1 Expressions

La fonction **abs** : $\mathbb{Z} \longrightarrow \mathbb{N}$ qui renvoie la valeur absolue d'un entier relatif est une fonction prédéfinie de notre langage d'algorithme. On suppose définis les algorithmes suivants :

Algorithme : proche ?
Données : `a` : Nombre, `b` : Nombre
Résultat : Booléen, qui vaut `true` si la distance entre `a` et `b` est inférieure à 2 et `false` sinon.
Le résultat est : `abs(b-a) < 2`
fin algorithme
Algorithme : reste
Données : x : *Nombre* ; x est entier.
Résultat : Nombre, égal au reste de la division entière de x par 2.
Le résultat est : `x mod 2`
fin algorithme

1. Quels sont les types et valeurs des expressions suivantes :

```
(2 + (3 * (7 - 5)))                              2 + (3 * true)
(true ou false)                                  true et false
non (true et false)                              (2 < 3) et false
proche?(2,7)                                     proche?(1,2,3)
proche?(1,1) et (proche?(2,3) et proche?(1,3))   proche?(1,2) ou proche?((5*48) quo 7,33)
(reste(3) et reste(5)) < 3                       proche?(reste(5),2)
(5 + (reste(5) + 1)) quo 2                       proche?(reste(12),3-reste(7))
cond( reste(5)=1, reste(2)+1, reste(2))          cond( proche?(4,2), 2/0, 1)
cond( proche?(1,4), false, proche?(2,3))         cond( 1 < 3, reste(3), true)
```

2. On suppose à présent que `x` et `y` sont des paramètres de type `Nombre` et que `z` est un paramètre de type `Booléen`. Quel est le type des expressions suivantes ?

```
x + y                                 x < y
z et (x=y)                            z + x
reste(x) + y                          proche?(x,y) et (x < y)
reste(x) = reste(y)                   abs(x+y) quo reste(x)
cond( non(z), reste(y), (x mod y))    cond( z, reste(x)<1, proche?(y,3))
cond( proche?(x,y), z, x)             proche?(reste(x),abs(y)) ou z
```

2 Algorithmes non récursifs

1. Écrivez l'algorithme `distance` qui étant donné deux nombres calcule leur distance, c'est à dire la valeur absolue de leur différence.
2. La fonction `max` calculant le maximum de deux nombres et la fonction `abs` calculant la valeur absolue d'un nombre sont deux fonctions prédéfinies de notre langage d'algorithme. On peut cependant définir leur algorithme en utilisant l'opérateur conditionnel. Écrivez ces deux algorithmes.
3. La règle du max de la Faculté des Sciences, ou comment calculer la note finale à une UE :
   ≪ Soit `noteExam` et `noteCC` vos deux notes (sur 20) au contrôle terminal et au contrôle continu. Soit `coeffExam` et `coeffCC` les poids respectifs de ces notes. On calcule la note sur 20 qui est la moyenne pondérée de `NoteExam` et `noteCC`. Votre note finale sur 20 est le max entre la note précédente et la note à l'examen. ≫ Écrire un algorithme qui prend en paramètre les deux notes et les deux coefficients et donne comme résultat la note finale.
4. Définissez l'algorithme `max3` qui calcule le maximum de 3 nombres. Vous donnerez deux versions d'algorithme, l'une n'utilisant que l'opérateur conditionnelle, l'autre n'utilisant que la fonction `max`.
5. Écrivez l'algorithme `plusProche` qui, étant donné trois nombres `a`, `b` et `x`, renvoie le nombre choisi parmi `a` et `b` qui est le plus proche de `x`. Par exemple, si `a=5`, `b=9` quand `x=6` le plus proche est `5`, quand `x=10` le plus proche est `9`, quand `x=7` le résultat est soit `5` soit `9`.
6. Écrivez un algorithme pour la fonction `médian` qui étant donné trois nombres fournit l'élément médian de ces nombres. Par exemple l'élément médian de 12, 3 et 8 est 8.

7. Le « ou exclusif » est la fonction booléenne dont la signature est
   ouExcl : $Bool \times Bool \longrightarrow Bool$ et dont la sémantique est donnée par la table :

a	b	ouExcl(a,b)
true	true	false
true	false	true
false	true	true
false	false	false

Écrivez deux versions d'algorithme calculant cette fonction, la première utilisant l'opérateur conditionnel, la seconde ne l'utilisant pas.

3 Algorithmes récursifs

1. Que calcule l'algorithme suivant ? Exprimez le résultat en fonction du paramètre n.

> **Algorithme :** mystere
> **Données :** `n` : Nombre ; `n` est un entier naturel.
> **Résultat :** Nombre, ? ?
> **Le résultat est :** `cond( n=0, 0, n*n + mystere(n-1))`
> **fin algorithme**

2. En utilisant la **récursivité** vous écrirez un algorithme qui calcule la somme $1+2+...+n$ en fonction de son paramètre n. En notant `somme` cette fonction, complétez la définition récurrente :
   — **Cas de base :** quand `n=0`, `somme(n)` vaut ? ? ?
   — **Équation de récurrence :** quand `n>0`, `somme(n)` vaut ? ? ?
   Écrivez l'algorithme.

3. Écrivez le corps de l'algorithme dont la spécification est :

> **Algorithme :** sommeInterv
> **Données :** a : Nombre, b : Nombre ; a et b sont 2 entiers tels que $a \leq b$.
> **Résultat :** Nombre, somme des entiers $a + (a+1) + \ldots + b$

4. Écrivez un algorithme `suiteU` qui étant donné un entier naturel n, calcule le $n^{ème}$ terme de la suite $(U_n)_{n\in\mathbb{N}}$ définie par : $U_0 = 1$ et $\forall n > 0\ U_n = 2 \times U_{n-1} + 3$

5. On cherche un algorithme calculant le carré d'un entier naturel n, sans utiliser de multiplication. Appelons `carré` cet algorithme et n son paramètre.
   Pour calculer `carré(n)` on utilise un schéma récursif :
   — **Cas de base :** quand `n=0` que vaut `carré(n)` ?
   — **Équation de récurrence :** quand `n`$\neq$`0` essayez de définir `carré(n)` en fonction de `carré(n-1)` (pour cela développez l'expression $(n-1)^2$).
   Écrivez l'algorithme `carré`.

6. Écrire un algorithme récursif `estPair` qui, étant donné un entier `n` positif ou nul renvoie `true` si `n` est un entier pair, `false` sinon. Les seules opérations autorisées sont la soustraction, les comparaisons et la conditionnelle.

7. Écrivez un algorithme qui renvoie la somme des entiers impairs inférieurs ou égaux à n : $1+3+5+\ldots n$ (si n est impair) ou $1+3+5+\ldots n-1$ (si n est pair).

8. Complétez l'algorithme dont les spécifications sont :

> **Algorithme :** plusPetitDiv
> **Données :** a : Nombre, b : Nombre ; a et b sont 2 entiers naturels tels que $a \leq b$.
> **Résultat :** Nombre, le plus petit entier d tel que $a \leq d \leq b$ et d divise b.

   Exemple : `plusPetitDiv(2,35)` vaut 5, `plusPetitDiv(3,9)` vaut 3 et `plusPetitDiv(8,35)` = 35.
   Utilisez l'algorithme `plusPetitDiv` pour définir un algorithme qui teste si un nombre est un nombre premier. Les spécifications de cet algorithme sont :

> **Algorithme :** estPremier
> **Données :** n : Nombre ; n est un entier natuel
> **Résultat :** Booléen, qui vaut `true` si n est un nombre premier, $false$ sinon.

On rappelle qu'un nombre $n \neq 1$ est premier si ses seuls diviseurs sont 1 et n. On rappelle également que 0 et 1 ne sont pas des nombres premiers.

9. (**) La multiplication dite « du paysan russe ». On souhaite multiplier deux entiers positifs a et b en n'utilisant que des additions, la multiplication par 2 et la division par 2. Soit `mul` le nom de cet algorithme. Pour définir `mul` on utilise un schéma récursif exprimant `mul(a,b)` en fonction de `mul(a quo 2,b)`. Complétez ce schéma :
   — **Cas de base :** quand `a=0` `mul(a,b)` vaut ...
   — **Équation de récurrence :** quand `a`$\neq$`0` il existe 2 cas :
     — quand `a` est pair `mul(a,b)` vaut ... `mul(a quo 2, b)` ...
     — quand `a` est impair `mul(a,b)` vaut ... `mul(a quo 2, b)` ...

   Écrivez l'algorithme `mul`.

10. (****) On cherche à calculer le nombre de façons de répartir n objets dans b boîtes différenciées. On compte les répartitions avec boîte vide. Par exemple les répartitions de 3 objets dans 2 boîtes sont | 0 | 3 | | 1 | 2 | | 2 | 1 | | 3 | 0 | et donc le nombre de répartitions est 4. Complétez l'algorithme :

> **Algorithme :** nbRepart
> **Données :** n : Nombre, b : Nombre ; n et b sont 2 entiers naturels, $b \neq 0$.
> **Résultat :** Nombre, le nombre de façons de répartir n objets dans b boîtes différenciées.

Le schéma de récurrence permettant de définir `nbRepart` utilise une récurrence sur les deux paramètres n et b. Comment modifier la définition pour ne compter que les répartitions dans des boîtes différenciées **non vides** ? Il n'y a que 2 façons de répartir 3 objets dans 2 boîtes non vides.

4 C/C++

Traduisez en C/C++ l'algorithme `plusProche` de l'exercice 2.5 et l'algorithme `carré` de l'exercice 3.5.

5 Listes.

1. Supposons que `li` soit un paramètre de type `Liste de Nombres`. Comment obtenir la valeur du premier élément de `li` ? du deuxième ? Comment savoir si `li` a plus d'un élément ? Comment insérer la valeur `5` en tête de la liste `li` ?
2. Dans les expressions suivantes, substituez à `li` la liste `( 2 3 2 4 )`, puis calculez la valeur de l'expression obtenue :

`queue(queue(li))`	`tête(queue(queue(li)))`
`tête(queue(queue(queue(queue(li)))))`	`estVide(queue(queue(queue(queue(li)))))`
`estVide(tete(li))`	`cons(7,li)`
`cons(7,queue(li))`	`cons(tête(li),queue(li))`
`cons(tête(queue(li)),queue(li))`	

3. Écrire un algorithme qui inverse les deux premiers éléments d'une liste de nombres composée d'au moins deux éléments.
4. Soit l'algorithme `listeEntiers`, qui étant donné un entier naturel `n`, calcule la liste `(n n-1 ... 1 0)`. Complétez la récurrence :
   — **Cas de base :** quand `n=0` que vaut `listeEntiers(0)` ?
   — **Équation de récurrence :** quand `n>0`, définissez `listeEntiers(n)` en fonction de `listeEntiers(n-1)`.

   Écrivez l'algorithme `listeEntiers`.
5. On cherche à écrire un algorithme `maxListe` qui étant donné `li`, une liste non vide de nombres, calcule le plus grand de ses éléments. Pour cela complétez la récurrence :
   — **Cas de base :** quand `li` ne contient qu'un élément que vaut `maxListe(li)` ?
   — **Équation de récurrence :** quand `li` contient plus d'un élément, définissez `maxListe(li)` en fonction de `maxListe(queue(li))` en utilisant la fonction `max` (maximum de 2 nombres).

Complétez alors l'algorithme :

> **Algorithme :** maxListe
> **Données :** li : Liste de Nombres ; li n'est pas vide.
> **Résultat :** Nombre, le plus grand élément de li

6. Écrire l'algorithme `derListe` qui, étant donnée une liste non vide `li`, calcule la valeur du dernier élément de `li`. Traduisez l'algorithme `derListe` en une fonction C/C++.
7. Écrire l'algorithme `appartientLi` qui étant donnés un nombre `n`, et une liste de nombres `li` vaut `true` si `n` est la valeur d'un élément de `li`, `false` sinon.
   Écrire l'algorithme `nbOccListe` qui étant donnés un entier `n`, et une liste d'entiers `li` calcule le nombre d'occurrences de `n` dans `li`, c'est à dire le nombre d'éléments de la liste `li` égaux à `n`.
8. Écrire l'algorithme `tousPairs` qui étant donné une liste de nombres `li` vaut `true` si chaque élément de `li` est un nombre pair, `false` sinon.
   Exemple, `tousPairs((2 8 2))` et `tousPairs(( ))` valent `true`. `tousPairs((2 5 2))` vaut `false`.
9. Écrire l'algorithme `listesEgales` qui, étant données deux listes `li1` et `li2` quelconques, vaut `true` si ces deux listes sont « égales », `false` sinon.
10. Soient les définitions suivantes ; `l1` et `l2` étant 2 listes :
    — `l1` est **préfixe** de `l2` si la séquence des éléments de `l2` est composée des éléments de `l1` dans le même ordre, puis d'éléments en nombre et valeur quelconques.
    — `l1` est **suffixe** de `l2` si la séquence des éléments de `l2` est composée d'éléments quelconques, suivis des éléments de `l1` dans le même ordre.
    — `l1` est **facteur** de `l2` si la séquence des éléments de `l2` est composée d'élements quelconques, suivis des éléments de `l1` dans le même ordre, suivis d'éléments quelconques.

    Exemples :
    — `(3 2 3)` est préfixe et facteur de la liste `(3 2 3 4 2)`.
    — `(3 2 3)` est préfixe, suffixe et facteur de la liste `(3 2 3)`.
    — `()` est préfixe, suffixe et facteur de toutes les listes.
    — `(3 2 3)` est suffixe et facteur de la liste `(4 3 2 3)`.
    — `(3 2 3)` est facteur de la liste `(4 2 3 2 3 8)`.

    Écrivez les algorithmes `prefixe`, `suffixe` et `facteur`.
11. Expliquez pourquoi l'algorithme suivant n'est pas correct :

> **Algorithme :** ajoutFin
> **Données :** n : Nombre, li : Liste de Nombres
> **Résultat :** Liste de Nombres, la liste `li` à laquelle on a ajouté `n` comme dernier élément.
>
> **Le résultat est :** `cons(li,n)`
> **fin algorithme**

   Modifiez le corps de cet algorithme pour obtenir une version correcte de `ajoutFin`.
12. En utilisant `ajoutFin`, écrivez un algorithme `listeInversee` qui étant une liste `li`, calcule la liste composée des éléments de `li`, mais dans l'ordre inverse.
13. (***) `l1` et `l2` étant 2 listes de nombres triées dans l'ordre croissant, la fusion de `l1` et `l2` est la liste triée composée des éléments de `l1` et des éléments de `l2`. Par exemple la fusion des listes `(2 2 4 7)` et `(1 2 3 4 4)` est la liste `(1 2 2 2 3 4 4 4 7)`.
    Écrivez un algorithme réalisant cette opération.
14. (****) Écrire une fonction C/C++ `listeSuffixes` qui étant donnée une liste de nombres `l1` calcule la liste des listes suffixes de `l1`. Le résultat est donc une liste dont les éléments sont des listes de nombres. L'ordre des préfixes dans la liste résultat est quelconque. Par exemple la valeur de `listeSuffixes((2 5 3))` est `( (2 5 3) (5 3) (3) () )` ou toute autre liste contenant les 4 éléments `() (3) (5 3) (2 5 3)`.
    Écrire une fonction C/C++ `listePréfixes` qui étant donnée une liste de nombres `l1` calcule la liste des listes préfixes de `l1`. Par exemple la valeur de `listePréfixes((2 5 3))` est `( (2 5 3) (2 5) (2) () )` ou toute autre liste contenant les 4 éléments `(2 5 3) (2 5) (2) ()`.