TP 2

Fonctions – Récursivité

1 Des fonctions simples

Pour cette partie, vous pouvez télécharcher le fichier `TP02_Squelette.c` sur « *cahier de prépa* ». Ce fichier permettra de tester vos fonctions.

Exercice 2.1 Écrire une fonction `minuscule` qui à partir d'un caractère en majuscule renvoie son pendant minuscule, tout autre caractère devra être renvoyé inchangé. On travaillera avec les caractères de la table ASCII classique (sur sept bits – dont le code va de 0 à 127 donc).

```c
char minuscule(char c);
```

Par exemple :

- `minuscule('A')` doit renvoyer `'a'`;
- `minuscule('a')` doit renvoyer `'a'`;
- `minuscule('1')` doit renvoyer `'1'`...

Exercice 2.2 Écrire une fonction `pythagore` qui calcule la longueur de l'hypothénuse à partir de la longueur des deux cathètes [1].

```c
double pythagore(double a, double b);
```

Exercice 2.3 Écrire une fonction `quotient` telle que l'appel `quotient(a, b)` calcule le quotient de la division euclidienne de a par b.
Attention, vous n'avez pas le droit d'utiliser le signe / !
Vous pouvez seulement utiliser les symboles `+`, `-` et éventuellement `*`.

```c
uint64_t quotient(uint64_t a, uint64_t b);
```

On pourra supposer, sans le vérifier, que b est non nul.

1. vous avez appris un mot là, avouez-le...

11 Récursivité

Exercice 2.4 Écrire une fonction récursive `binom` permettant de calculer $\binom{n}{p}$.

```
uint64_t binom(int n, int p);
```

On pourra supposer, sans le vérifier, que $p \leqslant n$.
On ne calculera aucune factorielle dans cette fonction.

Exercice 2.5 L'algorithme d'Euclide permet de calculer le PGCD de deux entiers naturels a et b ; mais il permet en réalité de faire plus que ça. L'identité de Bézout assure qu'il existe deux entiers relatifs x et y (non uniques) tels que :

$$ax + by = \text{pgcd}(a, b).$$

En étudiant l'exemple suivant, écrire une fonction `bezout` qui calcule un couple de Bézout des deux entiers donnés en paramètre. On pourra travailler avec une structure décrivant un couple d'entiers :

```
struct couple {
  int fst;
  int snd;
};

typedef struct couple couple;
```

```
couple bezout(int a, int b);
```

$\text{pgcd}(120, 23)$	$120 = 23 \times 5 + 5$	$1 = 1 \times \boxed{1} + 0 \times \boxed{0}$
$= \text{pgcd}(23, 5)$	$23 = 5 \times 4 + 3$	$1 = 1 \times 1 + \big(2 \times 1 + 1 \times (-2)\big) \times 0 = 2 \times \boxed{0} + 1 \times \boxed{1}$
$= \text{pgcd}(5, 3)$	$5 = 3 \times 1 + 2$	$1 = 2 \times 0 + \big(3 \times 1 + 2 \times (-1)\big) \times 1 = 3 \times \boxed{1} + 2 \times \boxed{-1}$
$= \text{pgcd}(3, 2)$	$3 = 2 \times 1 + 1$	$1 = 3 \times 1 + \big(5 \times 1 + 3 \times (-1)\big) \times (-1) = 5 \times \boxed{-1} + 3 \times \boxed{2}$
$= \text{pgcd}(2, 1)$	$2 = 1 \times 2 + 0$	$1 = 5 \times (-1) + \big(23 \times 1 + 5 \times (-4)\big) \times 2 = 23 \times \boxed{2} + 5 \times \boxed{-9}$
$= \text{pgcd}(1, 0) = 1$	$1 = 1 \times 1 + 0$	$1 = 23 \times 2 + \big(120 \times 1 + 23 \times (-5)\big) \times (-9) = 120 \times \boxed{-9} + 23 \times \boxed{47}$

Exercice 2.6 Les tours de Hanoï est un jeu inventé par Édouard Lucas en 1883. Il est formé de sept disques de tailles différentes répartis en trois colonnes (nommées `A`, `B` et `C`). Au départ, tous les disques sont empilés sur la colonne de gauche par taille croissante.

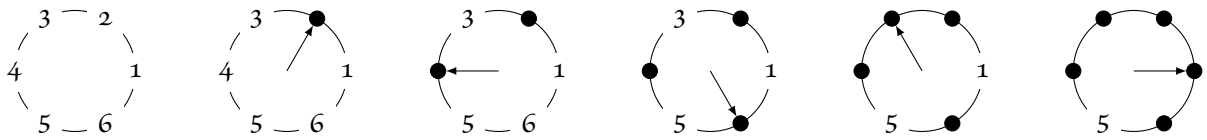
Le seul mouvement possible est le déplacement d'un disque situé au sommet d'une colonne vers le sommet d'une autre colonne, à condition que la colonne d'arrivée soit vide ou que le disque déplacé soit plus petit que son sommet. On note `n -> n'` le déplacement d'un disque de la colonne n vers la colonne n'. Le but du jeu est de déplacer tous les disques vers la colonne de droite.

Écrire un programme qui affiche une solution du jeu sous la forme d'une suite de mouvements. Par exemple, pour trois disques :

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

Indication : dans une variante, le jeu n'est formé que de six disques. Si l'on sait résoudre le jeu à six disques, comment le résoudre à sept disques ?

Exercice 2.7 *Problème de* JOSÈPHE. Une assemblée de n personnes, numérotées de 1 à n, forme un cercle. On décide de parcourir le cercle dans l'ordre croissant des numéros et d'éliminer une personne sur deux (qui sort alors du cercle), et ainsi de suite jusqu'à ce qu'il ne reste plus qu'une personne dans le cercle. On note J la fonction qui, à chaque entier $n \geqslant 1$, associe le numéro de la dernière personne restant après élimination de tous les autres candidats.
Par exemple, pour $n = 6$, l'ordre d'élimination des candidats est $(2, 4, 6, 3, 1)$ et le dernier candidat restant est le numéro 5 (donc $J(6) = 5$).



1. On suppose dans cette question que n est pair et on pose $p = \frac{n}{2}$.
   Après un tour du cercle, p candidats ont été éliminés. On renumérote les p candidats restants de 1 à p. Exprimer l'ancien numéro x_i de chaque candidat restant en fonction de son nouveau numéro i et en déduire une relation liant $J(2p)$ à $J(p)$.
2. Trouver de même une relation liant $J(2p+1)$ à $J(p)$.
3. Écrire une fonction récursive `josephe` calculant $J(n)$.

   ```
   int josephe(int n);
   ```

Tortue. La bibiothèque `turtle`, inspiré de la programmation Logo, permet de dessiner rapidement des figures simples en guidant une « tortue » (qui représente le stylo) avec des commandes simples.
Voici une partie des commandes utilisables :

```
/*
    Initialize the 2d field that the turtle moves on. This must be called
    before any of the other functions in this library.
*/
void turtle_init(int width, int height);

/*
    Move the turtle forward, drawing a straight line if the pen is down.
*/
void turtle_forward(int pixels);

/*
    Move the turtle backward, drawing a straight line if the pen is down.
*/
void turtle_backward(int pixels);

/*
    Turn the turtle to the left by the specified number of degrees.
*/
void turtle_turn_left(double angle);

/*
    Turn the turtle to the right by the specified number of degrees.
*/
void turtle_turn_right(double angle);

/*
    Set the pen status to ”up” (do not draw).
*/
void turtle_pen_up();

/*
    Set the pen status to ”down” (draw).
*/
void turtle_pen_down();

/*
    Move the turtle to the specified location, drawing a straight line if the
    pen is down. Takes integer coordinate parameters.
*/
void turtle_goto(int x, int y);

/*
    Set the current drawing color. Each component (red, green, and blue) may
    be any value between 0 and 255 (inclusive). Black is (0,0,0) and white is
    (255,255,255).
*/
void turtle_set_pen_color(int red, int green, int blue);
```

```
/*
    Draw a straight line between the given coordinates, regardless of current
    turtle location or pen status.
*/
void turtle_draw_line(int x0, int y0, int x1, int y1);

/*
    Draw a circle at the given coordinates with the given radius, regardless
of
    current turtle location or pen status.
*/
void turtle_draw_circle(int x, int y, int radius);

/*
    Save current field to a .bmp file.
*/
void turtle_save_bmp(const char *filename);

/*
    Returns the current x-coordinate.
*/
double turtle_get_x();

/*
    Returns the current x-coordinate.
*/
double turtle_get_y();
```

Le code suivant montre un exemple simple d'utilisation de cette tortue :

```
#include "turtle.h"
int NB_ITERATION = 200;
int ANGLE = 119;

int main(void) {
  turtle_init(500, 500);
  int longueur = 20;
  for (int i = 0; i < NB_ITERATION; i++) {
    turtle_forward(longueur);
    turtle_turn_left(ANGLE);
    longueur += 2;
  }
  turtle_save_bmp("exemple");
  return 0;
}
```

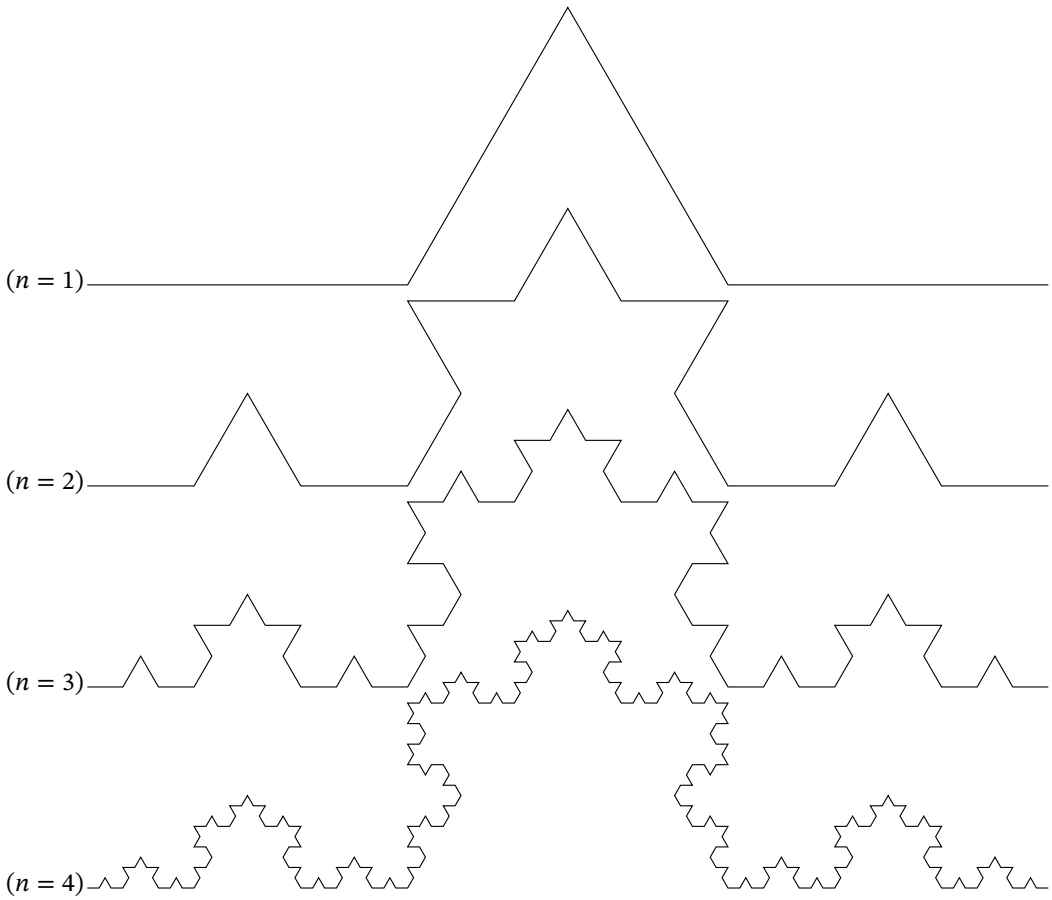
Avant de compiler, il faudra récupérer les deux fichiers `turtle.c` et `turtle.h` sur « *cahier de prépa* » et compiler avec la commande suivante :

```
gcc -o nom_de_sortie turtle.c fichier.c -lm
```

On indique ainsi qu'il est nécessaire de compiler aussi les commandes provenant du fichier `turtle.c` (et le `-lm` est là pour indiquer qu'on utilise des fonctions mathématiques).

Exercice 2.8 Le flocon de Von Koch est un exemple de courbe fractale. Cette courbe se définit comme la limite de la suite de courbes dont le premier élément est un segment :

et dans laquelle on passe d'un élément au suivant en divisant chaque segment en trois et en remplaçant le morceau du milieu par deux segments formant un triangle équilatéral avec le segment supprimé.



Écrire un programme qui dessine le n^e élément de cette suite.

Exercice 2.9 On souhaite dessiner la figure suivante, où les rayons de chaque cercle est divisé par deux à chaque génération :

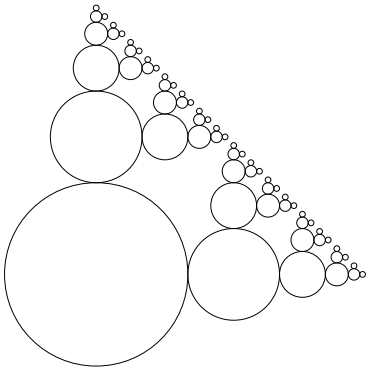


FIGURE 2.1 – Six générations de cercles dessinées

Écrire une fonction récursive permettant de tracer la figure voulue (*en bonus, on pourra réflechir à une fonction permettant de dessiner des petites bulles dans les « quatres » directions*).

Exercice 2.10 **Courbe du dragon.** Dans cet exercice, les points seront représentés par leur affixe (on travaille donc dans le plan complexe). Je reprends la structure définie dans le TP précédent pour représenter les complexes.

1. Soient A et B deux points d'affixe z_A et z_B. Soit C tel que ABC est un triangle rectangle isocèle en C direct. Calculer l'affixe z_C de C.
2. Programmer la fonction `calcule_zc` correspondante :

   ```
   complexe calcule_zc(complexe za, complexe zb);
   ```

3. La fonction `dragon` est alors définie comme ceci :
   - Sa signature est :

     ```
     void dragon(int n, complexe za, complexe zb);
     ```

     Elle ne renvoie rien, mais dessine une ligne brisée avec la tortue.
   - Pour tous points A et B, `dragon(0, za, zb)` est le segment $[AB]$.
   - Pour tous points A et B et tout entier $n \in \mathbb{N}^*$, soit C comme dans la question 1, alors `dragon(n, za, zb)` est la concaténation des segments de `dragon(n - 1, za, zc)` et de `dragon(n - 1, zb, zc)` (attention, l'ordre des points est important).

   Programmer la fonction `dragon`.

Fonctions mutuellement récursives (pour les rapides).

Exercice 2.11 Soit $a \in \mathbb{R}_+^*$. Soient u et v les deux suites définies par :

$$\begin{cases} u_0 = 1 \text{ et } v_0 = a, \\ \forall n \in \mathbb{N},\ u_{n+1} = \frac{2u_n v_n}{u_n + v_n} \text{ et } v_{n+1} = \frac{u_n + v_n}{2}. \end{cases}$$

On montre facilement que ces deux suites sont adjacentes et convergent vers $\sqrt{a}$. Écrire des fonctions récursives pour calculer u_n et v_n en fonction de n. On prendra ici :

- découvrir ce que sont des fonctions mutuellement récursives (par exemple sur Wikipedia), puis rechercher sur d'autres pages comment définir en C des fonctions mutuellement récursives ;
- dans un second temps, réfléchir à une méthode efficace pour faire ce calcul.