1 Affectation, séquence, instructions conditionnelles

1. Indiquez les valeurs de chacune des variables `a, b, c, d` à la fin de chacune des lignes de l'algorithme suivant :

```
    Algorithme : ...
    Variable : a : Nombre, b : Nombre, c : Nombre, d : Booléen
0
1       a := 10 ;
2       a := a + 5 ;
3       b := 2 * a - 1;
4       a := b - a;
5       d := (a >b);
6       si d alors c := a + b;
        fin si;
7       b := c + a ;
    fin algorithme
```

2. Les variables `a,b` sont supposées déclarées de type `Nombre`. Indiquez leur valeur après la **séquence** d'instructions :
   a := 5 ; b := 9 ; a := (a + b) ; b := (a - b) ; a := (a - b) ;
3. Les variables `FA, FB, FC` représentent les fortunes possédées par les personnes A, B et C. Ce jour là :
   — Le matin, A, qui a des loisirs, va porter à B la moitié de sa fortune.
   — À midi, B, qui ne veut pas être en reste, va porter à C la moitié de sa –nouvelle– fortune
   — Le soir venu, C, galvanisé par l'exemple, se précipite chez A pour lui donner la moitié de sa –nouvelle– fortune
   Écrire une séquence d'instructions qui calcule les fortunes respectives de A, B et C le soir venu.
4. Les variables `a,b,t` sont supposées déclarées de même type non nécessairement `Nombre` (`Booléen` par exemple). On suppose que les variables `a` et `b` ont une valeur. Écrire une **séquence** d'instructions dont l'effet est d'échanger les valeurs de `a` et `b`.
5. Les variables `exam,cc,note, noteFinale` sont supposées déclarées de type `Nombre`. Les variables `exam,cc` sont initialisées. Les 2 **séquences** d'instructions ci–dessous sont-elles équivalentes ? Pour répondre vous choisirez diverses valeurs des variables `exam,cc`.

```
si exam > cc
alors note := exam
finsi;
si cc > exam
alors note := ((exam + cc)/2)
finsi;
notefinale := (note + 1) ;
```

```
si exam > cc
alors note := exam
sinon note := ((exam + cc)/2)
finsi;
notefinale := (note + 1) ;
```

6. Mêmes hypothèses et question pour les 2 **séquences** d'instructions ci–dessous.

```
si exam > cc
alors note := exam
finsi;
si exam <= cc
alors note := ((exam + cc)/2)
finsi;
notefinale := (note + 1) ;
```

```
si exam > cc
alors note := exam
sinon note := ((exam + cc)/2)
finsi;
notefinale := (note + 1) ;
```

7. (*) Plus généralement, soit `b1` une expression booléenne, `i1` et `i2` des instructions.
   Montrez que les séquences d'instructions suivantes **ne sont pas** équivalentes.

```
si b1
alors i1
finsi;
si non(b1)
alors i2
finsi;
```

```
si b1
alors i1
sinon i2
finsi;
```

8. Écrire un algorithme calculant le montant d'une commande connaissant le nombre d'articles commandés et le prix unitaire d'un article. Deux remises successives peuvent être accordées.

   La première est une réduction de 10% lorsque le nombre d'articles dépasse 15.

   Après avoir appliqué la première remise, si le montant de la commande est supérieur à 200 € un rabais de 15 € est accordé. Sinon s'il est supérieur à 150 € on applique une réduction de 10 €.

9. Écrire un algorithme calculant le nombre de solutions réelles d'une équation du second degré $a.X^2 + b.X + c = 0$. Les données sont les coefficients a, b et c. On admettra que a et b ne peuvent pas être nuls tous les 2, c'est à dire que l'équation est de degré 2 ou 1, mais pas 0.

   Dans ce cas, on rappelle que le nombre de solutions réelles (valeurs réelles de X vérifiant $a.X^2 + b.X + c = 0$) est 0 ou 1 ou 2 selon les valeurs des coefficients :

   — si l'équation est du premier degré ($a = 0$), il vaut 1

   — si l'équation est du second degré, il dépend du signe du discriminant ($b^2 - 4.a.c$).

2 Itérations

1. Soit `f` l'algorithme ci–dessous. Quel est le résultat de `f(3)`, de `f(5)` ? Que calcule l'algorithme `f` ? Que se passe-t-il lorsque l'argument est inférieur ou égal à 2 ? En particulier que valent `f(2)` et `f(1)` ?

   Algorithme : f
   Données : `n` : Nombre ; `n` est un entier
   Résultat : ? ? ?
   Variable : som : Nombre

   som := 10 ;
   pour `k de 2 à n` **faire**
   som :=som+k*k ;
   fin pour;
   Le résultat est : `som`
   fin algorithme

2. Écrivez un algorithme qui calcule la somme $2 + 4 + 6 + ... + 2.n$ en fonction de son paramètre n.

3. Écrire un algorithme calculant en fonction de l'entier `n` la somme $\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \cdots + \frac{n}{n+1}$.

4. **Généralisation :** on suppose que `f` de signature **f** : $Nombre \longrightarrow Nombre$ est une fonction définie (prédéfinie ou définie par un algorithme). Complétez l'algorithme :

   Algorithme : sommePart
   Données : `a` : Nombre, `b` : Nombre ;
   $a, b \in \mathbb{N}$
   Résultat : Nombre, $\sum_{k=a}^{b} f(k)$

5. Écrivez un algorithme qui calcule la somme des diviseurs d'un entier naturel.

6. Écrivez un algorithme calculant la factorielle du nombre entier n.

7. Écrivez un algorithme `puissance` qui étant donné un nombre x et un entier naturel n calcule x^n, par multiplications successives.
8. Écrivez deux algorithmes calculant la même fonction `sommePuis` :

   Algorithme : sommePuis
   Données : `x` : Nombre, `n` : Nombre ; `n`$\in \mathbb{N}$
   Résultat : Nombre, $1 + x + x^2 + \cdots + x^n$

   — Pour le premier algorithme vous utiliserez la fonction `puissance` de l'exercice précédent.
   — Pour le second algorithme vous n'utiliserez pas la fonction `puissance`. Pour cela vous introduirez une nouvelle variable pour mémoriser les puissances successives de x.
9. Réécrivez les algorithmes des questions 2.2 et 2.3 en utilisant l'itération `Tant que`.

   Généralisez : écrivez une séquence d'instructions utilisant l'itération `Tant que` équivalente à l'instruction :

   pour `i de a à b` **faire**
   Inst
   fin pour;
10. On rappelle que la partie entière d'un réel x est le plus grand entier inférieur ou égal x. C'est, par conséquent, l'unique entier n tel que $n \leqslant x < n + 1$. On note souvent $\lfloor x \rfloor$ la partie entière de x, et la fonction correspondante est souvent nommée `floor`. Écrivez la fonction `floorPositif` qui est la restriction de la fonction $floor$ avec un paramètre réel positif.

    Algorithme : floorPositif
    Données : `x` : Nombre ; `x` est un réel positif
    Résultat : Nombre, $\lfloor x \rfloor$
11. De manière analogue on définit la partie entière *supérieure* $\textbf{ceil} : \begin{array}{ccc} \mathbb{R} & \longrightarrow & \mathbb{N} \\ x & \longmapsto & \lceil x \rceil \end{array}$

    telle que $n = ceil(x) \Leftrightarrow n - 1 < x \leqslant n$. Écrivez la fonction `ceilPositif`.
12. On rappelle qu'un nombre premier est un entier strictement supérieur à 1 qui n'a d'autre diviseur que 1 et lui-même. Écrire la fonction **estPremier** : $\mathbb{N} - \{0, 1\} \longrightarrow$ *Booléen* qui teste si son paramètre est un nombre premier.
13. Écrivez un algorithme calculant le $n^{ième}$ nombre multiple de 2 ou de 3 mais pas de 6. Les nombres multiples soit de 2 soit de 3 sont : 2, 3, 4, 8, 9, 10, 14, 15, 16, 20, ... Ainsi le $5^{ième}$ multiple soit de 3 soit de 2 est le nombre 9, le $9^{ième}$ est le nombre 20.
14. (*) Un autre algorithme calculant le PGCD de 2 nombres a et b est :
    (a) on calcule les listes `lDiva` et `lDivb` des diviseurs de a et b .
    (b) on calcule le plus grand nombre appartenant aux 2 listes `lDiva` et `lDivb`.

    Écrivez l'algorithme qui opère selon ce schéma. À vous de spécifier et d'écrire
    — un algorithme calculant la liste des diviseurs d'un nombre
    — un algorithme recherchant le plus grand nombre commun à 2 listes de nombres.

3 C/C++

1. Traduisez en C/C++ l'algorithme `f` de la question 2.1
2. Traduisez en C/C++ l'algorithme de la question 2.5 calculant la somme des diviseurs d'un entier naturel.
3. Écrivez une fonction C/C++ testant si un nombre est un nombre premier.

4 Tableaux

1. Indiquez les valeurs de la variable `tab` après chaque instruction de la séquence suivante. En cas d'erreur lors d'une instruction (de typage ou lors de son exécution), vous passerez à l'instruction suivante en ignorant l'instruction erronée.

```
   Variable : tab : Tableau de 5 Nombres
 1 tab[3] := 2;
 2 tab[4] := 0;
 3 tab[0] := tab[3]+1;
 4 si tab[0]>tab[4] alors tab[0] :=tab[0]+1;
   fin si;
 5 tab[2] := 2*tab[1];
 6 tab[2] := tab[0] + tab[3] ;
 7 tab[5] := tab[2] ;
 8 si tab[0]=4 alors tab[1] := 1 ;
   sinon  tab[2] := tab[4];
    fin si
```

2. Écrivez des algorithmes qui étant un tableau `tab` de nombres calculent les résultats suivants :
   (a) la moyenne des éléments du tableau `tab`
   (b) le nombre d'éléments de `tab` qui ont une valeur strictement inférieure à un nombre `e`. Traduisez cet algorithme en une fonction C/C++.
   (c) la plus grande valeur des éléments du tableau `tab`

3. Écrivez un algorithme qui pour un entier $n > 0$ donne en résultat le tableau de n nombres :

0	*1*	*2*	*3*		*n-1*
0	1	4	9	...	$(n-1)^2$

4. Écrivez un algorithme qui étant donné un tableau `tab` de nombres, calcule le tableau de nombres de même taille et dont la séquence des valeurs est la séquence en ordre inverse des valeurs de `tab`. Par exemple si la donnée est le tableau | 3 | 1 | 9 | 8 | 1 |, le résultat est le tableau | 1 | 8 | 9 | 1 | 3 |. Vous écrirez ensuite la traduction en C/C++ de votre algorithme.

5. Écrivez un algorithme calculant la somme de 2 tableaux de nombres de même taille : pour des données `t1` et `t2` le résultat est le tableau `t3` de même taille que `t1` et `t2`, tel que $\forall i \in [0, \text{taille(t3)-1}], t3[i] = t1[i] + t2[i]$.

6. Écrivez deux algorithmes qui pour un entier $n > 0$ donnent en résultat le tableau de n nombres :

0	*1*	*2*	*3*		*n-1*
1	2	4	8	...	2^{n-1}

Le premier algorithme que vous écrirez utilise l'algorithme `puissance` de la question 2.7, le second ne l'utilise pas.

7. Complétez l'algorithme ci–dessous :

Algorithme : nbOcc
Données : `e` : Nombre, `tab` : Tableau de Nombre
Résultat : Nombre, le nombre d'éléments de `tab` ayant la valeur `e`

8. Écrivez un algorithme qui étant donné un nombre `e` et un tableau de nombres `tab` calcule la liste des indices `i` de tab tels que `tab[i]=e`.

9. Écrivez un algorithme testant si la séquence des valeurs des éléments d'un tableau de nombre est croissante (le résultat est le booléen `true` si la séquence est croissante, `false` sinon).

10. On cherche à vérifier si un tableau est *injectif*, c'est à dire s'il ne contient pas d'éléments distincts ayant même valeur.

Par exemple $\begin{array}{|c|c|c|c|c|}\hline 3 & 1 & 9 & 8 & 1 \\ \hline\end{array}$ n'est pas injectif, alors que $\begin{array}{|c|c|c|c|c|}\hline 3 & 1 & 9 & 8 & 4 \\ \hline\end{array}$ l'est.
Votre algorithme utilisera l'algorithme `nbOcc`.

11. La valeur médiane d'un tableau de nombres est la valeur qui serait située au milieu du tableau si le tableau était trié. Plus précisément la valeur médiane d'un tableau injectif `tab` est la valeur `m` de `tab` telle que le nombre de valeurs dans `tab` inférieures strictement à `m` est $\lfloor \frac{n}{2} \rfloor$. Écrivez un algorithme qui étant donné un tableau de nombres, injectif mais non nécessairement trié, calcule sa valeur médiane.
   Par exemple la valeur médiane du tableau $\begin{array}{|c|c|c|c|c|c|c|}\hline 9 & 3 & 5 & 2 & 8 & 1 & 7 \\ \hline\end{array}$ est 5.
12. Écrivez un nouvel algorithme vérifiant si un tableau est `injectif`. Cette nouvelle version ne doit pas utiliser l'algorithme `nbOcc`. Traduisez votre algorithme en C/C++.
13. Écrivez un algorithme qui calcule la deuxième valeur la plus petite parmi l'ensemble des valeurs des éléments d'un tableau. Pour simplifier on supposera que le tableau est injectif.
14. Écrire un algorithme `fusion` qui, étant donné deux tableaux de nombres `t1, t2` triés croissant au sens large ($\forall i, j, 1 \leqslant i \leqslant j \leqslant taille(T),\ t[i] \leqslant t[j]$) renvoie le tableau trié croissant fusionnant les éléments de `t1` et `t2`.
15. (*) Soit `tab` un tableau de nombres. Un sous–tableau de `tab` est une séquence d'éléments de `tab` : tab[k], tab[k+1] ... tab[h] tel que $0 \leqslant k \leqslant h \leqslant$ taille(tab). On appelle somme d'un sous–tableau la somme de ses éléments. Écrivez l'algorithme qui renvoie la somme maximale des sous–tableaux du tableau passé en paramètre. Exemple : la somme maximale des sous–tableaux du tableau ci-dessous est 190, elle est atteinte pour le sous–tableau de bornes 3 et 11.

0	*1*	*2*	**3**	*4*	*5*	*6*	*7*	*8*	*9*	*10*	**11**	*12*	*13*	*14*	*15*
5	31	-41	59	26	-53	58	97	-93	-23	84	35	-98	-80	-72	-85

16. L'algorithme de recherche de nombres premiers par la technique du crible. La donnée est un entier `Nmax`. On cherche tous les nombres premiers inférieurs à `Nmax`. Le principe du crible d'Erathostène est le suivant :
   — Écrire tous les entiers de 2 à `Nmax`.
   — Rayer tous les multiples stricts de 2, puis tous les multiples stricts de 3, ...
   — De façon générale, après avoir rayé les multiples stricts de `p`, on recommence avec le plus petit nombre non rayé strictement supérieur à `p`.
   — À la fin du procédé, les nombres non rayés sont premiers.
   L'algorithme se schématise ainsi :

```
    p := 2;
    pmax := ???;
    tant que p ≤ pmax faire
1       Rayer les multiples stricts de p inférieurs à Nmax;
        Mettre à jour p
    fin tq;
  fin algorithme
```

   (a) Juste avant d'exécuter l'instruction 1, quel est le premier nombre non premier qui n'est pas encore rayé ?
   (b) Lorsqu'on raye les multiples stricts de `p`, à partir de quel multiple faut-il commencer ?
   (c) Précisez la valeur de `pmax` en fonction de `Nmax`.
   (d) Écrire l'algorithme `crible` qui a pour donnée le nombre entier `Nmax` et qui renvoie un tableau de booléens de taille `Nmax`, dont l'élément de rang `i` a pour valeur `true` si et seulement si `i` est premier.
   (e) Écrire enfin la fonction `listePremier` qui a pour donnée l'entier `Nmax` et qui renvoie la liste des entiers premiers inférieurs ou égaux à `Nmax`.

17. (*) **Un autre algorithme de tri.** Le tri par sélection consiste pour trier un tableau `tab` de `n` nombres :
    - à rechercher le plus grand élément du tableau et à échanger sa valeur avec celle du dernier élément (`tab[n-1]`)
    - à rechercher parmi les n-1 premiers éléments de `tab`, l'élément ayant la plus grande valeur et à échanger la valeur de cet élément avec celle de `tab[n-2]`
    - ...
    - à la $k^{ième}$ étape, on recherche le plus grand élément parmi les éléments `tab[0], tab[1], ..., tab[n-k]` et on échange la valeur de cet élément avec celle de `tab[n-k]`

    Écrivez l'algorithme de tri par sélection.

18. Dans ce qui suit on représentera un nombre entier strictement positif par un tableau dont les éléments sont les chiffres (nombres entre 0 et 9) de sa représentation en base 10.

    Par exemple le nombre 509 est représenté par le tableau | 5 | 0 | 9 |

    On impose que le 1^{er} élément du tableau représentant un nombre est différent de 0. Ainsi | 0 | 5 | 0 | 9 | n'est pas une représentation correcte de 509.

    (a) Écrivez $\boxed{\text{en langage C/C++}}$ une fonction `representeNombre` qui étant donné un tableau d'entiers `T` vérifie si `T` représente un nombre c'est à dire si tous ses éléments sont compris entre 0 et 9 et si son 1^{er} élément est différent de 0.

    Exemples :

    si `Tn` est le tableau | 2 | 1 | 11 | 9 | `representeNombre(Tn)=false`
    si `Tn` est le tableau | 8 | 0 | 1 | `representeNombre(Tn)=true`.
    si `Tn` est le tableau | 0 | 8 | 1 | `representeNombre(Tn)=false`.

    (b) Écrivez $\boxed{\text{en langage C/C++}}$ une fonction `estInferieur` qui étant donné 2 tableaux `T1` et `T2` représentant des nombres vérifie si le nombre représenté par `T1` est inférieur ou égal à celui représenté par `T2`.

    Exemples :

    si `Ta` est le tableau | 7 | 3 | 5 | 1 |, `Tb` le tableau | 9 | 5 | 1 | et `Tc` le tableau | 9 | 5 | 3 |
    on a `estInferieur(Ta,Tb)=false, estInferieur(Tb,Tc)=true`

    (c) Écrivez $\boxed{\text{en langage d'algorithmes}}$ un algorithme `plusUn` qui étant un tableau `Tn` représentant un nombre `n` donne comme résultat le tableau représentant le nombre `n+1`. Pour simplifier on supposera que le tableau donné ne contient pas que des 9.

    Exemples :

    si `Tn` est le tableau | 2 | 1 | 1 | 7 | `plusUn(Tn)` est le tableau | 2 | 1 | 1 | 8 |
    si `Tn` est le tableau | 2 | 8 | 9 | 9 | `plusUn(Tn)` est le tableau | 2 | 9 | 0 | 0 |.