1 Expressions

On suppose définies les variables et algorithmes suivants :

Algorithme : proche ?
Données : $a, b \in \mathbb{N}$.
Résultat : Un booléen qui teste la « proximité » de a et b
début
| /*abs est la fonction numérique primitive valeur absolue */
| **retourner** `abs(b-a) < 2`
fin algorithme
x,y : Entier ;
r : Booléen ;
z : Réel ;

1. Quelles sont les valeurs des expressions suivantes :

```
2 + (3 * (7 - 5))                                    2 + (3 * (x - y))
2 + (3 * true)                                       (2 < 3 ) et false
true ou false                                        true et false
non ( true et false )                                proche?( 2, 7)
proche?( 1,1 ) et (proche?( 2, 3 ) et proche?( 1, 3 ))  proche?(1, 2, 3)
proche?(1, 2) ou proche?(1, 2, 3)                    (x*x) + y
```

2. On suppose qu'après les définitions initiales identiques, on exécute la suite d'instructions suivantes :

```
x := 5 ; y := x mod 2 ; r := false ; z := 3.14159 ;
```

Quelles sont alors les valeurs des expressions suivantes :

```
( x < y ) et r                  2 + (3 * (x - y))
sin(z)                          proche?(y, 2)
(a*x) + b                       (x + (y + 1) ) quo 2
```

2 Affectation, séquence, instructions conditionnelles

1. Les variables `a,b` sont supposées déclarées de type `Entier`. Indiquez leur valeur après la **séquence** d'instructions.

```
a := 5 ; b := 9 ;
a := a + b ;
b := a - b ; a := a - b ;
```

2. Peut–on exécuter exactement la même séquence pour des variables de type `Booléen` ?
3. A,B et C possèdent respectivement 8000, 4000 et 4000 euros. 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 un algorithme qui calcule les fortunes respectives de A, B et C le soir venu et renvoie son résultat sous forme d'un triplet.
4. Les variables `a,b,t` sont supposées déclarées de même type non nécessairement `Entier`. `a,b` sont initialisées. 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 `Réel`. `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. Les variables `exam,cc,note, noteFinale` sont supposées déclarées de type `Réel`. `exam,cc` sont initialisées. Les 2 **séquences** d'instructions ci–dessous sont-elles équivalentes ?

```
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` et `b2` deux expressions booléennes mutuellement exclusives. C'est à dire telles que, si la valeur de l'une est `true` alors la valeur de l'autre est `false`, et vice versa. Soient `i1, i2` et `i3` trois instructions.

   Montrez que les séquences d'instructions suivantes **ne sont pas** équivalentes.

```
si b1
  alors i1
finsi;
si b2
  alors i2
finsi;
i3;
```

```
si b1
  alors i1
  sinon i2
finsi;
i3;
```

8. Dans la suite d'instructions suivantes, la fonction f calcule bizarrement la moyenne de ses paramètres. Peu importe.

   Quelles sonts les valeurs des variables `c,x,y,a,b` après exécution de l'instruction **1** ? Et après exécution de l'instruction **2** ?

Algorithme : f
Données : $a, b \in \mathbb{R}$
Résultat : Le réel moyenne de a et b
a1,b1 : Réel ;
début
  a1 := a + 1 ;
  b1 := b - 1 ;
  retourner (a1+b1)/2
fin algorithme
a,b,c,x,y : Réel;
x := 5 ; y := 11 ;
1 c := f(x,y) ;
a := 1 ; b := 5 ;
2 c := f(a,b) ;

9. La règle du max à l'UFR, ou comment calculer la note finale au module : « Soit `exam,cc` vos 2 notes (sur 20) au contrôle terminal et au contrôle continu. Soit coeffExam et coeffCc les poids respectifs de ces notes dans le calcul final. On calcule d'abord la note sur 20 qui est la pondération de `exam,cc` avec leurs poids. Votre note finale sur 20 est le max entre la note à l'examen et la note ainsi calculée. »

   (a) Écrire une fonction qui prend en paramètre les deux notes et les deux coefficients et qui renvoie la note finale.

   (b) Nos barêmes sont les suivants : L'examen compte coefficient 2 et le contrôle continu coefficient 1. Un étudiant a obtenu 12 à l'examen et 15 au contrôle continu. Écrire l'instruction qui permet de calculer sa note finale au module.

10. On dispose de la fonction **max :** $\mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ qui renvoie le plus grand de ses deux paramètres.

    Utilisez `max` pour écrire une fonction **max-3 :** $\mathbb{N} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ qui renvoie le plus grand de ses 3 paramètres.

    Écrivez une deuxième version de `max-3` qui n'utilise pas `max`.

3 Itérations simples

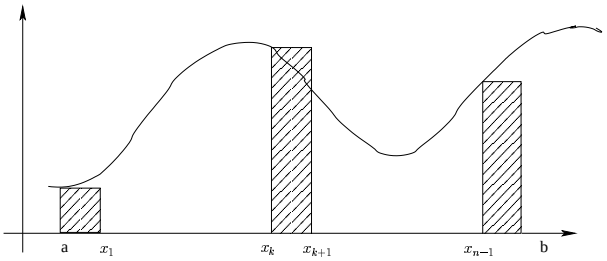
1. En utilisant une itération **pour** vous écrirez la fonction qui renvoie la somme $1 + 2 + ... + n$ en fonction de son paramètre n.

   Même question en utilisant cette fois une itération **tant que**.

2. `instr` est une instruction. `p` est une variable entière initialisée à une valeur supposée strictement positive. Soit le code général d'une itération `pour` générale. Écrivez le code équivalent en utilisant l'itération `tantque`.

```
k,a,b,p : Entier;
... ;
/*on suppose a,b et p initialisées          */
pour k de a à b par pas de p faire
| instr ;
finpour
```

3. Même question en supposant `p` est une variable entière initialisée à une valeur supposée strictement négative.
4. Écrivez la fonction `signe` qui renvoie `+1`,`0` ou `-1` selon que son argument entier est positif, nul ou négatif. Utilisez ensuite cette fonction pour généraliser la simulation d'une itération `pour` avec un pas quelconque, non nul évidemment, par une itération `tantque`.
5. On généralise la question **(1)** : Sommes partielles de séries.
   On définit : **sommePart** : $(\mathbb{N} \to \mathbb{R}) \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R}$
   $$(f, a, b) \longmapsto \sum_{k=a}^{b} f(k)$$
   Par exemple la somme des inverses des n premiers entiers strictement positifs est :
   $1 + 2 + ... + n = \sum_{k=1}^{n} \frac{1}{k}$ elle s'obtient donc par `sommePart(`$x \mapsto \frac{1}{x}$`,1,n)`.
   On pourrait aussi procéder en deux temps : d'abord définir la fonction `f` $:x \mapsto \frac{1}{x}$, ensuite faire l'appel à `sommePart(f,1,n)`. Cette manière de faire est « plus lourde », et la définition de la variable f ne s'impose pas (elle n'apporte rien).
   Écrivez la fonction `sommePart` qui a 3 paramètres.
6. 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** : $\mathbb{R} \longrightarrow \mathbb{N}$, $x \longmapsto \lfloor x \rfloor$
   (a) Écrivez la fonction `floorPositif` qui est la restriction de la fonction *floor* avec un paramètre réeel positif.
   (b) Soit alors x un réel négatif. Exprimez $floor(x)$ en fonction de $floorPositif(-x)$.
   (c) Déduire de ce qui précède l'écriture de l'algorithme `floor` dont le paramètre est un réeel quelconque.
   (d) De manière analogue on définit la partie entière *par valeurs supérieures* **ceil** : $\mathbb{R} \longrightarrow \mathbb{N}$, $x \longmapsto \lceil x \rceil$ telle que
   $n = ceil(x) \Leftrightarrow n - 1 < x \leqslant n$
   Écrivez la fonction `ceil`
7. On rappelle que `a mod b` renvoie le reste de la division entière de `a` par `b`. `a,b` sont des entiers strictement positifs.
   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 **premier ? :** $\mathbb{N} - \{0; 1\} \longrightarrow$ *Boolen* qui teste si son paramètre est un nombre premier.
8. On se donne $a < b$ deux réels, une fonction f continue sur $[a; b]$, et n un entier non nul. On découpe $[a; b]$ en n intervalles de même longueur. Le découpage définit une suite $x_0 = a, x_1, ..., x_n = b$ de $n+1$ points. On considère les n pavés du plan de base $x_{k+1} - x_k$ et de hauteur $f(x_k)$, pour $k \in [0..n-1]$. La somme des surfaces de ces pavés est une approximation de $\int_a^b f(x)dx$



Écrivez l'algorithme `approcheIntegrale` dont les données sont, dans l'ordre : la fonction f, les bornes réelles a, b, l'entier n, et qui renvoie le réel qui approche l'intégrale.

4 Tableaux

Dans toutes les questions, si un tableau est une donnée d'un algorithme, on suppose ce tableau initialisé au sens : tous les éléments du tableau ont une valeur.

1. Dans les questions qui suivent, on vous demande d'écrire un algorithme dont la donnée est un tableau d'entiers initialisé.
   (a) Algorithme qui renvoie la valeur maximum du tableau.
   (b) Algorithme qui renvoie la deuxième valeur dans l'ordre strictement décroissant. On supposera que le tableau contient au moins deux valeurs distinctes.
   (c) Algorithme qui renvoie un booléen : `true` si et seulement si la suite des éléments est strictement croissante.
   (d) Soit T un tableau $1..L$, un sous–tableau de T est la suite des éléments $T[k]$ pour $i \leqslant k \leqslant j$. i, j sont les bornes du sous–tableau qui doivent être entre 1 et $taille(T)$.
   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.
2. Écrire un algorithme `nbOcc` qui, étant donné un tableau et une valeur de même type que les éléments du tableau, renvoie le nombre d'*occurrences* de la valeur dans le tableau.

   Exemple 4.0.1 : Si T est le tableau d'entiers ayant pour éléments : $2, 3, 7, 2, 7, 2, 9, 2, 2$. L'entier 2 a 5 occurrences dans T (il apparait dans T comme élément d'indice 1,4,6,8,9). L'entier 4 n'a aucune occurrence dans T. □

3. Utilisez l'algorithme `nbOcc` pour écrire un algorithme `injectif ?` qui a pour donnée un tableau T et pour résultat un booléen qui vaut `true` si et seulement si les éléments de T sont 2 à 2 distincts.
4. Écrire un algorithme `fusion` qui, étant donné deux tableaux d'entiers T_1, T_2 triés croissant au sens large $(1 \leqslant i \leqslant j \leqslant taille(T) \Rightarrow T[i] \leqslant T[j])$ renvoie le tableau trié croissant fusionnant les éléments de T_1 et T_2.
5. Une chaîne de caractères est une suite de caractères. Par exemple un chromosome est une suite de bases, chaque base est représentée par la lettre "A" ou "C" ou "G" ou "T". On cherche souvent à apparier des chaînes entre elles.
   Dans cet exercice, nos chaînes de caractères seront représentées par des tableaux de caractères. Par exemple la chaîne `ch = "ACAGAT"` sera représentée par le tableau défini comme suit :
   ch : **array** 1..6 **of** Character ;
   ch[1] := "A" ; ch[2] := "C" ; ch[3] := "A" ; ch[4] := "G" ; ch[5] := "A" ; ch[6] := "T" ;
   La longueur de la chaîne `ch` est 6, c'est la taille du tableau `ch` qui la représente. La chaîne vide `""` se représente par un tableau de taille 0 `array(1..0)`

   Définition 4.0.1 Soit `ch1` et `ch2` deux chaînes de caractères.
   – `ch1` est dite *préfixe* de `ch2` si pour chaque caractère de `ch1`, il est au même rang dans `ch1` et `ch2`.
     Exemples : `""` est préfixe de toute chaîne, `"ACA"` est préfixe de `"ACAGAT"`. Mais `"ACG"` n'est pas préfixe de `"ACAGAT"`.
   – `ch1` est dite *facteur* de `ch2` si il existe un rang r tel que chacun des caractères de `ch1` sont consécutifs dans `ch2` à pertir du rang r.
     Exemples : `"AGA"` est facteur de `"ACAGATAGA"`, car ses 3 caractères sont conséctifs à partir du rand 3 dans `"ACAGATAGA"`. Mais `"AGTA"` n'est pas préfixe de `"ACAGATAGA"`.
   – `ch1` est dite *sous-mot* de `ch2` si chacun des caractères de `ch1` apparait dans `ch2` dans le même ordre mais pas nécessairement consécutivement.
     Exemples : `"AGTA"` est sous-mot de `"ACAGATAGA"`. Il n'est cependant pas sous–mot de `"ACAATAGA"`
   □

   Écrivez les algorithmes `préfixe`, `facteur`, `sous-mot` qui ont chacun comme paramètres deux chaînes `ch1` et `ch2` et qui renvoie le booléen indiquant que `ch1` est respectivement préfixe, facteur ou sous–mot de `ch2`.
6. Algorithme de recherche de nombre 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 p' le premier nombre non rayé de la suite.
   – À la fin du procédé, les nombres non rayés sont premiers.

L'algorithme se schématise ainsi :

début
  p := 2 ;
  /*Préciser la valeur de pmax */
1 **tant que** $p \leqslant pmax$ **faire**
2 rayer les multiples stricts de p ;
  mettre à jour p;
fin tq
fin algorithme

(a) Donner un invariant pertinent pour l'itération **1**.

(b) Juste avant d'exécuter l'instruction **2**, quelle est le premier nombre non premier qui n'est pas encore rayé ?

(c) Lorsqu'on raye les multiples stricts de p, à partir duquel faut–il commencer ?

(d) Précisez la valeur de $pmax$ en fonction de $Nmax$.

(e) Écrire la fonction `crible` qui a pour donnée l'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.

(f) Écrire enfin la fonction `TablePremier` qui a pour donnée l'entier `Nmax` et qui renvoie le vecteur formé des entiers premiers inférieurs ou égaux à `Nmax`.

5 Maple

1. On déja écrit un certain nombre d'algorithmes. Vous allez effectuer la **traduction** en Maple des algorithmes des questions :
   (a) Algorithme **(9 page 2)** « La règle du max à l'UFR »
   (b) Algorithme **(1 page 2)** « Itération pour et tant que calculant la somme des n premiers entiers »
   (c) Algorithme **(5 page 3)** « Sommes partielles de séries ».
   (d) 4 algorithmes de base qui manipulent des tableaux **(1 page 4)**
2. Vous devez écrire en MAPLE 4 procédures. Toutes les 4 ont un tableau initialisé d'entiers en paramètre et un booléen en résultat.
   (a) Écrivez une procédure qui vérifie si un tableau T contient un entier compris au sens large entre 0 et 20 ($\exists i \in [1..taille(T)], 0 \leqslant T[i] \leqslant 20$).
   (b) Écrivez une procédure vérifiant si tous les éléments d'un tableau sont compris au sens large entre 0 et 20.
   (c) Écrivez une autre procédure qui vérifie si tous les éléments d'un tableau d'entiers sont pairs. Pour vérifier si un entier est pair, on peut utiliser la fonction MAPLE `type(k,even)`, qui vaut `true` si et seulement si k est pair.
   (d) Généralisez : on passe en paramètre la propriété P que doivent satisfaire tous les éléments du tableau. Cette propriété est une fonction (ou procédure) qui étant donné un entier retourne un booléen
   P : $\mathbb{N} \longrightarrow$ *Booléen* .
   En MAPLE, ce paramètre est de type `procedure`.
3. (a) Donnez une procédure qui étant donné un tableau d'entiers T, renvoie un tableau S tel que $\forall i \in [1..taille(T)], S[i] = 2 * T[i]$
   (b) Généralisez la procédure précédente en passant en paramètre un tableau d'entiers T et une fonction **f :** $\mathbb{N} \longrightarrow \mathbb{N}$ et renvoie un tableau S tel que $\forall i \in [1..taille(T)], S[i] = f(T[i])$
4. On s'intéresse à l'écriture en base k d'un entier naturel n. Cette écriture sera représentée par un tableau T de chiffres ($\in [0..k-1]$), $T[1]$ étant le chiffre de poids faible (des unités), $T[2]$ le second chiffre (chiffre des dizaines si $k = 10$)... Plus précisément $n = \sum_{i=1}^{p} T[i] \times k^{i-1}$ où p est le nombre de chiffres de l'écriture en base k de n.
   Par exemple les écritures de 48 sont, en base 10 :

1	2
8	4

   , en base 3 :

1	2	3	4
0	1	2	1

   , en base 5 :

1	2	3
3	4	1

   .
   Vous aurez besoin de la division euclidienne. En MAPLE les fonctions **irem, iquo :** $\mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ fournissent respectivement le reste et le quotient de la division entière.
   (a) Comment calculer le nombre de chiffres de l'écriture décimale d'un entier n ? Donnez la procédure MAPLE correspondante.
   (b) Quelle expression permet de calculer le chiffre des unités d'un entier n ? Comment obtenir le chiffre des dizaines, ...
   En déduire une procédure MAPLE qui prend en entrée un entier naturel n et renvoie le tableau correspondant à l'écriture décimale de n.
   (c) Généralisez la procédure précédente en passant en paramètre la base k en plus de l'entier n.
5. Donnez une procédure MAPLE qui insère un élément dans un tableau trié. Étant donné T un tableau trié d'entiers et un entier k, la procédure renvoie un tableau trié S contenant les éléments de T, plus l'entier k (même si k figure déjà dans T).
6. Soit T un tableau trié d'entiers. Pour vérifier si T contient un élément donné x vous avez vu en cours un algorithme de recherche dichotomique.
   Si x apparaît plusieurs fois dans T on voudrait adapter l'algorithme du cours pour obtenir le plus petit indice i tel que $T[i] = x$. Par exemple rechDicho(

1	2	3	3	5	6	7
2	2	4	5	6	6	9

   , 6) doit renvoyer 5.

Algorithme : rechDicho
Données : T un tableau trié d'entiers ; x un entier
Résultat : un entier qui est 0 si $x \notin T$ et qui est le plus petit indice de x dans T sinon

L'algorithme recherché utilise 2 indices g et d et doit vérifier l'invariant suivant :

$$\forall i \in [1..g-1], T[i] < x \text{ et } \forall i \in [d+1..taille(T)], T[i] \geqslant x$$

Il s'agit à chaque itération de réduire strictement l'intervalle $[g, d]$ en calculant l'indice m milieu de cet intervalle et en comparant $T[m]$ à x.

(a) L'itération commence avec $g = 1$ et $d = taille(T)$ et s'arrête lorsque l'intervalle est vide ($g = d + 1$). Écrivez cette itération.

(b) À partir de l'invariant et du test d'arrêt comment, en fin d'itération, déterminer si l'élément recherché x appartient au tableau ? Terminez l'algorithme et traduisez-le en MAPLE.

6 Extensions Maple : ensembles, listes, tableaux et itérateurs

Dans cette partie, les algorithmes/programmes/fonctions doivent être écrits en Maple. Lorsque les éléments sont de type entier, on supposera des entiers naturels (ℕ). On utilisera comme type Maple `integer`.

1. On se donne un entier n et une base b entière supérieure à 1. Écrivez la procédure `ecritureBase` qui renvoie l'écriture de n en base b, sous la forme de la liste ordonnée de ses « chiffres ».

   EXEMPLE 6.0.2 : `ecritureBase(586,5)` renvoie `[4, 3, 2, 1]`. □

2. Utilisez `ecritureBase` pour écrire la procédure `passageBase` qui a pour paramètres `L1,b1,b2` tels que `L1` est une liste représentant l'écriture d'un nombre n en base `b1`. Le résultat est la liste représentant l'écriture de ce même n en base `b2`.

   EXEMPLE 6.0.3 : `ecritureBase(586,5)` renvoie `[4, 3, 2, 1]`
   `ecritureBase(586,2)` renvoie `[1, 0, 0, 1, 0, 0, 1, 0, 1, 0]`
   Donc, `passageBase([4,3,2,1],5,10)` renvoie `[5, 8, 6]`. □

3. Soit $n \in \mathbb{N}$ et $c_p c_{p-1} ... c_1 c_0$ sa représentation par une suite de $p+1$ chiffres en base 10. Pour tester si n est divisible par 11, on calcule la somme $S = c_0 - c_1 + c_2 - c_3 + ...$ On a : n est divisible par 11 si et seulement si S est divisible par 11.
   (a) Démontrez simplement l'affirmation précédente.
   (b) On rappelle que la divisibilité de l'entier `a` par l'entier `b` se teste en Maple en évaluant l'expression booléenne :
   `evalb( irem(a,b) = 0)`
   Écrivez la fonction `estDivisiblePar11?` qui teste si son argument `n` est divisible par 11, en utilisant la propriété qui précède (et non en utilisant directement `evalb( irem(n,11) = 0)`.

4. Écrire (encore ?) la fonction `estPremier?` qui renvoie un booléen indiquant que son paramètre entier n est un entier premier, donc supérieur à 1. On pourra, par exemple, tester la divisibilité de `n` successivement par $2, 3, 4, 5, ...$ jusqu'à une borne que vous déterminerez. On s'arrête et renvoie `false` *dès que* on a trouvé un diviseur de n (autre que 1 et$n$).

   EXEMPLE 6.0.4 : `estPremier?(63)` ; renvoie `false` □

5. Utilisez le prédicat [1] `estPremier?`, pour écrire la fonction `ensPremier` qui, étant donné un ensemble d'entiers `E`, renvoie le sous–ensemble des entiers premiers qui sont éléments de `E`.

   EXEMPLE 6.0.5 : `ensPremier({2, 5, 3, 4, 8, 5});` renvoie `{2, 3, 5}` □

6. Vous écrirez aussi la fonction `listePremier` qui, étant donné une liste d'entiers `L`, renvoie la sous–liste des entiers premiers qui sont éléments de `L`.

   EXEMPLE 6.0.6 : `listePremier([2, 5, 3, 4, 8, 5]);` renvoie `[2, 5, 3, 5]` □

7. Vous écrirez enfin la fonction `tableauPremier` qui, étant donné un tableau d'entiers `T`, renvoie le sous–tableau des entiers premiers qui sont éléments de `T`. Attention, le résultat est un nouveau tableau.

   EXEMPLE 6.0.7 : `tableauPremier(array(1..6,[2, 5, 3, 4, 8, 5]));` renvoie le tableau dont la « valeur » est `[2, 5, 3, 5]` □

8. On va reprendre la série de questions précédente, en utilisant cette fois le prédicat `estCarré?` qui renvoie un booléen indiquant que son paramètre entier n est un entier carré d'un entier.
   (a) Écrivez une première version `estCarré1?` qui utilise la racine carrée (`sqrt` en MAPLE) et la partie entière `floor` en MAPLE *(Rappel :* `floor(n) est l'unique entier` x `tel que` $x \leqslant n < x+1$).
   (b) Écrivez une deuxième version `estCarré2?` qui teste les carrés des entiers 1,2,... jusqu'à ... une certaine valeur fonction de l'argument.
   (c) On veut ensuite `ensCarré` qui, étant donné un ensemble d'entiers `E`, renvoie le sous–ensemble des entiers qui sont des carrés et qui sont éléments de `E`.

[1] Un prédicat est une fonction à valeur booléenne.

(d) De même `listeCarré` qui, étant donné une liste d'entiers `L`, renvoie la sous–liste des entiers qui sont des carrés et qui sont éléments de `L`.

(e) Enfin, `tableauCarré` qui, étant donné un tableau d'entiers `T`, renvoie le sous–tableau des entiers qui sont des carrés et qui sont éléments de `T`.

9. On peut généraliser (abstraire) le procédé précédent de la manière suivante.

(a) Écrivez la fonction `ensTelsQuePred` qui a pour paramètres un ensemble d'entiers `E` et un prédicat `P`. Cette fonction doit renvoyer le sous–ensemble des entiers qui vérifient `P` et qui sont éléments de `E`. Un entier `n` vérifie `P` si `P(n)` renvoie `true`.

(b) On se donne un ensemble `E`. Comment utiliser `ensTelsQuePred` pour retrouver le sous–ensemble des entiers de `E` qui sont premiers ? qui sont des carrés ? qui sont des entiers supérieurs à 2 ?

10. Divisibilité, décomposition en facteurs premiers, pgcd et ppcm : Dans cette partie, les entiers manipulés sont de entiers positifs, éléments de $\mathbb{N}^*$. On utilisera la fonction `estPremier?` à valeur booléenne qui renvoie `true` si son argument est un entier premier.

(a) Écrivez la fonction à un paramètre `EnsDiviseursPremiers` qui renvoie l'ensemble des diviseurs premiers du paramètre.

(b) Écrivez la fonction à un paramètre `EnsDecompPremiers` qui renvoie l'ensemble des couples sous forme de liste à deux éléments : `[d,p]` où `d` est un diviseur premier du paramètre et `p` est son ordre de multiplicité.

EXEMPLE 6.0.8 : `EnsDecompPremiers(360)` renvoie `{[2, 3], [3, 2], [5, 1]}`, mais il pourrait aussi bien renvoyer le même résultat sous la forme `{ [3, 2], [5, 1], [2, 3]}`
En effet $360 = 2^3.3^2.5^1$ □

(c) Écrivez la fonction à un paramètre `ListeDecompPremiers` qui renvoie la liste des couples `[d,p]` où `d` est un diviseur premier du paramètre et `p` est son ordre de multiplicité. La liste résultat devra être ordonnée suivant les valeurs croissantes des premiers éléments des couples, donc des diviseurs du paramètre.

EXEMPLE 6.0.9 : `ListeDecompPremiers(360)` renvoie `[[2, 3], [3, 2], [5, 1]]`. □

(d) Soit $n1 = 900 = 2^2.3^2.5^2$ et $n2 = 315 = 3^2.5^1.7^1$.
On calcule `L1 = ListeDecompPremiers(n1) = [[2, 2], [3, 2], [5, 2]]` et
`L2 = ListeDecompPremiers(n2) = [[3, 2], [5, 1], [7, 1]]`.
On « *parcourt* » `L1` et `L2`, pour trouver les diviseurs premiers communs à `n1` et `n2`, avec le plus grand exposant possible. On trouve ainsi le pgcd de `n1` et `n2`. Dans notre exemple `pgcd(900,360)` est $3^2.5^1$ soit 45.
En utilisant ce principe, écrivez un algorithme et la procédure `pgcd` MAPLE qui calcule le plus grand diviseur commun des ses deux paramètres entiers.

(e) Utilisez un principe similaire pour écrire la procédure `ppcm` qui calcule le plus petit multiple commun des ses deux paramètres entiers.

11. On note | la relation « divise » dans $\mathbb{N}^*$. Par exemple 5|15. Dans cette partie, tous les entiers manipulés sont naturels non nuls.

(a) Montrez que | est une relation d'ordre sur $\mathbb{N}^*$ et que l'ordre est partiel.

(b) Écrivez la fonction `ensDiviseursDansE` qui a pour paramètres un entier x et un ensemble d'entiers E et qui renvoie le sous–ensemble de E formé des diviseurs de x différents de x.

(c) Soit E un sous–ensemble fini non vide de $\mathbb{N}^*$. Soit $n \in E$. On dit que n est minimal pour | dans E si et seulement si il n'existe aucun élément n' de E différent de n tel que $n'|n$.

EXEMPLE 6.0.10 : Soit $E = \{2, 3, 20, 25, 28, 35\}$. 2 et 25 sont minimaux dans E, alors que 20 ne l'est pas. □

Écrivez la fonction `estMinimal?` a pour paramètres un entier x et un ensemble d'entiers E, tels que $x \in E$, qui renvoie le booléen `true` si x est minimal pour | dans E, `false` sinon.

(d) En déduire l'écriture de la fonction `ensMinimaux` qui renvoie le sous–ensemble de l'ensemble d'entiers E formé des éléments minimaux de E.

(e) Applications : Déduire de ce qui précède les fonctions :

i. `ensDiviseursDansN` qui renvoie l'ensemble des diviseurs dans $\mathbb{N}^*$ de l'entier paramètre

ii. `ensPremiersPlusPetitsQue` qui renvoie l'ensemble des nombres premiers plus petits que l'entier paramètre.

iii. Un nombre est dit *parfait* s'il est la somme de ses diviseurs différents de lui-même. Écrivez la fonction `estParfait?` qui teste si son paramètre est un entier parfait.

Écrivez enfin la fonction `ensParfaitsPlusPetitsQue` qui renvoie l'ensemble des nombres parfaits inférieurs à son paramètre.