HAUTE ECOLE DE BRUXELLES
Rue Royale 67 – 1000 Bruxelles
02/219.15.46 – esi@heb.be

Bachelor en Informatique

haute école de bruxelles

Logique

et

Techniques de programmation

1ère année

(SOLUTIONS – NE PAS DIFFUSER !)

2008 – 2009

révision mer. 20 août 08

Sommaire

Chapitre 1
Qu'est-ce qu'un algorithme ?

Étapes de la résolution d'un problème

Exercice : Un problème flou

On pourrait citer :
- Quels sont les nombres sur lesquels travailler ?
- La moyenne doit-elle être entière aussi ?
- Faut-il tenir compte du cas où il y a 0 nombre ? Que faire dans ce cas ?

Procédures de résolution

Exercice : Le dessin

A propos des questions de réflexion, bien faire faire passer le message qu'un ordinateur ne **comprend pas** ce qu'il fait. Par exemple, il ne comprendra pas qu'il est en train de calculer une moyenne.

Chapitre 2
Les bases de la logique

Les alternatives

Exercice : avancer de deux cases (amélioré)

```
Début
    Si non devantMur alors
        Avancer
        Si non devantMur alors
            Avancer
        Sinon
        Fin si
    Sinon
    Fin si
Fin
```

La répétition

Exercice : avancer jusqu'au mur

```
Début
    Tant que non devantMur faire
        Avancer
    Fin tant que
Fin
```

La mémoire du robot

Exercice : avancer jusqu'au mur et revenir

```
Début
    DéposerMarque
    Tant que non devantMur faire
        Avancer
    Fin tant que
    ADroite
    ADroite
    Tant que non surMarque faire
        Avancer
    Fin tant que
    ADroite
    ADroite
    EnleverMarque
Fin
```

La notion de procédure

EXERCICE : PROCÉDURE POUR LE DEMI-TOUR

```
Procédure DemiTour
Début
    ADroite
    ADroite
Fin
```

```
Début
    DéposerMarque
    Tant que non devantMur faire
        Avancer
    Fin tant que
    DemiTour
    Tant que non surMarque faire
        Avancer
    Fin tant que
    DemiTour
    EnleverMarque
Fin
```

Exercices

Exercices de synthèse

Ex. 1 Tour du domaine

```
Procédure AvancerJusqueMur
Début
    Tant que non devantMur faire
        Avancer
    Fin tant que
Fin
```

```
Début
    AvancerJusqueMur
    ADroite
    AvancerJusqueMur
    ADroite
    AvancerJusqueMur
    ADroite
    AvancerJusqueMur
    ADroite
Fin
```

Ex. 2 Trouver le trésor

1ère solution. D'abord tester si on n'est pas SUR le trésor. Mais il faut bouger pour le savoir. Sinon, cas général.

```
Procédure AvancerJusqueTrésor
Début
    Tant que non devantTrésor faire
        Avancer
    Fin tant que
Fin
```

```
Début
    Avancer
    DemiTour
    Si devantTrésor alors
        Avancer
    Sinon
        Avancer
        DemiTour
        AvancerJusqueTrésor
        Avancer
    Fin si
Fin
```

2ème solution : faire le cas général. Si le trésor n'est pas trouvé, c'est qu'on était dessus au départ.

```
Procédure AvancerJusqueTrésorOuMur
Début
    Tant que non devantMur ET non devantTrésor faire
        Avancer
    Fin tant que
Fin
```

```
Début
    AvancerJusqueTrésorOuMur
    Si non devantTrésor alors
        DemiTour
        AvancerJusqueTrésor
    Sinon
    Fin si
    Avancer
Fin
```

Ex. 3 Trouver le trésor (2°)

```
Début
    AvancerJusqueMur
    ADroite
    // suite idem problème 2
Fin
```

Exercices de réflexion

Ex. 4 Un algorithme correct

Bien faire comprendre que ce n'est pas parce que le programme fonctionne dans les cas testé qu'il fonctionne dans tous les cas (problème des cas particuliers) -> la

difficulté d'affirmer qu'un algorithme est correct.

Ex. 5 Unicité d'un algorithme

La comparaison des solutions proposées pour les exercices précédents suffit à s'en convaincre. Les faire réfléchir sur la question suivante : « Si plusieurs solutions sont correctes, est-ce qu'il y en a des meilleures que d'autres ? ».

Ex. 6 Un problème insoluble

Des problèmes qui demandent des actions que le robot ne sait pas faire ou une mémoire qu'il n'a pas : enlever le trésor, compter le nombre de marques, ...

Exercices complémentaires

Ex. 7 Trouver le trésor (3°)

```
Début
    AvancerJusqueMur
    ADroite
    Tant que non devantTrésor faire
        Si devantMur alors
            ADroite
        Sinon
        Fin si
        Avancer
    Fin tant que
    Avancer
Fin
```

Ex. 8 Marquer le domaine

Situation initiale : le robot en haut à gauche et regarde par la droite. On le remplit en spirale.

```
Procédure MarquerJusqueMurOuMarque
Début
    Tant que non devantMur ET non devantMarque faire
        DéposerMarque
        Avancer
    Fin tant que
Fin
```

```
Début
    Tant que non surMarque faire
        MarquerJusqueMurOuTrésor
        Adroite
        Si devantMarque alors
            DéposerMarque
        Sinon
        Fin si
    Fin tant que
Fin
```

Ex. 9 Trouver le trésor (4°)

```
Procédure Gauche
Début
    ADroite
    ADroite
    ADroite
Fin
```

```
// Avance de 2 lignes jusqu'au trésor ou au bout
Procédure DeuxLignesJusqueTrésor
Début
    AvancerJusqueTrésorOuMur
    Si non devantTrésor alors
        Droite
        Si non devantMur alors
            Avancer
            Droite
            AvancerJusqueTrésorOuMur
            Si non devantTrésor alors
                Gauche
                Si non devantTrésor alors
                    Avancer
                    Gauche
                Sinon
                Finsi
            Sinon
            Finsi
        Sinon
        Finsi
    Sinon
    Finsi
Fin
```

```
Début
    Avancer
    DemiTour
    Si devantTrésor alors
        Avancer
    Sinon
        Avancer
        DemiTour
        Tant que non devant Trésor faire
            DeuxLignesJusqueTrésor
        Fin tant que
        Avancer
    Fin si
Fin
```

Ex. 10 Trouver le trésor (5°)

L'idée est de se placer d'abord dans un coin pour revenir au problème précédent.

```
Début
    AvancerJusqueMur
    ADroite
    AvancerJusqueMur
    ADroite
    // On revient au problème précédent. Copier-coller la solution
Fin
```

Il est possible qu'on passe sur le trésor en allant au coin -> inefficacité de la logique. Pour remédier à cela, il faudrait utiliser **AvancerJusqueMurOuTrésor** et ajouter des tests d'arrêt.

Chapitre 3
Algorithmes séquentiels

Variables et types

Exercice : Une date

Ce qui vient d'abord à l'esprit c'est 3 entiers : jour, mois, année.

On peut envisager de représenter le mois avec une chaîne ou toute la date avec une chaîne. C'est plus rapide pour l'affichage (quoique peu souple) mais beaucoup plus lent pour les calculs.

On peut aussi envisager un seul entier, le nombre de jours écoulés depuis une date de référence. C'est le plus rapide pour les calculs mais plus lent pour les affichages.

Exercice : Un moment

Mêmes réponses avec l'entier représentant le nombre de secondes depuis minuit.

Exercice : Déclarer une date

```
jourAnniversaire, moisAnniversaire, annéeAnniversaire : Entiers
```

Exercice : Déclarer un rendez-vous

```
heureDébut, minuteDébut, heureFin, minuteFin : Entiers
motif : chaîne
```

Exercices

Pour s'échauffer

Ex. 1 Compréhension d'algorithme

- A : 8
- B : 1.5
- C : 2
- D : 5
- E : 5

Ex. 2 Le jeu des *n* erreurs

A : nombre non lus B : m non déclaré

Ex. 3 Simplification d'algorithme

```
hauteur ← 7
```

```
var ← 0
```

```
lire var
```

```
ok2 ← faux
```

Exercices d'apprentissage

Ex. 4 Surface d'un triangle

```
module surfaceTriangle
    base, hauteur : réels
    lire base, hauteur
    surface ← base*hauteur/2
    écrire surface
fin module
```

Ex. 5 Placement

```
module capital
    capital, taux, intérêt : réels
    lire capital, taux
    intérêt ← capital * taux /100
    écrire capital + intérêt
fin module
```

Ex. 6 Prix TTC

```
module prixTotal
    prixUnitaire, tauxTVA : réels
    quantité : entier
    lire prixUnitaire, tauxTVA, quantité
    écrire quantité*prixUnitaire*(1 + tauxTVA/100)
fin module
```

Ex. 7 Durée de trajet

```
module véhicule
    vitesse, distance : réels
    lire vitesse, distance
    distance ← distance*1000 // la distance est convertie en mètres
    écrire distance/vitesse
fin module
```

Ex. 8 Permutation

```
// permutation du contenu de a et b
aux ← a
a ← b
b ← aux
```

Ex. 9 Cote moyenne

```
module moyenneCotes
    cot1, pond1, cot2, pond2, cot3, pond3 : entiers
    lire cot1, pond1
    lire cot2, pond2
    lire cot3, pond3
    écrire (cot1*pond1 + cot2*pond2 + cot3*pond3)/(pond1 + pond2 + pond3)*0.05
fin module
```

Ex. 10 Somme des chiffres

```
module sommeDesChiffres
    n, centaines, dizaines, unités : entiers
    lire n          // on suppose que n est entre 100 et 999
    centaines ← n DIV 100
    dizaines ← (n MOD 100) DIV 10
    unités ← n MOD 100
    écrire centaines + dizaines + unités
fin module
```

Ex. 11 Conversion heure en secondes

```
module nombreSecondes
    heure, minute, seconde : entiers
    lire heure, minute, seconde
    écrire 3600*heure + 60*minute + seconde
fin module
```

Ex. 12 Conversion secondes en heures

```
module formatHMS
    totalSecondes, heure, minute, seconde : entiers
    lire totalSecondes
    heure ← totalSecondes DIV 3600
    minute ← (totalSecondes MOD 3600) DIV 60
    seconde ← totalSecondes MOD 60
    écrire heure, minute, seconde
fin module
```

Chapitre 4
Les alternatives

« Si – alors - sinon »

Exercice : Signe d'un nombre (amélioré)

```
Module SigneNombre
    nb : Entier
    lire nb
    si nb < 0 alors
        écrire « le nombre », nb, « est négatif »
    sinon
        si nb = 0 alors
            écrire « le nombre », nb, « est nul »
        sinon
            écrire « le nombre », nb, « est positif »
        fin si
    fin si
Fin module
```

« Selon-que »

Exercice : Signe d'un nombre (avec selon-que)

```
Module SigneNombre
    nb : Entier
    lire nb
    selon que
        nb < 0 : écrire « le nombre », nb, « est négatif »
        nb = 0 : écrire « le nombre », nb, « est nul »
        nb > 0 : écrire « le nombre », nb, « est positif »
    fin selon que
Fin module
```

Exercice : Nombre de jours dans un mois

```
Module NombreJoursMois
    mois : Entier
    lire mois
    selon que mois vaut
        1, 3, 5, 7, 8, 10, 12 : écrire 31
        4, 6, 9, 11 : écrire 30
        2 : écrire 28
    fin selon que
Fin module
```

Exercices

Ex. 1 Compréhension

	2 et 3	4 et 1
A	2	6
B	1	0
C	105	105

Ex. 2 Simplification d'algorithmes

```
si ok alors
    écrire nombre
fin si
```

```
si non ok alors
    écrire nombre
fin si
```

```
ok ← condition
```

```
si a > b alors
    ok ← faux
sinon
    ok ← vrai
fin si

==> ok ← non a > b
```

```
si ok1 ET ok2 alors
    écrire x
fin si
```

Ex. 3 Équation du deuxième degré

```
module secondDegré // résout l'équation ax² + bx + c = 0
    delta, x1, x2, a, b, c : réels
    lire a, b, c
    delta ← b*b – 4*a*c
    selon que
        delta > 0 :
            x1 ← (–b – √(delta) )/(2*a)
            x2 ← (–b + √(delta) )/(2*a)
            écrire « il y a deux solutions réelles », x1, « et », x2
        delta = 0 :
            x1 ← – b/(2*a)
            écrire « une seule solution réelle », x1
        delta > 0 :
            écrire « l'équation ne possède pas de solution réelle »
    fin selon que
fin module
```

Ex. 4 Maximum de 2 nombres

```
module max2
    a, b : réels
    lire a, b
    si a > b alors
        écrire a
    sinon
        écrire b
    fin si
fin module
```

Ex. 5 Maximum de 3 nombres

Beaucoup d'angles d'approche pour ce problème.

```
module max3
    a, b, c : réels
    lire a, b, c
    si a ≥ b alors        // b ne peut être le plus grand, on départage entre a et c
        si a ≥ c alors
            écrire a
        sinon
            écrire c
        fin si
    sinon                 // a ne peut être le plus grand, on départage entre b et c
        si b ≥ c alors
            écrire b
        sinon
            écrire c
        fin si
    fin si
fin module
```

```
module max3
    a, b, c : réels
    lire a, b, c
    si a ≥ b ET a ≥ c alors
        écrire a
    sinon
        si b ≥ a ET b ≥ c alors
            écrire b
        sinon
            écrire c
        fin si
    fin si
fin module
```

Ex. 6 Test d'intervalle

```
module intervalle
    a, b, c, aux : réels
    // il faut vérifier si a est dans l'intervalle ] b, c [
    lire a, b, c
    si b < a ET a < c OU c < a ET a < b alors      // ≤ si on inclut les bornes
        écrire a, « est dans l'intervalle »
    sinon
        écrire a, « n'est pas dans l'intervalle »
    fin si
fin module
```

Ex. 7 Calcul de salaire

```
module salaireNet
    constante tauxRetenue = 15       // en %
    constante plafond = 1200
    salaireBrut, salaireNet : réels
    lire salaireBrut
    si salaireBrut > plafond alors
        salaireNet ← plafond + (salaireBrut – plafond)*(1 – tauxRetenue/100)
    sinon
        salaireNet ← salaireBrut
    fin si
    écrire salaireNet
fin module
```

Ex. 8 Année bissextile

```
module bissextile
    année : entier
    bissextile : booléen
    lire année
    si (année MOD 4 = 0 ET année MOD 100 ≠ 0) OU année MOD 400 = 0 alors
        bissextile ← vrai
    sinon
        bissextile ← faux
    fin si
    écrire bissextile
fin module
```

ou encore

```
module bissextile
    année : entier
    bissextile : booléen
    lire année
    bissextile ← (année MOD 4 = 0 ET année MOD 100 ≠ 0) OU année MOD 400 = 0
    écrire bissextile
fin module
```

Ex. 9 Valider une date

```
module dateValide
    année, mois, jour : entiers
    ok, bissextile: booléen
    maxJour : entier
    lire année, mois, jour
    ok ← année > 0 et jour > 0 et mois > 0 et mois < 13
    si ok alors // Reste à tester si le mois ne dépasse pas la limite
        // calcul nb jours dans mois : cf. supra
        si mois = 2 alors
            // calcul bissextile : cf. Supra
            si bissextile alors
                maxJour ← maxJour + 1
            fin si
        fin si
        ok ← jour ≤ maxJour
    fin si
    si ok alors
        écrire « date valide »
    sinon
        écrire « date non valide »
    fin si
fin module
```

Chapitre 5
Les modules

Exercices

Ex. 1 Compréhension

- Ex 1 : 7, 8
- Ex 2 : 7, 8
- Ex 3 : 18, 3, 4, 18, 3, 4, 17
- Ex 4 : 3, 4, 19

Ex. 2 Appels de module

OK : 1, 2, 3, 4 à discuter, 6, 7, 9

Ex. 3 Comparaison d'algorithmes

J'éviterais la première et la dernière version pour des questions de lisibilité.

Ex. 4 Validité d'une date

```
module bissextile ( année ↓: entier ) →booléen
   retourner (année MOD 4 = 0 ET année MOD 100 ≠ 0) OU année MOD 400 = 0
fin module

module nombreJoursMois( mois ↓: entier, année ↓: entier ) →entier
   nbJours : Entier
   selon que mois vaut
         1, 3, 5, 7, 8, 10, 12 :  nbJours ← 31
         4, 6, 9, 11  :  nbJours ← 30
         2 :
                  si bissextile( année ) alors
                           nbJours ← 29
                  sinon
                           nbJours ← 28
                  fin si
   fin selon que
   retourner  nbJours
fin module

module dateValide (jour ↓: entier, mois ↓: entier, année ↓: entier ) →booléen
   retourner  année > 0
         ET mois > 0 ET mois ≤ 12
         ET jour > 0 ET jour ≤ nbJoursMois( mois, année )
fin module
```

Chapitre 6
Les boucles

Exercices

Ex. 1 Compréhension d'algorithmes.

- Boucle 1 : 12
- Boucle 2 : 33
- Boucle 3 : 11, 14, 15, 18, 19, 22, 3
- Boucle 4 : 31
- Boucle 5 : 36
- Boucle 6 : 21, 31, 34, 37

Ex. 2 Simplification

```
a ← 1
tant que a < 10 faire
    a ← a + 1
    écrire a
fin tant que
```

```
b ← 10
```

Ex. 3 Maximum de nombres

```
module coteMax
    cote, max : entiers
    max ← 0
    lire cote
    tant que cote ≥ 0 faire
        si cote > max alors
            max ← cote
        fin si
        lire cote
    fin tant que
    écrire « la plus grande cote vaut », max
fin module
```

Ex. 4 Afficher les multiples de 3

```
module multiple3
    nombre, cpt : entiers
    cpt ← 0
    lire nombre
    tant que nombre ≠ 0 faire
        si nombre MOD 3 = 0 alors
            écrire nombre
            cpt ← cpt + 1
        fin si
        lire nombre
    fin tant que
    écrire « le nombre de multiples de 3 est », cpt
fin module
```

Ex. 5 Placement d'un capital

```
module capital( capital, taux : réels, n : entier )
    constante année = 2008   // à adapter chaque année !
    newCapital : réel
    i :entier
    newCapital ← capital
    pour i de 1 à n faire
        newCapital ← newCapital * (1 + taux/100)
        écrire « le 1er janvier », année + i
        écrire « le capital vaudra », newCapital
        écrire « et l'intérêt obtenu sera », newCapital – capital
    fin pour
fin module
```

Ex. 6 Produit de 2 nombres

```
module produitSansMultiplication(a↓, b↓: entiers) → entier
    i, somme : entiers
    somme ← 0
    si a > b alors
        pour i de 1 à b faire
            somme ← somme + a
        fin pour
    sinon
        pour i de 1 à a faire
            somme ← somme + b
        fin pour
    fin si
    retourner somme
fin module
```

Ex. 7 Génération de suites

```
module suite1( N : entier )
    i, nombre : entiers
    nombre ← 1
    pour i de 1 à N faire
        écrire nombre
        nombre ← nombre + i
    fin pour
fin module
```

```
module suite2( N : entier )
   i, nombre : entiers
   nombre ← 1
   pour i de 1 à n faire
         écrire nombre
         si i MOD 2 = 1 alors
               nombre ← nombre + 1
         sinon
               nombre ← nombre + 2
         fin si
   fin pour
fin module

ou encore

module suite2( N : entier )
   i, nombre, pas : entiers
   nombre ← 1
   pas ← 1
   pour i de 1 à N faire
         écrire nombre
         nombre ← nombre + pas
         pas ← 3 - pas
   fin pour
fin module
```

```
module suite3( N : entier )              // Fibonaci
   x, y, z : entiers
   si N ≥ 1 alors       // on écrit le premier
         écrire 0
         x ← 0
   fin si
   si N ≥ 2 alors       // on écrit le deuxième
         écrire 1
         y ← 1
   fin si
   pour i de 3 à N faire        // on ne rentre dans cette boucle que si n ≥ 3
         z ← x + y
         écrire z
         x ← y
         y ← z
   fin pour
fin module
```

```
module suite4( N : entier )
   i, nombre, pas : entiers
   nombre ← 1
   pas ← 1
   pour i de 1 à N faire
         écrire nombre
         si i MOD 10 = 0 alors
               nombre ← nombre + 10
               pas ← – pas
         sinon
               nombre ← nombre + pas
         fin si
   fin pour
fin module
```

```
module suite5( N : entier )
    i, nombre : entiers
    nombre ← 1
    pour i de 1 à N faire
        écrire nombre
        selon que i MOD 5 vaut
            1, 2, 3 : nombre ← nombre + 1
            0, 4 :    nombre ← nombre – 1
        fin si
    fin pour
fin module
```

```
module suite6( N : entier )
    // il s'agit de l'alternance des deux suites 1, 3, 5, 7, 9,… et 2, 3, 4, 5, 6,…
    x, y, i : entiers
    x ← 1
    y ← 2
    pour i de 1 à N faire
        si i MOD 2 = 1 alors
            écrire x
            x ← x + 2
        sinon
            écrire y
            x ← x + 1
        fin si
    fin pour
fin module
```

Ex. 8 Factorielle

```
module factorielle(n↓ : entier) → entier
    produit, i : entiers
    produit ← 1
    pour i de 2 à n faire
        produit ← produit * i
    fin pour
    retourner produit
fin module
```

Ex. 9 Somme de chiffres

```
module sommeChiffres(n↓ : entier) → entier
    somme, chiffre : entiers
    somme ← 0
    tant que n ≠ 0 faire
        chiffre ← n MOD 10
        somme ← somme + chiffre
        n ← n DIV 10
    fin tant que
    retourner somme
fin module
```

Ex. 10 Conversion binaire-décimale

```
module binaireVersDécimal(n↓: entier) → entier    // n ne contient que des 0 et des 1
    somme, chiffre, puis : entiers
    somme ← 0
    puis ← 1      // contiendra les puissances de 2
    tant que n ≠ 0 faire
        chiffre ← n MOD 10
        somme ← somme + chiffre*puis
        puis ← puis * 2
        n ← n DIV 10
    fin tant que
    retourner somme
fin module
```

```
// version avec signalement d'erreur
module binaireVersDécimal(n↓: entier, ok↑ : booléen) → entier
    // ok recevra faux si l'entier n contient un chiffre autre que 0 ou 1
    somme, chiffre, puis : entiers
    somme ← 0
    puis ← 1      // contiendra les puissances de 2
    ok ← vrai
    tant que n ≠ 0 ET ok faire
        chiffre ← n MOD 10
        ok ← chiffre = 0 OU chiffre = 1
        somme ← somme + chiffre*puis
        puis ← puis * 2
        n ← n DIV 10
    fin tant que
    retourner somme
fin module
```

Ex. 11 Conversion décimale-binaire

```
module DécimalVersBinaire(n↓: entier) → entier
    somme, reste, puis : entiers
    somme ← 0
    puis ← 1      // contiendra les puissances de 10
    tant que n ≠ 0 faire
        reste ← n MOD 2
        somme ← somme + reste*puis
        puis ← puis * 10
        n ← n DIV 2
    fin tant que
    retourner somme
fin module
```

Ex. 12 PGCD

```
module PGCD(a↓, b↓: entiers) → entier
    c : entier
    faire
        c ← a
        a ← b
        b ← c MOD b
    jusqu'à ce que b = 0
    retourner a
fin module
```

Ex. 13 PPCM

```
module PPCM(a↓, b↓: entiers) → entier
    c, mutliple : entiers
    si a < b alors
        c ← a
        a ← b
        b ← c
    fin si
    // on recherche le premier multiple de a divisible par b
    multiple ← a
    tant que multiple MOD b ≠ 0 faire
        multiple ← multiple + a
    fin tant que
    retourner multiple
fin module
```

Variante faisant appel au module PGCD

```
module PPCM(a↓, b↓: entiers) → entier
    retourner a * b / PGCD(a,b)
fin module
```

Ex. 14 Nombre premier.

```
module premier(n↓: entier) → booléen
    diviseur : entier
    divisible : booléen          // devient vrai dès qu'un diviseur de n est trouvé
    si n = 1 alors
        retourner faux
    sinon
        divisible ← n MOD 2 = 0      // on teste la divisibilité par 2
        diviseur ← 3                 // ensuite, on teste les diviseurs impairs
        tant que NON divisible ET diviseur < √(n) faire
            divisible ← n MOD diviseur = 0
            diviseur ← diviseur + 2
        fin tant que
        retourner NON divisible
    fin si
fin module
```

Ex. 15 Nombres premiers.

```
module listeNombresPremiers(n↓ : entier)
    i : entier
    pour i de 2 à n faire
        si premier(i) alors
            écrire i
        fin si
    fin pour
fin module
```

Ex. 16 Nombre parfait.

Première version "directe": on calcule la somme des diviseurs jusqu'à la moitié du nombre, et on vérifie à la fin si cette somme est égale au nombre donné.

```
module parfait( n↓: entier ) → booléen
    d, somme : entiers
    somme ← 0
    pour d de 1 à n DIV 2 faire
        si n MOD d = 0 alors
            somme ← somme + d
        fin si
    fin pour
    retourner somme = n
fin module
```

1ère amélioration: dans le cas de nombres ayant beaucoup de diviseurs, la somme peut dépasser n, on arrête alors l'algorithme lorsque somme > n. Exemple pour n=36: 1 + 2 + 3 + 4 + 6 + 9 + 12 = 37, donc inutile d'ajouter encore 18.

```
module parfait( n↓: entier ) → booléen
    d, somme : entiers
    somme ← 0
    d ← 1
    tant que d ≤ n DIV 2 ET somme < n faire
        si n MOD d = 0 alors
            somme ← somme + d
        fin si
        d ← d + 1
    fin tant que
    retourner somme = n
fin module
```

2ème amélioration: le nombre d'itérations dans la 1ère version est de n DIV 2, et dans la 2ème version, au maximum n DIV 2. Comme dans l'algorithme de détermination d'un nombre premier, on peut s'arrêter à la racine carrée, en groupant les diviseurs associés par 2 (d et n DIV d). Exception: il ne faut pas grouper 1 avec n, et aussi √(n) dans le cas où n est un carré parfait, pour ne pas le compter deux fois:

1. somme des diviseurs de 28 (sauf 28) = 1 + (2 + 14) + (4 + 7)
2. somme des diviseurs de 36 (sauf 36) = 1 + (2 + 18) + (3 + 12) + (4 + 8) + 6

```
module parfait( n↓: entier ) → booléen
    d, somme : entiers
    somme ← 1
    d ← 2
    tant que d < √(n) ET somme < n faire
        si n MOD d = 0 alors
            somme ← somme + d + n MOD d
        fin si
        d ← d + 1
    fin tant que
    si d * d = n alors
        somme ← somme + d
    fin si
    retourner somme = n ET n ≠ 1    // l'algorithme général ne fonctionne plus pour 1
fin module
```

Ex. 17 Décomposition en facteurs premiers.

Pour la décomposition en facteurs premiers, il faut compter combien de fois le nombre est divisible par les diviseurs premiers à partir de 2, et le diviser au fur et à mesure. Pour simplifier, on peut prendre comme diviseurs 2 et puis les impairs 3, 5, 7, 9, 11, 13... La division par 9 ne donnera forcément rien, puisqu'on aura déjà testé la division par 3, seuls les entiers premiers auront encore un rôle à jouer.

```
module décomposition(n↓: entier)
    d, cpt : entiers
    écrire « décomposition de », n, « en facteurs premiers »
    d ← 2                    // premier diviseur test
    tant que n > 1 faire
        cpt ← 0
        tant que n MOD d  = 0 faire
            cpt ← cpt + 1
            n ← n DIV d
        fin tant que
        écrire « le nombre est divisible », cpt, « fois par », d
        si d = 2 alors           // calcul du diviseur suivant
            d ← 3
        sinon
            d  ← d + 2
        fin si
    fin tant
fin module
```

Ex. 18 Palindrome.

```
module palindrome( n↓: entier ) → booléen
    // on construit le nombre « miroir » de n
    m, reste, puis : entiers
    puis ← 1
    m ← 0
    tant que n ≠ 0 faire
        reste ← n MOD 10
        m ← m + reste * puis
        puis ← puis * 10
        n ← n DIV 10
    fin tant que
    retourner m = n
fin module
```

Ex. 19 Jeu de la fourchette.

```
module jeuFourchette()
    nombre, essai, proposition : entiers
    trouvé : booléen     // devient vrai quand le joueur a trouvé le nombre
    nombre ← hasard()
    trouvé ← faux
    essai ← 0
    faire
        lire proposition
        selon que
            proposition > nombre :      écrire «  nombre donné trop grand »
            proposition < nombre :      écrire «  nombre donné trop petit »
            proposition = nombre :      trouvé ← vrai
        fin selon que
        essai ← essai +1
    jusqu'à ce que trouvé OU essai = 8
    si trouvé alors
        écrire « bravo, vous avez trouvé en », essai, « essais »
    sinon
        écrire « désolé, vous avez épuisé vos huit essais, le nombre était », nombre
    fin si
fin module
```

Exercices récapitulatifs

Ex. 20 Dates valides.

```
module DatesValide()
    nbDates, jour, mois, année, i : entiers
    lire nbDates
    pour i de 1 à nbDates faire
        lire jour, mois, année
        afficherDate( jour, mois, année )
    fin pour
fin module

module afficherDate( jour, mois, année : entiers )
    si dateValide( jour, mois, année ) alors   // déjà écrit
        écrire jour, moisChaîne( mois ), année
    sinon
        écrire « date invalide »
    fin si
fin module

module moisChaîne( mois : entiers ) → Chaîne
    résultat : Chaîne
    selon que mois vaut
        1 : résultat ← « Janvier »
        2 : résultat ← « Février »
        ...
        11 : résultat ← « Novembre »
        12 : résultat ← « Décembre »
    fin selon que
    retourner  résultat
fin module
```

Ex. 21 IMC.

```
module IMC( poids, taille : réels ) → Réel
    retourner  poids / taille / taille
fin module

module corpulence( IMC : réel, sexe : caractère ) → Chaîne
    // sexe = 'H' ou 'F'
    corpulence : Chaîne
    si sexe = 'H' alors
        selon que
                                IMC < 20 : corpulence ← « maigre »
                20 ≤ IMC ET IMC < 25 : corpulence ← «normale»
                25 ≤ IMC ET IMC < 30 : corpulence ← «excès pondéral»
                30 ≤ IMC             : corpulence ← «obèse»
        fin selon que
    sinon
        selon que
                                IMC < 19 : corpulence ← « maigre »
                19 ≤ IMC ET IMC < 24 : corpulence ← «normale»
                24 ≤ IMC ET IMC < 30 : corpulence ← «excès pondéral»
                30 ≤ IMC             : corpulence ← «obèse»
        fin selon que
    fin si
    retourner  corpulence
fin module

module CorpulencePersonnes()
    nbPersonnes  : entiers
    sexe : caractère
    poids, taille : réels
    lire  nbPersonnes
    pour i de 1 à nbPersonnes faire
        lire sexe, poids,  taille
        écrire corpulence( IMC(poids,taille), sexe )
    fin pour
fin module
```

Ex. 22 Cotes.

```
module résultatEtudiant(cote1, cote2, cote3  : entiers)
    nbInterros, sommeInterros, totalExamen, minimumExamen : entiers
    sommeInterros ← 0
    nbInterros ← 0
    si cote1 > -1 alors
        nbInterros ← nbInterros + 1
        sommeInterros ← sommeInterros + cote1
    fin si
    si cote2 > -1 alors
        nbInterros ← nbInterros + 1
        sommeInterros ← sommeInterros + cote2
    fin si
    si cote3 > -1 alors
        nbInterros ← nbInterros + 1
        sommeInterros ← sommeInterros + cote3
    fin si
    écrire « cote moyenne », sommeInterros / nbInterros
    totalExamen ← 160 – 20 * nbInterros
    minimumExamen ← 96 – sommeInterros
    écrire « cote minimale examen », minimumExamen / totalExamen * 100
fin module
```

```
module résultatsEtudiant()
    nbEtudiants, cote1, cote2, cote3  : entiers
    lire   nbEtudiants
    pour i de 1 à nbEtudiants faire
          lire cote1, cote2,  cote3
          résultatEtudiant( cote1, cote2, cote3 )
    fin pour
fin module
```

Chapitre 7
Les variables structurées

Exercices sur les structures

Ex. 1 Conversion secondes-moment

```
module SecVersHMS( totalSecondes : entier ) → Moment
    moment : Moment
    moment.heure ← totalSecondes DIV 3600
    moment.minute ← (totalSecondes MOD 3600) DIV 60
    moment.secondes ← totalSecondes MOD 60
    retourner moment
fin module
```

Ex. 2 Conversion moment-secondes

```
module SecVersHMS( moment : Moment ) → entier
    retourner  3600 * moment.heure + 60 moment.minute + moment.seconde
fin module
```

Ex. 3 Écart entre 2 moments

```
module SecVersHMS( début, fin: Moment ) → entier
    retourner  SecVersHMS(fin) - SecVersHMS(début)
fin module
```

Ex. 4 Milieu de 2 points

```
module milieu( point1, point2 : Point ) → Point
    milieu : Point
    milieu.x ← ( point1.x + point2.x ) /2
    milieu.y ← ( point1.y + point2.y ) /2
    retourner milieu
fin module
```

Ex. 5 Distance entre 2 points

```
module distance( point1, point2 : Point ) → réel
    distX, distY : réel
    distX ← ( point2.x - point1.x )
    distY ← ( point2.y - point1.y )
    retourner  √( distX² + distY² )
fin module
```

Ex. 6 Un rectangle

```
structure Rectangle
    bordSG : Point      // bord supérieur gauche
    bordSD : Point      // bord supérieur droit
    bordID : Point      // bord inférieur droit
fin structure
```

```
module longueur1( rectangle : Rectangle ) → réel
    retourner distance( rectangle.bordSG, rectangle.bordSD)
fin module

module longueur2( rectangle : Rectangle ) → réel
    retourner distance( rectangle.bordID, rectangle.bordSD)
fin module

module périmètre( rectangle : Rectangle ) → réel
    retourner 2 * longueur1(rectangle) + 2 * longueur2(rectangle)
fin module

module surface( rectangle : Rectangle ) → réel
    retourner longueur1(rectangle) * longueur2(rectangle)
fin module
```

Chapitre 8
L' orienté objet

La notion d'objet

EXERCICES - ATTRIBUTS

1. trois entiers représentant le jour (1 à 31) le mois (1 à 12) et l'année
2. un entier (1 à 6) représentant la face visible du dé
3. des attributs pour l'état non variable de la télé. Par exemple : ses dimensions (3 réels), la taille de l'écran (réel), couleur ou N/B (booléen), le nombre de stations (entier), technologie (une énumération avec CATHODIQUE, PLASMA, ...)
   Et aussi des attributs pour l'état variable : allumé ou éteint (un booléen), la station en cours (un entier), les éléments de réglage comme la luminosité, l'intensité des couleurs, ...

EXERCICE - COMPORTEMENT

Comportement d'un téléviseur

- l'interroger sur son état permanent (dimension, ...)
- l'interroger sur son état variable : est-elle allumée ? , ...
- l'allumer/l'éteindre
- changer de chaîne
- demander/changer le volume
- modifier les réglages (modifie la luminosité, le niveau des couleurs, ...)

EXERCICES - MÉTHODES

```
// allumé : booléen est l'attribut choisi pour représenter l'état allumé/éteint
          du téléviseur
méthode allumer()
   allumé ←vrai
fin méthode

méthode éteindre()
   allumé ←faux
fin méthode
```

```
// jour, mois, année sont les attributs (entiers)
méthode jourSuivant()
    si jour < nbJoursMois() alors
        jour ←jour + 1
    sinon
        jour ←1
        si mois < 12 alors
            mois ←mois + 1
        sinon
            mois ←1
            année ←année + 1
        fin si
    fin si
fin méthode

// Ces méthodes feraient aussi probablement partie du comportement
méthode nbJoursMois() →entier
    nbJours : entier
    selon que mois vaut
        1, 3, 5, 7, 8, 10, 12 :
            nbJours ←31
        4, 6, 9, 11 :
            nbJours ←30
        2 :
            si estBisextile() alors
                nbJours ←29
            sinon
                nbJours ←28
            fin si
    fin selon que
    retourner nbJours
fin méthode

méthode estBissextile() →booléen
    retourner ( multipleDe( année, 4 ) ET non multipleDe( année, 100 ) )
        OU multipleDe( année, 400 )
fin méthode

// Ce module est d'intérêt général et sert à la lisibilité du code
module multipleDe( nombre, diviseur : entiers ) →booléen
    retourner ( nombre MOD diviseur ) = 0
fin module
```

EXERCICE – ACTIVER UN COMPORTEMENT

```
maTélévision.allumer()
maTélévision.éteindre()
```

EXERCICES – PARAMÈTRES DU COMPORTEMENT

```
énumération JourSemaine est
        {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE}
méthode getJourSemaine() →JourSemaine
méthode setAnnée( nouvelleAnnée : entier )
méthode setDate ( nouveauJour, nouveauMois, nouvelleAnnée : entiers )
méthode estAntérieure ( autreDate : Date ) →booléen
méthode écart ( autreDate : Date ) →entier
```

```
date1.setDate( 12, 11, 2007 )
date2.setDate( 15, 2, 2008 )
si date1.estAntérieure( date2 ) alors
   afficher « date1 est avant date2 »
sinon
   afficher « date2 est avant date1 »
fin si
écrire date1.écart( date2 )
```

```
méthode getTailleEcran() →réel
méthode estAllumée() →booléen
méthode getChaîne() →entier
méthode setChaîne( uneChaîne : entier )
méthode getVolume() →entier
méthode setVolume( vol : entier )

si non maTélévision.estAllumée() alors
   maTélévision.allumer()
fin si
maTélévision.setVolume(8)
maTélévision.setChaîne(3)
maTélévision.éteindre()
```

L'encapsulation

EXERCICE - ENCAPSULATION

???

La notion de classe et d'instance

EXERCICES – CLASSE ET INSTANCE

```
classe Date
   privé :
        jour, mois, année : entiers
   public :
        // accesseurs
        méthode getJour() →entier
        méthode getMois() →entier
        méthode getAnnée() →entier
        // mutateurs
        méthode setJour( j : entier )
        méthode setMois( m : entier )
        méthode setAnnée( a : entier )
        // autres méthodes
        méthode setDate ( nouveauJour, nouveauMois, nouvelleAnnée : entiers )
        méthode jourSuivant()
        méthode nbJoursMois() →entier
        méthode estBissextile() →booléen
        méthode getJourSemaine() →JourSemaine
        méthode estAntérieure ( autreDate : Date ) →booléen
        méthode écart ( autreDate : Date ) →entier
fin classe
```

```
module écartDates
    date1, date2 : Date
    date1←nouveau Date()
    date1.setDate( 15, 9, 2007 )
    date2←nouveau Date()
    date2.setDate( 30, 6, 2008 )
    afficher date1.écart( date2 )
fin module
```

Les constructeurs

EXERCICES - CONSTRUCTEUR

```
constructeur Date( j, m, a : Entiers )
    setAnnée( a )
    setMois( m )
    setJour ( j )
fin constructeur

constructeur Date( a : Entiers )
    setAnnée( a )
    setMois( 1 )
    setJour ( 1 )
fin constructeur
```

```
module écartDates
    date1, date2 : Date
    date1 ←nouveau Date( 15, 9, 2007 )
    date2 ←nouveau Date( 30, 6, 2008 )
    afficher date1.écart( date2 )
fin module
```

Du choix de la représentation de l'état

EXERCICES – REPRÉSENTATION DE L'ÉTAT

```
// Pour une représentation avec 3 attributs : heure, minute, seconde
méthode getHeure() →entier
    retourner heure
fin méthode

méthode estPlusPetit ( autreMoment : Moment ) →booléen
    retourner  heure < autreMoment.heure
            OU ( heure = autreMoment.heure ET minute < autreMoment.minute )
            OU ( heure = autreMoment.heure
                    ET minute = autreMoment.minute ET seconde < autreMoment.seconde )
fin méthode
```

```
// Pour une représentation avec 1 attribut : totalSecondes
méthode getHeure() →entier
    retourner totalSecondes DIV 3600
fin méthode

méthode estPlusPetit ( autreMoment : Moment ) →booléen
    retourner  totalSecondes < autreMoment.totalSecondes
fin méthode
```

```
module Test
    m1, m2 : Moment
    m1 ←nouveau Moment ( 12, 34, 15 )
    m2 ←nouveau Moment ( 17, 11, 8 )
    si m1.estPlusPetit( m2 ) alors
        afficher « m1 est bien avant m2 »
    sinon
        afficher « m1 n'est pas avant m2 »
    fin si
fin module
```

Quelques éléments de syntaxe

EXERCICE – MÉTHODE ÉGAL()

```
// dépend de la représentation
méthode égal( autre : Moment ) →booléen
    retourner heure = autre.heure ET minute = autre.minute
                ET seconde = autre.seconde
    // ou retourner totalSecondes = autre.totalSecondes
fin méthode
```

Exercices

Ex. 1 La date

```
classe Date
    privé :
        jour, mois, année : entiers
    public :
        // constructeurs
        constructeur Date( j, m, a : entiers )
        constructeur Date( a : entiers )
        // accesseurs
        méthode getJour() →entier
        méthode getMois() →entier
        méthode getAnnée() →entier
        méthode setJour( j : entier )
        méthode setMois( m : entier )
        méthode setAnnée( a : entier )
        // autres méthodes
        méthode setDate ( nouveauJour, nouveauMois, nouvelleAnnée : entiers )
        méthode jourSuivant()
        méthode nbJoursMois() →entier
        méthode estBissextile() →booléen
        méthode getJourSemaine() →JourSemaine
        méthode estAntérieure ( autreDate : Date ) →booléen
        méthode écart ( autreDate : Date ) →booléen
    privé :
        méthode nbJoursDepuis1900 () →entier
fin classe

constructeur Date( j, m, a : entiers )
    setDate( j, m, a )
fin constructeur

constructeur Date( a : entiers )
    setDate( 1, 1, a )
fin constructeur
```

```
méthode getJour() →entier
    retourner jour
fin méthode

méthode getMois() →entier
    retourner mois
fin méthode

méthode getAnnée() →entier
    retourner année
fin méthode

méthode méthode setJour( j : entier )
    si j < 1 OU j > nbJoursMois() alors erreur « jour invalide » fin si
    jour ←j
fin méthode

méthode méthode setMois( m : entier )
    si m < 1 OU m > 12 alors erreur « mois invalide » fin si
    mois ←m
fin méthode

méthode méthode setAnnée( a : entier )
    année ←a
fin méthode

méthode setDate ( nouveauJour, nouveauMois, nouvelleAnnée : Entiers )
    setAnnée( nouvelleAnnée  )
    setMois( nouveauMois )
    setJour ( nouveauJour )
fin méthode

méthode jourSuivant()
    si jour < nbJoursMois() alors
        jour ←jour + 1
    sinon
        jour ←1
        si mois < 12 alors
            mois ←mois + 1
        sinon
            mois ←1
            année ←année + 1
        fin si
    fin si
fin méthode

méthode nbJoursMois() →entier
    retourner nbJoursMois( mois, année )
fin méthode

méthode estBissextile() →booléen
    retourner estBissextile( année )
fin méthode

méthode getJourSemaine() →JourSemaine       // Le 1 janvier 1900 tombe un lundi
    selon que nbJoursdepuis1900() MOD 7 vaut
        0 : retourner LUNDI
        1 : retourner MARDI
        2 : retourner MERCREDI
        3 : retourner JEUDI
        4 : retourner VENDREDI
        5 : retourner SAMEDI
        6 : retourner DIMANCHE
    fin selon que
fin méthode
```

```
méthode estAntérieure ( autreDate : Date ) →booléen
    retourner nbJoursDepuis1900() < autreDate.nbJoursDepuis1900()
fin méthode

méthode écart ( autreDate : Date ) →entier
    retourner absolu( nbJoursDepuis1900() - autreDate.nbJoursDepuis1900() )
fin méthode

méthode nbJoursDepuis1900 () →entier
    // On présente une version qui ne fonctionne que pour des dates
    // à partir du 1/1/1900. Elle n'est pas optimisée
    nbJours : entier
    nbJours ←0
    pour a de 1900 à année -1 faire  // Toutes les années complètes
        si estBissextile( a ) alors
            nbJours ←nbJours + 366
        sinon
            nbJours ←nbJours + 365
        fin si
    fin pour
    pour m de 1 à mois – 1 faire      // Tous les mois complets de la dernière année
        nbJours ←nbJours + nbJoursMois( m, année )
    fin pour
    // Les jours du dernier mois
    nbJours ←nbJours + jour – 1
    retourner nbJours
fin méthode

// Pour des raisons d'efficacité, on fournit aussi des modules (cf. nbJoursDepuis1900)
// On évite ainsi bon nombre d'instanciations d'objets
module estBissextile( année : entier ) →booléen
    retourner ( multipleDe( année, 4 ) ET non multipleDe( année, 100 )
        OU multipleDe( année, 400 )
fin module

module nbJoursMois( mois, année : entiers ) →entier
    nbJours : entier
    selon que mois vaut
        1, 3, 5, 7, 8, 10, 12 :
            nbJours ←31
        4, 6, 9, 11 :
            nbJours ←30
        2 :     si estBisextile(année) alors
                    nbJours ←29
                sinon
                    nbJours ←28
                fin si
    fin selon que
    retourner nbJours
fin module

// Ces modules sont d'intérêt général et servent à la lisibilité du code
module multipleDe( nombre, diviseur : entiers ) →booléen
    retourner ( nombre MOD diviseur ) = 0
fin module

module absolu( nombre : entier ) →entier
    si nombre > 0 alors
        retourner nombre
    sinon
        retourner ( -nombre )
    fin si
fin module
```

Ex. 2 Une personne

```
classe Personne
   privé :
        nom, prénom : chaînes
        naissance : Date
   public :
        // Constructeurs
        constructeur Personne( n, p : chaîne, naiss : Date)
        constructeur Personne( n, p : chaîne )
        // Accesseurs
        méthode getNom() → chaîne
        méthode getPrénom() → chaîne
        méthode getDateNaissance() →Date
        // Mutateurs
        méthode setNom( n : chaîne)
        méthode setPrénom( p : chaîne)
   privé :
        méthode setDateNaissance( d : Date )
fin classe

constructeur Personne( n, p : chaîne, naiss : Date)
   setNom( n )
   setPrénom( p )
   setDatenaissance( naiss )
fin constructeur

constructeur Personne( n, p : chaîne )
   setNom( n )
   setPrénom( p )
   naissance ←rien
fin constructeur

méthode getNom() →chaîne
   retourner nom
fin méthode

méthode getPrénom() →chaîne
   retourner prénom
fin méthode

méthode getDateNaissance() →Date
   retourner naissance
fin méthode

méthode setNom( n : chaîne)
   nom ←n
fin méthode

méthode setPrénom( p : chaîne)
   prénom ←p
fin méthode
méthode setDateNaissance( d : Date )
   si getDateJour().estAntérieur( d ) alors erreur « né dans le futur » fin si
   naissance ←d
fin méthode
```

```
module Test
   p1, p2, p3 : Personne
   p1 ←nouvelle Personne()
   p2 ←nouvelle Personne( « Durant », « Zébulon » )
   p3 ←nouvelle Personne( « Durant », « Zébulon », nouvelle Date(1,1,2005) )
fin module
```

Ex. 3 Le carré

```
classe Carré
   privé :
        centre : Point
        demiHauteur : Réel
   public :
        constructeur Carré ()
        constructeur Carré ( c : Point )
        constructeur Carré ( h : réel )
        constructeur Carré ( c : Point, h : réel )
        méthode getCentre() →Point
        méthode getSupérieurGauche() →Point
        méthode getInférieurDroit() →Point
        méthode getSurface() →réel
        méthode getPérimètre() →réel
        méthode déplacer( dx, dy : réels )
   privé :
        méthode setDemiHauteur( h : Réel )
fin classe
```

```
constructeur Carré ()
   centre.abs ←0
   centre.ord ←0
   setHauteur( 1 )
fin constructeur

constructeur Carré ( c : Point )
   centre ←c
   setHauteur( 1 )
fin constructeur

constructeur Carré ( h : Réel )
   centre.abs ←0
   centre.ord ←0
   setHauteur( h )
fin constructeur

constructeur Carré ( c : Point, h : Réel )
   centre ←c
   setHauteur( h )
fin constructeur

méthode getCentre() →Point
   retourner centre
fin méthode

méthode getSupérieurGauche () →Point
   point : Point
   point.abs ←centre.abs - hauteur
   point.ord ←centre.ord + hauteur
   retourner point
fin méthode

méthode getInférieurDroit () →Point
   point : Point
   point.abs ←centre.abs + hauteur
   point.ord ←centre.ord - hauteur
   retourner point
fin méthode
```

```
méthode déplacer ( dx, dy : Réels )
    centre.abs ←centre.abs + dx
    centre.ord ←centre.ord + dy
fin méthode

méthode getSurface() →Réel
    retourner 4 * hauteur * hauteur
fin méthode

méthode getPérimètre () →Réel
    retourner 8 * hauteur
fin méthode
```

```
module Test
    car1, car2, car3, car4 : Carré
    pt : Point
    car1 ←nouveau Carré()
    car2 ←nouveau Carré( 7 )
    pt ←{ 6, 3 }
    car3 ←nouveau Carré( pt )
    car4 ←nouveau Carré( {-5,4}, 12 )
    écrire car1.getSurface()
    car4.déplacer( 5, 3 )
    écrire car4
    pt ←car2.getSupérieurGauche()
    écrire pt.abs, pt.ord
fin module
```

Ex. 4 Le sablier

```
classe Sablier
    privé :
        centre : Point
        hauteur : Réel
        largeur : Réel
    public :
        constructeur Sablier ()
        constructeur Sablier ( c : Point )
        constructeur Sablier ( c : Point, h, l : réel )
        méthode getSupérieurGauche() →Point
        méthode getPérimètre() →réel
        méthode déplacer( dx, dy : réels )
    privé :
        méthode setHauteur( h : réel )
        méthode setLargeur( l : réel )
fin classe
```

```
constructeur Sablier ()
    centre.abs ←0
    centre.ord ←0
    setHauteur( 1 )
    setLargeur ( 1 )
fin constructeur

constructeur Sablier ( c : Point )
    centre ←c
    setHauteur( 1 )
    setLargeur ( 1 )
fin constructeur
```

```
constructeur Sablier ( c : Point, h, l : Réel )
    centre ←c
    setHauteur( h )
    setLargeur ( l )
fin constructeur

méthode getSupérieurGauche () →Point
    point : Point
    point.abs ←centre.abs - largeur
    point.ord ←centre.ord + hauteur
    retourner point
fin méthode

méthode déplacer ( dx, dy : Réels )
    centre.abs ←centre.abs + dx
    centre.ord ←centre.ord + dy
fin méthode

méthode getPérimètre () →Réel
    L ←largeur
    H ←hauteur
    retourner 4 * (L + √(H² + L²) ).
fin méthode
```

```
module Test
    sab1, sab2, sab3 : Sablier
    pt : Point
    sab1 ←nouveau Sablier()
    sab2 ←nouveau Sablier({ -4, 3 })
    sab3 ←nouveau Sablier({-4,3},6,2)

    écrire sab3. getPérimètre ()
    sab2.déplacer( 5, -3 )
    pt ←sab1.getSupérieurGauche()
    écrire pt.abs, pt.ord
fin module
```

Ex. 5 Anniversaire des personnes

```
module Anniversaires
    personne : Personne
    moisCrt, nbPersonnes : Entier
    moisCrt ←getDateJour().getMois()
    nbPersonnes ←0
    lire personne
    tant que personne ≠ rien faire
        si personne.getDateAnniversaire.getMois() = moisCrt alors
            afficher personne.getNom()
            nbPersonneș ←nbPersonnes + 1
        fin si
        lire personne
    fin tant que
    écrire nbPersonnes
fin module
```

Chapitre 9
Les tableaux

Exercices sur les tableaux à une dimension

Exercices de base

Ex. 1 Somme.

```
module somme ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + tab[i]
    fin pour
    retourner somme
fin module
```

Ex. 2 Maximum/minimum.

```
// pour le min, on remplace max par min et > par <
module max ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    max : Entier
    max  ← tab[1]
    pour i de 2 à n faire
        si tab[i] > max alors
            max ←tab[i]
        fin si
    fin pour
    retourner max
fin module
```

Ex. 3 Indice du maximum/minimum.

```
// pour les variantes, tout est dans le signe de comparaison dans le test
module posMax ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    pos : Entier
    pos  ← 1
    pour i de 2 à n faire
        si tab[i] > tab[pos] alors
            pos ←i
        fin si
    fin pour
    retourner pos
fin module

module max ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    retourner tab[ posMax(tab) ]
fin module
```

Ex. 4 Nombre d'éléments d'un tableau.

```
module posMax ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    retourner n
fin module
```

Ex. 5 Plus grand écart absolu.

```
module maxEcart ( tab : tableau [ 1 à n ] d'entiers ) →Entier
    max, écart : Entier
    max ←0
    pour i de 1 à n-1 faire
        écart ←abs( tab[i+1] – tab[i] )
        si écart > max alors
            max ←écart
        fin si
    fin pour
    retourner max
fin module
// Pour le plus petit écart, on peut remplacer 0 par l'infini
// ou calculer le premier écart en dehors de la boucle
```

Ex. 6 Remplacer des valeurs.

```
module mult3par0 ( tab : tableau [ 1 à n ] d'entiers )
    pour i de 1 à n faire
        si tab[1] MOD 3 = 0 alors
            tab[i] ←0
        fin si
    fin pour
fin module
```

Ex. 7 Tableau ordonné ?

```
module mult3par0 ( tab : tableau [ 1 à n ] d'entiers ) →Booléen
    i : Entier
    croissant : Booléen
    croissant ←vrai
    i ←2
    tant que i ≤ n ET croissant faire
        croissant ←tab[i] ≤ tab[i-1]
        i ←i + 1
    fin tant que
    retourner croissant
fin module
```

Ex. 8 Position des minimums.

```
// 8a
module indicesMin ( tab : tableau [ 1 à n ] d'entiers )
    min : Entier
    min ← min(tab)     // cf. exercice précédent
    pour i de 1 à n faire
        si tab[1] = min alors
            afficher i
        fin si
    fin pour
fin module
```

```
// 8b
module indicesMin ( tab : tableau [ 1 à n ] d'entiers )
        min, nbMin : Entier
    indMin : tableau [ 1 à n ] d'entiers
    min ← tab[1]
    indMin[1] ← 1
    nbMin ← 1
    pour i de 2 à n faire
        selon que
            tab[i] = min :
                nbMin ← nbMin + 1
                indMin[ nbMin ] ← i
            tab[i] < min :
                min ← tab[i]
                nbMin ← 1
                indMin[ nbMin ] ← i
        fin selon
    fin pour
    afficherTab( indMin, nbMin )
fin module

module afficherTab ( tab : tableau [ 1 à n ] d'entiers, nbEffectif : entier )
    pour i de 1 à nbEffectif faire
        afficher tab[i]
    fin pour
fin module
```

Ex. 9 Renverser un tableau.

```
module renverser ( tab ⇵ : tableau [ 1 à n ] d'entiers )
    pour i de 1 à n MOD 2 faire
        swap ( tab[i] , tab[n+1-i] )
    fin pour
fin module

module swap ( a ⇵, b ⇵ : Entiers )
    temp : Entier
    temp ← a
    a ← b
    b ← temp
fin module
```

Ex. 10 Tableau symétrique ?

```
module estSymétrique ( tab ⇵ : tableau [ 1 à n ] d'entiers ) →Booléen
    i : Entier
    i ←1
    tant que i < ( n MOD 2 ) ET tab[i] = tab[n+1-i] faire
        i ←i + 1
    fin tant que
    retourner tab[i] = tab[n+1-i]
fin module
```

Exercices de difficulté moyenne

Ex. 11 Palindrome.

```
module estPalindrôme ( phrase ⇵: tableau [ 1 à n ] de caractères ) →Booléen
    gauche, droite : Entiers
    gauche ←0
    droite ←n + 1
    répéter
        avancer( gauche, droite, tab )
        reculer( droite, gauche, tab )
    jusqu'à ce que droite ≤ gauche OU phrase[ gauche ] ≠ phrase[ droite ]
    retourner droite ≤ gauche
fin module

// Avance jusqu'à la prochaine lettre sans toutefois dépasser la limite
module avancer ( g ⇵: Entier, limite : Entier, phrase : tableau [ 1 à n ] de caractères )
    répéter
        g ←g + 1
    jusqu'à ce que g ≥ limite OU estLettre( phrase[g] )
fin module

// Recule jusqu'à la prochaine lettre sans toutefois dépasser la limite
module reculer ( d ⇵: Entier, limite : Entier, phrase : tableau [ 1 à n ] de caractères )
    répéter
        d ←d - 1
    jusqu'à ce que d ≤ limite OU estLettre( phrase[d] )
fin module

module estLettre ( c : Caractère )
    retourner non estEspace( c ) ET non estPonctuation( c )
fin module
```

Ex. 12 Occurrence des chiffres.

```
module nbOccChiffre ( nb : Entier )
    nbOccurences : tableau [ 0 à 9 ] d'entiers
    initialiser( nbOccurences )
    compter( nb, nbOccurences )
    afficher( nbOccurences )
fin module

module initialiser ( tab ⇵: tableau [ 0 à 9 ] de Entiers )
    pour i de 0 à 9 faire
        tab[i] ←0
    fin pour
fin module

module compter ( nb : Entier, nbOccurences ⇵: tableau [ 0 à 9 ] de Entiers )
    chiffre : Entier
    tant que nb > 0 faire
        chiffre ←nb MOD 10
        nb ←nb DIV 10
        nbOccurences [ chiffre ] ←nbOccurences [ chiffre ] + 1
    fin tant que
fin module

module afficher (nbOccurences: tableau [ 0 à 9 ] de Entiers )
    pour i de 0 à 9 faire
        si nbOccurences[ i ] > 0 alors
            afficher i, nbOccurences[ i ]
        fin si
    fin pour
fin module
```

Ex. 13 Cumul des ventes.

```
module cumul ( tab : tableau [ 1 à 12 ] de Entiers ) →tableau [ 1 à 12 ] de Entiers
    cumul : tableau [ 1 à 12 ] de Entiers
    pour i de 2 à 12 faire
        cumul[ i ] ←cumul[ i - 1 ] + ventes[ i ]
    fin pour
    retourner cumul
fin module
```

Ex. 14 Moyenne d'éléments.

```
module moyenneMinMax ( tab : tableau [ 1 à n ] de Entiers ) →Réel
    indMin, indMax, début, fin, somme, nbValeurs : Entiers
    indMin      ←indMin( tab )      // cf. exercice précédent
    indMax      ←indMax( tab )
    début       ←min( indMin, indMax )
    fin         ←max( indMin, indMax )
    somme       ←somme( tab, début, fin )
    nbValeurs   ←fin – début + 1
    retourner somme / nbValeurs
fin module

module somme ( tab : tableau [ 1 à n ] de Entiers, début, fin : Entiers ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + tab[ i ]
    fin pour
    retourner somme
fin module
```

Ex. 15 OXO.

```
module Oxo ( oxo : tableau [ 1 à n ] de caractères  ) →Entier
    nbOxo, i : Entiers
    nbOxo  ←0
    i ←1
    tant que i < n - 1 faire
        si oxo[ i ] = 'O' ET oxo[ i+1 ] = 'X' ET oxo[ i+2 ] = 'O' alors
            nbOxo  ←nbOxo + 1
            i ←i + 3
        sinon
            i ←i + 1
        fin si
    fin tant que
    retourner nbOxo
fin module
```

Ex. 16 Mastermind.

```
module testerProposition( proposition, solution : tableaux [1 à k ] de Couleur,
                                                  bienPlacés ↑, malPlacés ↑: entiers)
    utilisé : tableau [1 à k] de Booléen
    j : Entier

    initialiserTableau( utilisés, faux )
    // Bien placés
    bienPlacés ←0
    pour i de 1 à k faire
        si proposition[ i ] = solution [ i ] alors
            utilisé[ i ]  ←vrai
            bienPlacés ←bienPlacés + 1
        fin si
    fin pour
```

```
    // Mal placés
    malPlacés ←0
    pour i de 1 à k faire
        si proposition[ i ] ≠ solution [ i ] alors
            j ←1
            tant que j ≤ k ET ( utilisé[ j ] OU proposition[ i ] ≠ solution [ j ] ) faire
                j ←j + 1
            fin tant que
            si j ≤ k alors
                utilisé[ j ] ←vrai
                malPlacés ←malPlacés + 1
            fin si
        fin si
    fin pour

fin module
```

Exercices sur les tableaux à 2 dimensions

Ex. 17 Tous positifs

```
module tousPositifs ( tab : tableau [ 1 à m, 1 à n ] d'entiers ) →Booléen
    ok : Booléen
    i, j : Entiers
    ok ←vrai
    i ←1
    tant que i < m ET ok faire
        j ←1
        tant que j < n ET ok faire
            ok ←tab[ i, j ] > 0
            j ← j + 1
        fin tant que
        i ← i + 1
    fin tant que
    retourner ok
fin module
```

Ex. 18 Le carré magique

```
module estCarréMagique ( carré : tableau [ 1 à n, 1 à n ] d'entiers ) →Booléen
    somme : Entier
    ok : Booléen
    somme ←sommeDiagonale( carré )
    ok ←( somme = sommeDiagonaleInverse(carré ) )
    i ←1
    tant que i < n ET ok faire
        ok ←somme = sommeLigne( carré, i )
    fin tant que
    i ←1
    tant que i < n ET ok faire
        ok ←somme = sommeColonne( carré, i )
    fin tant que
    retourner ok
fin module
```

```
module sommeDiagonale ( carré : tableau [ 1 à n, 1 à n ] d'entiers ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + carré[ i, i ]
    fin pour
    retourner somme
fin module

module sommeDiagonaleInverse ( carré : tableau [ 1 à n, 1 à n ] d'entiers ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + carré[ i, n-i+1 ]
    fin pour
    retourner somme
fin module

module sommeLigne ( carré : tableau [ 1 à n, 1 à n ] d'entiers, lg : Entier ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + carré[ lg, i ]
    fin pour
    retourner somme
fin module

module sommeColonne ( carré : tableau [ 1 à n, 1 à n ] d'entiers, col:Entier ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à n faire
        somme ←somme + carré[ i, col ]
    fin pour
    retourner somme
fin module
```

Ex. 19 Le contour du tableau

```
module sommeContour ( tab : tableau [ 1 à m, 1 à n ] d'entiers ) →Entier
    somme : Entier
    somme ←sommeLigne( tab, 1 )
    si m > 1 alors
        somme ←somme + sommeLigne( tab, m )
    fin si
    somme ←somme + sommeColonne( tab, 1 )
    si c > 1 alors
        somme ←somme + sommeColonne ( tab, n )
    fin si
    retourner somme
fin module

module sommeLigne ( tab : tableau [ 1 à m, 1 à n ] d'entiers, lg : Entier ) →Entier
    somme : Entier
    somme ←0
    pour c de 1 à n faire
        somme ←somme + tab[ lg, c ]
    fin pour
    retourner somme
fin module
```

```
module sommeColonne ( tab : tableau [ 1 à m, 1 à n ] d'entiers, col: Entier ) →Entier
    somme : Entier
    somme ←0
    pour l de 2 à m - 1 faire
        somme ←somme + tab[ col, l ]
    fin pour
    retourner somme
fin module
```

Ex. 20 Le triangle de Pascal

```
module trianglePascal ( n : Entier ) →tableau [ 1 à n, 1 à n ] d'entiers
    pascal : tableau [ 1 à n, 1 à n ] d'entiers
    pour lg de 1 à n faire
        pascal[ lg, 1 ] ←1
        pascal[ lg, lg ] ←1
        pour col de 2 à lg faire
            pascal[ lg, col ] ←pascal[ lg-1, col-1 ] + pascal[ lg-1, col ]
        fin pour
    fin pour
    retourner pascal
fin module
```

Ex. 21 Le calendrier du mois.

```
module calendrier ( jourPremier, nbJours: Entiers ) →tableau [ 1 à 6, 1 à 7 ] d'Entiers
    cal : tableau [ 1 à 6, 1 à 7 ] d'Entiers
    lg, col : Entiers
    initialiserTableau ( cal, 0 )
    lg ← 1
    col ← jourPremiers
    pour i de 1 à nbJours faire
        cal[ lg, col ] ← i
        si col < 7 alors
            col ←col + 1
        sinon
            col ←1
            lg ←lg + 1
        fin si
    fin pour
    retourner cal
fin module
```

Ex. 22 A vos pinceaux !

```
module A-Contour ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    initialiserTableau (tab, 'blanc' )
    ligne( tab, 1 )
    ligne( tab, n )
    colonne( tab, 1 )
    colonne( tab, n )
fin module

module B-X ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    initialiserTableau (tab, 'blanc' )
    diagonale( tab )
    diagonaleInverse( tab )
fin module

module C-Zorro ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    initialiserTableau (tab, 'blanc' )
    ligne( tab, 1 )
    ligne( tab, n )
    diagonaleInverse( tab )
fin module
```

```
module D-Zèbre ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    initialiserTableau (tab, 'blanc' )
    pour i de 1 à 2*n par pas de 3 faire
        si i ≤ n alors
            diagonaleInverse( 0, i )
        sinon
            diagonaleInverse( i – n , n )
        fin si
    fin pour
fin module

module E-Tunnel ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    initialiserTableau ( tab, 'blanc' )
    pour i de 1 à n par pas de 2 faire
        carré( tab, i, i, n+1-i, n+1-i )
    fin pour
fin module

énumération Sens {DROITE, BAS, GAUCHE, HAUT}

module F-Spirale ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    lg, col : Entiers
    sens : Sens
    initialiserTableau ( tab, 'blanc' )
    lg ←1
    col ←1
    sens ←DROITE
    avancer( tab, lg, col, sens, n )
    long ←n-1
    pour i de 2 à n faire
        tourner( sens )
        avancer( tab, lg, col, sens, long )
        si impair( i ) alors
            long ←long - 2
        fin si
    fin pour
fin module

module ligne ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs, lg : Entier )
    ligne( tab, lg, 1, n )
fin module

module ligne ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs, lg, deb, fin : Entier )
    pour col de deb à fin faire
        tab[ lg, col ] ←'noir'
    fin pour
fin module

module colonne ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs, col : Entier )
    colonne ( tab, col, 1, n )
fin module

module colonne ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs, col, deb, fin : Entier )
    pour lg de deb à fin faire
        tab[ lg, col ] ←'noir'
    fin pour
fin module

module diagonale ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    pour i de 1 à n faire
        tab[ i, i ] ←'noir'
    fin pour
fin module
```

```
module diagonaleInverse ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs )
    diagonaleInverse ( tab, 1, n )
fin module

module diagonaleInverse ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs,
                                                    lgDeb, colDeb : Entier )
    lg, col : Entiers
    lg ← lgDeb
    col ← colDeb
    répéter
        tab[ lg, col ] ← 'noir'
        lg ← lg - 1
        col ← col + 1
    jusqu'à ce que lg < 1 OU col > n
fin module

module carré ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs,
                                                    lgSG, colSG, lgID, colID : Entiers )
    ligne ( tab, lgSG, colSG, colID )
    colonne ( tab, colID, lgSG, lgID )
    ligne ( tab, lgID, colSG, colID )
    colonne ( tab, colSG, lgSG, lgID )
fin module

module avancer ( tab ⇵ : tableau [ 1 à n, 1 à n ] de couleurs, lg ⇵, col ⇵ : Entiers
                                                    sens : Sens, long : Entier )
    pour i de 1 à long faire
        selon que sens vaut
            DROITE      : col ← col + 1
            BAS         : lg ← lg + 1
            GAUCHE      : col ← col - 1
            HAUT : lg ← lg – 1
        fin selon que
        tab[ lg, col ] ← 'noir'
    fin pour
fin module

module tourner ( sens ⇵ : Sens )
    selon que sens vaut
        DROITE      : sens ← BAS
        BAS         : sens ← GAUCHE
        GAUCHE      : sens ← HAUT
        HAUT  : sens ← DROITE
    fin selon que
fin module
```

Ex. 23 Le Flipper à le dos fin.

```
module chercherTrou ( tab : tableau [ 1 à L, 1 à C ] de caractères,
                            ligne ⇵, col ⇵ : Entier ) → Booléen
    trouvé: Booléen
    répéter
        si col < C alors
            col ← col + 1
        sinon
            col ← 1
            ligne ← ligne + 1
        fin si
        trouvé ← tab[ ligne, col ] = 'T'
    jusqu'à ce que ligne = L OU trouvé
    retourner trouvé
fin module
```

```
module Flipper ( tab :  tableau [ 1 à L, 1 à C ] de caractères, colDépart : Entier )
    ligne, col : Entier
    impasse, trouvé : Booléen
    ligne ←1
    col ←colDépart
    impasse ←faux
    répéter
        selon que tab[ ligne, col ] vaut
                ' '     : ligne ←ligne + 1
                'G'     : col ←col - 1
                'D'     : col ←col + 1
                'T'     : trouvé ←chercherTrou( ligne, col )
                          si trouvé alors
                              afficher ligne, col
                              ligne ←ligne + 1
                          sinon
                              impasse ←vrai
                          fin si
        fin selon que
        afficher ligne, col
    jusqu'à ce que ligne = L OU impasse
fin module
```

Exercices de synthèse

Ex. 24 Un jeu de poursuite.

```
classe JeuPoursuite
    privé :
        circuit : tableau [ 1 à 50 ] de booléens
        avancement : tableau [0 à 1] d'entiers  //nb de cases avancées par ch. joueur
        début : tableau [ 0 à 1 ] d'entiers       // case de début de ch joueur
        joueurCourant, autreJoueur, gagnant : Entier
        fini : booléen
    public :
        constructeur JeuPoursuite( c : tableau [ 1 à 50 ] de booléens )
        méthode initialiser()
        méthode jouer()
    privé :
        méthode jouerCoup()
        méthode jouerTour()
        méthode joueurSuivant()
        méthode positionJoueurCourant ()) →Entier
        méthode afficherRésultat()
fin classe

constructeur JeuPoursuite ( c : tableau [ 1 à 50 ] de booléens )
    circuit ←c
    début[ 0 ] ←1
    début[ 1 ] ←26
    initialiser()
fin constructeur

méthode initialiser ()
    avancement  [ 0 ] ←0
    avancement  [ 1 ] ←0
    joueurCourant ←0
    autreJoueur ←1
    fini ←faux
fin méthode
```

```
méthode jouer ()
   répéter
        jouerTour()
        joueurSuivant()
   jusqu'à ce que fini
   afficherRésultat()
fin méthode

méthode jouerTour ()
   répéter
        jouerCoup()
   jusqu'à ce que fini OU non circuit[ positionJoueurCourant() ]
fin méthode

méthode jouerCoup ()
   avancement[ joueurCourant ] ←avancement[ joueurCourant ] + LancerDé()
   si avancement[ joueurCourant ] > avancement[ autreJoueur ] + 26 alors
        fini ←vrai
        gagnant ←joueurCourant
   fin si
fin méthode

méthode joueurSuivant ()
   joueurCourant ←1 - joueurCourant
   autreJoueur ← 1 - autreJoueur
fin méthode

méthode positionJoueurCourant ()) →Entier
   retourner (avancement[ joueurCourant ] + début[ joueurCourant ] – 1) MOD 50 + 1
fin méthode

méthode afficherRésultat ()
   nomJoueur : Chaine
   nbTours : Entier
   si gagnant = 0 alors
        nomJoueur ←« A »
   sinon
        nomJoueur ←« B »
   fin si
   nbTours ←avancement[ gagnant ] DIV 50
   écrire nomJoueur, nbTours
fin méthode
```

Ex. 25 La course à la case 64.

```
classe Course64
   privé :
        nbJoueurs : Entiers
        joueurs : tableau de chaînes          // les noms des joueurs
        position : tableau d'entiers          // la position de chaque joueur
        joueurCourant : Entier
        fini : booléen
   public :
        constructeur Course64 ( j : tableau [1 à n] de Chaîne )
        méthode recommencer()
        méthode jouer()
   privé :
        méthode jouerTour()
        méthode joueurSuivant()
        méthode reculerAutres()
fin classe
```

```
constructeur Course64 ( j : tableau [1 à n] de Chaîne )
    nbJoueurs ←n
    joueurs ←j
    position  ← nouveau tableau [ 1 à nbJoueurs ] d'entiers
    recommencer ()
fin constructeur

méthode recommencer ()
    initialiserTableau( position, 0 )
    joueurCourant ←1
    fini ←faux
fin méthode

méthode jouer ()
    jouerTour()
    tant que non fini
          joueurSuivant()
          jouerTour()
    fin tant que
    afficher joueurCourant
fin méthode

méthode jouerTour ()
    dé : Entier
    répéter
          dé ←lancerDé()
          position[ joueurCourant ] ←position[ joueurCourant ] + dé
          fini ←( position[ joueurCourant ] ≥ 64 )
          reculerAutres()
    jusqu'à ce que dé ≠ 6
fin méthode

méthode reculerAutres ()
    pour i de 1 à nbJoueurs faire
          si i != joueurCourant ET position[i] = position[joueurCourant] alors
                position[ i ] ←0
          fin si
    fin pour
fin méthode

méthode joueurSuivant ()
    joueurCourant ←joueurCourant + 1
    si joueurCourant > nbJoueurs alors
          joueurCourant ←1
    fin si
fin méthode
```

Ex. 26 Mots croisés

```
classe Grille
   privé :
        grille : tableau [1 à 10, 1 à 10] de TCase
   public :
        constructeur Grille ( g : grille : tableau [1 à 10, 1 à 10] de TCase )
        méthode placer( i,j : Entiers, lettre : caractère )
        méthode nbCasesNoires( ) →Entier
        méthode nbTotalMots( ) →Entier
        méthode nbMotsPlacés( ) →Entier
   privé :
        méthode nbTotalMotsHorizontaux() →Entier
        méthode nbTotalMotsVerticaux() →Entier
        méthode nbMotsPlacésHorizontaux() →Entier
        méthode nbMotsPlacésVerticaux() →Entier
        méthode examinerMotsHorizontal(i,j:Entiers, lg↑:Entier, complet ↑: Booléen)
        méthode examinerMotsVertical(i,j : Entiers, lg ↑: Entier, complet ↑: Booléen)
        méthode estMotHorizontalComplet(i,j : Entiers) →Booléen
        méthode estMotVerticalComplet(i,j : Entiers) →Booléen
fin classe

constructeur Grille ( g : grille : tableau [1 à 10, 1 à 10] de TCase )
   grille ←g
fin constructeur

méthode placer( i,j : Entiers, lettre : caractère )
   si i< 1 OU i > 10 OU j < 1 OU j > 10 OU grille[i,j].NOIR alors
        erreur « paramètre invalide »
   fin si
   grille[i,j] ←lettre
fin méthode

méthode nbCasesNoires( ) →Entier
   cpt : Entier
   cpt ←0
   pour i de 1 à 10 faire
        pour j de 1 à 10 faire
             si grille[i,j].NOIR alors
                  cpt ←cpt + 1
             fin si
        fin pour
   fin pour
   retourner cpt
fin méthode

méthode nbTotalMots ( ) →Entier
   retourner nbTotalMotsHorizontaux() + nbTotalMotsVerticaux()
fin méthode

méthode nbTotalMotsHorizontaux ( ) →Entier
   cpt : Entier
   inutilisé : Booléen
   cpt ←0
   pour i de 1 à 10 faire
        j ←1
        tant que j < 10 faire
             examinerMotsHorizontal ( i, j, lg, inutilisé )
             si lg > 1 alors cpt ←cpt + 1 fin si
             j ←j + lg + 1
        fin tant que
   fin pour
   retourner cpt
fin méthode
```

```
méthode examinerMotsHorizontal(i,j : Entiers, lg ↑: Entier, complet ↑: Booléen)
    lg ←0
    complet ←vrai
    tant que j < 10 ET NON grille[i, j].NOIR faire
        lg ←lg + 1
        complet ←complet ET NON grille[i,j].LETTRE = ' '
        j ←j + 1
    fin tant que
fin méthode

// Les versions verticales sont fort proches

méthode nbMotsPlacés ( ) →Entier
    retourner nbMotsPlacésHorizontaux () + nbMotsPlacésVerticaux ()
fin méthode

méthode nbMotsPlacésHorizontaux ( ) →Entier
    cpt, lg : Entier
    complet : Booléen
    cpt ←0
    pour i de 1 à 10 faire
        j ←1
        tant que j < 10 faire
            examinerMotsHorizontal ( i, j, lg, complet )
            si lg > 1 ET complet alors
                cpt ←cpt + 1
            fin si
            j ←j + lg + 1
        fin tant que
    fin pour
    retourner cpt
fin méthode
```

Ex. 27 Le Jeu du Millionnaire .

```
structure Question
    libellé : Chaîne
    réponses : tableau [1 à 4]  de Chaînes
    bonneRéponse : Entier
fin structure

structure Gain
    somme : Entier
    palier : Booléen
fin structure

classe Millionnaire
    privé :
        questionnaire : tableau [1 à 15] de Question
        gains : tableau [1 à 15] de Gain
        questionCrt, gainAssuré, gainCrt : Entier
        fini : Booléen
    public :
        constructeur Millionnaire (   q : tableau [1 à 15] de Question,
                                      g : tableau [1 à 15] de Gain )
        méthode initialiser(    q : tableau [1 à 15] de Question,
                                g : tableau [1 à 15] de Gain )
        méthode getQuestion( ) →Question
        méthode donnerRéponse( num : Enter )
        méthode estFini( ) →Booléen
        méthode arrêter( )
        méthode getGain( ) →Entier
fin classe
```

```
constructeur Millionnaire(q:tableau [1 à 15] de Question,g:tableau [1 à 15] de Gain )
    initialiser ( q, g )
fin constructeur

méthode initialiser( q : tableau [1 à 15] de Question,  g : tableau [1 à 15] de Gain )
    questionnaire ←q
    gains ←g
    questionCrt ←1
    fini ←faux
    gainsAssuré ←0
    gainsCrt ←0
fin méthode

méthode donnerRéponse( num : Enter )
    si fini alors erreur « le jeu est fini » fin si
    si questionnaire[ questionCrt ].bonneRéponse = num alors
        gainsCrt ←gains[ questionCrt ].montant
        si gains[ questionCrt ].palier alors
            gainsAssuré ←gainsCrt
        fin si
        questionCrt ←questionCrt + 1
        si questionCrt = 15 alors
            fini ←vrai
        fin si
    sinon
        fini ←vrai
    fin si
fin méthode

méthode arrêter( )
    si fini alors erreur « le jeu est déjà fini » fin si
    fini ←vrai
fin méthode

méthode estFini( ) →Booléen
    retourner fini
fin méthode

méthode getQuestion( ) →Question
    retourner questionnaire[ questionCrt ]
fin méthode

méthode getGain( ) →Entier
    retourner gainCrt
fin méthode

module jeuMillionnaireConsole(q:tableau [1 à 15] Question,g:tableau [1 à 15]  Gain)
    jeu : Millionnaire
    question : Question
    réponse : Entier
    jeu ←nouveau Millionnaire( q, g )
    tant que NON jeu.estFini () faire
        question  ← jeu.getQuestion()
        écrire question.libellé, question.réponses[1], question.réponses[2],
                    question.réponses[3], question.réponses[4]
        lire réponse
        si réponse = 0 alors
            jeu.arrêter()
        sinon
            jeu.donnerRéponse( réponse )
        fin si
    fin tant que
    écrire jeu.getGain()
fin module
```

Chapitre 10
La liste

La classe Liste

EXERCICE : MISE À JOUR D'UNE LISTE DE RENDEZ-VOUS

```
module màjRDV ( rdvs : Liste de RendezVous )
    i : Entier
    aujourdhui : Date
    aujourdhui ←nouvelle Date()
    pour i de liste.taille() à 1 par -1 faire
        si rvds.get(i).date.estAntérieure( aujourdhui ) alors
            rvds.supprimer( i )
        fin si
    fin pour
fin module
```

Exercices

Ex. 1 Somme d'une liste.

```
module somme ( liste : Liste d'entiers ) →Entier
    somme : Entier
    somme ←0
    pour i de 1 à liste.taille() faire
        somme ←somme + liste.get( i )
    fin pour
    retourner somme
fin module
```

Ex. 2 Maximum d'une liste.

```
module max ( liste : Liste d'entiers ) →Entier
    max, i : Entier
    max  ← liste.get( 1 )
    pour i de 2 à n faire
        si liste.get( i ) > max alors
            max ←liste.get( i )
        fin si
    fin pour
    retourner max
fin module
```

Ex. 3 Anniversaires.

```
module anniversaires ( personnes : Liste de Personne ) →Liste de Personne
    personne : Personne
    moisCourant, i : Entier
    anniversaires : Liste de Personne

    moisCourant ←getDateJour().getMois()
    anniversaires ←nouvelle Liste de Personne
    pour i de 1 à personnes.taille() faire
        personne ←personnes.get(i)
        si personne.getDateAnniversaire.getMois() = moisCourant alors
            anniversaires.add( personne )
        fin si
    fin pour
    retourner anniversaires
fin module
```

Ex. 4 Concaténation de deux listes.

```
module concaténation ( liste1, liste2 : Liste )
    i: Entier
    pour i de 1 à liste2.taille() faire
        liste1.ajouter( liste2.get(i) )
    fin pour
fin module
```

Ex. 5 Fusion de deux listes.

```
module concaténation ( liste1, liste2 : Liste d'entiers ) →Liste d'entiers
    liste : Liste d'entier
    i,i1,i2 : Entier
    liste ← nouvelle Liste d'entiers()
    i1 ← 1
    i2 ← 1
    tant que i1 ≤ liste1.taille() ET i2 ≤ liste2.taille() faire
        si liste1.get(i1) < liste2.get(i2) alors
            liste.ajouter( liste1.get(i1) )
            i1 ← i1 + 1
        sinon
            liste.ajouter( liste2.get(i2) )
            i2 ← i2 + 1
        fin si
    fin tant que
    pour i de i1 à liste1.taille() faire
        liste.ajouter( liste1.get(i) )
    fin pour
    pour i de i2 à liste2.taille() faire
        liste.ajouter( liste2.get(i) )
    fin pour
    retourner liste
fin module
```

Ex. 6 Éliminer les doublons d'une liste.

```
module éliminerDoublon-A ( données: Liste d'entiers ) →Liste d'entiers
    résultats : Liste d'entier
    i : Entier
    résultats ← nouvelle Liste d'entiers()
    si données.estVide() alors
        retourner résultats
    fin si
    résultats.ajouter( données.get(1) )
    pour i de 2 à données.taille() faire
        si données.get(i) = données.get(i-1) alors
            résultats.ajouter( données.get(i) )
        fin si
    fin pour
    retourner résultats
fin module
```

```
module éliminerDoublon-B ( liste: Liste d'entiers )
    i : Entier
    i ← 2
    tant que i ≤ liste.taille() faire
        si liste.get(i) = liste.get(i-1) alors
            liste.supprimer( i )
        sinon
            i ← i + 1
        fin si
    fin pour
fin module
```

On pourrait aussi utiliser l'algorithme de rupture (cf. Chapitre sur les ruptures)

Ex. 7 Chambre avec vue.

A FAIRE

Ex. 8 Mastermind.

A FAIRE

Ex. 9 La chaîne.

A FAIRE

Chapitre 11
La liste ordonnée

La classe ListeOrdonnée

EXERCICE : ÉVITER LES DOUBLONS

```
module motsTriés ()
    mot : chaîne
    mots : ListeOrdonnée de chaîne
    i, pos : entier
    mots ←nouvelle ListeOrdonnée de chaîne
    lire mot
    tant que mot ≠ "" faire
        si non mots.existe( mot, pos ) alors
            mots.ajouter( mot )
        fin si
        lire mot
    fin tant que
    pour i de 1 à mots.taille() faire
        écrire mots.get( i )
    fin pour
fin module
```

Exercices

Exercices sur la complexité

Ex. 1 Manipulation d'une liste.

[A] O(N)

[B] O(N)

Ex. 2 Manipulation d'un tableau.

[A] $O(N^2)$

[B] $O(N^2)$

Ex. 3 Réflexion.

Pas sur des petites listes car plus lent dans ces cas là. La complexité n'est pas assez précise pour donner une indication pour les petites valeurs.

Exercices sur la liste ordonnée

Ex. 4 Un agenda.

A FAIRE

Chapitre 12
Le tri

Tri par sélection des minima successifs

Exercices

Ex. 1 A FAIRE

Ex. 2 A FAIRE

Tri bulle

Exercices

Ex. 3 A FAIRE.

Ex. 4 A FAIRE.

Chapitre 13
Le fichier séquentiel

Un accès différent aux fichiers/séquences

EXERCICE – ADAPTATION AUX TABLEAUX ET AUS LISTES

A FAIRE.

Exercices

Ex. 1 Algorithmes de base.

A FAIRE.

Ex. 2 Copies et modifications de fichiers.

A FAIRE.

Ex. 3 Fichiers ordonnés.

A FAIRE.

Ex. 4 Statistiques sur les profs.

A FAIRE.

Ex. 5 La sélection des mannequins.

A FAIRE.

Ex. 6 Le top 10.

A FAIRE.

Ex. 7 Le méli-mélo de cartes.

A FAIRE.

Ex. 8 La partie d'échec.

A FAIRE.

Ex. 9 Les stages.

A FAIRE.

Ex. 10 Séquence.

A FAIRE.

Chapitre 14
Les traitements de rupture

Exercices

Ex. 1 La chasse au gaspi.

A FAIRE.

Ex. 2 Vos papiers, SVP !

A FAIRE.

Ex. 3 Statistiques de ventes de voitures.

A FAIRE.

Ex. 4 Les fanas d'info.

A FAIRE.

Ex. 5 Degré ou de force.

A FAIRE.

Ex. 6 NBA actions.

A FAIRE.

Ex. 7 Le meilleur site.

A FAIRE.

Ex. 8 Quoi de neuf, doc ?

A FAIRE.

Ex. 9 Bruxelles-national

A FAIRE.

Ex. 10 Une suite logique

A FAIRE.

Ex. 11 Éliminer les doublons d'une liste.

A FAIRE.

Chapitre 15
La pile

Chapitre 16
La file

Exercices

Ex. 1 Une suite logique

A FAIRE.

Ex. 2 Monte-charge.

A FAIRE.

Chapitre 17
L'ensemble

Exercices

Ex. 1 Autres opérations ensemblistes.

A FAIRE.

Ex. 2 Autres implémentation de l'état.

A FAIRE.

Ex. 3 Nombres d'un fichier.

A FAIRE.