# SERIE D'EXERCICES EN INFORMATIQUE

**Proposée par Pr. D. El Ghanami**

**Ecole Mohammadia d'Ingénieurs**

**Mai 2011**

## Enoncés

**Exercice 1.1 :**

Les types manipulés en algorithmique sont : Entier, Réel, Caractère et Booléen (les types qui peuvent être représentés en binaire). Donner le type et la valeur des expressions suivantes :

1. *2 + 3 * 4*
2. *2.0 + 3 * 4*
3. *vrai et (faux ou vrai)*
4. *(2 < 3) et (4 > 5)*

**Exercice 1.2 :**

Parmi les instructions suivantes, lesquelles sont correctement écrites (justifier votre réponse en indiquant le type possible de chaque variable).

1. *z ← (x > 2) et (y < 5)*
2. *z ← (x > 2) et y*
3. *z ← (x > 2 et y) < 5*
4. *z ← (x + y) > 2 et y*
5. *si (y ≠ 0) alors z ← x/y sinon z ← faux ; finsi ;*

**Exercice 1.3 :**

Donner la table de vérité des expressions booléennes suivantes :

- *(a et non b) ou c*
- *(a et non b) ou (non a et b)*

**Exercice 1.4 :**

Soient *x, y, z, t* des entiers. Donner les expressions booléennes correspondant aux situations suivantes:

1. *Les valeurs de x et de y sont toutes les deux supérieures à 3*
2. *Les variables x, y et z sont identiques*
3. *Les valeurs de x, y et z sont identiques mais différentes de celle de t*
4. *Les valeurs de x sont strictement comprises entre les valeurs de y et t*
5. *Parmi les valeurs de x, y et z deux valeurs au moins sont identiques*
6. *Parmi les valeurs de x, y et z deux valeurs et seulement deux sont identiques*
7. *Parmi les valeurs de x, y et z deux valeurs au plus sont identiques*

*Indication : Utiliser les parenthèses si l'expression logique comporte des Ou et des Et*

**Exercice 1.5 :**

Quelles seront les valeurs des variables *a*, *b* et *c* après exécution des instructions suivantes :

*a ← 1 ;*
*b : ← 5 ;*
*c ← a – b ;*
*a ← 2 ;*
*c ← a + b ;*

**Exercice 1.6 :**

On considère trois variables entières *x, y, z*. Donner des expressions booléennes pour déterminer si :

1. *x* est pair.
2. *x* est impair.
3. *x* et *y* ont la même parité.
4. L'une au moins des trois est paire.
5. Deux d'entre-elles au moins ont la même parité.
6. Exactement deux sur les trois sont paires.

*Indication : Utiliser la fonction mod qui permet d'avoir le reste de la division de deux entiers.*

**Exercice 1.7 :**
Ecrire un algorithme qui permet d'échanger les valeurs de deux variables entières *a* et *b*.
*Indication : Lors de l'affectation, la valeur précédente d'une variable est remplacée par la nouvelle.*

**Exercice 1.8 :**
Ecrire un algorithme qui affiche à l'écran "Bonjour"

**Exercice 1.9 :**
Écrire un algorithme qui à partir de 3 notes d'un étudiant et 3 coefficients calcule et affiche la moyenne.
*Indication : Définir d'abord les données d'entrées et de sorties du problème, leur type et par la suite le traitement à faire.*

**Exercice 1.10 :**
Écrire un algorithme qui à partir d'une somme d'argent donnée, donne le nombre minimal de : billets de 50Dh, 20Dh, les pièces de 2Dh et de 1Dh qui la compose.
*Indication : On suppose que le montant est la différence entre le prix à payer par un client dans un magasin et le montant qu'il donne au caissier.*
*Pour avoir un minimum de billets et de pièces à rendre, il faut maximiser le nombre de billets de grandes valeurs et minimiser celui de pièces de petites valeurs.*

**Exercice 1.11 :**
Écrire un algorithme qui permet d'effectuer une permutation circulaire des valeurs entières de trois variables *x, y, z* (la valeur de *y* dans *x*, la valeur de *z* dans *y* et la valeur de *x* dans *z*).
*Indication : Sauvegarder la valeur d'une variable et commencer à partir d'elle.*

**Exercice 2.1 :**
Écrire un algorithme qui détermine si une année est bissextile ou non. Les années bissextiles sont multiples de *4*, mais pas de *100*, sauf pour les millénaires qui le sont.

**Exercice 2.2 :**
Écrire un algorithme qui prend en entrée trois entiers et qui les trie par ordre croissant.

**Exercice 2.3 :**
Ecrire un programme qui lit deux valeurs entières quelconques (*A* et *B*) au clavier et qui affiche le signe de la somme de *A* et *B* sans faire l'opération de l'addition.
Exemple : $A = -8, B = 3$ le signe de la somme est négatif.

**Exercice 2.4 :**
Écrire un algorithme qui pour un temps donné (représenté sous la forme : heure, minute, seconde) affiche le temps (sous la même représentation) après avoir ajouté une seconde.
On suppose que $0 \leq h < 24, 0 \leq m < 60, 0 \leq s < 60$.

**Exercice 2.5 :**
Écrire un algorithme qui détermine le numéro d'un jour dans l'année en fonction du jour, du mois, et de l'année.

**Exercice 2.6 :**
Programmer une petite calculatrice qui demande à l'utilisateur une opération à effectuer sous forme de caractère (par exemple '*', '+', '-', '/'), demande ensuite *2* nombres et effectue le calcul demandé et affiche le résultat.

**Exercice 2.7 :**

Dans une entreprise, le calcul des jours de congés payés s'effectue de la manière suivante : si une personne est entrée dans l'entreprise depuis moins d'un an, elle a droit à deux jours de congés par mois de présence, sinon à *28* jours au moins. Si c'est un cadre et s'il est âgé d'au moins *35* ans et si son ancienneté est supérieure à *3* ans, il lui est accordé *2* jours supplémentaires. S'il est âgé d'au moins *45* ans et si son ancienneté est supérieure à *5* ans, il lui est accordé *4* jours supplémentaires, en plus des *2* accordés pour plus de *35* ans. Écrire un algorithme qui calcule le nombre de jours de congés à partir de l'âge, l'ancienneté et l'appartenance au collège cadre d'un employé.

**Exercice 2.8 :**

Résolution d'une équation du second degré dans C : $a.x^2 + b.x + c = 0$

**Exercice 3.1 :**

1- Écrire un algorithme qui calcule la somme des *1/i* pour *i* allant de *1* à *n* pour une valeur de *n* rentrée au clavier. Faire cette somme en partant de *1* à *n* et la recalculer en partant de *n* à 1. Comparer ces deux sommes et conclure pour différentes valeurs de *n*.

2- Chercher la valeur de *n* à partir de laquelle les deux sommes deviennent différentes.

Remarque : pour utiliser un entier en tant que réel, il suffit de le multiplier par *1.0*

**Exercice 3.2 :**

Écrire un algorithme qui effectue la multiplication de deux entiers positifs (notés *x* et *y*) donnés en utilisant uniquement l'addition entière.

**Exercice 3.3 :**

Afficher la table de conversion entre les degrés Fahrenheit et Celsius de *250* à *−20* degré F par pallier de *10* degrés. On passe de *x* degré F au degré C en calculant (*5/9 x − 160/9*).

**Exercice 3.4 :**

Écrire un algorithme qui dessine, à l'aide du signe '+', les figures de hauteur n suivantes :

*N.B : on suppose l'existence de la procédure retourLigne() qui permet un retour à la ligne.*

1-

```
+
+       +
+       +       +
```

2-

```
                +
        +       +       +
+       +       +       +       +
```

3-

```
+       +       +
+               +
+       +       +
```

4-

```
                +
        +               +
+                               +
        +               +
                +
```

**Exercice 3.5 :**

Une école primaire veut disposer d'un logiciel pour enseigner aux enfants quelques rudiments d'arithmétique, en particulier les tables de multiplication. Le fonctionnement d'un tel système doit

permettre à tout élève de vérifier ses connaissances au cours d'une session (suite de questions du système, réponses de l'élève). Chaque session se déroulera ainsi :

- Le système propose deux nombres, entre 0 et 10, tirés au hasard par la fonction hasard(10).
- L'élève en donne le produit.
- En cas de réponse, un message s'affiche et une nouvelle question est posée.

La fin de la session survient quand l'élève a fourni 20 bonnes réponses ou 10 fausses avec droit à trois essais successifs maximum pour un essai donné.
Ecrire un algorithme qui simule le jeu.
*N.B. La fonction rand() permet de générer une valeur aléatoire.*

**Exercice 3.6 :**
Écrire un algorithme qui vérifie si N un entier positif est un carré parfait.
Exemple : 4, 9, 16 sont des carrés parfaits.

**Exercice 3.7 :**
Ecrire un algorithme qui permet de calculer et d'afficher le nombre d'occurrences d'un chiffre ($0 \leq chiffre < 10$) dans un nombre positif.
Exemples : L'occurrence du chiffre *7* dans le nombre *778* est *2*.
L'occurrence du chiffre *8* dans le nombre *20681* est *1*.
L'occurrence du chiffre *5* dans le nombre *2771* est *0*.

**Exercice 3.8 :**
Trouver tous les entiers entre 1 et n vérifiant $x^2 + y^2 = z^2$
Exemples de résultats : *(8, 6, 10), (8, 15, 17)*
*Indication : Tester la formule pour un triplet (x, y, z) puis inclure les actions dans un ensemble de répétitions imbriquées afin d'obtenir tous les triplets possibles.*

**Exercice 3.9 :**
Lorsque *x* est proche de *0*, *arcsin (x)* peut être approximé à l'aide de la formule suivante :

$$\sum_{i=1}^{n} \frac{1*3*...*(2i-1)}{2^i * i! * (2i+1)} x^{2i+1}$$

Écrire un algorithme qui calcule une approximation de *arcsin(x)*.

**Exercice 3.10 :**
Ecrire un algorithme qui recherche le premier nombre entier naturel dont le carré se termine par *n* fois le même chiffre.
Exemple : pour $n = 2$, le résultat est *10* car *100* se termine par *2* fois le même chiffre

**Exercice 3.11 :**
Ecrire un algorithme qui simule le problème de Syracuse définit par :
$u_0 = m > 1$
$u_{n+1} = u_n/2$ si $u_n$ est pair
$u_{n+1} = 3u_n+1$ sinon
Exemple : *6 → 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1*

**Rappel sur les tableaux**
Déclaration de tableau :
*int T[3] ;* déclare un tableau *T* avec réservation de *3* cases mémoires de la taille d'un entier.
*int T[3] = {12, 14, 6} ;* déclare *T* un tableau d'entiers initialisés de dimension *3*.
Déclaration de matrice :
*int T[2][3] ;* déclare *T* comme un tableau de dimension *2* de tableaux de *3* entiers.
*int T[2][3] = {{1, 5, 1}, {2, 2, 0}} ;* déclare *T* une matrice d'entiers initialisés de dimension *2*x*3*.

Un tableau ou une matrice des paramètres d'une fonction sont passés par référence.
L'indice d'un tableau commence de *0* en langage C et de *1* en pseudo code (en algo).
Procédure utilisées :
On donne les procédures suivantes qui permettent d'initialiser un tableau ou une matrice.
*remplirTab(T)* remplit un tableau *T* d'entiers par *N* valeurs.
*remplirMat(T)* remplit une matrice *T* (*N*x*M*) d'entiers par des valeurs.

**Exercice 4.1 :**
Ecrire un algorithme qui affiche un tableau à l'envers.

**Exercice 4.2 :**
Soit un tableau *t* d'entiers. Écrire un algorithme qui change de place les éléments de ce tableau de telle façon que le nouveau tableau *t* soit une sorte de "miroir" de l'ancien.
Exemple : *1 2 4 6* → *6 4 2 1*

**Exercice 4.3 :**
Ecrire un algorithme qui vérifie si une chaîne est un carré ou pas.
Définition : Une chaîne de caractères est un carré si elle se compose de 2 chaînes identiques.
Exemple : "chercher" et "bonbon" sont des carrés.

**Exercice 4.4 :**
Un programme qui permet de :
- Lire à partir du clavier un nombre entier *N* ($0 < N \leq 10$)
- Lire à partir du clavier N entiers et les mettre dans un tableau t
- Décaler chaque élément du tableau t vers le haut (le $2^{ème}$ élément devient le $1^{er}$, le $3^{ème}$ devient le $2^{ème}$, …, le $1^{er}$ devient dernier)

Exemple : *t = {-3, 0, 5, -8, 1, 4}*
Décalage vers le haut *t = {0, 5, -8, 1, 4, -3}*

**Exercice 4.5 :** Recherche dans un tableau
1- Ecrire un algorithme qui recherche l'indice d'une valeur dans un tableau quelconque
Principe : Rechercher l'indice d'une valeur dans un tableau quelconque, parcourir le tableau jusqu'à trouver cette valeur et s'arrêter (recherche séquentielle).
2- Ecrire un algorithme qui recherche l'indice d'une valeur dans un tableau trié.
Principe : Le tableau est trié c'est-à-dire que les éléments sont rangés dans l'ordre (en général croissant). L'algorithme précédent s'exécute parfaitement, mais comment l'améliorer ?
Pour une valeur qui existe dans le tableau, aucune différence : on s'arrête dès qu'on l'a trouvée ; mais, pour une valeur qui ne se trouve pas dans le tableau, on peut s'arrêter dès que l'on arrive sur une valeur trop grande, on forcera donc la boucle à s'arrêter.

**Exercice 4.6 :**
Écrire un algorithme qui à partir d'un tableau d'entiers *t* d'au moins un entier, fournit le nombre de sous-séquences croissantes de ce tableau, ainsi que les indices de début et de fin de la plus grande sous-séquence.
Par exemple, soit t un tableau de *15* éléments :

*1, 2, 5, 3, 12, 25, 13, 8, 4, 7, 24, 28, 32, 11, 14*

Les séquences strictement croissantes sont :

*< 1, 2, 5 >;< 3, 12, 25 >;< 13 >;< 8 >;< 4, 7, 24, 28, 32 >;< 11, 14 >*

Le nombre de sous-séquence est : 6 et la plus grande sous-séquence est :

*< 4, 7, 24, 28, 32 >*

**Exemple 4.7 :**

Ecrire un algorithme qui permet de remplir et d'afficher une matrice par des valeurs saisies.

**Exercice 4.8 :** Moyenne de notes par élève et par matière

On considère une matrice de notes dont les *M* lignes correspondent à *m* élèves et dont les *n* colonnes correspondent à *N* matières.

De plus on dispose d'un vecteur dont les *N* valeurs correspondent à des coefficients.

On demande de calculer la moyenne par élève et, distinctement, de calculer la moyenne par matière

**Exercice 4.9 :** Produit matriciel

Écrire un algorithme qui réalise le produit *C* de deux matrices *A* et *B*.

*Méthode :* En multipliant une matrice *A(N,M)* avec une matrice *B(M,P)* on obtient une matrice *C(N,P)*.

La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes :

$$C_{i,j} = \sum_{k=1}^{M} A_{i,k} * B_{k,j}$$

**Exercice 4.10 :**

Ecrire un algorithme qui affiche un triangle de Pascal d'ordre n.

| | | | | | |
|---|---|---|---|---|---|
| *1* | | | | | $(a+b)^0 = 1$ |
| *1* | *1* | | | | $(a+b)^1 = 1*a + 1*b$ |
| *1* | *2* | *1* | | | $(a+b)^2 = 1*a^2 + 2*a*b + 1*b^2$ |
| *1* | *3* | *3* | *1* | | $(a+b)^3 = 1*a^3 + 3*a^2*b + 3*a*b^2 + 1*b^3$ |
| *1* | *4* | *6* | *4* | *1* | $(a+b)^4 = 1*a^4 + 4*a^3*b + 6*a^2*b^2 + 4*a*b^3 + 1*b^4$ |

*Indication : il ne faut pas utiliser la Formule du triangle de Pascal suivante :*

$$C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$

**Exercice 5.1 :**

Ecrire une fonction qui à partir d'un réel retourne la valeur absolue de ce réel.

**Exercice 5.2 :**

Ecrire une fonction qui à partir d'un nombre entier strictement positif retourne *VRAI* si ce nombre est pair et *FAUX* sinon.

**Exercice 5.3 :**

Ecrire une fonction qui retourne le nombre de chiffres dans un nombre.

Exemple: Pour le nombre entier *21668*, la fonction retourne *5* (le nombre de chiffres).

**Exercice 5.4 :**

Ecrire un algorithme qui permet de calculer la distance nécessaire à une voiture pour s'arrêter, connaissant la vitesse de la voiture en km/h et l'état de la route : sèche ou mouillée.

La formule de calcul de la distance est :

*Distance d'arrêt = chemin de réaction + distance de freinage.*

Sachant que la réaction du conducteur est :

*Le chemin de réaction = 3mètres pour 10km/h.*

Distance de freinage :

*Distance sur route mouillée = vitesse$^2$/100* *(ex. à 20km/h, on a 4m).*

*Distance sur route sèche = 3/4 * Distance sur route mouillée.*

L'algorithme permet aussi de retournez le problème et calcule la vitesse de la voiture d'après la longueur des traces de freinage (les traces de freinage ne représente pas la réaction du conducteur).

**Exercice 5.5 :**

Écrire une fonction qui à partir d'un entier strictement positif donné, retourne le résultat booléen *VRAI* ou *FAUX* selon que le nombre est premier ou non.
Afficher la liste des nombres premiers inférieurs à *n*
Exemple : *n = 50* donne *1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.*

**Exercice 5.6 :**
Écrire un algorithme qui affiche la suite de tous les nombres parfaits inférieurs ou égaux à un nombre, entier positif donné noté *n*. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs stricts.
Exemple : *28 = 1 + 2 + 4 + 7 + 14*
Voici la liste des nombres parfaits inférieurs à *10000* : *6, 28, 496, 8128.*
*Indication : Pour un nombre donné, chercher les diviseurs et en faire la somme ; vérifier que le nombre est parfait, puis inclure ces actions dans une répétition permettant de tester tous les nombres.*

**Exercice 5.7 :**
Programmer une procédure qui échange les valeurs de deux entiers.

**Exercice 5.8 :**
Donner un algorithme qui permet d'afficher les nombres jumeaux compris entre *1* et *n*.
On dit que *2* entiers positifs *p* et *q* sont jumeaux, s'ils sont tous les *2* premiers et si $q=p+2$ (ou $p=q+2$).
Par exemple, *5* et *7* sont jumeaux.

**Exercice 5.9 :**
Deux nombres entiers n et m sont qualifiés d'amis, si la somme des diviseurs de *n* est égale à *m* et la somme des diviseurs de *m* est égale à *n* (on ne compte pas comme diviseur le nombre lui même et *1*).
Exemple : les nombres *48* et *75* sont deux nombres amis puisque :
Les diviseurs de *48* sont : *2 + 3 + 4 + 6 + 8 + 12 + 16 + 24 = 75*
Les diviseurs de *75* sont : *3 + 5 + 15 + 25 = 48.*
Ecrire un algorithme qui permet de déterminer si deux entiers *n* et *m* sont amis ou non.

**Exercice 5.10 :**
Lorsque *x* est proche de *0*, *sin (x)* peut être approximé à l'aide de la formule suivante :

$$\sum_{i=0}^{n} \frac{(-1)^i * x^{2i+1}}{(2i+1)!}$$

Écrire une fonction qui retourne une approximation de *sin(x)*.
On s'arrête lorsque l'ajout est inférieur à une précision donnée (par exemple $10^{-6}$) ou que l'on a atteint un certain nombre d'itérations *n*.

**Exercice 5.11 : Nombre d'Armstrong**
Ecrire un algorithme qui vérifie si un Entier positif est un nombre d'Armstrong.
Un nombre d'Armstrong *n* est un entier naturel pour lequel il existe un entier positif ou nul *p* tel que la somme de chacun des chiffres de *n* mis à la puissance *p* est égale à *n*.
Ecrire un algorithme qui vérifie si un Entier positif est un nombre d'Armstrong.
Exemple : $153 = 1^3 + 5^3 + 3^3$
$548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6$

**Exercice 5.12 :**
Programmer une fonction qui insère un élément à sa place dans un tableau qu'on supposera déjà trié.
Principe : Ajouter la valeur de façon que le tableau reste trié, mais sans exécuter de tri.
Décaler les éléments en partant de la fin jusqu'à trouver une place correcte pour placer la nouvelle valeur ; il faut supposer que la place est suffisante c'est-à-dire que le nombre effectif d'éléments est strictement plus petit que la taille du tableau en mémoire.

**Exercice 5.13 :**
Écrire une fonction qui compte le nombre d'occurrences d'un caractère dans une chaîne de caractères.

**Exercice 5.14 :**
Ecrire une procédure qui permet de créer une copie d'une chaine de caractères.

**Exercice 5.15 :**
Ecrire une fonction qui retourne le minimum et sa position et le maximum et sa position dans un tableau d'entiers.
*Indication : une fonction ne retourne qu'une seule valeur. Il faut passer les variables par référence.*

**Exercice 5.16 :**
Ecrire une procédure qui permet de concaténer deux mots dans le premier.
Exemple : concaténer "bon" et "jour" donne "bonjour".

**Exercice 5.17 :**
Ecrire une fonction qui permet de comparer deux mots et qui retourne :
*–1* si le premier mot est plus petit
*0* si les deux mots sont égaux
*1* sinon
Exemple : "*avant*" *et* "*plus*" → *-1*
"*pareil*" *et* "*pareil*" → *0*
"*avant*" *et* "*apres*" → *1*

**Exercice 5.18 :**
Ecrire un algorithme qui permet de supprimer les espaces supplémentaires (plus d'un espace) dans une chaîne de caractère.

**Exercice 5.19 :**
Ecrire un algorithme qui permet de purger un tableau (supprimer les éléments qui se répètent) d'entiers positifs de taille N sans utiliser un autre tableau.
Exemple : *t = {1, 5, 5, 10, 9, 1, 1, 30}*
devient *t = {1, 5, 10, 9, 30}*

**Exercice 5.20 :**
Ecrire une procédure qui permet de fusionner deux tableaux triés passés en paramètre dans un troisième tableau.
Mettre bout à bout deux tableaux triés de façon que le tableau résultant soit trié, et ce, sans exécuter de tris (coûteux).
*Indication : Parcourir les 2 tableaux (avec 2 indices), et recopier dans le 3$^{ème}$ tableau la plus petite des 2 valeurs du 1$^{er}$ et du 2$^{nd}$ tableau ; puis avancer dans le tableau d'où on a recopié la valeur. A la fin de ce traitement, l'un des tableaux est copié, il suffit de copier les éléments restant de l'autre.*

**Exercice 5.21 :**
Ecrire une procédure qui échange deux éléments d'un tableau d'entiers.

**Exercice 5.22 :** Tri de tableau
Écrire une procédure qui réalise le tri d'un tableau d'entiers par la méthode du :
1- tri par sélection.
Principe :
On recherche le plus petit élément *T[m]* de la suite *T[0] ... T[N–1]* et on l'échange avec *T[0]*,
On recherche le plus petit élément *T[m]* de la suite *T[1] ... T[N–1]* et on l'échange avec *T[1]*,

etc.
2- tri par insertion.
Principe :
On suppose les *i–1* premiers éléments triés et on insère le *i*-ème à sa place parmi les *i* premiers.
3- tri à bulles.
Principe :
On parcourt la suite *t[0] ... t[N–1]* en échangeant les éléments consécutifs *t[j–1] > t[j]*
On parcourt la suite *t[0] ... t[N–2]* en échangeant les éléments consécutifs *t[j–1] > t[j]*, etc.

**Exercice 5.23 :** Anagramme
Deux mots sont des anagrammes si l'un est une permutation des lettres de l'autre. Par exemple les mots suivants sont des anagrammes :
– aimer et maire
– chien et niche
– . . .
Par définition, on considère que deux mots vides sont des anagrammes.
Ecrire une fonction qui vérifie si deux mots sont des anagrammes.

**Exercice 5.24 :**
Soit *t* un tableau d'entiers non trié. Écrire une fonction qui retourne l'élément qui apparaît le plus souvent dans le tableau *t*, ainsi que son nombre d'occurrences. Si plusieurs éléments différents répondent au problème, votre algorithme doit en fournir un, quel qu'il soit. Vous ne devez utiliser aucun autre tableau que celui sur lequel vous travaillez.

**Exercice 5.25 :**
Ecrire une fonction qui calcule le produit scalaire de deux vecteurs.
Utiliser cette fonction pour écrire une procédure qui calcule le produit de deux matrices *A* et *B*

**Exercice 5.26 :**
Soit une matrice *T* de dimensions *N* x *N* x *2* d'entiers représentant l'existence de liaisons ferroviaires directs entre *N* villes et le prix de la liaison s'il existe.
Une ville est indiquée par un numéro (c'est un indice dans la matrice).
*T[i,j,0] = 1* (ou *0*) indique l'existence (ou pas) d'une liaison directe entre les villes *i* et *j*
*T[i,j,1] = 100* indique le prix de la liaison entre les villes *i* et *j* s'il existe (*T[i,j,1] = 1*).
Sur la diagonale est présente la même ville (*i = j*) et donc pas de liaison (cases vides).
1- Ecrire une fonction qui pour deux villes *k* et *l*, permet de savoir si les deux villes sont joignables directement en donnant le prix du trajet.
2- En cas d'absence de liaison directe, écrire une fonction qui cherche une ville qui permet de joindre indirectement ces deux villes (une seule escale) en donnant le prix du trajet.
3- Ecrire un algorithme qui simule une recherche de billet de train.

**Exercice 5.27 :**
1- Ecrire une fonction qui retourne la valeur d'un polynôme *P* de degré *n* en un point *x* donné.
Les coefficients ($a_n$, $a_{n-1}$, …., $a_0$) de *P* sont contenus dans le tableau *t* (*t[0]*=$a_0$, …, *t[n]*=$a_n$)
$p(x) = a_n*x^n + a_{n-1}*x^{n-1} + ... + a_1*x^1 + a_0$
2- Schéma de Horner
$p(x) = (((a_n*x + a_{n-1})*x + ... )*x + a_1)*x + a_0$

**Exercice 5.28 :**
En considérant un tableau de trois couleurs 'A', 'B', 'C' (ex. *Bleu, Blanc, Rouge)* réparties aléatoirement de dimension MAX, écrire une procédure qui le tri de telle façon que toutes les couleurs 'A' apparaissent au début du tableau, suivies des 'B' puis des 'C'.

**Exercice 5.29 :** Transposition de matrice
Écrire un algorithme qui réalise la transposition d'une matrice (en utilisant une seule matrice).

**Exercice 5.30 :**
Donnez une fonction qui retourne l'indice de la première occurrence de sous chaîne dans une chaîne. Si sous chaîne n'est pas présente dans la chaîne, alors la fonction retourne *-1*.

**Exercice 6.1 :**
Calculer le *n-ème* nombre de Fibonacci $F_n$ qui est défini de la manière suivante :

$$\begin{cases} F_0 = F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \text{ pour } n > 1 \end{cases}$$

**Exercice 6.2 :**
Programmer l'exponentiation binaire de manière récursive. Cette fonction consiste à calculer $x^n$ en appelant $(x*x)^{n/2}$ si *n* est pair, et $x*x^{n-1}$ si *n* est impair et retourne *1* si *n = 0*.

**Exercice 6.3 :**
Ecrire une fonction récursive qui affiche à l'écran des entiers positifs lus au clavier dans l'ordre inverse de leur saisi.
*Indication : Afficher dans l'ordre inverse nécessite une mémorisation des valeurs saisies avant l'affichage. Le fait que la dernière valeur saisie est la première à afficher, l'avant dernière valeur saisie et la deuxième à afficher et ce jusqu'à la première valeur saisie qui sera afficher en dernier pousse à utiliser une pile (la pile de la récursivité).*

**Exercice 6.4 :** Ackerman
Ecrire une fonction qui permet de calculer et retourner la valeur de la fonction d'Ackermann pour deux entiers positifs m et n donnés comme paramètres
La fonction d'Ackermann est définie comme suit :
*Ackermann(0, n) = n + 1* *si* $n \geq 1$
*Ackmann(m, 0) = Ackmann(m–1, 1)* *si* $m \geq 1$
*Ackmann(m, n) = Ackmann(m–1, Ackmann(m, n–1))* *si* $n \geq 1$ *et* $m \geq 1$.

**Exercice 6.5 :** Suites récurrentes de Mycielski
On considère les deux suites récurrentes de Mycielski définies par :

$$\begin{cases} m_1 = 2 \\ m_k = 2 * m_{k-1} + 1 \end{cases}$$

$$\begin{cases} c_1 = 1 \\ c_k = 3 * c_{k-1} + m_{k-1} \end{cases}$$

Ecrire un algorithme qui affiche le n-ème terme de la suite $c_n$.

**Exercice 6.6 :** PGCD
Programmer le *PGCD* de deux entiers en utilisant l'algorithme d'Euclide par récursivité.
*Principe*
*si x > y alors* *PGCD (x, y) = PGCD (y, x)*
*si* $1 \leq x \leq y$ *alors* *PGCD (x, y) = PGCD (x, y modulo x)*
*si x = 0 alors* *PGCD (x, y) = y*

**Exercice 6.7 :** PPCM

Le *PPCM* (plus petit commun multiplicateur) de deux entiers *a* et *b* est le plus petit entier *m* tel que *m* est divisible par *a* et *b*.
Exemple *PPCM (4, 3) = 12*.
Proposer un algorithme qui calcule le *PPCM* de deux entiers strictement positifs.
Indication : le PPCM est un multiple d'un des nombres qui doit être multiple de l'autre, ou en d'autres termes divisible par l'autre ; et si les multiples sont pris par ordre croissant, le premier trouvé sera le bon.

**Exercice 6.8 :**
Ecrire une procédure qui affiche un entier *x* (exprimé en base 10) en base *b* (*1<b<10*).
Exemple : $(42)_{10} = (101010)_2 = (52)_8$
Indication : La base b est une représentation des nombres qui utilise des chiffres de 0 à b–1 alors que la base 10 utilise de 0 à 9 (ex. base binaire *b=2* est représentée par 0 et 1). Pour transformer un nombre écrit en base 10 en un nombre écrit en base b, il suffit d'effectuer des divisions successives par b en prenant à chaque fois comme diviseur le quotient de la division précédente (sauf pour la première où le diviseur est le nombre donné) et ceci jusqu'à ce que ce diviseur soit égal à 0. Les restes successifs de ces divisions sont alors la représentation inverse en base b du nombre de départ.
Un affichage des restes au fur à mesure des divisions (solution 1) donnera un affichage à l'envers (du bit de poids faible vers le bit de poids fort) alors que pour afficher correctement le résultat, il faut mémoriser les restes jusqu'à la fin et par la suite les affichés à l'envers (c.à.d le dernier reste calculé est le premier affiché, etc.). Ainsi, la solution passe par l'utilisation d'une pile.

**Exercice 6.9 :**
Ecrire une fonction récursive qui affiche les éléments d'un tableau d'entiers du premier au dernier.
*Indication : L'appel de la fonction récursive remplacera l'itération.*

**Exercice 6.10 : Nombres méchants**
Les nombres méchants sont des nombres dont les facteurs premiers sont uniquement 2, 3 ou 5. Les 11 premiers nombres méchants sont : 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15.
Par convention, 1 est suppose un nombre méchant.
Ecrire un programme qui permet de trouver le centième nombre méchant.

**Exercice 6.11 :**
Ecrire une procédure qui transforme un entier en chaîne de caractères et une procédure qui réalise l'opération inverse
Exemple : *875 → "875"*

**Exercice 6.12 :**
Les égyptiens de l'antiquité savaient additionner deux entiers strictement positifs, soustraire 1 à un entier strictement positif, multiplier par *1* et *2* tout entier strictement positif et diviser par *2* un entier strictement positif pair.
Exemple de multiplication de *15* par *13* en n'utilisant que ces opérations :

$$\begin{aligned} 15 \times 13 &= 15 + 15 \times 12 \\ &= 15 + 30 \times 6 \\ &= 15 + 60 \times 3 \\ &= 75 + 60 \times 2 \\ &= 75 + 120 \times 1 \\ &= 75 + 120 \\ &= 195 \end{aligned}$$

Ecrire l'algorithme qui permet la multiplication de *2* entiers positifs suivant cette méthode (Donner une version itérative et une version récursive).

**Exercice 6.13 :**

Ecrire un algorithme qui permet de calculer le produit de *2* naturels en utilisant la Multiplication russe dont le principe est le suivant :

On divise par 2 autant de fois que possible un des deux nombres et en multiplie le $2^{ème}$ nombre par 2. Le produit est égal à la somme des multiples correspondant aux divisions impaires.

Exemple : 37 x 15

| | | | | | |
|---|---|---|---|---|---|
| 37 | 18 | 9 | 4 | 2 | 1 |
| 15 | 30 | 60 | 120 | 240 | 480 |

37 x 15 = 15 + 60 + 480 = 555

**Exercice 6.14 :**

On forme une suite de nombres de la façon suivante :

- le point de départ est un entier naturel donné, non nul, multiple de *3* ;
- chaque nombre a pour successeur la somme des cubes de ses chiffres ;
- théorème : toute suite de ce type devient constante, égale à *153* ;
- exemple : partons de *33*

$3^3 + 3^3 = 54$

$5^3 + 4^3 = 189$

$1^3 + 8^3 + 9^3 = 1242$

$1^3 + 2^3 + 4^3 + 2^3 = 81$

$8^3 + 1^3 = 513$

$5^3 + 1^3 + 3^3 = 153$

Ecrire un algorithme pour démontrer ce théorème.

**Exercice 6.15 :** Palindrome

Écrire un algorithme qui permet de savoir si une chaîne de caractères est un palindrome (un palindrome est un mot qui se lit aussi bien de la gauche vers la droite que de la droite vers la gauche comme par exemple les mots "été" ou "radar").

(Donner une version itérative et une version récursive).

**Exercice 6.16 :** Recherche dichotomique

Ecrire une fonction qui retourne la position d'un entier dans un tableau trié en utilisant la recherche dichotomique.

Principe :

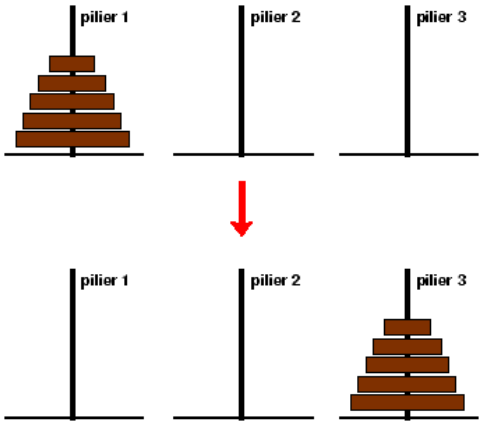
La recherche séquentielle fait apparaître un temps d'exécution en O(n) (proportionnel au nombre effectif de valeurs) ; comment améliorer l'algorithme ?

Le nombre d'éléments étant important, on décide de n'étudier que la moitié, on coupe en deux, on vérifie que la valeur recherchée est exactement au milieu, auquel cas on s'arrête car on a trouvé ; sinon on sait de quel côté (à gauche ou à droite) doit se trouver la valeur si elle existe ; et on recommence pour la partie considérée ; on définit la partie considérée par deux indices désignant la borne inférieure et la borne supérieure.

**Exercice 6.17 :** Tours de Hanoï

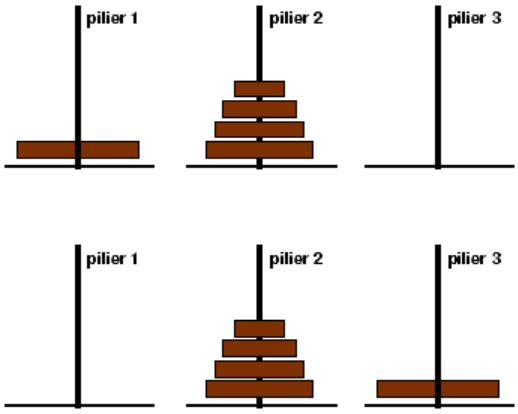
Les tours de Hanoï est un jeu solitaire dont l'objectif est de déplacer les disques qui se trouvent sur une tour (par exemple ici la première tour, celle la plus à gauche) vers une autre tour (par exemple la dernière, celle la plus à droite) en suivant les règles suivantes :

- on ne peut déplacer que le disque se trouvant au sommet d'une tour ;
- on ne peut déplacer qu'un seul disque à la fois ;
- un disque ne peut pas être posé sur un disque plus petit.




*Méthode :* Résoudre le problème des tours de Hanoï, c'est-à-dire déplacer *n* disques d'une tour source 'S' vers une tour destination 'D' en utilisant une tour intermédiaire 'I', revient à :

- déplacer *n – 1* disques de la tour source vers la tour intermédiaire;
- déplacer *1* disque de la tour source vers la tour destination;
- déplacer *n – 1* disques de la tour intermédiaire vers la tour destination.




**Exercice 6.18 :** Tri rapide (Quick sort)

Le principe est de placer une valeur du tableau à trier à sa bonne place (appelée pivot) puis recommencer la démarche sur les deux moitiés droite et gauche du pivot jusqu'à la fin du tri.

**Exercice 6.19 :** Tri par fusion

Écrire des procédures qui réalisent le tri d'un tableau de *n* entiers par la méthode du tri par fusion.

Principe :

Etant donné un tableau de taille *MAX*, on le partage en deux tableaux de taille (approximative) *MAX/2* que l'on trie récursivement et que l'on fusionne ensuite. Bien entendu les appels récursifs se terminent lorsque le tableau est de taille 1 auquel cas il est tout trié.

La procédure de tri par fusion d'un tableau est simple.

*Méthode : Il nous faut utiliser une procédure de fusion de 2 sous-tableaux consécutifs d'un même tableau. Le 1$^{er}$ sous tableau commence de l'indice min à m et le 2$^{ème}$ de l'indice m + 1 à max*

**Exercice 6.20 :**

Écrire un algorithme qui calcule le *zéro* d'une fonction *f(x)* sur l'intervalle [*a*; *b*], avec une précision *epsilon*. La fonction f et les réels *a*, *b* et *epsilon* sont donnés. Soit *f(x)* une fonction continue sur l'intervalle [*a*; *b*], où elle ne s'annule qu'une seule et unique fois. Pour trouver ce *zéro*, on procède par dichotomie, c'est-à-dire que l'on divise l'intervalle de recherche par *deux* à chaque étape. Soit *m* le milieu de [*a*; *b*]. Si *f(m)* et *f(a)* sont de même signe, le *zéro* recherché est dans l'intervalle [*m*; *b*], sinon il est dans l'intervalle [*a*; *m*].

**Exercice 6.21 :**

Calculer le maximum d'un tableau de manière récursive.

**Exercice 6.22 :** Fractal : courbes de Koch

La courbe de Koch est un fractal dont le détail varie en fonction d'un niveau donné *n*.

La figure ci-dessous à droite présente la courbe de Koch de niveau *2*. En fait, dessiner la courbe de Koch de niveau *n* entre deux points $A(x_a, y_a)$ et $E(x_e, y_e)$ revient à :

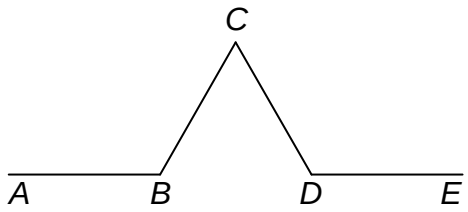
– dessiner le segment de droite [*A*, *E*] si *n = 0*

– dessiner *4* segments de Koch de niveau *n = 1* entre les points *A*, *B*, *C*, *D* et *E* ([*A*, *B*], [*B*, *C*], [*C*, *D*] et [*D*, *E*]) telles que :

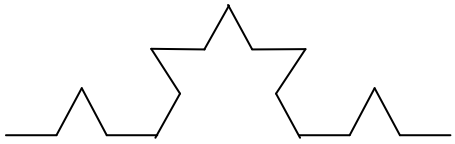
– *AB = AE/3*

– *ED = EA/3*

– Les points *B*, *C* et *D* forment un triangle équilatéral.




*n = 1*



*n = 2*

**Exercice 6.23**

On considère la fonction récursive f sur l'ensemble des entiers positifs donnée par :

*f(1) = 0*

*f(2) = 1*

*f(n) = f(n-f(n-1)) + f(n-1 - f(n-2));*

1. calculer les valeurs de *f(n)* pour la valeur *n = 3*.
2. Ecrire une version itérative du calcul de *f(n)* qui pour obtenir ce résultat.