
Algorithmique – Travaux Dirigés

Master “*Technologie et Handicap*” : Intensifs 1

Corrigé

Exercice 1 – Affectations

1. Considérons les algorithmes ci-dessous.

- (a) Quel sera le contenu des variables a , b et éventuellement c après leur exécution ?
- (b) Dans chacun des cas ci-dessus, y a-t-il des lignes inutiles, et si oui lesquelles ?

```
1 // algo 1.1
2 var a, b, c : entier
3 debut
4 a ← 1
5 b ← a + 2
6 c ← b - 3
7 fin
```

Solution

```
// a = 1
// b = 3
// c = 0
```

```
1 // algo 1.2
2 var a, b : entier
3 debut
4 a ← 1
5 b ← a + 2
6 a ← 3
7 fin
```

Solution

```
// a = 3
// b = 3
```

```
1 // algo 1.3
2 var a, b : entier
3 debut
4 a ← 2
5 b ← a + 2
6 a ← a + 2
7 b ← a - 2
8 fin
```

Solution

```
// a = 4
// b = 2
// ligne 5 inutile
```

```
1 // algo 1.4
2 var a, b, c : entier
3 debut
4 a ← 2
5 b ← 4
6 c ← a + b
7 a ← 1
8 c ← b - a
9 fin
```

Solution

```
// a = 1
// b = 4
// c = 3
// lignes 6 et 4 inutile
// (la 4 parce que la 6 est inutile)
```

```
1 // algo 1.5
2 var a, b, c : entier
3 debut
4 a ← 1
5 b ← 2
6 b ← a + b
7 c ← a + b
8 fin
```

Solution

```
// a = 1
// b = 3
// c = 4
```

```
1 // algo 1.6
2 var a, b : car
3 debut
4 a ← '1'
5 b ← '2'
6 a ← a + b
7 fin
```

Solution

Ne marche pas. On ne peut pas additionner des caractères.

Solution (Complément)

Dans la plupart des langages de programmation le dernier exemple (1.6) ne générera pas d'erreur mais le résultat ne sera pas souvent '3'. Selon le langage, ce sera la concaténation de ces caractères en une chaîne de caractères "ab" (cas des langages **Python** ou **javascript**), voire "a+b" (cas du **shell**) ou bien la somme des codes ASCII correspondant aux caractères '1' et '2', soit $49 + 50 = 99$: le caractère 'c' (cas des langages **C**, **C++** ou **java**). Dans très peu de langages, comme **PHP** ou **perl**, en revanche, le résultat sera bien 3. Enfin dans d'autres langages comme **pascal**, qui est un langage fortement typé, la compilation générera une erreur.

2. Considérons l'algorithme ci-contre

- Permet-il de permuter les valeurs des variables a et b ?
- Proposer des solutions pour permuter les valeurs de 2 variables numériques ? Chacune des solutions proposées marche-t-elle dans le cas de variables non numériques.
- Étant données 3 variables a , b et c , proposer un algorithme pour les permuter circulairement, en transférant les valeurs initiales de a à b , de b à c , et de c à a .

```

1  var a, b : entier
2  debut
3  a ← 1
4  b ← 2
5  a ← b
6  b ← a
7  fin

```

Solutions

(a) Bien sûr que non ! Le résultat est // $a = 2, b = 2$

(b)

```

var a, b, c : entier
debut
a ← 1
b ← 2
c ← a
a ← b
b ← c
fin
// Fonctionne avec
// n'importe quel
// type de données

```

```

var a, b, c : entier
debut
a ← 1
b ← 2
a ← a + b
b ← a - b
a ← a - b
fin
// Ne fonctionne
// qu'avec des nombres !

```

(c) Permutation de 3 variables

```

var a, b, c, t : entier
debut
a ← 1
b ← 2
c ← 3
t ← a
a ← c
c ← b
b ← t
fin

```

Exercice 2

1. Quels sont les affichages provoqués par les algorithmes ci-contre.

Solutions

algo 2.1	algo 2.2
4.21 8.42	5 2
16.84	2
16	2.0

Pour l'algo 2.2, si on veut que le résultat soit bien la moyenne de 5 et de 2, c'est à dire 2.5 et non 2.0, il faut forcer la conversion en réel avant la division, soit en faisant successivement, à la place de la ligne 9, $c \leftarrow a$ et $c \leftarrow c / b$, ou bien directement : $c \leftarrow (1.0 * a) / 2$.

```

1 // algo 2.1
2 var a, b : reel
3 var c : entier
4 debut
5 a ← 4.21
6 b ← a * 2
7 ecrire a, b
8 ecrire b * 2
9 c ← b * 2
10 ecrire c
11 fin

```

```

1 // algo 2.2
2 var a, b : entier
3 var c : reel
4 debut
5 a ← 5
6 b ← 2
7 ecrire a, b
8 ecrire a / b
9 c ← a / b
10 ecrire c
11 fin

```

2. Écrire un algorithme qui demande un entier à l'utilisateur, puis affiche son carré.

Solution

```

var a : entier
debut
lire "Saisissez_un_entier_", a
ecrire a * a
fin

```

Exercice 3 – Conditionnelles

1. Écrire un algorithme qui demande un entier à l'utilisateur, teste si ce nombre est positif (≥ 0) ou non, et affiche "positif" ou "négatif".
2. Écrire un algorithme qui demande un entier à l'utilisateur, teste si ce nombre est strictement positif, nul ou strictement négatif, et affiche ce résultat.
3. Écrire un algorithme qui demande un réel à l'utilisateur et affiche sa valeur absolue (sans utiliser de fonction prédéfinie évidemment).
4. Écrire un algorithme qui demande un réel à l'utilisateur et l'arrondit à l'entier le plus proche (les x,5 seront arrondis à l'entier supérieur).
5. Écrire un algorithme qui demande le numéro d'un mois et affiche le nombre jours que comporte ce mois (sans tenir compte des années bissextiles).
6. Écrire un algorithme qui vérifie si une année est bissextile. On rappelle qu'il y a des années bissextiles tous les 4 ans, mais la première année d'un siècle ne l'est pas (1800, 1900 n'étaient pas bissextiles) sauf tous les 400 ans (2000 était une année bissextile).
7. Écrire un algorithme qui demande une date sous la forme de 2 nombres entiers (numéro du jour et numéro du mois) et affiche la saison (ex : 12/02 \rightsquigarrow hiver). On supposera que le premier jour de la saison est toujours le 21.
8. Écrire un programme qui demande les coordonnées (x, y) des sommets A, B et C d'un triangle et affiche la nature du triangle (isocèle, équilatéral, rectangle ou quelconque).

Solutions

1. Entier positif

```
var a : entier
debut
lire "tapez_un_entier", a
si a  $\geq$  0
alors
    ecrire a, "est_positif"
sinon
    ecrire a, "est_négatif"
finsi
fin
```

3. Valeur absolue

```
var a : reel
debut
lire "tapez_un_réel", a
si a > 0
alors
    ecrire "|", a, "|=", a
sinon
    ecrire "|", a, "|=", -a
finsi
fin
```

4. Arrondi

```
var a, d : reel
var b : entier
debut
lire "tapez_un_réel", a
b  $\leftarrow$  a
d  $\leftarrow$  a - b
si d < .5
alors
    b  $\leftarrow$  b + 1
finsi
ecrire a, "arrondi_à", b
fin
```

2. Signe d'un entier

```
var a : entier
debut
lire "tapez_un_entier", a
ecrire "ce_nombre_est_:"
si a > 0
alors
    ecrire a, "est_strict_positif"
sinon
    si a < 0
    alors
        ecrire a, "est_strict_négatif"
    sinon
        ecrire a, "est_nul"
    finsi
finsi
fin
```

5. Nombre de jours du mois

```
var mois : entier
debut
lire "tapez_un_numéro_de_mois
    (entre 1 et 12)", mois
si mois = 2
alors
    ecrire "28_ou_29_jours"
sinon
    si mois = 4 ou mois = 6
    ou mois = 9 ou mois = 11
    alors
        ecrire "30_jours"
    sinon
        ecrire "31_jours"
    finsi
finsi
fin
```

6. Années bissextiles

```
var annee : entier
debut
lire "tapez_une_année", annee
si annee%4 = 0 et annee%100 ≠ 0
alors
  ecrire annee, "est_bissextile"
sinon
  si annee%400 = 0
  alors
    ecrire annee, "est_bissextile"
  sinon
    ecrire annee, "n'est_pas_bissextile"
  finsi
finsi
fin
```

ou encore

```
var annee : entier
debut
lire "tapez_une_année", annee
si ( annee%4 = 0 et annee%100 ≠ 0 ) ou annee%400 = 0
alors
  ecrire annee, "est_bissextile"
sinon
  ecrire annee, "n'est_pas_bissextile"
finsi
fin
```

7. Saisons

```
var jour, mois : entier
debut
lire "Quel_est_le_jour?", jour
lire "Quel_est_le_mois_(entre_1_et_12)", mois
si (mois=12 et jour≥21) ou mois=1 ou mois=2 ou (mois=3 et jour<21)
alors
  ecrire "C'est_l'hiver!"
sinon
  si (mois=3 et jour≥21) ou mois=4 ou mois=5 ou (mois=6 et jour<21)
  alors
    ecrire "Vive_le_printemps!"
  sinon
    si (mois=6 et jour≥21) ou mois=7 ou mois=8 ou (mois=9 et jour<21)
    alors
      ecrire "Enfin_l'été!"
    sinon
      ecrire "Déjà_l'automne!"
    finsi
  finsi
finsi
fin
```

8. Triangles

```
var xA, yA, xB, yB, xC, yC : reel
    lAB, lAC, lBC, precision : reel
debut
precision=.001
lire "Donnez_les_coordonnées_(x,y)_du_point_A", xA, yA
lire "Donnez_les_coordonnées_(x,y)_du_point_B", xB, yB
lire "Donnez_les_coordonnées_(x,y)_du_point_C", xC, yC

lAB= $\sqrt{|x_A - x_B|^2 + |y_A - y_B|^2}$ 
lAC= $\sqrt{|x_A - x_C|^2 + |y_A - y_C|^2}$ 
lBC= $\sqrt{|x_B - x_C|^2 + |y_B - y_C|^2}$ 

si lAB-lAC < precision et lAB-lBC < precision
alors
    ecrire "Le_triangle_est_équilatéral"
sinon
    si lAB*lAB + lAC*lAC -lBC*lBC < precision
    alors
        si lAB - lAC < precision
        alors
            ecrire "Le_triangle_est_isocèle_et_rectangle_en_A"
        sinon
            ecrire "Le_triangle_est_rectangle_en_A"
        finsi
    sinon
        si lAB*lAB + lBC*lBC -lAC*lAC < precision
        alors
            si lAB - lBC < precision
            alors
                ecrire "Le_triangle_est_isocèle_et_rectangle_en_B"
            sinon
                ecrire "Le_triangle_est_rectangle_en_B"
            finsi
        sinon
            si lAC*lAC + lBC*lBC -lAB*lAB < precision
            alors
                si lAC - lBC < precision
                alors
                    ecrire "Le_triangle_est_isocèle_et_rectangle_en_C"
                sinon
                    ecrire "Le_triangle_est_rectangle_en_C"
                finsi
            sinon
                si lAB-lAC < precision ou lAB-lBC < precision ou lAC-lBC < precision
                alors
                    ecrire "Le_triangle_est_isocèle"
                sinon
                    ecrire "Le_triangle_est_quelconque"
                finsi
            finsi
        finsi
    finsi
fin
fin
```

Exercice 4 – Itérations

1. Écrire un algorithme qui demande un entier positif, et le rejette tant que le nombre saisi n'est pas conforme.

Solution

```
var a : entier
debut
ecrire "Saisir_un_entier_positif"
lire a
tantque a < 0
faire
    écrire "on_a_dit_positif!,_recommencez"
    lire a
fait
fin
```

2. Écrire un algorithme qui demande 10 entiers, compte le nombre d'entiers positifs saisis, et affiche ce résultat.

Solution

```
var a, i, cc : entier
debut
ecrire "Saisir_des_entiers"
i ← 0
cc ← 0
tantque i < 10
faire
    lire a
    i ← i + 1
    si a ≥ 0
    alors
        cc ← cc + 1
    finsi
fait
ecrire cc, "sont_positifs"
fin
```

ou bien, avec une boucle de type "pour"

```
var a, cc, n : entier
debut
ecrire "Saisir_des_entiers"
cc ← 0
pour i allantde 1 a 10
faire
    lire a
    si a ≥ 0
    alors
        cc ← cc + 1
    finsi
fait
ecrire cc, "sont_positifs"
fin
```

3. Écrire un algorithme qui demande des entiers positifs à l'utilisateur, les additionne, et qui s'arrête en affichant le résultat dès qu'un entier négatif est saisi.

Solution

```
var a, s : entier
debut
ecrire "Saisir_des_entiers"
lire a
s ← 0
tantque a ≥ 0
faire
    s ← s + a
    lire a
fait
ecrire "Résultat:_:", s
fin
```

-
4. Modifier ce dernier algorithme pour afficher la moyenne de la série d'entiers positifs saisis.

Solution

```
var a, s, cc : entier
var m : reel
debut
  ecrire "Saisir_des_entiers"
  lire a
  s ← 0
  cc ← 0
  tantque a ≥ 0
  faire
    cc += 1
    s ← s + a
    lire a
  fait
  m ← s // pour le convertir en reel
  si cc > 0 alors
    m ← m / cc
    ecrire "Moyenne_:_", m
  sinon
    ecrire "Aucun_nombre_saisi,_pas_de_moyenne!"
fin
```

Attention, si on écrit directement $m \leftarrow s/cc$ le résultat sera faux car converti en entier.

Exercice 5 – Conversion en binaire

1. Écrire un algorithme de conversion d'un nombre entier en binaire.
2. (Facultatif) Écrire un algorithme de conversion d'un nombre entier en une base b quelconque.

Solutions

1. Conversion en binaire

```
var n : entier
var bin : chaine
debut
  lire "Saisir_un_entier", n
  bin=""

  si n=0
  alors
    bin="0"
  finsi

  tantque n>0
  faire
    si n%2=0
    alors
      ch = "0" + ch
    sinon
      ch = "1" + ch
    finsi
  fait

  ecrire(n, "en_binaire_:_", bin)
fin
```

2. Conversion en base b ($b < 10$)

```
var n, base : entier
var out : chaine
debut
  lire "Saisir_un_entier", n
  lire "Saisir_la_base_(<10)", b
  out=""

  si n=0 alors out="0" finsi

  tantque n>0
  faire
    ch=char(n%b=0) + ch
    n=n/b
  fait

  ecrire(n, "en_base",b,"_:_", out)
fin
```

Notes :

- On suppose que `char(int n)` est une fonction qui transforme un entier n en la chaîne de caractères correspondante.
- Pour les bases supérieures à 10, on a besoin de plus de 10 "chiffres", (en général on utilise des lettres), qu'il faut placer dans un tableau.

Exercice 6 – Suites

1. Écrire un algorithme pour afficher les n premiers termes des suites suivantes (n demandé à l'utilisateur) :

(a) Suite arithmétique

$$\begin{cases} u_0 = 1 \\ u_{n+1} = u_n + 2 \end{cases}$$

(b) Suite de Newton

$$\begin{cases} u_0 = \frac{a}{2} \\ u_{n+1} = \frac{1}{2}(u_n + \frac{a}{u_n}) \end{cases}$$

(a réel demandé à l'utilisateur)

(c) Suite de Fibonacci

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

2. (facultatif) La suite de Newton converge vers la racine carrée de a . Modifier l'algorithme (b) pour calculer la racine carrée d'un nombre selon une précision donnée.

1. Solution

```
var u, i, n : entier
debut
lire "nb_de_termes_", n
u ← 1
i ← 0
tantque i ≤ n faire
    ecrire "U(", i, ")=", u
    u ← u + 2
    i ← i + 1
fait
fin
```

2. Solution

```
var n : entier
var u, a : reel
debut
lire "a", a
lire "nb_de_termes_", n
u ← 1
i ← 0
tantque i ≤ n faire
    ecrire "U(", i, ")=", u
    u ← (u + a / u) / 2
    i ← i + 1
fait
fin
```

Note : il s'agit d'une l'approximation de la racine carrée

3. Solution

```
var u, u1, u2, n : entier
debut
lire "nb_de_termes_", n
u1 ← 0
u2 ← 1
ecrire "U(0)=0"
ecrire "U(1)=1"
i ← 1
tantque i < n faire
    u ← u1 + u2
    u1 ← u2
    u2 ← u
    i ← i + 1
    ecrire "U(", i, ")=", u
fait
fin
```

Exercice 7 – Devinez un nombre

1. Écrire un algorithme permettant de jouer au jeu du *plus petit-plus grand*. On tire un nombre au hasard pour le faire deviner au joueur en lui disant à chaque tour si le nombre proposé est plus grand ou plus petit que le nombre à chercher. Lorsque le joueur a trouvé, l'algorithme se termine en affichant le nombre de tours.

Note : On suppose qu'on a une fonction **entier** *nombreAléatoire*(**var** max : **entier**) qui tire un nombre au hasard et le renvoie.

2. Modifier ensuite cet algorithme pour limiter à 10 le nombre de propositions du joueur, et afficher "Perdu !" si le joueur n'a pas trouvé.

Solutions

```
var x, prop, cc : entier
debut
cc ← 1
x ← nombreAléatoire(1000)
lire "Ta_proposition_", prop
tantque prop ≠ x faire
    si prop > x alors
        ecrire "trop_grand"
    sinon
        ecrire "trop_petit"
    finsi
    lire "Ta_proposition_", prop
    cc ← cc + 1
fait
ecrire "Trouvé_en_", cc, "coups"
fin
```

```
var x, prop, cc : entier
debut
cc ← 1
x ← nombreAléatoire(1000)
lire "Ta_proposition_", prop
tantque (prop ≠ x) et (cc < 10) faire
    si prop > x alors
        ecrire "trop_grand"
    sinon
        ecrire "trop_petit"
    finsi
    lire "Ta_proposition_", prop
    cc ← cc + 1
fait
si prop = x
alors
    ecrire "Trouvé_en_", cc, "coups"
sinon
    ecrire "Perdu!"
fin
fin
```


Exercice 8 – Boucles imbriquées

1. Échiquiers

- Écrire un algorithme permettant d'écrire un carré de 8 fois 8 caractères 'x'.
- Écrire un algorithme permettant d'écrire un échiquier. On représentera les cases noires par des 'x' et les cases blanches par des espaces.
- Modifier l'algorithme précédent pour afficher un cadre autour de l'échiquier, en utilisant les caractères '|', '-' et '+
- Modifier de nouveau cet algorithme pour afficher l'ensemble des cases avec ces mêmes caractères (voir exemple 4 ci-dessous – incomplet).

// a	// b	// c	// d
xxxxxxxx	x x x x	+-----+	+-----+ ... +
xxxxxxxx	x x x x	x x x x	x x ...
xxxxxxxx	x x x x	x x x x	+-----+ ... +
xxxxxxxx	x x x x	x x x x	x ...
xxxxxxxx	x x x x	x x x x	+-----+ ... +
xxxxxxxx	x x x x	x x x x	x x ...
xxxxxxxx	x x x x	x x x x	+-----+ ... +
xxxxxxxx	x x x x	x x x x	x x x x
xxxxxxxx	x x x x	x x x x
xxxxxxxx	x x x x	x x x x
		+-----+	+-----+ ... +

Solutions

On suppose dans ce qui suit que **ecrire** n'insère pas de fin de ligne. On utilisera le caractère **EOL** comme caractère de retour à la ligne. On procède de façon incrémentale, en modifiant le code de (a) pour faire (b) et ainsi de suite.

(a) Carré

```

var i, j : entier
debut
pour i allantde 1 a 8
faire
  pour j allantde 1 a 8
  faire
    écrire "x"
  fait
  écrire EOL
fait
fin
    
```

Autre solution construisant la solution dans une chaîne de caractères

```

var i, j : entier
var s : chaîne
debut
s ← ""
pour i allantde 1 a 8
faire
  pour j allantde 1 a 8
  faire
    s ← s + "x"
  fait
  s ← s + EOL
fait
écrire s
fin
    
```

(b) Échiquier

```

var i, j : entier
debut
pour i allantde 1 a 8
faire
  pour j allantde 1 a 8
  faire
    si (i+j)%2=0
    alors
      écrire "x"
    sinon
      écrire " "
    finsi
  fait
  écrire EOL
fait
fin
    
```

(c) Échiquier encadré

```

var i, j : entier
debut
écrire "+"
pour i allantde 1 a 8
faire
  écrire "-"
fait
écrire "+\n"
pour i allantde 1 a 8
faire
  écrire "|"
  pour j allantde 1 a 8
  faire
    si (i+j)%2=0
    alors
      écrire "x"
    sinon
      écrire " "
    finsi
  fait
  écrire "|\n"
fait
écrire "+"
pour i allantde 1 a 8
faire
  écrire "-"
fait
écrire "+\n"
fin
    
```

(d) Échiquier avec cases dessinées.

On construira la solution à partir de celle de (c) en déplaçant simplement quelques caractères (un '|', deux '+' et en déplaçant 1 'fait'.

2. Écrire un algorithme permettant d'écrire une table de multiplication comme celle présentée ci-contre. Dans un premier temps on ne s'occupera pas du nombre d'espaces entre les nombres, puis on affina en en tenant compte.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
...									
10	20	30	40	50	60	70	80	90	100

Solutions

(sans tenir compte des espaces)

```

var i, j : entier
debut
pour i allantde 1 a 10 faire
  pour j allantde 1 a 10 faire
    ecrire i * j, " "
  fait
ecrire EOL
fait
fin

```

(en alignant correctement les colonnes)

```

var i, j : entier
debut
pour i allantde 1 a 10 faire
  pour j allantde 1 a 10 faire
    si i*j < 100 alors ecrire " " finsi
    si i*j < 10 alors ecrire " " finsi
    // Attention ici les 2 tests ne
    // peuvent pas être combinés
    // car si i*j<10 il faut 2 espaces
    ecrire i * j, " "
  fait
ecrire EOL
fait
fin

```

Exercice 9 – Tableaux

1. Que font les algorithmes suivants ?

```

1 var i, n[10] entier
2 debut
3 n[0] ← 1
4 pour i allantde 1 a 9
5 faire
6 n[i] ← n[i-1] + 2
7 fin

```

Solution : Initialisation d'un tableau avec les 10 premiers nombres impairs

```

1 var i, n[10] entier
2 debut
3 pour i allantde 0 a 9
4 faire
5 n[i] ← 2 * i
6 fin

```

Solution : Initialisation d'un tableau avec les 10 premiers nombres pairs

2. Écrire un algorithme qui déclare un tableau de 10 éléments et initialise toutes ses valeurs à 1

Solution

```

var i, n[10] entier
debut
pour i allantde 0 a 9
faire
n[i] ← 1
fin

```

3. Écrire un algorithme qui calcule les n premiers nombres premiers.

Solution

```

var premiers[], nb , n, cc, i : entier
debut
lire "Combien_de_nombres_premiers_voulez_vous?", nb
dimension premiers[nb]
// Invariant de boucle: à chaque itération, on connaît les cc premiers
// nombres premier rangés de p[0] à p[cc-1]
// version 1
cc ← 0
n ← 2
tantque cc < nb faire
  i ← 2
  tantque (i < n) et (n % i ≠ 0)
  faire
    i ← i + 1
  fait
  si i = n alors
    premiers[cc] ← n
    cc ← cc+1
  finsi
  n ← n+1
fait
fin

```

On peut l'améliorer en remplaçant $(i < n)$ par $(i < n/2)$, par contre il serait inefficace de le remplacer par $(i < \sqrt{n})$ en raison du temps de calcul d'une racine.

```

// version 2
cc ← 0
n ← 2
tantque cc < nb faire
  i ← 0
  tantque (i < cc)
  et (n % premier[i] ≠ 0)
  faire
    i ← i + 1
  fait
  si i = cc alors
    premiers[cc] ← n
    cc ← cc+1
  finsi
  n ← n+1
fait
fin

```

Exercice 10 – Le triangle de Pascal

Écrire un algorithme permettant de calculer le triangle de Pascal au rang n , dans lequel à la ligne i et à la colonne j ($0 \leq j \leq i$) est placé le coefficient binomial C_j^i . On le construit aisément par récurrence, en remarquant qu'à chaque ligne i , le coefficient numéro j est la somme des coefficients $j - 1$ et j de la ligne précédente (pour $0 < j < i$). Les lignes 1 et 2 sont initialisées à 1 ainsi que les coefficients 1 et i de chaque ligne i .

Autrement dit :

$$\begin{cases} \forall i, j \mid 0 < j < i, C_j^i = C_{j-1}^{i-1} + C_j^{i-1} \\ C_i^0 = C_i^i = 1 \end{cases}$$

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

Solution

```

var i, j, n : entier
var pascal[][] : entier
debut
lire "Jusqu'à_quelle_ligne_", n
dimension pascal[n][n]
i ← 0
tantque i ≤ n faire
  pascal[i,0] ← 1
  j ← 1
  tantque j < i faire
    pascal[i,j] ← pascal[i-1,j-1] + pascal[i-1,j]
    j ← j + 1
  fait
  pascal[i,i] ← 1
  i ← i + 1
fait
fin

```

Exercice 11 – Fonctions numériques

1. Écrire une fonction qui calcule la somme de 3 entiers fournis en arguments

Solution

```
fonction sommeDeTrois(var a, b, c : entier par valeur)
debut
    retourner a+b+c
fin
```

2. Écrire une fonction qui calcule la moyenne des éléments d'un tableau. Le tableau et la taille du tableau sont fournis en argument

Solution

```
fonction moyenne(var t[] : reel par reference, n : entier par valeur)
var s : reel
var i : entier
debut
    pour i allant de 0 a n-1 faire
        s ← s + t[i]
    faire
    retourner s/n
fin
```

3. Écrire une fonction qui affiche la décomposition en facteurs premiers d'un nombre. Indication : le plus petit diviseur strictement supérieur à 1 d'un nombre est nécessairement premier.

Solution

```
fonction decomposition(var n : entier par valeur) : chaîne
var dec : chaîne
    i : entier
debut
    dec = ""
    i ← 2
    tantque i ≤ n faire
        si n % i = 0 alors
            n ← n / i
            dec ← dec + enChaine(i)
            si n ≠ 1 alors
                dec ← dec + "*"
            finsi
        sinon
            i ← i + 1
        finsi
    fait
    retourner dec
fin
```

4. En utilisant la fonction PGCD vue en cours, écrire une fonction PPCM. On rappelle que $PPCM(a, b) = \frac{|ab|}{PGCD(a, b)}$.

Solution

```
fonction PPCM(var a, b : entier par valeur)
var m : entier
debut
    m ← a * b
    si m < 0 alors
        m ← -m
    finsi
    retourner m / PGCD(a, b)
fin
```

Exercice 12 – Fonctions de traitement de chaînes de caractères

On donne le type chaîne pour représenter les chaînes de caractères. On utilisera l'opérateur + pour concaténer des chaînes et on donne les 2 fonctions suivantes :

fonction *taille*(**var** *s* : chaîne **par** variable) : renvoie la taille de la chaîne, en entier.

fonction *charAt*(**var** *s* : chaîne **par** variable, **var** *i* : entier **par** valeur) : renvoie le caractère numéro *i* de la chaîne.

1. Écrire une fonction *contient* qui prend 2 chaînes en paramètres et renvoie vrai si la seconde est incluse dans la première.

Solution

```
fonction contient(var s, c : chaîne par référence)
var i, j : entier
debut
  i ← 0
  tantque i + taille(c) ≤ taille(s) faire
    j ← 0
    tantque j < taille(c) et charAt(s, i + j) = charAt(c, j) faire
      j ← j + 1
    fait
    si j = taille(c) alors
      retourner vrai
    finsi
    i ← i + 1
  fait
  retourner faux
fin
```

2. Écrire une fonction qui purge une chaîne d'un caractère, la chaîne comme le caractère étant passés en argument. Si le caractère spécifié ne fait pas partie de la chaîne, celle-ci devra être retournée intacte. Par exemple :
 - *purge*("Bonjour", "o") renverra "Bnjur".
 - *purge*("J'aime pas les espaces", " ") renverra "J'aimepaslesespaces".
 - *purge*("Moi, je m'en fous", "y") renverra "Moi, je m'en fous".

Solution

```
fonction purge(var s : chaîne par référence, var c : car par valeur)
var s2 : chaîne
  i : entier
debut
  s2 ← ""
  pour i allant de 0 à taille(s) - 1 faire
    si charAt(s, i) ≠ c alors
      s ← s + charAt(s, i)
    finsi
  fait
  retourner s2
fin
```

3. Écrire deux fonctions *start* et *end* qui prennent 1 chaîne *s* et un entier *n* en paramètres et qui renvoient une chaîne contenant respectivement les *n* premiers caractères et les *n* derniers caractères de *s*

Solutions

```
fonction start(var s : chaîne par référence, var n : entier par valeur)
var s2 : chaîne
debut
  s2 ← ""
  if n > taille(s) alors
    n = taille(s)
  finsi
  pour i allant de 0 à n faire
    s2 ← s2 + charAt(s, i)
```

```

fait
retourner s2
fin

```

```

fonction end(var s : chaine par reference, var n : entier par valeur)
var s2 : chaine
debut
  s2 ← ""
  if n > taille(s) alors
    n = taille(s)
  finsi
  pour i allantde 0 a n faire
    s ← charAt(s, taille(s)-i-1) + s
  fait
  retourner s2
fin

```

4. Écrire une fonction middle qui prend 1 chaîne et 2 entiers p et q en paramètres et qui renvoie la partie de la chaîne comprise entre les caractères p et q. On utilisera les fonctions définies précédemment.

Solution

```

fonction middle(var s : chaine par reference, var p, q : entier par valeur)
debut
  si p ≥ q alors retourner ""
  retourner end(start(s, q), q-p)
fin

```

Exercice 13 – Fonctions récursives

1. Écrire un algorithme de PGCD récursif. On rappelle que si $q < p$ alors $PGCD(p, q) = PGCD(p - q, q)$.

Solution

```

fonction PGCD(var p, q : entier par valeur)
debut
  si p = q alors
    retourner p
  finsi
  si q < p alors
    retourner PGCD(p - q, q)
  finsi
  retourner PGCD(q - p, p)
fin

```

2. On reprend la Suite de Fibonacci, qu'on a déjà utilisée dans l'exercice 6 sur les suites.

$$\begin{cases} u_0 = 0, & u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

- (a) Écrire une fonction récursive de calcul du terme de rang n de cette suite. Qu'en pensez vous ?

Solution

```

fonction fibo(var n : entier par valeur)
debut
  si n < 2 alors
    retourner n
  finsi
  retourner fibo(n-1) + fibo(n-2)
fin

```

Ce n'est pas efficace du tout. La complexité est exponentielle.

L'algo itératif vu dans l'exercice 6 est bien meilleur (complexité linéaire).

Algorithmique – Travaux Dirigés

Master “*Technologie et Handicap*” : Intensifs 1

Corrigé additif

Exercice 13 – Fonctions récursives (*suite*)

2. Suite de Fibonacci (*suite*)

(b) On peut montrer que :

$$\begin{cases} u_{2k} = 2u_{k-1}u_k + u_k^2 = (2u_{k-1} + u_k)u_k \\ u_{2k+1} = u_{k+1}^2 + u_k^2 \end{cases}$$

En déduire une nouvelle fonction récursive. Est-elle plus efficace ?

Solution

```
fonction fibo(var n : entier par valeur)
debut
  si n < 2 alors
    retourner n
  finsi

  si n % 2 = 0 alors // cas où n est pair
    u_k1 = fibo(n/2-1)
    u_k0 = fibo(n/2)
    retourner (2*u_k1 + u_k0) * u_k0
  sinon
    u_k1 = fibo3(n/2+1)
    u_k0 = fibo3(n/2)
    retourner u_k1 * u_k1 + u_k0 * u_k0
  finsi
fin
```

*C'est le meilleur algorithme connu pour calculer les termes de la suite de Fibonacci.
(La complexité est en $\log(n)$.)*

Exercice 14 – Problème des 8 reines

Il s'agit de trouver les différentes façons de placer 8 reines sur un échiquier (8x8 cases) sans qu'elles ne se menacent mutuellement, conformément aux règles du jeu d'échecs (on ne tient pas compte de la couleur des pièces).