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 – 1ère année

Cours enseigné par :

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

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 LaTeX.
Version du 14 octobre 2013.

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

Table des matières

Correction des exercices 4.4

Simplification d'algorithme

```
si ok alors
    afficher nombre
fin si

ok ← condition
```

```
si NON ok alors
    afficher nombre
fin si

ok ← a ≥ 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 ≥ nb1 alors
        max ← nb2
    sinon
        max ← 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 ≥ nb1 alors
        max ← nb2
    sinon
        max ← nb1
    fin si
    si nb3 ≥ max alors
        max ← 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 ± √delta)/(2 * coeffCarré)
        delta = 0 : afficher -coeff/(2 * 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 ≤ jour ≤ 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 : chaine
    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 ≥ dateJour ≥ 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:
        erreur "mois inconnu"
    fin selon que
fin module
```

Exercice 7 – Valider une date

```
module dateValide(jour, mois, annee : entiers) → booléen
    jourMois : entier
    bissextile : booléen
    selon que mois vaut
     1, 3, 5, 7, 8, 10, 12:
        jourMois ← 31
     4, 6, 9, 11:
        jourMois ← 30
     2:
        bissextile ← annee MOD 4 = 0 ET annee MOD 100 <> 0 OU annee MOD 400 = 0
        si bissextile alors
            jourMois ← 29
        sinon
            jourMois ← 28
        fin si
     autres :
        erreur "mois inconnu"
    fin selon que
    retourner 1 ≤ jour ≤ jourMois
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 / 3600 , (secondes MOD 3600) / 60 , secondes MOD 60 }
    retourner moment
fin module
```

Exercice 3 – Temps écoulé entre 2 moments

```
module écartEntreMoments(moment1, moment2 : Moments) → entier
    retourner SecondesMinuit(moment1) - SecondesMinuit(moment2)
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 $\pi *$ (cercle.rayon)^2
fin module
module *CréeCercle*(a, b : Points) → Cercle
  cercle : Cercle
  cercle.centre ← a
  cercle.rayon ← LongueurSegment(a,b)
  retourner cercle
fin module
module *pointDansCercle*(point : Point, cercle : Cercle) → booléen
  retourner $\sqrt{(\text{point.x} - \text{cercle.centre.x})^2 + (\text{cercle.centre.y} - \text{point.y})} < \text{cercle.rayon}$
fin module
module *IntersectionCercle*(cercle1, cercle2 : Cercles) → booléen
  distanCentre : 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
    bd : Point
    hg : Point
    hb : Point
fin structure
module périmètreRectangle(rect : Rectangle) → réel
    retourner rect.bg + rect.bg + rect.hg + rect.hd
fin module
module aireRectangle(rect↓ : Rectangle) → réel
    retourner LongueurSegment(rect.bg, rect.bd) * LongueurSegment(rect.hg, rect.hd)
fin module
module pointDansRectangle(rect : Rectangle, point : Point) → booléen
    retourner (rect.bg.x ≤ point.x ≤ rect.bd.x) ET (rect.hg.y ≤ point.y ≤ rect.bg.y)
fin module
module pointSurBordRectangle(rect : Rectangle, point : Point) → booléen
    selon que
        rect.bg.x ≤ point.x ≤ rect.bd.x ET rect.bg.y == point.y : retourner vrai
        rect.hg.x ≤ point.x ≤ rect.hd.x ET rect.hg.y == point.y : retourner vrai
        rect.bg.x == point.x ET rect.bg.y ≤ point.y ≤ rect.hg.y : retourner vrai
        rect.bg.x == point.x ET rect.bd.y ≤ point.y ≤ rect.hd.y : retourner vrai
        autres : retourner faux
    fin selon que
fin module
```