Département d'Informatique
DUT Informatique S1
Année 2014/2015
Algorithmique et programmation

Algorithmique et programmation

Exercices de TD avec solutions

Table des matières

Version du 18 décembre 2014.

1 Séquences d'actions

1.1 Écluse

Écrire l'*idée de l'algorithme* permettant de modéliser le franchissement d'une écluse par un bateau.

Les portes d'une écluse sont munies de vannes que l'on peut ouvrir ou fermer et qui permettent de laisser passer l'eau afin de mettre l'écluse à niveau (figure 1). Les portes sont également équipées de feux pouvant être vert ou rouge, pour autoriser ou non le passage du bateau.

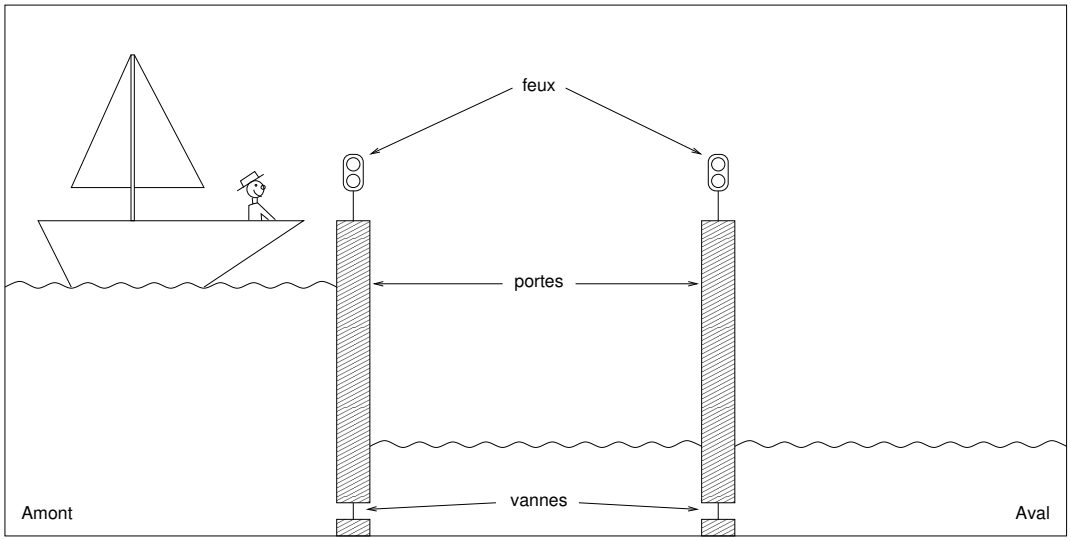

FIGURE 1 – Schéma d'un écluse

On considérera dans un premier temps que le bateau descend la rivière et que lorsqu'il se présente devant l'écluse, celle-ci est en niveau bas.

Correction

On suppose qu'au départ, les deux portes sont fermées, les vannes sont fermées, et les deux feux sont au rouge. Une idée d'algorithme est la suivante :

- ouvrir la vanne amont (l'écluse se remplit)
- ouvrir la porte amont
- passer le feu amont au vert (le bateau entre dans l'écluse)
- passer le feu amont au rouge
- fermer la porte amont
- fermer la vanne amont
- ouvrir la vanne aval (l'écluse se vide, le bateau descend)
- ouvrir la porte aval
- passer le feu aval au vert (le bateau sort de l'écluse)
- passer le feu aval au rouge
- fermer la porte aval
- fermer la vanne aval

1.2 Radiateurs

1. Concevoir un algorithme pour calculer le nombre de radiateurs dont on a besoin pour chauffer une pièce. On sait qu'un radiateur est capable de chauffer $8\,\mathrm{m}^3$. L'utilisateur donnera la longueur, la largeur et la hauteur de la pièce en mètres.

Correction

Données

Aucune donnée.

Résultat

Nombre de radiateurs nécessaires pour chauffer une pièce.

Idée

Demander les dimensions de la pièce à l'utilisateur, calculer le volume de la pièce et le diviser par la capacité de chauffage d'un radiateur. En prendre ensuite la partie entière supérieure pour obtenir le nombre de radiateurs recherché.

Lexique des constantes

capacité	(réel)	$= 8$	capacité de chauffage d'un radiateur (m^3)

Lexique des variables

nombre	(entier)	nombre de radiateurs	RÉSULTAT
longueur	(réel)	longueur de la pièce	INTERMÉDIAIRE
largeur	(réel)	largeur de la pièce	INTERMÉDIAIRE
hauteur	(réel)	hauteur de la pièce	INTERMÉDIAIRE
volume	(réel)	volume de la pièce	INTERMÉDIAIRE

Algorithme

longueur $\leftarrow$ lire
largeur $\leftarrow$ lire
hauteur $\leftarrow$ lire
volume $\leftarrow$ *longueur* $\times$ *largeur* $\times$ *hauteur*
nombre $\leftarrow \lceil$ *volume*/*capacité* $\rceil$

---

2. Détailler le déroulement de l'algorithme pour une pièce de dimensions $6 \times 4 \times 2{,}5$ m.

Correction

Au cours du déroulement de l'algorithme, les variables prennent les valeurs suivantes : *longueur* $= 6$, *largeur=4*, *hauteur* $= 2{,}5$, *volume* $= 60$, *nombre* $= 8$.

3. Traduire l'algorithme dans un programme Java complet.

Correction

```
import java.util.*;

class Radiateurs {

    static final Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        final double capacite = 8;

        int nombre;
        double longueur;
        double largeur;
        double hauteur;
        double volume;

        longueur = input.nextDouble();
        largeur = input.nextDouble();
        hauteur = input.nextDouble();
        volume = longueur * largeur * hauteur;
        nombre = (int)Math.ceil(volume / capacite);

        System.out.println("Nombre = " + nombre);
    }

}
```

1.3 Gazon

On veut calculer la surface du gazon et le temps de tonte du gazon pour un terrain rectangulaire sur lequel sont bâtis une maison rectangulaire et un appentis triangulaire, et qui comporte une piscine circulaire. On sait que la tonte du gazon prend x secondes au m^2 (x est donné), et on donne les dimensions indiquées sur la figure 2.

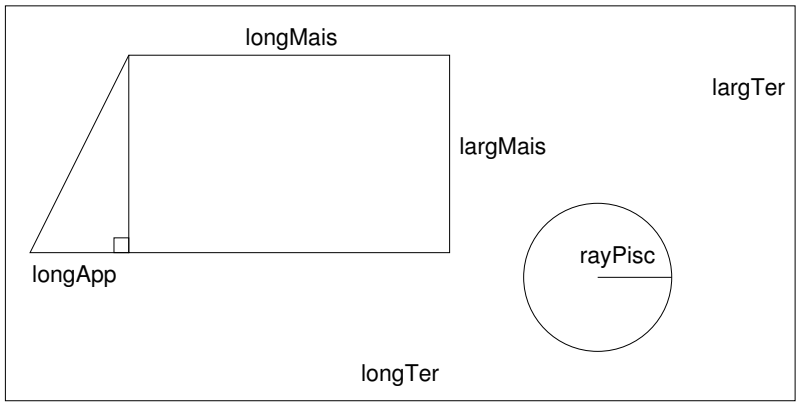

FIGURE 2 – Schéma du terrain à tondre

Correction

Données
Dimensions du terrain et des éléments, comme sur le plan (m), temps de tonte unitaire (s/m^2).

Résultat
Surface de gazon et temps total de tonte.

Idée
Calculer la surface de gazon en retranchant de la superficie du terrain, la superficie occupée par les différents éléments. Calculer ensuite le temps total de tonte en multipliant la surface de gazon par le temps de tonte unitaire.

Lexique des constantes
PI (réel) = 3,141 592 653 59 constante mathématique π

Lexique des variables

longTer	(réel)	longueur du terrain (m)	DONNÉE
largTer	(réel)	largeur du terrain (m)	DONNÉE
longMais	(réel)	longueur de la maison (m)	DONNÉE
largMais	(réel)	largeur de la maison (m)	DONNÉE
longApp	(réel)	longueur de l'appentis (m)	DONNÉE
rayPisc	(réel)	rayon de la piscine (m)	DONNÉE
x	(réel)	temps de tonte unitaire (s/m^2)	DONNÉE
surface	(réel)	surface du terrain (m^2)	RÉSULTAT
temps	(réel)	temps total de tonte (s)	RÉSULTAT

Algorithme
$surface \leftarrow longTer \times largTer$
$surface \leftarrow surface - longMais \times largMais$
$surface \leftarrow surface - (longApp \times largMais)/2$
$surface \leftarrow surface - PI \times rayPisc \times rayPisc$
$temps \leftarrow x \times surface$

1.4 Heures, minutes, secondes

Écrire un algorithme pour convertir un nombre de secondes en un nombre d'heures, de minutes et de secondes. On utilisera les opérateurs modulo et division entière.

Correction

Données
Une durée, en secondes.

Résultat
La même durée, en heures, minutes, secondes.

Idée
Décomposer le nombre total des secondes par division entières successives, sachant qu'il y a 60 secondes dans une minute et 3600 secondes dans une heure.

Lexique des variables

duree	(entier)	durée à convertir, en secondes	DONNÉE
heures	(entier)	nombre d'heures	RÉSULTAT
minutes	(entier)	nombre de minutes	RÉSULTAT
secondes	(entier)	nombre de secondes	RÉSULTAT

Algorithme

heures ← *duree*/3600
minutes ← (*duree* mod 3600)/60
secondes ← *duree* mod 60

1.5 Programme Gazon

Utiliser les algorithmes précédents pour écrire un programme Java qui calcule et affiche surface de gazon et temps de tonte. Les informations nécessaires seront demandées à l'utilisateur. Le temps de tonte sera affiché en heures, minutes, secondes.

Indications

En Java,
- l'opération modulo (mod) se note `%`. Par exemple : `duree % 3600`.
- la constante mathématique π est définie par `Math.PI`.

Correction

```
import java.util.*;

class Gazon {

    static final Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        /* Données de l'algorithme du gazon */
        float longTer;
        float largTer;
        float longMais;
        float largMais;
        float longApp;
        float rayPisc;
        float x;
        /* Résultats de l'algorithme du gazon */
        float surface;
        float temps;

        /* Donnée de l'algorithme de la durée */
        int duree;
        /* Résultats de l'algorithme de la durée */
        int heures;
        int minutes;
        int secondes;

        /***** Entrée des données *****/

        System.out.println("Entrer la longueur et la largeur du terrain (m)");
        longTer = input.nextFloat();
        largTer = input.nextFloat();
        System.out.println("Entrer la longueur et la largeur de la maison (m)");
        longMais = input.nextFloat();
        largMais = input.nextFloat();
        System.out.println("Entrer la longueur de l'appentis (m)");
        longApp = input.nextFloat();
        System.out.println("Entrer le rayon de la piscine (m)");
        rayPisc = input.nextFloat();
        System.out.println("Entrer le temps de tonte unitaire (s/m^2)");
        x = input.nextFloat();

        /***** Algorithme du gazon *****/
        surface = longTer * largTer;
        surface -= longMais * largMais;
        surface -= (longApp * largMais) / 2.0;
        surface -= Math.PI * rayPisc * rayPisc;
        temps = x * surface;

        /***** Préparation des données pour l'algorithme de la durée *****/
        duree = Math.round(temps);

        /***** Algorithme de la durée *****/
        heures = duree / 3600;
        minutes = (duree % 3600) / 60;
        secondes = duree % 60;

        /***** Affichage des résultats *****/
        System.out.println("Surface de gazon : " + surface + " m^2");
        System.out.println("Temps de tonte : " +
                           heures + " heures, " +
                           minutes + " minutes, " +
                           secondes + " secondes");
    }

}
```

2 Conditionnelles

2.1 Dérouler un algorithme

Réalisez l'exécution de l'algorithme suivant en donnant successivement les valeurs 20, 150, 50, 51 et 400 à *valeur*.

Lexique des variables

valeur	(entier)	entier donné	DONNÉE

Algorithme

```
si valeur > 100 alors
|  écrire "A"
|  si valeur > 200 alors
|  |  écrire "B"
|  sinon
|  |  écrire "C"
|  fsi
sinon
|  si valeur > 50 alors
|  |  écrire "D"
|  sinon
|  |  écrire "E"
|  fsi
fsi
```

Correction

Donnée	Sorties de l'algorithme (impression)
20	E
150	A C
50	E
51	D
400	A B

2.2 Pizzas

On ne fait pas forcément des économies en achetant plus gros. Est-ce vrai quand on achète des pizzas ? Concevoir un algorithme qui lit le diamètre de deux pizzas, leur numéro et leur prix puis qui imprime le numéro de celle qui a le meilleur rapport *taille/prix*.

Correction

Données

Aucune donnée.

Résultat

Aucun résultat (affichage).

Idée

On remarquera que le « meilleur » rapport taille/prix est celui qui est le plus grand. Pour la définition de la taille, le plus naturel est d'utiliser l'aire de la pizza. Il suffit ensuite de calculer les rapports taille/prix de chacune des pizzas, et de les comparer pour afficher le numéro de celle qui a le meilleur rapport.

Lexique des constantes

PI	(réel)	= 3,141 592 653 59	constante mathématique π

Lexique des variables

$numéro_1$	(entier)	numéro de la première pizza	INTERMÉDIAIRE
$numéro_2$	(entier)	numéro de la deuxième pizza	INTERMÉDIAIRE
$diamètre_1$	(réel)	diamètre de la première pizza	INTERMÉDIAIRE
$diamètre_2$	(réel)	diamètre de la deuxième pizza	INTERMÉDIAIRE
$prix_1$	(réel)	prix de la première pizza	INTERMÉDIAIRE
$prix_2$	(réel)	prix de la deuxième pizza	INTERMÉDIAIRE
$rapport_1$	(réel)	Rapport taille/prix de la première pizza	INTERMÉDIAIRE
$rapport_2$	(réel)	Rapport taille/prix de la deuxième pizza	INTERMÉDIAIRE

Algorithme

écrire "`Entrer numéro, diamètre et prix de la première pizza`"
$numéro_1 \leftarrow$ lire
$diamètre_1 \leftarrow$ lire
$prix_1 \leftarrow$ lire
écrire "`Entrer numéro, diamètre et prix de la deuxième pizza`"
$numéro_2 \leftarrow$ lire
$diamètre_2 \leftarrow$ lire
$prix_2 \leftarrow$ lire
$rapport_1 \leftarrow diamètre_1 \times diamètre_1 \times PI/(4 \times prix_1)$
$rapport_2 \leftarrow diamètre_2 \times diamètre_2 \times PI/(4 \times prix_2)$
si $rapport_1 > rapport_2$ **alors**
| écrire "`C'est la pizza`", $numéro_1$, "`qui a le meilleur rapport taille/prix.`"
sinon si $rapport_1 < rapport_2$ **alors**
| écrire "`C'est la pizza`", $numéro_2$, "`qui a le meilleur rapport taille/prix.`"
sinon
| écrire "`Les deux pizzas on le même rapport taille/prix.`"
fsi

Remarque : Ce n'est pas équivalent de prendre le périmètre et l'aire comme mesure de la taille. Par exemple, si $diamètre_1 = 4$, $diamètre_2 = 6$, $prix_1 = 2$ et $prix_2 = 4$, alors

– avec le périmètre, c'est la pizza n° 1 qui a le meilleur rapport :

$$périmètre_1 = 4\pi \qquad rapport_1 = 4\pi/2 = 2\pi$$
$$périmètre_2 = 6\pi \qquad rapport_2 = 6\pi/4 = 1{,}5\pi$$

– avec l'aire, c'est la pizza n° 2 qui a le meilleur rapport :

$$aire_1 = (4/2)^2 = 4\pi \qquad rapport_1 = 4\pi/2 = 2\pi$$
$$aire_2 = (6/2)^2 = 9\pi \qquad rapport_2 = 9\pi/4 = 2{,}25\pi$$

2.3 Coordonnées

Écrire un algorithme qui, étant donné les coordonnées x et y d'un point, détermine dans quelle partie (A, B, C ou D) du plan se trouve le point (cf. figure 3).

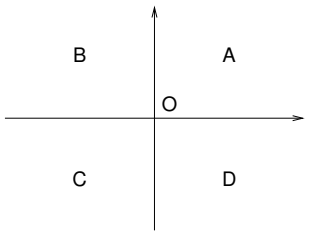

FIGURE 3 – Parties du plan

Correction

Données
Coordonnées (x, y) d'un point.

Résultat
Affichage de la partie du point dans laquelle se trouve le point.

Idée
Un simple enchaînement de structures conditionnelles.

Lexique des variables

x	(réel)	abscisse du point	DONNÉE
y	(réel)	ordonnée du point	DONNÉE

Algorithme

si $x < 0$ **alors**
  si $y < 0$ **alors**
    écrire "`C`"
  sinon si $y > 0$ **alors**
    écrire "`B`"
  sinon // $y = 0$
    écrire "`BC`"
  fsi
sinon si $x > 0$ **alors**
  si $y < 0$ **alors**
    écrire "`D`"
  sinon si $y > 0$ **alors**
    écrire "`A`"
  sinon // $y = 0$
    écrire "`AD`"
  fsi
sinon // $x = 0$
  si $y < 0$ **alors**
    écrire "`CD`"
  sinon si $y > 0$ **alors**
    écrire "`AB`"
  sinon // $y = 0$
    écrire "`O (ABCD)`"
  fsi
fsi

Pour aller plus loin. Écrire un algorithme de conversion des coordonnées du point en coordonnées polaires. La valeur de l'angle doit être dans l'intervalle $[0 ; 2\pi[$.

2.4 Mois de l'année

Écrire un algorithme qui demande un numéro de mois à l'utilisateur et indique en retour son nom et le nombre de jours dans ce mois.

Correction

Données
Aucune donnée.

Résultat
Aucun (affichage).

Idée

(i) demander un numéro de mois à l'utilisateur ;

(ii) afficher le nom et le nombre de jours correspondant au numéro de mois (utiliser pour cela une structure **selon que**).

Lexique des variables

mois	(entier)	numéro de mois	INTERMÉDIAIRE

Algorithme

écrire “`Entrer un numéro de mois`”
mois ← lire
selon que *mois* **est**
  cas *1* **:** écrire “`janvier (31 jours)`”
  cas *2* **:** écrire “`février (28 ou 29 jours)`”
  cas *3* **:** écrire “`mars (31 jours)`”
  cas *4* **:** écrire “`avril (30 jours)`”
  cas *5* **:** écrire “`mai (31 jours)`”
  cas *6* **:** écrire “`juin (30 jours)`”
  cas *7* **:** écrire “`juillet (31 jours)`”
  cas *8* **:** écrire “`août (31 jours)`”
  cas *9* **:** écrire “`septembre (30 jours)`”
  cas *10* **:** écrire “`octobre (31 jours)`”
  cas *11* **:** écrire “`novembre (30 jours)`”
  cas *12* **:** écrire “`décembre (31 jours)`”
  défaut : écrire “`numéro invalide`”
fselon

---

3 Itérations

3.1 Lignes d'étoiles

1. Concevoir un algorithme qui, pour un caractère imprimable et un nombre n donnés, imprime une barre horizontale de n de ces caractères.

   ****

Correction

Données

Un caractère imprimable, une longueur de ligne n.

Résultat

Affichage d'une ligne de n fois le caractère.

Idée

Répéter n fois l'affichage du caractère, à l'aide d'une boucle **pour**

Lexique des variables

car	(caractère)	un caractère imprimable	DONNÉE
n	(entier)	longueur de la ligne	DONNÉE
i	(entier)	compteur de caractères	INTERMÉDIAIRE

Algorithme

pour i **de** 1 **à** n **faire**
  écrire *car*
fpour
écrire '`\n`'

---

2. Modifier l'algorithme pour l'impression d'une barre double.

   ****
   ****

Correction

Données

Un caractère imprimable, une longueur de ligne n.

Résultat

Affichage de deux lignes de n fois le caractère.

Idée

Répéter 2 fois l'algorithme précédent.

Lexique des variables

car	(caractère)	un caractère imprimable	DONNÉE
n	(entier)	longueur de la ligne	DONNÉE
i	(entier)	compteur de caractères	INTERMÉDIAIRE

Algorithme

pour i **de** 1 **à** n **faire**
  écrire *car*
fpour
écrire '`\n`'
pour i **de** 1 **à** n **faire**
  écrire *car*
fpour
écrire '`\n`'

---

3. Modifier l'algorithme pour l'impression d'une barre d'épaisseur quelconque donnée.

Correction

Données

Un caractère imprimable, une longueur de ligne n, une épaisseur m.

Résultat

Affichage de m lignes de n fois le caractère.

Idée

Répéter m fois le premier algorithme. Utiliser pour cela une boucle **pour**.

Lexique des variables

car	(caractère)	un caractère imprimable	DONNÉE
n	(entier)	longueur de la ligne	DONNÉE
m	(entier)	nombre de lignes	DONNÉE
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de caractères	INTERMÉDIAIRE

Algorithme

pour i **de** 1 **à** m **faire**
  pour j **de** 1 **à** n **faire**
    écrire *car*
  fpour
  écrire '`\n`'
fpour

4. (optionnel) Transformer les algorithmes ci-dessus en fonctions.
5. Écrire un programme Java implémentant la dernière version de l'algorithme (épaisseur quelconque).

Correction

```
import java.util.*;

class Lignes {

    static final Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        char car;                   // caractère à imprimer
        int n;                      // longueur de ligne
        int m;                      // épaisseur
        int i;                      // compteur de lignes
        int j;                      // compteur de caractères

        System.out.print("Caractère ? ");
        car = input.next().charAt(0);
        System.out.print("Longueur ? ");
        n = input.nextInt();
        System.out.print("Épaisseur ? ");
        m = input.nextInt();

        for (i = 1; i <= m; ++i) {
            for (j = 1; j <= n; ++j)
                System.out.print(car);
            System.out.println();
        }
    }
}
```

3.2 Carrés et cubes

Écrire un algorithme qui imprime la suite des carrés et des cubes des entiers compris entre 10 et 100.

Correction

Données
Aucune donnée.

Résultat
Aucun (affichage).

Idée
Énumérer les valeurs de *borneInf* à *borneSup* (utiliser une boucle **pour**), et utiliser la valeur de l'indice de boucle pour en calculer le carré et le cube.

Lexique des constantes

borneInf	(entier)	$= 10$	borne inférieure de l'intervalle
borneSup	(entier)	$= 100$	borne supérieure de l'intervalle

Lexique des variables

i	(entier)	compteur	INTERMÉDIAIRE
carré	(entier)	valeur de i^2	INTERMÉDIAIRE
cube	(entier)	valeur de i^3	INTERMÉDIAIRE

Algorithme

pour i **de** *borneInf* **à** *borneSup* **faire**
  $carré \leftarrow i \times i$
  $cube \leftarrow carré \times i$
  écrire i, *carré*, *cube*
fpour

3.3 Triangle de nombres

Concevoir un algorithme qui imprime pour n donné :

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
..........
...........
............
1 2 3 4 5 6 ... n
```

Correction

Données
Un entier n.

Résultat
Aucun (affichage suivant énoncé).

Idée
Énumérer les lignes et, pour chaque ligne, imprimer les nombres de 1 au numéro de ligne. Utiliser pour cela deux boucles imbriquées.

Lexique des variables

n	(entier)	le paramètre donné	DONNÉE
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE

Algorithme

pour i **de** 1 **à** n **faire**
  pour j **de** 1 **à** i **faire**
    écrire j
  fpour
  écrire '`\n`'
fpour

3.4 Tables de multiplications

Concevoir un algorithme qui imprime l'ensemble des tables de multiplication par 1, 2, ..., 12.
Modifier ensuite l'algorithme pour qu'il imprime 3 tables côte à côte.

Correction

Affichage table par table

Données
Aucune.

Résultat
Aucun (affichage).

Idée
Utiliser deux boucles imbriquées pour énumérer tous les produits : une première boucle sur les tables (de 1 à 12), et une seconde sur les lignes dans les tables (de 1 à 12 également).

Lexique des variables

i	(entier)	compteur pour les tables	INTERMÉDIAIRE
j	(entier)	compteur pour les lignes dans les tables	INTERMÉDIAIRE
produit	(entier)	produit de i par j	INTERMÉDIAIRE

Algorithme

pour i **de** 1 **à** 12 **faire**
  pour j **de** 1 **à** 12 **faire**
    $produit \leftarrow i \times j$
    écrire i, "×", j, "=", *produit*, "`\n`"
  fpour
  écrire "`\n`"
fpour

Affichage des tables trois par trois

Données
Aucune.

Résultat
Aucun (affichage).

Idée
Utiliser trois boucles imbriquées pour énumérer tous les produits : une première boucle sur les lignes de tables (de 1 à 4), une seconde sur les lignes dans les tables (de 1 à 12), et une troisième sur les tables (3 tables à chaque fois, les bornes sont définies à partir de la ligne de table courante).

Lexique des variables

i	(entier)	compteur pour les tables	INTERMÉDIAIRE
j	(entier)	compteur pour les lignes dans les tables	INTERMÉDIAIRE
k	(entier)	compteur pour les lignes de tables	INTERMÉDIAIRE
produit	(entier)	produit de i par j	INTERMÉDIAIRE

Algorithme

pour k **de** 1 **à** 4 **faire**
  pour j **de** 1 **à** 12 **faire**
    pour i **de** $3 \times (k-1) + 1$ **à** $3 \times k$ **faire**
      $produit \leftarrow i \times j$
      écrire i, "×", j, "=", *produit*
    fpour
    écrire "`\n`"
  fpour
  écrire "`\n`"
fpour

3.5 Puissances

Imprimer la suite des puissances de x (nombre réel donné) : x, x^2, ..., x^n, avec n un entier positif donné. La solution ne devra pas utiliser d'opération « puissance ».

Correction

Données
Un réel x et un entier positif n.

Résultat
Affichage de x, x^2, …, x^n.

Idée
La valeur de x^k est calculée comme $x \times x^{k-1}$, avec $x^0 = 1$.

Lexique des variables

x	(réel)	paramètre	DONNÉE
n	(entier)	paramètre	DONNÉE
i	(entier)	compteur d'itérations	INTERMÉDIAIRE
p	(réel)	valeurs successives de la suite des puissances x^i	INTERMÉDIAIRE

Algorithme

$p \leftarrow 1$
pour i **de** 1 **à** n **faire**
  $p \leftarrow p \times x$
  écrire p
fpour

3.6 Suite alternée

Imprimer les termes de la suite alternée :

$$1, -2!, 3!, -4!, \ldots, (-1)^{n+1} \times n!$$

Correction

Données
Un entier positif n.

Résultat
Affichage des termes de la suite donnée.

Idée
Notons $u_k = (-1)^{k+1} \times k!$ le k^{e} terme de la suite. Il s'agit alors d'afficher les termes u_1, u_2, …, u_n de la suite. On remarque que les termes de la suite peuvent être calculés par la relation $u_k = -k \times u_{k-1}$, avec $u_0 = -1$.

Lexique des variables

n	(entier)	paramètre	DONNÉE
i	(entier)	compteur d'itérations	INTERMÉDIAIRE
u	(entier)	valeurs successives de la suite $(-1)^{i+1} \times i!$	INTERMÉDIAIRE

Algorithme

$u \leftarrow -1$
pour i **de** 1 **à** n **faire**
  $u \leftarrow -i \times u$
  écrire u
fpour

3.7 Sommes de fractions

1. Écrire un algorithme qui, pour n donné, calcule la somme :

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Correction

Données

Un entier n.

Résultat

Valeur de $\sum_{k=1}^{n} \frac{1}{k}$.

Idée

Énumérer les valeurs $i = 1 \dots n$, et accumuler la somme des $\frac{1}{i}$.

Lexique des variables

n	(entier)	le paramètre donné	DONNÉE
somme	(réel)	valeur de $\sum_{k=1}^{n} \frac{1}{k}$	RÉSULTAT
i	(entier)	compteur d'itérations	INTERMÉDIAIRE

Algorithme

$somme \leftarrow 0$
pour i **de** 1 **à** n **faire**
  $somme \leftarrow somme + 1{,}0/i$ `// attention à la division entière !`
fpour

---

2. Même question avec :

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^n}$$

Correction

Données

Un entier n.

Résultat

Valeur de $\sum_{k=0}^{n} \frac{1}{2^k}$.

Idée

Énumérer les valeurs $i = 0 \dots n$, et accumuler la somme des $\frac{1}{2^i}$. On remarque que $\frac{1}{2^0} = 1$, et $\frac{1}{2^i} = \frac{1}{2} \times \frac{1}{2^{i-1}}$.

Lexique des variables

n	(entier)	le paramètre donné	DONNÉE
somme	(réel)	valeur de $\sum_{k=0}^{n} \frac{1}{2^k}$	RÉSULTAT
i	(entier)	compteur d'itérations	INTERMÉDIAIRE
u	(réel)	valeurs successives de $\frac{1}{2^i}$	INTERMÉDIAIRE

Algorithme

$somme \leftarrow 0$
$u \leftarrow 1$
pour i **de** 0 **à** n **faire**
  $somme \leftarrow somme + u$
  $u \leftarrow u/2{,}0$
fpour

3.8 Hauteurs de pluie

Écrire un algorithme qui lit les hauteurs de pluie (en mm) tombée durant un an (de janvier à décembre), puis imprime la hauteur de pluie la plus forte ainsi que le numéro du mois où cela s'est produit, la hauteur de pluie la plus faible ainsi que le numéro du mois où cela s'est produit et la moyenne des hauteurs de pluie tombées par mois. On suppose que les hauteurs de pluie tombée sont différentes chaque mois et l'on ne vérifie pas ce fait.

Correction

Données
Aucune.

Résultat
Aucun.

Idée
Lire une première hauteur, et initialiser les différentes variables en conséquence (hauteur et mois min., hauteur et mois max., somme des hauteurs). Lire ensuite les hauteurs suivantes, et mettre les variables à jour au fur et à mesure. Enfin calculer la moyenne et afficher les résultats.

Lexique des constantes

NB_MOIS	(entier)	= 12	nombre de mois à traiter

Lexique des variables

mois	(entier)	numéro de mois	INTERMÉDIAIRE
hauteur	(entier)	hauteur de pluie pour le mois	INTERMÉDIAIRE
h_min	(entier)	hauteur minimale	INTERMÉDIAIRE
m_min	(entier)	mois avec la hauteur minimale	INTERMÉDIAIRE
h_max	(entier)	hauteur maximale	INTERMÉDIAIRE
m_max	(entier)	mois avec la hauteur maximale	INTERMÉDIAIRE
somme	(entier)	somme des hauteurs de pluie	INTERMÉDIAIRE
moyenne	(entier)	moyenne des hauteurs de pluie	INTERMÉDIAIRE

Algorithme

hauteur ← lire
h_min ← *hauteur*
m_min ← 1
h_max ← *hauteur*
m_max ← 1
somme ← *hauteur*
pour *mois* **de** 2 **à** *NB_MOIS* **faire**
  hauteur ← lire
  si *hauteur* < *h_min* **alors**
    h_min ← *hauteur*
    m_min ← *mois*
  fsi
  si *hauteur* > *h_max* **alors**
    h_max ← *hauteur*
    m_max ← *mois*
  fsi
  somme ← *somme* + *hauteur*
fpour
moyenne ← *somme*/*NB_MOIS*
écrire "`Hauteur min.: `", *h_min*, "` pour le mois `", *m_min*
écrire "`Hauteur max.: `", *h_max*, "` pour le mois `", *m_max*
écrire "`Hauteur moy.: `", *moyenne*

3.9 Puissance de 2

On donne un réel x supposé positif, trouver le plus petit entier n tel que 2^n soit plus grand que x.

Correction

Données
Un nombre positif x.

Résultat
Plus petit entier n tel que $2^n \geq x$.

Idée
L'idée générale est de calculer les puissance de 2, jusqu'à dépasser la valeur de x.
Il faut cependant distinguer le cas où $x \geq 1$ et le cas où $x < 1$.
Dans le premier cas, on calcule les puissances 2^1, 2^2, 2^3, ... jusqu'à ce que $2^n \geq x$.
Dans le second cas, on calcule les puissances 2^{-1}, 2^{-2}, 2^{-3}, ... jusqu'à ce que $2^n < x$.

Lexique des variables

x	(réel)	le nombre positif donné	DONNÉE
n	(entier)	plus petit entier tel que $2^n \geq x$	RÉSULTAT
p	(réel)	valeur de 2^n	INTERMÉDIAIRE

Algorithme

$n \leftarrow 0$
$p \leftarrow 1$
si $x \geq 1$ **alors**
  tant que $p < x$ **faire**
    $n \leftarrow n + 1$
    $p \leftarrow 2 \times p$
  ftant
sinon
  tant que $p \geq x$ **faire**
    $n \leftarrow n - 1$
    $p \leftarrow p/2$
  ftant
  $n \leftarrow n + 1$
fsi

Remarque : la solution peut aussi être calculée directement par la formule $n = \lceil log_2 x \rceil$.

3.10 Chute libre

On veut imprimer les effets de la gravité sur un objet qui tombe depuis une tour de hauteur donnée en mètres. Pour cela, on imprime la hauteur à laquelle l'objet se trouve toutes les x (réel) secondes pendant sa chute. On sait que la distance d parcourue par un objet en fonction du temps de parcours t est donnée par la formule :

$$d = \frac{1}{2} g t^2$$

où g est la constante de gravitation (9,806 65). On écrira le message « boum » quand l'objet heurte le sol.

Correction

Données
Hauteur de la tour h, intervalle de temps pour la simulation x.

Résultat
Aucun (affichage).

Idée
Simuler la chute de l'objet. Pour cela, calculer la hauteur de l'objet en fonction du temps écoulé depuis le début. Si la hauteur est positive, recommencer en ajoutant x au temps écoulé. Finir par l'affichage du message « boum ».

Lexique des constantes

G	(réel)	= 9,806 65	constante de gravitation

Lexique des variables

h	(réel)	hauteur de la tour, en mètres	DONNÉE
x	(réel)	intervalle de temps pour la simulation, en secondes	DONNÉE
t	(réel)	temps écoulé depuis le début	INTERMÉDIAIRE
haut	(réel)	hauteur de l'objet	INTERMÉDIAIRE

Algorithme

$t \leftarrow 0$
$haut \leftarrow h$
tant que $haut \geq 0$ **faire**
  écrire "`t = `", t, "`; hauteur = `", $haut$
  $t \leftarrow t + x$
  $haut \leftarrow h - G \times t \times t/2{,}0$
ftant
écrire "`BOUM !`"

---

4 Fonctions

4.1 Calcul d'âge

Écrire une fonction qui calcule le nombre d'années (entières) qui s'écoulent entre deux dates (jour, mois, année).

L'utiliser dans un algorithme qui doit imprimer l'âge d'une personne étant donné sa date de naissance et indique sous forme de message si la personne est mineure, adulte ou peut bénéficier de la carte Vermeil (plus de 60 ans) !

Correction

La fonction...

fonction diffDates(**in** jA : entier, **in** mA : entier, **in** aA : entier, **in** jB : entier, **in** mB : entier, **in** aB : entier) : **ret** entier

Retourne le nombre d'années entières écoulées entre les dates $jA/mA/aA$ et $jB/mB/aB$. On suppose que la première date est antérieure à la seconde.

Idée

Calculer la différence des années, et en retrancher 1 si la dernière année n'est pas complète.

Lexique local des variables

diff (entier) nombre d'année entières entre les deux dates

Algorithme de diffDates

$diff \leftarrow aB - aA$
si $mB < mA \vee (mB = mA \wedge jB < jA)$ **alors**
  `// la dernière année n'est pas complète`
  $diff \leftarrow diff - 1$
fsi
retourner *diff*

... et l'algorithme...

Données

La date de naissance d'une personne, et la date du jour, sous la forme jour/mois/année.

Résultat

Aucun (affichage).

Idée

Utiliser la fonction définie précédemment pour calculer l'âge de la personne.

Lexique des fonctions

fonction diffDates(**in** jA : entier, **in** mA : entier, **in** aA : entier, **in** jB : entier, **in** mB : entier, **in** aB : entier) : **ret** entier

Retourne le nombre d'années entières écoulées en les dates $jA/mA/aA$ et $jB/mB/aB$. On suppose que la première date est antérieure à la seconde.

Lexique des variables

jNaiss	(entier)	jour de naissance de la personne	DONNÉE
mNaiss	(entier)	mois de naissance de la personne	DONNÉE
aNaiss	(entier)	année de naissance de la personne	DONNÉE
jCour	(entier)	jour courant	DONNÉE
mCour	(entier)	mois courant	DONNÉE
aCour	(entier)	année courant	DONNÉE
age	(entier)	âge de la personne	INTERMÉDIAIRE

Algorithme

$age \leftarrow$ diffDates($jNaiss, mNaiss, aNaiss, jCour, mCour, aCour$)
écrire "`Cette personne est âgée de `", *age*, "` ans.`"
si $age < 18$ **alors**
  écrire "`Elle est mineure.`"
sinon
  écrire "`Elle est adulte.`"
fsi
si $age \geq 60$ **alors**
  écrire "`Et elle peut bénéficier de la carte Vermeil.`"
fsi

4.2 Multiplication récursive

Écrire une fonction récursive qui fait le produit de deux nombres entiers positifs en utilisant des additions. L'idée est que :

$$a \times b = \begin{cases} 0 & \text{si } a = 0 \\ (a-1) \times b + b & \text{si } a > 0 \end{cases}$$

Exécuter la fonction pour $a = 4$ et $b = 5$ (pour cela, présenter les appels récursifs successifs sous forme d'un *arbre*), puis pour $a = 5$ et $b = 4$. Pour quel appel l'arbre est-il le plus haut ? Noter sur quel argument porte la récursion. Conclure.

Correction

Algorithme

---

fonction produit(**in** a : entier, **in** b : entier) : **ret** entier
Retourne le produit $a \times b$.

Algorithme de produit

si $a = 0$ **alors**
  retourner 0
sinon
  retourner produit$(a-1, b) + b$
fsi

L'arbre des appels pour le calcul de 4×5 est :

```
-> produit(4, 5) ........: 20
  -> produit(3, 5) ......: 15
    -> produit(2, 5) ....: 10
      -> produit(1, 5) ..:  5
        -> produit(0, 5) :  0
```

L'arbre des appels pour le calcul de 5×4 est :

```
-> produit(5, 4) ..........: 20
  -> produit(4, 4) ........: 16
    -> produit(3, 4) ......: 12
      -> produit(2, 4) ....:  8
        -> produit(1, 4) ..:  4
          -> produit(0, 4) :  0
```

On remarque que la hauteur de l'arbre est égale $a+1$ (avec a la valeur du premier paramètre). C'est normal, car c'est cette valeur qui est utilisée pour contrôler la récursion. Si on veut minimiser la hauteur de l'arbre, et donc le nombre d'appels de fonction, il faut donc passer la plus petite valeur en premier.

4.3 Calculs de puissances

1. Écrire une fonction récursive qui calcule x^n sachant que :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ x^{n-1} \times x & \text{si } n > 0 \end{cases}$$

Correction

---

fonction puissance1(**in** x : réel, **in** n : entier) : **ret** réel
Retourne la valeur de x^n.

Lexique local des variables
res (réel) résultat de la fonction

Algorithme de puissance1

si $n = 0$ **alors**
  $res \leftarrow 1$
sinon
  $res \leftarrow x \times \text{puissance1}(x, n-1)$
fsi
retourner *res*

---

2. Écrire une fonction récursive qui calcule x^n sachant que :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ \left(x^{n/2}\right)^2 & \text{pour } n > 0 \text{ pair} \\ \left(x^{(n-1)/2}\right)^2 \times x & \text{pour } n > 0 \text{ impair} \end{cases}$$

Correction

---

fonction puissance2(**in** x : réel, **in** n : entier) : **ret** réel
  Retourne la valeur de x^n.

Lexique local des variables

y	(réel)	pour la valeur de $x^{n/2}$
res	(réel)	résultat de la fonction

Algorithme de puissance2

si $n = 0$ **alors**
  $res \leftarrow 1$
sinon si $n \bmod 2 = 0$ **alors**
  $y \leftarrow \text{puissance2}(x, n/2)$
  $res \leftarrow y \times y$
sinon
  $y \leftarrow \text{puissance2}(x, (n-1)/2)$
  $res \leftarrow y \times y \times x$
fsi
retourner *res*

---

3. Comparer les exécutions (avec *arbre* des appels) des deux fonctions ci-dessus pour $n = 11$ et $x = 2$. Quel est l'arbre le plus court ? Qu'en conclure ?

Correction

Arbres des appels pour le calcul de 2^{11}, première version :

```
-> puissance1 (2, 11) ....................: 2048
  -> puissance1 (2, 10) ...................: 1024
    -> puissance1 (2, 9) ..................:  512
      -> puissance1 (2, 8) ................:  256
        -> puissance1 (2, 7) ..............:  128
          -> puissance1 (2, 6) ............:   64
            -> puissance1 (2, 5) ..........:   32
              -> puissance1 (2, 4) ........:   16
                -> puissance1 (2, 3) ......:    8
                  -> puissance1 (2, 2) ....:    4
                    -> puissance1 (2, 1) ..:    2
                      -> puissance1 (2, 0) :    1
```

Arbres des appels pour le calcul de 2^{11}, deuxième version :

```
-> puissance2 (2, 11) .......: 2048
  -> puissance2 (2, 5) ......:   32
    -> puissance2 (2, 2) ....:    4
      -> puissance2 (2, 1) ..:    2
        -> puissance2 (2, 0) :    1
```

On peut montrer que la hauteur de l'arbre est de $n + 1$ pour la première version, et de $1 + \lceil \log_2(n+1) \rceil$ pour la deuxième. L'arbre est donc plus court pour la deuxième version, dès que $n > 2$. On peut en déduire que la deuxième version s'exécutera plus rapidement.

4.4 Tours de Hanoï

Autrefois, à Hanoï, l'empereur a conçu un puzzle de n disques et trois piquets : A (départ), B (échangeur) et C (arrivée). Les disques étaient tous de tailles différentes et avaient un trou au milieu afin que l'on puisse passer les disques dans les piquets. À cause de leur poids important, on ne pouvait pas mettre un disque plus lourd au dessus de disques qui étaient plus petits. Le but du jeu est de transférer tous les disques du piquet A au piquet C, en ne déplaçant qu'un seul disque à la fois. Concevoir une solution récursive qui imprime un message décrivant la solution.

Correction

Pour lancer la résolution, il faut faire l'appel de fonction suivant : hanoï(n, '`A`', '`C`', '`B`').

fonction hanoï(**in** n : entier, **in** *départ* : caractère, **in** *arrivée* : caractère, **in** *échangeur* : caractère) : **vide**
Imprime une description des étapes de résolution du problème des tours de Hanoï, où il faut déplacer n disques du piquet *départ* vers le piquet *arrivée*, en utilisant éventuellement un troisième piquet : *échangeur*.

Idée
Le problème des tours de Hanoï à n disques se résout très simplement de manière récursive. Il suffit de remarquer que pour déplacer n disques du piquet A au piquet C, il faut :
- déplacer $(n-1)$ disques du piquet A vers le piquet B ;
- déplacer le dernier disque du piquet A vers le piquet C ;
- déplacer $(n-1)$ disques du piquet B vers le piquet C.

Algorithme de hanoï

si $n = 1$ **alors**
  écrire "`déplacement de `", *départ*, "` vers `", *arrivée*
sinon
  hanoï($n-1$, *départ*, *échangeur*, *arrivée*)
  hanoï(1, *départ*, *arrivée*, *échangeur*)
  hanoï($n-1$, *échangeur*, *arrivée*, *départ*)
fsi

4.5 Flocon de Von Koch

Une famille de fonctions mathématiques donne des représentations graphiques intéressantes, il s'agit des fractales.

Le principe de dessin d'une famille de fractales est de remplacer chaque segment de la figure initiale par un motif et de réitérer le processus sur les segments de la figure obtenue. On arrête le processus en fixant le nombre maximal de subdivisions à effectuer (niveau de récursion).

Dans l'exemple de la fractale de Von Koch (cf. figure 4), on considère l'ajout des points C, D et E dans la figure initiale $[AB]$, en considérant que les longueurs des nouveaux segments $[AC]$, $[CD]$, $[DE]$ et $[EB]$ sont toutes égales à $1/3$ de la longueur du segment initial $[AB]$.

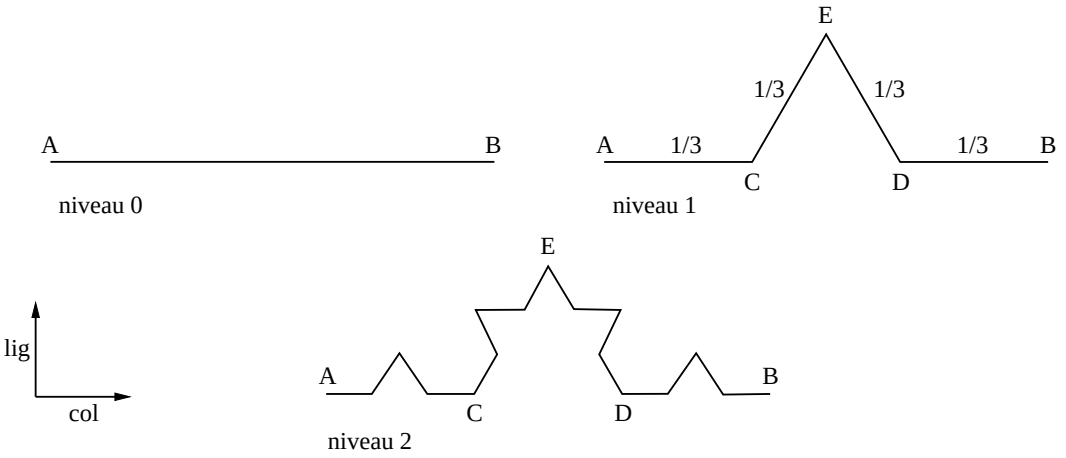

FIGURE 4 – Exemple de la fractale de Von Koch

1. Écrire un algorithme permettant de calculer les coordonnées C, D et E, à partir des coordonnées des points A et B.

Correction

Données

Coordonnées (Ax, Ay) et (Bx, By) des points A et B.

Résultat

Coordonnées (Cx, Cy), (Dx, Dy) et (Ex, Ey) des points C, D et E.

Idée

Les point C et D sont placés respectivement à 1/3 et 2/3 du segment $[AB]$. En notant $\overrightarrow{AB}$ le vecteur AB, on a :

$$C = A + \overrightarrow{AC} = A + \frac{1}{3}\overrightarrow{AB}$$
$$D = A + \overrightarrow{AD} = A + \frac{2}{3}\overrightarrow{AB}$$

et donc

$$Cx = Ax + \frac{1}{3}(Bx - Ax) = \frac{1}{3}(2Ax + Bx)$$
$$Cy = Ay + \frac{1}{3}(By - Ay) = \frac{1}{3}(2Ay + By)$$
$$Dx = Ax + \frac{2}{3}(Bx - Ax) = \frac{1}{3}(Ax + 2Bx)$$
$$Dy = Ay + \frac{2}{3}(By - Ay) = \frac{1}{3}(Ay + 2By)$$

Le vecteur $\overrightarrow{CE}$ est l'image du vecteur $\overrightarrow{CD}$ par la rotation d'angle $\Pi/3$, et $E = C + \overrightarrow{CE}$. Soit (x, y) les coordonnées du vecteur $\overrightarrow{CD}$, et (x', y') les coordonnées du vecteur $\overrightarrow{CE}$. Alors :

$$x = Dx - Cx$$
$$y = Dy - Cy$$

et

$$x' = x\cos(\frac{\Pi}{3}) - y\sin(\frac{\Pi}{3}) = \frac{1}{2}x - \frac{\sqrt{3}}{2}y = \frac{1}{2}(x - \sqrt{3}y)$$
$$y' = x\sin(\frac{\Pi}{3}) + y\cos(\frac{\Pi}{3}) = \frac{\sqrt{3}}{2}x + \frac{1}{2}y = \frac{1}{2}(\sqrt{3}x + y)$$

Or

$$E = C + \overrightarrow{CE}$$

donc

$$Ex = Cx + x' = Cx + \frac{1}{2}(x - \sqrt{3}y)$$
$$Ey = Cy + y' = Cy + \frac{1}{2}(\sqrt{3}x + y)$$

Lexique des variables

Ax	(réel)	abscisse du point A	DONNÉE
Ay	(réel)	ordonnée du point A	DONNÉE
Bx	(réel)	abscisse du point B	DONNÉE
By	(réel)	ordonnée du point B	DONNÉE
Cx	(réel)	abscisse du point C	RÉSULTAT
Cy	(réel)	ordonnée du point C	RÉSULTAT
Dx	(réel)	abscisse du point D	RÉSULTAT
Dy	(réel)	ordonnée du point D	RÉSULTAT
Ex	(réel)	abscisse du point E	RÉSULTAT
Ey	(réel)	ordonnée du point E	RÉSULTAT
x	(réel)	abscisse du vecteur $\overrightarrow{CD}$	INTERMÉDIAIRE
y	(réel)	ordonnée du vecteur $\overrightarrow{CD}$	INTERMÉDIAIRE

Algorithme

$Cx \leftarrow (2 \times Ax + Bx)/3$
$Cy \leftarrow (2 \times Ay + By)/3$
$Dx \leftarrow (Ax + 2 \times Bx)/3$
$Dy \leftarrow (Ay + 2 \times By)/3$
$x \leftarrow Dx - Cx$
$y \leftarrow Dy - Cy$
$Ex \leftarrow Cx + (x - \sqrt{3} \times y)/2$
$Ey \leftarrow Cy + (\sqrt{3} \times x + y)/2$

---

2. Écrire une fonction récursive de dessin de la fractale de Von Koch pour deux points de départ et un niveau de récursion donnés.
   Pour effectuer le dessin, on considère que l'on dispose d'une fonction qui trace une ligne entre deux points donnés :

---

fonction ligne(**in** x_1 : entier, **in** y_1 : entier, **in** x_2 : entier, **in** y_2 : entier) : **vide**
Trace une ligne entre les points (x_1, y_1) et (x_2, y_2).

Correction

---

fonction vonKoch(**in** *recur* : entier, **in** Ax : réel, **in** Ay : réel, **in** Bx : réel, **in** By : réel) : **vide**
Trace la courbe de Von Koch entre les points A et B, avec le niveau de récursion donné.

Idée

Calculer les coordonnées des points C, D et E, et recommencer récursivement sur chacun des segments tant que le niveau de récursion souhaité est supérieur à 0.

Lexique local des fonctions

fonction ligne(**in** x_1 : entier, **in** y_1 : entier, **in** x_2 : entier, **in** y_2 : entier) : **vide**
Trace une ligne entre les points (x_1, y_1) et (x_2, y_2).

Lexique local des variables

Cx	(réel)	abscisse du point C
Cy	(réel)	ordonnée du point C
Dx	(réel)	abscisse du point D
Dy	(réel)	ordonnée du point D
Ex	(réel)	abscisse du point E
Ey	(réel)	ordonnée du point E

Algorithme de vonKoch

si *recur* > 0 **alors**
    `// ...`
    `// ici,`
    `// calculer les coordonnées des points` C`,` D `et` E `en utilisant l'algorithme précédent`
    `// ...`
    vonKoch($recur - 1$, Ax, Ay, Cx, Cy)
    vonKoch($recur - 1$, Cx, Cy, Ex, Ey)
    vonKoch($recur - 1$, Ex, Ey, Dx, Dy)
    vonKoch($recur - 1$, Dx, Dy, Bx, By)
sinon
    ligne(Ax, Ay, Bx, By)
fsi

---

5 Tableaux

5.1 Notes

Écrire un algorithme qui, étant donné les notes des 30 élèves d'une classe, calcule le nombre de ceux qui ont obtenu une note supérieure à la moyenne et le nombre de ceux qui ont obtenu une note inférieure à la moyenne.

Question subsidiaire Résoudre le même problème, mais en utilisant la moyenne de la classe.

Correction

Données
Les notes des 30 élèves d'une classe.

Résultat
Le nombre d'élèves qui ont une note ≥ 10, et le nombre de ceux qui ont une note < 10

Idée
Les notes sont stockées dans un tableau de réels.
Parcourir ensuite les notes pour compter le nombre d'élèves ayant la moyenne. Le nombre de ceux qui n'ont pas la moyenne est ensuite obtenu par soustraction du nombre total d'élèves.

Lexique des constantes

NB_NOTES	(entier)	$= 30$	le nombre de notes

Lexique des variables

notes	(tableau [*NB_NOTES*] de réels)	les notes	DONNÉE
nb_sup	(entier)	nombre d'élèves avec une note ≥ 10	RÉSULTAT
nb_inf	(entier)	nombre d'élèves avec une note < 10	RÉSULTAT
i	(entier)	indice de parcours	INTERMÉDIAIRE

Algorithme

```
nb_sup ← 0
pour i de 0 à NB_NOTES − 1 faire
    si notes[i] ≥ 10 alors
        nb_sup ← nb_sup + 1
    fsi
fpour
nb_inf ← NB_NOTES − nb_sup
```

5.2 Températures

Écrire un algorithme qui, étant donné un relevé de températures sur un mois de 30 jours, imprime le ou les jours du mois où il y a eu le plus grand écart par rapport à la moyenne de ces températures.

Correction

Données
Un relevé de températures pour les 30 jours d'un mois.

Résultat
Aucun (affichage des jours où il y a eu le plus grand écart par rapport à la moyenne des températures).

Idée
Le relevé de températures va être modélisé à l'aide d'un tableau de 30 réels.
Le calcul se fera ensuite à l'aide de trois parcours successifs du tableau :

(i) un premier parcours pour calculer la somme, puis la moyenne des températures ;

(ii) un deuxième parcours pour calculer les écarts par rapport à la moyenne, et le plus grand d'entre eux (ces écarts seront stockés dans un tableau) ;

(iii) enfin un troisième parcours pour trouver les jours atteignant l'écart maximal, et les afficher.

Lexique des constantes

NB_JOURS	(entier)	$= 30$	le nombre total de jours

Lexique des variables

températures	(tableau [*NB_JOURS*] de réels)	le relevé de températures	DONNÉE
somme	(réel)	la somme des températures	INTERMÉDIAIRE
moyenne	(réel)	la moyenne des températures	INTERMÉDIAIRE
écarts	(tableau [*NB_JOURS*] de réels)	les écarts de température par rapport à la moyenne	INTERMÉDIAIRE
écart_max	(réel)	la valeur du plus grand écart par rapport à la moyenne	INTERMÉDIAIRE
i	(entier)	indice de parcours	INTERMÉDIAIRE

Algorithme

```
// calcul de la moyenne des températures
somme ← 0
pour i de 0 à NB_JOURS − 1 faire
    somme ← somme + températures[i]
fpour
moyenne ← somme/NB_JOURS
// calcul des écarts à la moyenne, et de leur maximum
écart_max ← 0                              // un écart est toujours positif
pour i de 0 à NB_JOURS − 1 faire
    écarts[i] ← |températures[i] − moyenne|          // prendre la valeur absolue
    si écarts[i] > écart_max alors
        écart_max ← écarts[i]
    fsi
fpour
// affichage des jours atteignant l'écart maximal
pour i de 0 à NB_JOURS − 1 faire
    si écarts[i] = écart_max alors
        écrire "Écart maximal atteint au jour", i + 1
    fsi
fpour
```

5.3 Vote

Une organisation vote pour renouveler son directeur. Il y a 5 candidats numérotés de 1 à 5. Les noms des candidats sont, dans l'ordre de leur numéros : Ariane, Jean, Lucie, Marc, et Michel. Chaque membre de l'organisation envoie un bulletin de vote qui porte le numéro qui correspond au candidat de son choix. Écrire un algorithme pouvant servir à faire le décompte des voix, et afficher le ou les noms des candidats ayant le plus de voix.

Correction

Données

Aucune.

Résultat

Aucun (lecture des votes, puis affichage des vainqueurs).

Idée

Chaque candidat se voit attribuer un numéro entre 1 et le nombre de candidats. De cette manière, les noms des candidats, et leurs nombres de voix peuvent être stockées dans des tableaux.

Le calcul s'effectue en trois étapes :

(i) lecture des votes, et décomptes des voix ;

(ii) calcul du nombre de voix maximal ;

(iii) affichage des vainqueurs.

Lexique des constantes

NB_CANDIDATS	(entier)	= 5	le nombre de candidats
CANDIDATS	(tableau [*NB_CANDIDATS*] de chaînes)	= {"Ariane", "Jean", "Lucie", "Marc", "Michel"}	les noms des candidats

Lexique des variables

voix	(tableau [*NB_CANDIDATS*] d'entiers)	le décompte des voix	INTERMÉDIAIRE
vote	(entier)	un vote	INTERMÉDIAIRE
voix_max	(entier)	le maximum de voix atteint par un candidat	INTERMÉDIAIRE
i	(entier)	indice de parcours	INTERMÉDIAIRE

Algorithme

```
// initialisation du décompte des voix
pour i de 0 à NB_CANDIDATS − 1 faire
    voix[i] ← 0
fpour
// saisie des votes
vote ← Lire
tant que vote ≠ EOF faire
    si vote ≥ 1 ∧ vote ≤ NB_CANDIDATS alors
        voix[vote − 1] ← 1 + voix[vote − 1]
    sinon
        écrire "Vote invalide :", vote
    fsi
    vote ← Lire
ftant
// calcul du maximum
voix_max ← 0
pour i de 0 à NB_CANDIDATS − 1 faire
    si voix[i] > voix_max alors
        voix_max ← voix[i]
    fsi
fpour
// affichage du résultat
pour i de 0 à NB_CANDIDATS − 1 faire
    si voix[i] = voix_max alors
        écrire "Le gagnant est", CANDIDATS[i], "avec", voix[i], "voix."
    fsi
fpour
```

5.4 Mélange

Écrire une fonction permettant de mélanger les valeurs d'un tableau d'entiers. On suppose que l'on dispose d'une fonction de tirage aléatoire :

---

fonction nombreAléatoire(**in** *a* : entier, **in** *b* : entier) : **ret** entier
Retourne un nombre entier tiré aléatoirement dans l'intervalle [*a* ; *b*[.

Correction

---

fonction mélange(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Mélange aléatoirement les valeurs du tableau *tab* de taille *taille*.

Idée

Le tableau est parcouru de gauche à droite, et la valeur de chaque case est échangée avec celle d'une autre case se trouvant après elle dans le tableau. Cette autre case est choisie de manière aléatoire. On utilise pour cela une fonction qui retourne un nombre aléatoire, et qui n'est pas détaillée ici.

Lexique local des fonctions

fonction nombreAléatoire(**in** *a* : entier, **in** *b* : entier) : **ret** entier
Retourne un nombre entier tiré aléatoirement dans l'intervalle [*a* ; *b*[.

Lexique local des variables

i	(entier)	compteur pour le parcours du tableau
j	(entier)	indice de la valeur à échanger
tmp	(réel)	variable temporaire pour l'échange

Algorithme de mélange

pour i **de** 0 **à** $taille - 2$ **faire**
  $j \leftarrow$ nombreAléatoire(i, $taille$)
  $tmp \leftarrow tab[i]$
  $tab[i] \leftarrow tab[j]$
  $tab[j] \leftarrow tmp$
fpour

5.5 Nombres en toutes lettres

Écrire un algorithme qui lit des nombres entiers supposés positifs et inférieurs à 99999 et les imprime en toutes lettres. Par exemple, avec 367, on imprime *trois cent soixante-sept*. On utilisera des tableaux de constantes chaînes de caractères appropriés.

Correction

Données
Aucune donnée.

Résultat
Aucun résultat (affichage).

Idée
L'algorithme est très simple : dès qu'un nombre est lu, on vérifie que ce nombre est valide et on utilise la fonction imprimeNombre999999 pour l'affichage.

Lexique des fonctions
fonction imprimeNombre999999(**in** *nombre* : entier) : **vide**
  Imprime en toutes lettres le nombre *nombre* compris entre 0 et 999 999.

Lexique des variables
nombre (entier) le nombre à afficher INTERMÉDIAIRE

Algorithme

nombre ← lire
tant que *nombre* ≠ EOF **faire**
  si *nombre* ≥ 0 ∧ *nombre* ≤ 999 999 **alors**
    imprimeNombre999999(*nombre*)
  sinon
    écrire "`Le nombre doit être compris entre 0 et 999 999.`"
  fsi
  nombre ← lire
ftant

Il faut cependant définir la fonction imprimeNombre999999. Avant cela, on définit des constantes globales (des tableaux de chaînes de caractères) et les fonctions auxiliaires imprimeNombre99 et imprimeNombre999.

Quelques constantes globales

Idée
L'idée, pour ces constantes, est de définir des tableaux avec les valeurs correspondant aux indices (0 pour "`zéro`" dans *CH_UNITÉS* ou 2 pour "`vingt`" dans *CH_DIZAINES* par exemple). Plus tard, cela permettra de retrouver facilement les constantes.

Lexique des constantes

CH_UNITÉS	(tableau [20] de chaînes)	= ["`zéro`", "`un`", "`deux`", "`trois`", "`quatre`", "`cinq`", "`six`", "`sept`", "`huit`", "`neuf`", "`dix`", "`onze`", "`douze`", "`treize`", "`quatorze`", "`quinze`", "`seize`", "`dix-sept`", "`dix-huit`", "`dix-neuf`"]
CH_DIZAINES	(tableau [9] de chaînes)	= ["", "", "`vingt`", "`trente`", "`quarante`", "`cinquante`", "`soixante`", "", "`quatre-vingt`"]

La fonction imprimeNombre99

fonction imprimeNombre99(**in** *nombre* : entier, **in** *esse* : booléen) : **vide**
  Imprime en toutes lettres le nombre *nombre* compris entre 0 et 99. Si le paramètre *esse* est faux, n'imprime pas de « s » terminal pour « quatre-vingts. »

Idée

On commence par imprimer les dizaines, en faisant attention aux cas soixante-... et quatre-vingt-... On imprime ensuite les unités restantes.

Lexique local des variables

dizaines	(entier)	nombre de dizaines
unités	(entier)	nombre d'unités restantes

Algorithme de imprimeNombre99

si *nombre* ≥ 20 **alors**
  si *nombre* ≥ 60 **alors**
    `// soixante ou quatre-vingt`
    dizaines $\leftarrow 2 \times$ (*nombre*/20)
    unités $\leftarrow$ *nombre* mod 20
  sinon
    dizaines $\leftarrow$ *nombre*/10
    unités $\leftarrow$ *nombre* mod 10
  fsi
  écrire *CH_DIZAINES*[*dizaines*]
  si *esse* $\wedge$ *dizaines* $= 8 \wedge$ *unités* $= 0$ **alors**
    écrire "`s`"
  fsi
  si *unités* mod $10 = 1 \wedge$ *dizaines* $\neq 8$ **alors**
    écrire " `et` "
  sinon si *unités* $\neq 0$ **alors**
    écrire "`-`"
  fsi
sinon
  unités $\leftarrow$ *nombre*
fsi
si *nombre* $= 0 \vee$ *unités* $\neq 0$ **alors**
  écrire *CH_UNITÉS*[*unités*]
fsi

La fonction imprimeNombre999

---

fonction imprimeNombre999(**in** *nombre* : entier, **in** *esse* : booléen) : **vide**

Imprime en toutes lettres le nombre *nombre* compris entre 0 et 999. Si le paramètre *esse* est faux, n'imprime pas de « s » terminal pour « quatre-vingts » et « cents. »

Idée

On imprime le nombre de centaines, puis on utilise la fonction imprimeNombre99 pour imprimer le reste.

Lexique local des fonctions

fonction imprimeNombre99(**in** *nombre* : entier, **in** *esse* : booléen) : **vide**

Lexique local des variables

centaines	(entier)	nombre de centaines
nombre99	(entier)	dizaines et unités du nombre

Algorithme de imprimeNombre999

centaines $\leftarrow$ *nombre*/100
nombre99 $\leftarrow$ *nombre* mod 100
si *centaines* ≥ 1 **alors**
  si *centaines* ≥ 2 **alors**
    écrire *CH_UNITÉS*[*centaines*]
    écrire "␣"
  fsi
  écrire "`cent`"
  si *esse* $\wedge$ *centaines* $\geq 2 \wedge$ *nombre99* $= 0$ **alors**
    écrire "`s`"
  fsi
  si *nombre99* $\neq 0$ **alors**
    écrire "␣"
  fsi
fsi
si *nombre* $= 0 \vee$ *nombre99* $\neq 0$ **alors**
  imprimeNombre99(*nombre99*, *esse*)
fsi

La fonction imprimeNombre999999

fonction imprimeNombre999999(**in** *nombre* : entier) : **vide**
Imprime en toutes lettres le nombre *nombre* compris entre 0 et 999 999.

Idée
On utilise deux fois la fonction imprimeNombre999 : une première fois pour imprimer le nombre de milliers et une deuxième fois pour imprimer le reste.

Lexique local des fonctions
fonction imprimeNombre999(**in** *nombre* : entier, **in** *esse* : booléen) : **vide**

Lexique local des variables

milliers	(entier)	nombre de milliers
nombre999	(entier)	nombre modulo 1000

Algorithme de imprimeNombre999999

milliers ← *nombre*/1000
nombre999 ← *nombre* mod 1000
si *milliers* ≥ 1 **alors**
  si *milliers* ≥ 2 **alors**
    imprimeNombre999(*milliers*, faux)
    écrire "␣"
  fsi
  écrire "mille"
  si *nombre999* ≠ 0 **alors**
    écrire "␣"
  fsi
fsi
si *nombre* = 0 ∨ *nombre999* ≠ 0 **alors**
  imprimeNombre999(*nombre999*, vrai)
fsi

5.6 Recherche dichotomique

Exécuter la trace de l'algorithme de recherche dichotomique appliqué aux recherches successives des entiers 6, 1, 12, 30, 0, 21, 29 dans un tableau d'entiers contenant les valeurs 1, 5, 9, 12, 15, 21, 29. Vous vous appuierez sur les deux versions (récursive et itérative) données en cours.

Correction

```
Tableau: [ 1, 5, 9, 12, 15, 21, 29 ]

Recherche de 6
==============
-> rechDichoRec(tab, 0, 6, 6): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 0, 2, 6): milieu = 1, tab[milieu] = 5
    -> rechDichoRec(tab, 2, 2, 6): milieu = 2, tab[milieu] = 9
      -> rechDichoRec(tab, 2, 1, 6): deb > fin
      <- -1
    <- -1
  <- -1
<- -1

-> rechDichoIter(tab, 0, 6, 6):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (0) <= max (2), milieu = 1, tab[milieu] = 5
  min (2) <= max (2), milieu = 2, tab[milieu] = 9
  min (2) > max (1)
<- -1

Recherche de 1
==============
-> rechDichoRec(tab, 0, 6, 1): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 0, 2, 1): milieu = 1, tab[milieu] = 5
    -> rechDichoRec(tab, 0, 0, 1): milieu = 0, tab[milieu] = 1
```

```
    <- 0
  <- 0
<- 0

-> rechDichoIter(tab, 0, 6, 1):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (0) <= max (2), milieu = 1, tab[milieu] = 5
  min (0) <= max (0), milieu = 0, tab[milieu] = 1
<- 0

Recherche de 12
===============
-> rechDichoRec(tab, 0, 6, 12): milieu = 3, tab[milieu] = 12
<- 3

-> rechDichoIter(tab, 0, 6, 12):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
<- 3

Recherche de 30
===============
-> rechDichoRec(tab, 0, 6, 30): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 4, 6, 30): milieu = 5, tab[milieu] = 21
    -> rechDichoRec(tab, 6, 6, 30): milieu = 6, tab[milieu] = 29
      -> rechDichoRec(tab, 7, 6, 30): deb > fin
      <- -1
    <- -1
  <- -1
<- -1

-> rechDichoIter(tab, 0, 6, 30):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (4) <= max (6), milieu = 5, tab[milieu] = 21
  min (6) <= max (6), milieu = 6, tab[milieu] = 29
  min (7) > max (6)
<- -1

Recherche de 0
==============
-> rechDichoRec(tab, 0, 6, 0): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 0, 2, 0): milieu = 1, tab[milieu] = 5
    -> rechDichoRec(tab, 0, 0, 0): milieu = 0, tab[milieu] = 1
      -> rechDichoRec(tab, 0, -1, 0): deb > fin
      <- -1
    <- -1
  <- -1
<- -1

-> rechDichoIter(tab, 0, 6, 0):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (0) <= max (2), milieu = 1, tab[milieu] = 5
  min (0) <= max (0), milieu = 0, tab[milieu] = 1
  min (0) > max (-1)
<- -1

Recherche de 21
===============
-> rechDichoRec(tab, 0, 6, 21): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 4, 6, 21): milieu = 5, tab[milieu] = 21
  <- 5
<- 5

-> rechDichoIter(tab, 0, 6, 21):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (4) <= max (6), milieu = 5, tab[milieu] = 21
<- 5
```

```
Recherche de 29
===============
-> rechDichoRec(tab, 0, 6, 29): milieu = 3, tab[milieu] = 12
  -> rechDichoRec(tab, 4, 6, 29): milieu = 5, tab[milieu] = 21
    -> rechDichoRec(tab, 6, 6, 29): milieu = 6, tab[milieu] = 29
    <- 6
  <- 6
<- 6

-> rechDichoIter(tab, 0, 6, 29):
  min (0) <= max (6), milieu = 3, tab[milieu] = 12
  min (4) <= max (6), milieu = 5, tab[milieu] = 21
  min (6) <= max (6), milieu = 6, tab[milieu] = 29
<- 6
```

5.7 Reversi

L'algorithme doit simuler un jeu appelé *Reversi*. Le jeu commence par afficher les chiffres de 1 à 9 dans le désordre. Le joueur donne un entier compris entre 1 et 9 qui indique le nombres de chiffres à inverser à partir de la gauche. La nouvelle configuration du jeu est affichée. Par exemple pour la configuration :

5 4 6 2 1 7 9 8 3

si le joueur donne l'entier 5, la nouvelle configuration sera :

1 2 6 4 5 7 9 8 3

Le jeu s'arrête quand on atteint une configuration où les chiffres sont placés par ordre croissant. L'objectif du joueur est de réaliser ce but avec le moins de tours possible. L'algorithme permettra d'imprimer en combien de tours le joueurs a réalisé le tri et lui permettra de recommencer le jeu sur la même configuration initiale autant de fois qu'il le désire. La configuration initiale est supposée fournie par une fonction dont on ne décrira pas l'algorithme :

---

fonction initTab(**out** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Remplit le tableau *tab* avec les nombre de 1 à *taille*, dans le désordre.

Correction

Données
Aucune donnée.

Résultat
Aucun résultat (programme interactif).

Idée
On utilisera des tableaux pour stocker les configurations.
On commence par générer la configuration initiale.
Pour une partie, on prend une copie de la configuration initiale, on l'affiche et on initialise le nombre de tours à 0.
Ensuite, les tours de jeu sont enchaînés tant que les valeurs ne sont pas dans l'ordre (une fonction sera définie pour vérifier ce fait).
Pour chaque tour, le joueur entre son choix (k), puis les k premières valeurs du tableau sont inversées à l'aide d'une fonction appropriée. Le nombre de tours est incrémenté et le nouvel état du tableau est affiché.
À la fin de la partie, le nombre de tours est affiché et il est proposé au joueur de recommencer une partie.

Lexique des constantes
TAILLE (entier) = 9 taille du tableau de jeu

Lexique des fonctions

fonction initTab(**out** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Remplit le tableau *tab* avec les nombre de 1 à *taille*, dans le désordre.

fonction afficheTab(**in** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Affiche le tableau *tab* de taille *taille* passé en paramètre.

fonction copieTab(**in** *src* : tableau d'entiers, **out** *dest* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Copie les valeurs du tableau *src* dans le tableau *dest*. Les deux tableaux sont de taille *taille*.

fonction vérifTab(**in** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** booléen
Retourne vrai si le tableau *tab* contient les valeurs de 1 à *taille* dans l'ordre. Retourne faux sinon.

fonction permuteTab(**in-out** *tab* : tableau d'entiers, **in** *n* : entier) : **ret** vide
Inverse les *n* premières valeurs du tableau *tab*.

Lexique des variables

configInitiale	(tableau [*TAILLE*] d'entiers)	la configuration initiale	INTERMÉDIAIRE
config	(tableau [*TAILLE*] d'entiers)	le tableau de jeu	INTERMÉDIAIRE
ntours	(entier)	compteur de tours de jeu	INTERMÉDIAIRE
jeu	(entier)	jeu du joueur lors d'un tour	INTERMÉDIAIRE
réponse	(caractère)	réponse (o/n) du joueur	INTERMÉDIAIRE

Algorithme

```
initTab(configInitiale, TAILLE)
répéter
    copieTab(configInitiale, config, TAILLE)
    afficheTab(config, TAILLE)
    ntours ← 0
    tant que ¬vérifTab(config, TAILLE) faire
        écrire "Jeu ? "
        jeu ← lire
        si jeu ≥ 1 ∧ jeu ≤ TAILLE alors
            permuteTab(config, jeu)
            afficheTab(config, TAILLE)
            ntours ← ntours + 1
        sinon
            écrire "Jeu invalide."
        fsi
    ftant
    écrire "Gagné en ", ntours, "tours."
    écrire "Recommencer (o/n) ? "
    réponse ← lire
tant que réponse = 'o'
```

---

fonction initTab(**out** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Remplit le tableau *tab* avec les nombre de 1 à *taille*, dans le désordre.

Idée

Initialiser le tableau avec les valeurs de 1 à *taille*, puis mélanger les valeurs (cf. fonction mélange() de l'exercice 5.4 p. 27). Si jamais les valeurs mélangées sont déjà rangée, recommencer le mélange.

Lexique local des fonctions

fonction mélangeTab(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Mélange aléatoirement les valeurs du tableau *tab* de taille *taille*.

fonction vérifTab(**in** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** booléen
Retourne vrai si le tableau *tab* contient les valeurs de 1 à *taille* dans l'ordre. Retourne faux sinon.

Lexique local des variables

i (entier) indice de parcours

Algorithme de initTab

```
pour i de 0 à taille − 1 faire
    tab[i] ← i + 1
fpour
répéter
    mélangeTab(tab, taille)
tant que vérifTab(tab, taille)
```

---

fonction afficheTab(**in** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** vide
Affiche le tableau *tab* de taille *taille* passé en paramètre.

Idée

Parcourir les éléments du tableau pour les afficher.

Lexique local des variables

i (entier) indice de parcours

Algorithme de afficheTab

écrire "[”
pour i **de** *0* **à** *taille* -1 **faire**
  écrire " ", *tab*[i]
fpour
écrire "]”

---

fonction copieTab(**in** *src* : tableau d'entiers, **out** *dest* : tableau d'entiers, **in** *taille* : entier) : **ret** vide

Copie les valeurs du tableau *src* dans le tableau *dest*. Les deux tableaux sont de taille *taille*.

Idée

Parcourir les éléments du tableau *src* et les copier dans le tableau *dest* à la même position.

Lexique local des variables

i (entier) indice de parcours

Algorithme de copieTab

pour i **de** *0* **à** *taille* -1 **faire**
  dest[i] $\leftarrow$ *src*[i]
fpour

---

fonction vérifTab(**in** *tab* : tableau d'entiers, **in** *taille* : entier) : **ret** booléen

Retourne vrai si le tableau *tab* contient les valeurs de 1 à *taille* dans l'ordre. Retourne faux sinon.

Idée

Vérifie qu'on a *tab*[i] $= i + 1$ pour tous les indices valides i. Retourne faux dès qu'un indice ne vérifie pas la relation. Retourne vrai si tous les indices valides vérifient la relation.

Lexique local des variables

i (entier) indice de parcours

Algorithme de vérifTab

pour i **de** *0* **à** *taille* -1 **faire**
  si *tab*[i] $\neq i + 1$ **alors**
    retourner faux
  fsi
fpour
retourner vrai

---

fonction permuteTab(**in-out** *tab* : tableau d'entiers, **in** n : entier) : **ret** vide

Inverse les n premières valeurs du tableau *tab*.

Idée

Procède par échange de valeurs. On note i l'indice de la première valeur à échanger (0 au départ), et j l'indice de la dernière valeur à échanger ($n - 1$ au départ).

Les valeurs de *tab*[i] et de *tab*[j] sont échangées, i est incrémenté et j est décrémenté. On recommence tant que $i < j$.

Lexique local des variables

i	(entier)	premier indice à échanger
j	(entier)	dernier indice à échanger
tmp	(entier)	temporaire pour l'échange

Algorithme de permuteTab

$i \leftarrow 0$
$j \leftarrow n - 1$
tant que $i < j$ **faire**
  tmp $\leftarrow$ *tab*[i]
  tab[i] $\leftarrow$ *tab*[j]
  tab[j] $\leftarrow$ *tmp*
  $i \leftarrow i + 1$
  $j \leftarrow j - 1$
ftant

---

6 Tableaux 2D

6.1 Transposée de matrice

Écrire un algorithme calculant la transposée d'une matrice $m \times n$ de réels.

Correction

Données

Deux entiers m et n, et une matrice $m \times n$ (tableau 2D de $m \times n$ réels).

Résultat

Une autre matrice (tableau 2D de $n \times m$ réels), transposée de la matrice donnée.

Idée

Parcourir toutes les valeurs de la matrice pour construire la transposée ($A^t_{j,i} = A_{i,j}$).

Lexique des variables

n	(entier)	nombre de lignes de la matrice	DONNÉE
m	(entier)	nombre de colonnes de la matrice	DONNÉE
A	(tableau $[m, n]$ de réels)	la matrice donnée	DONNÉE
Atrans	(tableau $[n, m]$ de réels)	la transposée de A	RÉSULTAT
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE

Algorithme

pour i **de** 0 **à** $m - 1$ **faire**
  pour j **de** 0 **à** $n - 1$ **faire**
    $Atrans[j, i] \leftarrow A[i, j]$
  fpour
fpour

6.2 Matrice symétrique

Écrire un algorithme permettant de vérifier si une matrice carrée $n \times n$ est symétrique.

Correction

Données

Un entier n, et une matrice $n \times n$ (tableau 2D de $n \times n$ réels).

Résultat

Vrai si la matrice donnée est symétrique, faux sinon.

Idée

Parcourir toutes les valeurs de la matrice se trouvant sous la diagonale ($A_{i,j}$ avec $j < i$), et les comparer avec leurs symétriques par rapport à la diagonale ($A_{j,i}$). Le parcours s'arrête dès que deux de ces valeurs sont différentes.

Lexique des variables

n	(entier)	dimension de la matrice	DONNÉE
A	(tableau $[n, n]$ de réels)	la matrice donnée	DONNÉE
sym	(booléen)	vrai si la matrice est symétrique	RÉSULTAT
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE

Algorithme

$sym \leftarrow$ vrai
$i \leftarrow 0$
tant que $sym \wedge i < n$ **faire**
  $j \leftarrow 0$
  tant que $sym \wedge j < i$ **faire**
    si $A[i, j] \neq A[j, i]$ **alors**
      $sym \leftarrow$ faux
    fsi
    $j \leftarrow j + 1$;
  ftant
  $i \leftarrow i + 1$
ftant

6.3 Somme et produit de matrices

Écrire un algorithme permettant de calculer la somme de deux matrices de réels. Écrire ensuite un algorithme permettant de calculer le produit de deux matrices de réels.

Rappels :

- La somme de deux matrices A et B de même taille $m \times n$ donne une matrice C de taille $m \times n$ également, calculée par :
$$C_{i,j} = A_{i,j} + B_{i,j} \qquad 1 \leq i \leq m, 1 \leq j \leq n$$
- Le produit de deux matrices A de taille $m \times n$ et B de taille $n \times p$ donne une matrice C de taille $m \times p$, calculée par :
$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} \times B_{k,j} \qquad 1 \leq i \leq m, 1 \leq j \leq p$$

Correction

Somme de deux matrices

Données

Deux entiers m et n, et deux matrices de réels de taille $m \times n$.

Résultat

La somme des deux matrices.

Lexique des variables

m	(entier)	nombre de lignes des matrices	DONNÉE
n	(entier)	nombre de colonnes des matrices	DONNÉE
A	(tableau $[m, n]$ de réels)	première matrice	DONNÉE
B	(tableau $[m, n]$ de réels)	deuxième matrice	DONNÉE
C	(tableau $[m, n]$ de réels)	matrice, somme de A et B	RÉSULTAT
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE

Algorithme

pour i **de** 0 **à** $m - 1$ **faire**
  pour j **de** 0 **à** $n - 1$ **faire**
    $C[i, j] \leftarrow A[i, j] + B[i, j]$
  fpour
fpour

Produit de deux matrices

Données

Trois entiers m, n et p, et deux matrices de réels de tailles $m \times n$ et $n \times p$.

Résultat

Le produit des deux matrices.

Lexique des variables

m	(entier)	nombre de lignes de la première matrice	DONNÉE
n	(entier)	nombre de colonnes de la première matrice	DONNÉE
p	(entier)	nombre de colonnes de la deuxième matrice	DONNÉE
A	(tableau $[m, n]$ de réels)	première matrice	DONNÉE
B	(tableau $[n, p]$ de réels)	deuxième matrice	DONNÉE
C	(tableau $[m, p]$ de réels)	matrice, produit de A et B	RÉSULTAT
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE
k	(entier)	compteur intermédiaire	INTERMÉDIAIRE
somme	(réel)	somme intermédiaire	INTERMÉDIAIRE

Algorithme

pour i **de** 0 **à** $m-1$ **faire**
  pour j **de** 0 **à** $p-1$ **faire**
    $somme \leftarrow 0$
    pour k **de** 0 **à** $n-1$ **faire**
      $somme \leftarrow somme + A[i,k] \times B[k,j]$
    fpour
    $C[i,j] \leftarrow somme$
  fpour
fpour

6.4 Jeu de la vie

Le *jeu de la vie* est un automate cellulaire imaginé par John Horton Conway en 1970. Malgré des règles très simples, le jeu de la vie permet le développement de motifs extrêmement complexes.

Dans ce jeu, on dispose d'un certain nombre de cellules, organisées en grille torique. Chaque cellule a donc 8 voisins. Les cellules du bords ont pour voisines celle du bord opposé. Une cellule a deux états possibles : morte ou vivante.

À chaque itération, l'état de toutes les cellules est mis à jour selon les règles suivantes :

naissance : une cellule morte qui a exactement 3 voisines vivantes devient vivante.

isolement : une cellule vivante qui a moins de 2 voisines vivantes meurt.

étouffement : une cellule vivante qui a plus de 3 voisines vivantes meurt.

L'état des autres cellules ne change pas.

Écrire un algorithme permettant de simuler l'évolution des cellules. On se contentera ici de calculer l'état des cellules, et on ne s'occupera pas d'un affichage éventuel.

Remarque : Au départ, l'état des cellules sera tiré aléatoirement (vivante ou morte, avec un probabilité de 1/2).

Question subsidiaire Modifier l'algorithme pour ne pas recalculer entièrement à chaque itération le nombre de voisines vivantes pour chaque cellules. Lorsqu'une cellule change d'état, le nombre de voisines de ses voisines est modifié.

Correction

Données

Hauteur et largeur de la grille à simuler.

Résultat

Aucun (boucle infinie).

Idée

Les cellules sont modélisées un tableau 2D de booléens, avec les valeurs vrai pour une cellule vivante, et faux pour une cellule morte.

Initialiser la grille de cellules, puis répéter indéfiniment les deux étapes suivantes :

1. calculer le nombre de voisines vivantes pour chaque cellule ;
2. calculer le nouvel état de chaque cellule.

On définira une fonction pour compter le nombre de voisines vivantes d'une cellule.

Lexique des fonctions

fonction nombreAléatoire(vide) : **ret** réel
  Retourne un nombre entier tiré aléatoirement dans l'intervalle [0 ; 1[.

fonction nombreVoisines(**in** *hauteur* : entier, **in** *largeur* : entier,
  in *cellules* : tableau [*hauteur*, *largeur*] de booléens,
  in i : entier, **in** j : entier) : **ret** entier
  Retourne le nombre de voisines vivantes pour la cellule (i, j) dans la grille des états donnée.

Lexique des variables

hauteur	(entier)	hauteur de la grille	DONNÉE
largeur	(entier)	largeur de la grille	DONNÉE
cellules	(tableau [*hauteur*, *largeur*] de booléens)	les états des cellules (vrai pour vivante, faux pour morte)	INTERMÉDIAIRE
voisines	(tableau [*hauteur*, *largeur*] d'entiers)	les nombres de voisines vivantes pour chaque cellule	INTERMÉDIAIRE
i	(entier)	compteur de lignes	INTERMÉDIAIRE
j	(entier)	compteur de colonnes	INTERMÉDIAIRE

Algorithme

```
// initialisation
pour i de 0 à hauteur − 1 faire
    pour j de 0 à largeur − 1 faire
        cellules[i, j] ← (nombreAléatoire() < 0,5)
    fpour
fpour
tant que vrai faire // boucle infinie
    // calcul du nombre de voisines vivantes
    pour i de 0 à hauteur − 1 faire
        pour j de 0 à largeur − 1 faire
            voisines[i, j] ← nombreVoisines(hauteur, largeur, cellules, i, j)
        fpour
    fpour
    // calcul des nouveau états
    pour i de 0 à hauteur − 1 faire
        pour j de 0 à largeur − 1 faire
            selon que voisines[i, j] est
                cas 2 :
                    // 2 voisines vivantes : l'état ne change pas
                cas 3 :
                    cellules[i, j] ← vrai // la cellule se régénère
                défaut :
                    cellules[i, j] ← faux // la cellule meurt
            fselon
        fpour
    fpour
ftant
```

---

fonction nombreVoisines(**in** *hauteur* : entier, **in** *largeur* : entier,
in *cellules* : tableau [*hauteur*, *largeur*] de booléens,
in i : entier, **in** j : entier) : **ret** entier

Retourne le nombre de voisines vivantes pour la cellule (i, j) dans la grille des états donnée.

Idée

Parcourir les 9 cases autour de (i, j), et compter le nombre de cellules voisines vivantes. Les bords de la grille sont gérés à l'aide d'opérations modulo.

Lexique local des variables

k	(entier)	compteur de lignes
l	(entier)	compteur de colonnes
vi	(entier)	numéro de ligne d'un voisin
vj	(entier)	numéro de colonne d'un voisin
voisines	(entier)	nombre de voisines vivantes

Algorithme de nombreVoisines

```
voisines ← 0
pour k de i − 1 à i + 1 faire
    vi ← (k + hauteur) mod hauteur
    pour l de j − 1 à j + 1 faire
        vj ← (l + largeur) mod largeur
        si (vi ≠ i ∨ vj ≠ j) ∧ cellules[vi, vj] alors
            voisines ← voisines + 1
        fsi
    fpour
fpour
retourner voisines
```

Pour la question subsidiaire, l'idée est la suivante : pour chaque cellule, on doit avoir son état, son état suivant et son nombre de voisines. Ensuite :

- initialiser la grille des états
- initialiser de la même manière la grille des états suivants
- calculer le nombre de voisines vivantes pour chaque cellule
- l'évolution se fait alors de la manière suivante (boucle infinie) :
  - calculer le nouvel état suivant pour toutes les cellules
  - si l'état d'une cellule a changé, corriger le nombre de voisines de ses voisines, et changer l'état courant