



Haute École de Bruxelles
École Supérieure d'Informatique

Rue Royale 67 – 1000 Bruxelles
02/219.15.46 – esi@heb.be

Logique & Techniques de programmation

Bachelor en Informatique – 1^{ère} année

Cours enseigné par :

<i>L. Beeckmans</i>	<i>M. Codutti</i>	<i>G. Cuvelier</i>	<i>J. Dossogne</i>	<i>A. Hallal</i>
<i>C. Leruste</i>	<i>E. Levy</i>	<i>N. Pettiaux</i>	<i>F. Servais</i>	<i>W. Willame</i>

Ce syllabus a été écrit à l'origine par M. Monbaliu. Il a ensuite été adapté par Mme Leruste, M. Beeckmans et M. Codutti. Qu'ils en soient tous remerciés. Nous remercions également tout ceux qui ont contribué à son amélioration grâce à leur lecture attentive et leurs remarques.

Document produit avec L^AT_EX.
Version du 6 novembre 2013.

**Ce syllabus couvre la matière du premier quadrimestre
(jusque fin décembre).
La suite sera diffusée en janvier.**



Ce document est distribué sous licence
Creative Commons Paternité - Partage à l'Identique 2.0 Belgique
(<http://creativecommons.org/licenses/by-sa/2.0/be/>).
Les autorisations au-delà du champ de cette licence
peuvent être obtenues à www.heb.be/esi - mcodutti@heb.be.

Table des matières

Correction des exercices 4.4

Simplification d'algorithme

```
si ok alors
  afficher nombre
fin si
```

ok \leftarrow condition

```
si NON ok alors
  afficher nombre
fin si
```

ok $\leftarrow a \geq b$

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

Exercice 3 – Maximum de 2 nombres

```
module max2Nb()
  nb1, nb2 : entiers
  max : entier
  lire nb1, nb2
  si nb2  $\geq$  nb1 alors
    max  $\leftarrow$  nb2
  sinon
    max  $\leftarrow$  nb1
  fin si
  afficher max
fin module
```

Exercice 4 – Maximum de 3 nombres

```
module max3Nb()
  nb1, nb2, nb3 : entiers
  max : entier
  lire nb1, nb2, nb3
  si nb2  $\geq$  nb1 alors
    max  $\leftarrow$  nb2
  sinon
    max  $\leftarrow$  nb1
  fin si
  si nb3  $\geq$  max alors
    max  $\leftarrow$  nb3
  fin si
  afficher max
fin module
```

Exercice 5 – Signe

```
module signe()
  nb : entier
  lire nb
  selon que
    nb > 0 : afficher "positif"
    nb < 0 : afficher "négatif"
    autre : afficher "nul"
  fin selon que
fin module
```

Exercice 6 – La fourchette

```
module fourchette()
  nb1, nb2, nb3, petit, grand : entiers
  ok : booléen
  lire nb1, nb2, nb3
  si nb2 > nb3 alors
    petit ← nb3
    grand ← nb2
  sinon
    petit ← nb2
    grand ← nb3
  fin si
  afficher ← nb1 > petit ET nb1 < grand
fin module
```

Exercice 7 – Équation du second degré

```
module racinesÉquation()
  coeffCarré, coeff, termeIndé : entiers
  delta : entier
  lire coeffCarré, coeff, termeIndé
  delta ← (coeff)2 - 4 * coeffCarré * termeIndé
  selon que
    delta > 0 : afficher  $(-coeff \pm \sqrt{\text{delta}}) / (2 * \text{coeffCarré})$ 
    delta = 0 : afficher  $-coeff / (2 * \text{coeffCarré})$ 
    autres : afficher "pas de racine"
  fin selon que
fin module
```

Exercice 8 – Une petite minute

```
module plusUneMin()
    heure, minute : entiers
    lire heure, minute
    si minute = 59 alors
        minute ← 0
        heure ← heure + 1
    sinon
        minute ← minute + 1
    fin si
    afficher heure, minute
fin module
```

Exercice 9 – Calcul de salaire

```
module salaireNet()
    salaireBrut : entier
    constante RETENUE : 15
    salaireNet : entier
    lire salaire
    si salaire > 1200 alors
        salaireNet ← salaire - (salaire * RETENUE) / 100
        afficher salaireNet
    sinon
        afficher salaireBrut
    fin si
fin module
```

Exercice 10 – Nombres de jours dans un mois

```
module nbJours()
    mois : chaine
    jours : entier
    lire mois
    selon que mois vaut
        "JANVIER", "MARS", "MAI", "JUILLET", "AOÛT", "OCTOBRE", "DÉCEMBRE":
            afficher 31
        "AVRIL", "JUIN", "SEPTEMBRE", "NOVEMBRE":
            afficher 30
        "FÉVRIER":
            afficher 28
    fin selon que
fin module
```

Exercice 11 – Année bissextile

```
module estBissextile()  
  annee : entier  
  lire annee  
  afficher annee MOD 4 = 0 ET NON(annee MOD 100 = 0) OU annee MOD 400 = 0  
fin module
```

Exercice 12 – Valider une date

```
module dateValide()  
  annee, mois, jour, jourMois : entiers  
  bissextile : booléen  
  lire jour, mois, annee  
  bissextile ← annee MOD 4 = 0 ET annee MOD 100 <> 0 OU annee MOD 400 = 0  
  selon que mois vaut  
    1, 3, 5, 7, 8, 10, 12:  
      jourMois ← 31  
    4, 6, 9, 11:  
      jourMois ← 30  
    2:  
      si bissextile alors  
        jourMois ← 29  
      sinon  
        jourMois ← 28  
      fin si  
    autres :  
      afficher "mois inconnu"  
  fin selon que  
  afficher  $1 \leq \text{jour} \leq \text{jourMois}$   
fin module
```

Exercice 13 – Le jour de la semaine

```
module jourSemaine()  
  dateMois : entier  
  lire dateMois  
  selon que dateMois MOD 7 vaut  
    0 : afficher "vendredi"  
    1 : afficher "samedi"  
    2 : afficher "dimanche"  
    3 : afficher "lundi"  
    4 : afficher "mardi"  
    5 : afficher "mercredi"  
    6 : afficher "jeudi"  
  fin selon que  
fin module
```

Exercice 14 – Quel jour serons-nous ?

```
module jourFutur()
  jour : chaîne
  n, jourFutur : entier
  lire jour, n
  selon que jour vaut
    "lundi" : jourFutur ← 1
    "mardi" : jourFutur ← 2
    "mercredi" : jourFutur ← 3
    "jeudi" : jourFutur ← 4
    "vendredi" : jourFutur ← 5
    "samedi" : jourFutur ← 6
    "dimanche" : jourFutur ← 7
  fin selon que
  selon que (jourFutur + n) MOD 7 vaut
    0 : afficher "lundi"
    1 : afficher "mardi"
    2 : afficher "mercredi"
    3 : afficher "jeudi"
    4 : afficher "vendredi"
    5 : afficher "samedi"
    6 : afficher "dimanche"
  fin selon que
fin module
```

Exercice 15 – Un peu de trigono

```
module cosinus()
  n : entier
  cosinus : entier
  lire n
  si NON(n MOD 2 = 0) alors
    cosinus ← 0
  sinon
    si (n/2) MOD 2 = 0 alors
      cosinus ← 1
    sinon
      cosinus ← -1
    fin si
  fin si
  afficher cosinus
fin module
```

Exercice 16 – Le stationnement alternatif

```
module bienStationné()  
  dateJour, numMaison : entiers  
  si  $1 \geq \text{dateJour} \geq 15$  alors  
    afficher NON(numMaison MOD 2 = 0)  
  sinon  
    afficher numMaison MOD 2 = 0  
  fin si  
fin module
```

Correction des exercices 5.8

Exercice 4 – Échange de variables

```
module swap(a↓↑,b↓↑ : entiers)
  a ← a + b
  b ← a - b
  a ← a - b
fin module
```

Exercice 5 – Valeur absolue

```
module abs(a : réel) → réel
  si a < 0 alors
    a ← -a
  fin si
  retourner a
fin module
```

Exercice 6 – Maximum de 4 nombres

```
module max4Nb(a↓,b↓,c↓,d↓ : entiers) → entier
  max : entier
  max ← max2(max3(a,b,c), d)
  retourner max
fin module
```

Exercice 7 – Validité d'une date

```
module dateValide(jour, mois, annee : entiers) → booléen
  retourner 1 ≤ jour ≤ nbJours(mois, annee)
fin module
module estBissextile(annee : entier) → booléen
  retourner annee MOD 4 = 0 ET NON(annee MOD 100 = 0) OU annee MOD 400 = 0
fin module
module nbJours(mois, annee : entiers) → entier
  selon que mois vaut
    1, 3, 5, 7, 8, 10, 12:
      retourner 31
    4, 6, 9, 11:
      retourner 30
    2:
      si estBissextile(annee) alors
        retourner 29
      sinon
        retourner 28
      fin si
    autres:
      retourner 0
  fin selon que
fin module
```

Correction des exercices 6.6

Structure – Moment

```
structure Moment
|   heure : entier
|   minute : entier
|   seconde : entier
fin structure
```

Exercice 1 – Conversion moment-secondes

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

Exercice 2 – Conversion secondes-moment

```
module secondesVersMoments(secondes↓ : entier) → Moment
|   moment : Moment
|   moment ← {secondes DIV 3600, (secondes MOD 3600) DIV 60, secondes MOD 60}
|   retourner moment
fin module
```

Exercice 3 – Temps écoulé entre 2 moments

```
module écartEntreMoments(moment1, moment2 : Moments) → entier
|   si secondeMinuit(moment1) > secondeMinuit(moment2) alors
|       |   retourner secondeMinuit(moment1) - secondeMinuit(moment2)
|   sinon
|       |   retourner secondeMinuit(moment2) - secondeMinuit(moment1)
|   fin si
fin module
```

Exercice 4 – Milieu de deux points

```
module MilieuSegment(a, b : Points) → Point
|   retourner  $\left\{ \frac{a.x + b.x}{2}, \frac{a.y + b.y}{2} \right\}$ 
fin module
```

Exercice 5 – Distance entre deux points

```
module LongueurSegment(a, b : Points) → entier
|   retourner  $\sqrt{(b.x - a.x)^2 + (b.y - a.y)^2}$ 
fin module
```

Exercice 6 – Un cercle

```
structure Cercle
|   centre : Point
|   rayon : réel
fin structure

module SurfaceCercle(cercle : Cercle) → réel
|   retourner  $\pi * (\text{cercle.rayon})^2$ 
fin module

module CréeCercle(a, b : Points) → Cercle
|   cercle : Cercle
|   cercle.centre ← MilieuSegment(a,b)
|   cercle.rayon ← LongueurSegment(a,b) / 2
|   retourner cercle
fin module

module pointDansCercle(point : Point, cercle : Cercle) → booléen
|   retourner LongueurSegment(point, cercle.centre) < cercle.rayon
fin module

module IntersectionCercle(cercle1, cercle2 : Cercles) → booléen
|   distanceCentre : réel
|   distanceCentre ← LongueurSegment(cercle1.centre, cercle2.centre)
|   retourner distanceCentre - (cercle1.rayon * 2 - cercle2.rayon * 2) ≤ 0
fin module
```

Exercice 7 – Un rectangle

```
structure Rectangle
  bg : Point
  hd : Point
fin structure

module périmètreRectangle(rect : Rectangle) → réel
  base, hauteur : réels
  base ← LongueurSegment(rect.bg.x, rect.hd.x)
  hauteur ← LongueurSegment(rect.bg.y, rect.hd.y)
  retourner 2 * base + 2 * hauteur
fin module

module aireRectangle(rect : Rectangle) → réel
  base, hauteur : réels
  base ← LongueurSegment(rect.bg.x, rect.hd.x)
  hauteur ← LongueurSegment(rect.bg.y, rect.hd.y)
  retourner base * hauteur
fin module

module pointDansRectangle(rect : Rectangle, point : Point) → booléen
  retourner (rect.bg.x ≤ point.x ≤ rect.hd.x) ET (rect.bg.y ≤ point.y ≤ rect.hd.y)
fin module

module pointSurBordRectangle(rect : Rectangle, point : Point) → booléen
  surBaseBas, surBaseHaut, surHauteurGauche, surHauteurDroit : booléens
  surBaseBas ← rect.bg.x ≤ point.x ≤ rect.hd.x ET (rect.bg.y = point.y)
  surBaseHaut ← rect.bd.x ≤ point.x ≤ rect.hd.x ET (rect.hd.y = point.y)
  surCotéGauche ← rect.bg.y ≤ point.y ≤ rect.hd.y ET (rect.bg.x = point.x)
  surCotéDroit ← rect.bg.y ≤ point.y ≤ rect.hd.y ET (rect.hd.x = point.x)
  retourner surBaseBas OU surBaseHaut OU surCotéGauche OU surCotéDroit
fin module
```

Correction des exercices 7.4

Exercice 3 – Afficher les n premiers

```
module strictementPositifs()
  n, i : entiers
  lire n
  pour i de 1 à n faire
    afficher i
  fin pour
fin module

module strictementPositifsDécroissants()
  n, i : entiers
  lire n
  pour i de n à 1 par -1 faire
    afficher i
  fin pour
fin module

module carrésParfaits()
  n, i : entiers
  lire n
  pour i de 1 à n faire
    afficher  $i^2$ 
  fin pour
fin module

module naturelsImpairs()
  n, nb, i : entiers
  lire n
  nb  $\leftarrow$  1
  pour i de 1 à n faire
    afficher nb
    nb  $\leftarrow$  nb + 2
  fin pour
fin module

module naturelsImpairsInférieurs()
  n, i : entiers
  lire n
  pour i de 1 à n par 2 faire
    afficher i
  fin pour
fin module
```

Exercice 4 – Maximum de cotes

```
module maxCote()  
  cote, max : entier  
  cote  $\leftarrow$  0  
  max  $\leftarrow$  0  
  lire cote  
  tant que cote  $\geq$  0 faire  
    si cote > max alors  
      max  $\leftarrow$  cote  
    fin si  
    lire cote  
  fin tant que  
  afficher max  
fin module
```

Exercice 5 – Afficher les multiples de 3

```
module multiplesDe3()  
  nombre, multiple3 : entiers  
  nombre  $\leftarrow$  1  
  lire nombre  
  tant que nombre  $\neq$  0 faire  
    si nombre MOD 3 = 0 alors  
      afficher nombre  
      multiple3  $\leftarrow$  multiple3 + 1  
    fin si  
    lire nombre  
  fin tant que  
  afficher multiple3  
fin module
```

Exercice 6 – Placement d'un capital

```
module placementCapital()  
  capitalDépart, nbAnnées, tauxPlacement, capitalIntérêt : entiers  
  lire capitalDépart, nbAnnées, tauxPlacement  
  pour année de 2013 à n faire  
    capitalIntérêt  $\leftarrow$  capitalDépart + (tauxPlacement * capitalDépart)/100  
    afficher année, capitalDépart, capitalIntérêt - capitalDépart  
  fin pour  
fin module
```


Exercice 7 – Produit de 2 nombres

```
module min(nb1, nb2 : entiers) → entier
  si nb1 < nb2 alors
    retourner nb1
  sinon
    retourner nb2
  fin si
fin module
module produit2Nb(nb1, nb2 : entiers) → entier
  max, min, somme : entiers
  somme ← 0
  max ← max(nb1, nb2)
  min ← min(nb1, nb2)
  pour i de 1 à abs(min) faire
    somme ← somme + max
  fin pour
  si min < 0 alors
    somme ← -somme
  fin si
  retourner somme
fin module
```

Exercice 8 – Génération de suites (1/2)

```
module pasCroissant(n : entier)
  nb : entier
  nb ← 1
  pour i de 1 à n faire
    afficher nb
    nb ← nb + i
  fin pour
fin module

module boiteuse(n : entier)
  nb, pas : entiers
  nb ← 1
  pas ← 2
  pour i de 1 à n faire
    afficher nb
    pas ← 3 - pas
    nb ← nb + pas
  fin pour
fin module

module suiteDeFibonacci(n : entier)
  val, prec : entnbers
  val ← 0
  prec ← 1
  pour i de 1 à n faire
    afficher val
    val ← val + prec
    prec ← val - prec
  fin pour
fin module
```

Exercice 8 – Génération de suites (2/2)

```
module processionEchternach(n : entier)
  val, i : entier
  val ← 1
  pour i de 1 à n faire
    afficher val
    si  $1 \leq i \bmod 5$  ET  $i \bmod 5 \leq 3$  alors
      val ← val + 1
    sinon
      val ← val - 1
    fin si
  fin pour
fin module
```

```
module combinaison2Suites(n : entier)
  première : booléen
  val1, val2 : entiers
  première ← vrai
  val1 ← 1
  val2 ← 2
  pour i de 1 à n faire
    si première alors
      afficher val1
      val1 ← val1 + 2
    sinon
      afficher val2
      val2 ← val2 + 1
    fin si
    première ← NONpremière
  fin pour
fin module
```

```
module capricieuse(n : entier)
  val, pas : entiers
  descendant : booléen
  val, pas ← 1
  descendant ← faux
  pour i de 1 à n faire
    afficher val
    pas ← pas + 1
    si pas = 10 alors
      val ← val + 10
      descendant ← NON descendant
    fin si
    si descendant alors
      val ← val - 1
    sinon
      val ← val + 1
    fin si
  fin pour
fin module
```

Exercice 9 – Factorielle

```
module factorielle(n : entier) → entier
  si n > 1 alors
    retourner n * factorielle(n - 1)
  sinon
    retourner 1
  fin si
fin module
```

Exercice 10 – Somme de chiffres

```
module sommeChiffre()
  chiffre, nombre : entiers
  chiffre ← 0
  lire nombre
  tant que nombre > 0 faire
    chiffre ← chiffre + nombre MOD 10
    nombre ← nombre / 10
  fin tant que
  afficher chiffre
fin module
```

Exercice 11 – Conversion binaire-décimal

```
module BinaireDécimal()
  décimal, i : réels
  nb : entier
  décimal ← 0
  i ← 0
  lire nb
  tant que nb > 0 faire
    décimal ← nb MOD 10 * 2i + décimal
    nb ← nb / 10
    i ← i + 1
  fin tant que
  retourner décimal
fin module
```

Exercice 12 – Conversion décimal-binaire

```
module DécimalBinaire()  
  binaire : chaîne  
  nb : entier  
  lire nb  
  tant que nb > 0 faire  
    binaire ← concat(binaire, nb MOD 2)  
    nb ← nb / 2  
  fin tant que  
  pour i de long(binaire) à 1 par -1 faire  
    afficher car(binaire, i)  
  fin pour  
  afficher binaire  
fin module
```

Exercice 13 – PGCD

```
module PGCD(a, b : entiers) → entier  
  reste : entier  
  tant que reste != 0 faire  
    reste ← a MOD b  
    a ← b  
    b ← reste  
  fin tant que  
  retourner a  
fin module
```

Exercice 14 – PPCM

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

Exercice 15 – Nombre premier

```
module estPremier(nb : entier) → booléen
  si nb = 0 OU nb = 1 alors
    retourner faux
  fin si
  si nb = 2 alors
    retourner vrai
  fin si
  si nb MOD 2 = 0 alors
    retourner faux
  fin si
  pour i de 3 à  $\sqrt{nb}$  par 2 faire
    si nb MOD i = 0 alors
      retourner faux
    fin si
  fin pour
  retourner vrai
fin module
```

Exercice 16 – Nombres premiers

```
module nombresPremiers(nb : entier)
  pour i de 1 à nb par 2 faire
    si estPremier(i) alors
      afficher i
    fin si
  fin pour
fin module
```

Exercice 17 – Nombre parfait

```
module nbParfait(nb : entier) → booléen
  p : entier
  si nb = 6 alors
    retourner vrai
  fin si
  p ← 3
  tant que nb <  $2^{(p-1)}(2^p - 1)$  faire
    p ← p + 2
  fin tant que
  retourner n =  $2^{(p-1)}(2^p - 1)$ 
fin module
```

Exercice 18 – Décomposition en facteurs premiers

```
module facteursPremiers(nb : entier)
  i : entier
  i ← 2
  tant que i ≤ nb faire
    tant que nb MOD i = 0 ET estPremier(i) faire
      afficher i
      nb ← nb/i
      si nb ≥ 1 alors
        afficher " * "
      fin si
    fin tant que
    i ← i + 1
  fin tant que
fin module
```

Exercice 19 – Palindrome

```
module palindrome(nb : entier) → booléen
  palindrome, inverse : entiers
  inverse ← 0
  palindrome ← nb
  tant que palindrome ≠ 0 faire
    chiffre ← palindrome MOD 10
    inverse ← inverse * 10 + chiffre
    palindrome ← palindrome / 10
  fin tant que
  retourner nb = inverse
fin module
```

Exercice 20 – Jeu de la fourchette

```
module fourchette()
  nbRandom, essai, tentative : entiers
  tentative ← 0
  nbRandom ← hasard(100)
  faire
    afficher "Entrez un nombre"
    lire essai
    selon que
      nbRandom > essai:
        afficher "le nb est plus grand"
      nbRandom < essai:
        afficher "le nb est plus petit"
    fin selon que
    tentative ← tentative + 1
  jusqu'à ce que tentative = 8 OU nbRandom = essai
  si nbRandom ≠ essai alors
    afficher "désolé, le nb était ", nbRandom
  sinon
    afficher "bravo, vous avez trouvé en ", tentative, " coups"
  fin si
fin module
```

Exercice 21 – IMC

```
module IMC()
  imc : réel
  sexe, poids, taille : réels
  obèse, nbPersonnes : entier
  faire
    lire sexe, poids, taille
    imc ← poids / taille2
    si imc > 30 alors
      obèse ← obèse + 1
    fin si
    nbPersonnes ← nbPersonnes + 1
  jusqu'à ce que sexe ≠ 'H' ET sexe ≠ 'F'
  afficher obèse / nbPersonnes * 100
fin module
```

Exercice 22 – Cotes

```
module cotesEtudiants()
  coteInt, coteExam, coteTotal, coteMax : réel
  absence, réussite, nbCote : entier
  fin : booléen
  réponse : chaîne
  tant que fin = faux faire
    pour i de 1 à 3 faire
      lire cote
      si cote = -1 alors
        cote ← absence
      sinon
        coteInt ← coteInt + cote
        nbCote ← nbCote + 1
      fin si
    fin pour
    lire coteExam
    nbCote ← nbCote + 1
    si coteExam = -1 alors
      coteExam ← 0
    fin si
    selon que absence vaut
      1:
        coteExam ← coteExam / 100 * 120
      2:
        coteExam ← coteExam / 100 * 140
      3:
        coteExam ← coteExam / 100 * 160
    fin selon que
    coteTotal ← (coteInt + coteExam) / 8
    si coteTotal > 12 alors
      réussite ← réussite + 1
    fin si
    coteMax ← max2Nb(coteMax, coteTotal)
    afficher "Y a-t-il encore des cotes à rentrer ?"
    lire réponse
    fin ← réponse = "non"
  fin tant que
  afficher nbCote, coteMax, réussite / nbCote * 100
fin module
```

Exercice 23 – Normaliser une chaîne

```
module versAlpha(chaine : chaîne) → chaîne
  chaineAlpha : chaîne
  lettre : caractère
  pour lettre de 1 à long(chaine) faire
    lettre ← majuscule(car(chaineAlpha, lettre))
    si estLettre(lettre) alors
      chaineAlpha ← concat(chaineAlpha, lettre)
    fin si
  fin pour
  retourner chaineAlpha
fin module
```


Exercice 24 – Le chiffre de César (1/2)

```
module chiffrementCésar(msgClair : chaine, k : entier) → chaine
  msgChiffré : chaine
  carClair, carChiffré : caractères
  msgChiffré ← ""
  pour i de 1 à long(msgClair) faire
    carClair ← numLettre(car(msgClair, carClair)) + k
    si carClair > 26 alors
      carClair ← carClair MOD 26
    fin si
    si carClair < 1 alors
      carClair ← abs(carClair - 1) MOD 26 + 1
    fin si
    carChiffré ← lettre(carClair)
    msgChiffré ← concat(msgChiffré, carChiffré)
  fin pour
  retourner msgChiffré
fin module
```

Exercice 24 – Le chiffre de César (2/2)

```
module déchiffrementCésar(message : chaine, k : entier) → chaine
  retourner chiffrementCésar(message, -k)
fin module
```

Exercice 24 – Le chiffre de César (solution Github)

```
module chiffrerCésar(msgClair↓ : chaine, déplacement↓ : entier) → chaine
  msgChiffré : chaine
  carClair, carChiffré : caractères
  i : entier

  msgChiffré ← ""
  pour i de 1 à long(msgClair) faire
    carClair ← car(msgClair, i)
    carChiffré ← avancer(carClair, déplacement)
    msgChiffré ← concat( msgChiffré, chaine(carChiffré) )
  fin pour
  retourner msgChiffré
fin module

// Calcule la lettre qui est "delta" position plus loin dans l'alphabet (circulairement)
module avancer(lettre↓ : caractère, delta↓ : entier) → caractère
  retourner lettre( (numLettre(lettre) + delta - 1) MOD 26 + 1 )
fin module

// Déchiffrer un message chiffré avec le chiffre de César
module déchiffrerCésar(msgClair↓ : chaine, déplacement↓ : entier) → chaine
  retourner chiffrerCésar(msgClair, 26 - déplacement)
fin module
```