avec une seule case noire, voici ce que l'on obtient à chaque pas de temps :



Pour représenter l'évolution d'un automate cellulaire élémentaire, on peut le simuler sur plusieurs centaines d'itérations, et superposer les différentes lignes obtenues. Et on obtient ici un triangle.

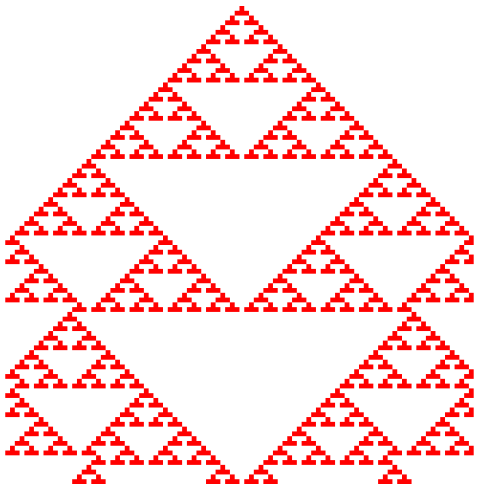
On peut remarquer que chacune des cellules pouvant prendre deux états, il existe $2^3 = 8$ configurations (ou motifs) possibles d'un tel voisinage. Pour que l'automate cellulaire fonctionne, il faut définir quel doit être l'état, à la génération suivante, d'une cellule pour chacun de ces motifs. Il y a $2^8 = 256$ façons différentes de s'y prendre, soit donc 256 automates cellulaires différents de ce type.

Les automates de cette famille sont dits « élémentaires ». On les désigne souvent par un entier entre 0 et 255 dont la représentation binaire est la suite des états pris par l'automate sur les motifs successifs 111, 110, 101, etc (1 pour vrai et 0 pour faux). À titre d'exemple, l'automate précédent peut être défini via la table suivante :



En fait chacune des configurations va correspondre à un entier n qui donne le valeur du $n^{\text{ème}}$ bit de la représentation binaire d'un nombre entre 0 et 255. Pour l'exemple précédent ce nombre est 254.

Le but de l'exercice est de proposer une solution permettant d'obtenir une figure `ppm` représentant la simulation des n premières itérations de l'automate. Dans cet exercice la règle et le nombre d'itération sont donnés via la ligne de commande. La figure suivante représente la figure générée avec la règle 22 :



Exercice 4 : Le but de cet exercice est de réaliser un programme permettant de jouer au Tic-Tac-Toe. Pour rappel, le Tic-Tac-Toe est un jeu à deux joueurs. Ils doivent remplir chacun à leur tour une case de la grille avec le symbole qui leur est attribué : O ou X. Le gagnant est celui qui arrive à aligner trois symboles identiques, horizontalement, verticalement ou en diagonale.

Vous devrez réaliser votre travail dans un contexte de ressources très limitées, puisque vous n'aurez le droit de déclarer qu'une seule variable de type `unsigned long int` ! Vous n'avez pas le droit de déclarer de fonctions si celles-ci prennent des paramètres ou retourne autre chose que `void` (même pas `void *`) !

```
   |   |
-----------
   |   |
-----------
   |   |
Ligne: 1
Colonne: 1

 X |   |
-----------
   |   |
-----------
   |   |
Ligne: 2
Colonne:
```

Il est possible de demander à l'utilisateur un `unsigned char` à l'aide de la séquence `%hhu` de la fonction `scanf`.

Est-ce que vous pensez qu'il est possible de faire le même exercice dans une contexte encore plus critique en ressource ? Par exemple avec un `unsigned int` ?