Algorithmique & Programmation
Semestre 2 ST
Les types agrégés

Rappels du semestre 1

Types agrégés	Structuration en sous-algorithmes	Tableaux de variables	Algorithmes de recherche et de tri	Précision, rapidité et complexité
Matrices de variables	Récursivité	Travaux dirigés	Travaux pratiques	Informations
Sujets de projet	Evaluation intermédiaire	Evaluation finale	Evaluation complémentaire	Archives

Cours TD - Corrections TP

Fichier PDF

Tous les exercices sont à réaliser en langage algorithmique.

Exercice n°1

Pour une fédération sportive, on souhaite développer un programme de gestion des licences.
Concevoir un type agrégé "licence" de stockage des informations relatives à un licencié:

- nom,
- prénom,
- numéro de licence,
- club,
- vétéran ou non.

TypeLicence.lda

```
{ Type agrege de stockage des informations     }
{ relatives a un licencie sportif              }

structure licence
  nom : chaine                  <- ""
```

```
    prenom : chaine              <- ""
    numeroLicence : entier       <- 0
   club : chaine                <- ""
    categorieAge : caractere     <- ' '
    sexe : booleen               <- vrai
fin structure
```

Exercice n°2

a) Concevoir un type agrégé "position3D" de stockage d'une position définie dans un espace à trois dimensions réelles.

b) Concevoir un algorithme de lecture au clavier d'une variable de type position3D, puis d'affichage à l'écran de cette variable.

LectureEcriturePosition3D.lda

```
{ Type agrege de stockage des informations     }
{ relatives a une position en trois dimensions }

structure position3D
  x : reel <- 0.0
  y : reel <- 0.0
  z : reel <- 0.0
fin structure

action principale()
   locales
    pos : position3D
  pos.x <- Clavier.saisirReel()
  pos.y <- Clavier.saisirReel()
  pos.z <- Clavier.saisirReel()
  Ecran.afficher("(",pos.x,",",pos.y,",",pos.z,")")
fin action
```

Exemple d'exécution

Exercice n°3

a) Concevoir un type agrégé "temps" de stockage d'une heure représentée par un nombre d'heures, de minutes et de secondes.

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture au clavier de deux variables de type temps.
- Affichage de l'heure la plus tardive parmis les deux, puis affichage de l'autre heure.
- Affichage du nombre de secondes entre les deux.

Pour comparer deux temps, on pourra les convertir en secondes et effectuer la comparaison sur les nombres de secondes obtenus.

ManipulationTemps.lda

```
{ Type agrege de stockage des informations      }
{ relatives a un temps code sous la forme       }
{ heure, minute, seconde                        }

structure temps
  h : entier  <- 0
  mn : entier <- 0
  s : entier  <- 0
fin structure

{ Programme principal                           }

action principale()
   Locales
    t1 : temps
    t2 : temps
    inferieurEgalT1T2 : booleen
    difference : entier
    s1 : entier
    s2 : entier
  Ecran.afficherln("SVP, t1 (h, mn & s)?")
  t1.h <- Clavier.saisirEntier()
  t1.mn <- Clavier.saisirEntier()
  t1.s <- Clavier.saisirEntier()
  Ecran.afficherln("SVP, t2 (h, mn & s)?")
  t2.h <- Clavier.saisirEntier()
  t2.mn <- Clavier.saisirEntier()
  t2.s <- Clavier.saisirEntier()
  s1 <- 3600*t1.h+60*t1.mn+t1.s
  s2 <- 3600*t2.h+60*t2.mn+t2.s
  inferieurEgalT1T2 <- (s1 <= s2)
   si inferieurEgalT1T2 = vrai alors
    difference <- s2-s1
    Ecran.afficherln("[",t2.h,":",t2.mn,":",t2.s,"]")
    Ecran.afficherln("[",t1.h,":",t1.mn,":",t1.s,"]")
     sinon
    difference <- s1-s2
    Ecran.afficherln("[",t1.h,":",t1.mn,":",t1.s,"]")
    Ecran.afficherln("[",t2.h,":",t2.mn,":",t2.s,"]")
   fsi
  Ecran.afficherln("Ecart : ",difference," secondes")
fin action
```

Exemple d'exécution

Exercice n°4

a) Concevoir un type agrégé "couleur" de stockage d'une couleur représentée par une valeur de rouge, une valeur de vert et une valeur de bleu. Ces trois valeurs sont de type entier. Usuellement en informatique, elles sont comprises entre 0 et 255 (8 bits).
Exemples: (255,255,255) -> blanc, (0,0,0) -> noir, (255,0,0) -> rouge, (255,255,0) -> jaune

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture au clavier d'une variable de type couleur.
- Lecture au clavier d'un pourcentage d'assombrissement.
- Calcul et affichage de cette couleur après assombrissement de la valeur lue.
- Lecture au clavier d'un pourcentage d'éclaircissement.
- Calcul et affichage de cette couleur après éclaircissement de la valeur lue.

Pour assombrir une composante de couleur de n %, on poura la multiplier par (1.0-n/100.0).
Pour éclaircir une composante de couleur c de n %, on pourra lui appliquer la formule f(c) = 255.0-((255.0-c)*(1.0-n/100.0)).

ManipulationCouleurs.lda

```
{ Type agrege de stockage des informations      }
{ relatives a une couleur RVB                   }

structure couleur
  r : entier <- 0
  v : entier <- 0
  b : entier <- 0
fin structure

{ Programme principal                           }

action principale()
   Locales
    c : couleur
    assombrissement : reel
    ca : couleur
    eclaircissement : reel
    ce : couleur
    fa : reel
    fe : reel
  Ecran.afficherln("SVP, r, v & b?")
  c.r <- Clavier.saisirEntier()
  c.v <- Clavier.saisirEntier()
```

```
  c.b <- Clavier.saisirEntier()
  Ecran.afficherln("SVP, assombrissement?")
  assombrissement <- Clavier.saisirReel()
  Ecran.afficherln("SVP, eclaircissement?")
  eclaircissement <- Clavier.saisirReel()
  fa <- 1.0-assombrissement/100.0
  ca.r <- round(fa*c.r)
  ca.v <- round(fa*c.v)
  ca.b <- round(fa*c.b)
  fe <- 1.0-eclaircissement/100.0
  ce.r <- 255-round(fe*(255-c.r))
  ce.v <- 255-round(fe*(255-c.v))
  ce.b <- 255-round(fe*(255-c.b))
  Ecran.afficherln("[",c.r,":",c.v,":",c.b,"]")
  Ecran.afficherln("[",ca.r,":",ca.v,":",ca.b,"]")
  Ecran.afficherln("[",ce.r,":",ce.v,":",ce.b,"]")
fin action
```

Exemple d'exécution

Exercice n°5

a) Concevoir un type agrégé "position2D" de stockage d'une position définie dans un espace à deux dimensions réelles.

b) Concevoir un type agrégé "triangle2D" de stockage d'un triangle de 3 position2D.

c) Concevoir un algorithme réalisant les traitements suivants:
- Lecture au clavier d'un triangle2D.
- Calcul et affichage de la surface de ce triangle2D.

Pour calculer la surface d'un triangle défini par les 3 sommets (x1,y1), (x2,y2) et (x3,y3), on pourra utiliser la formule suivante:
surface = abs((x1+x2)*(y1-y2)+(x2+x3)*(y2-y3)+(x3+x1)*(y3-y1))/2.0

ManipulationTriangle2D.lda

```
{ Type agrege de stockage des informations     }
{ relatives a une position en deux dimensions  }

structure position2D
  x : reel <- 0.0
  y : reel <- 0.0
fin structure

{ Type agrege de stockage des informations     }
{ relatives a un triangle en deux dimensions   }
```

```
structure triangle2D
  p0 : position2D
  p1 : position2D
  p2 : position2D
fin structure

{ Programme principal                                    }

action principale()
   Locales
    t : triangle2D
    surface : double
  Ecran.afficherln("SVP, x & y du sommet 1?")
  t.p0.x <- Clavier.saisirReel()
  t.p0.y <- Clavier.saisirReel()
  Ecran.afficherln("SVP, x & y du sommet 2?")
  t.p1.x <- Clavier.saisirReel()
  t.p1.y <- Clavier.saisirReel()
  Ecran.afficherln("SVP, x & y du sommet 3?")
  t.p2.x <- Clavier.saisirReel()
  t.p2.y <- Clavier.saisirReel()
  surface <- (t.p0.x-t.p1.x)*(t.p0.y+t.p1.y)+
             (t.p1.x-t.p2.x)*(t.p1.y+t.p2.y)+
             (t.p2.x-t.p0.x)*(t.p2.y+t.p0.y)
  surface <- abs(surface)/2.0
  Ecran.afficherln("Surface: ",surface)
fin action
```

Exemple d'exécution

Exercice n°6

a) Concevoir un type agrégé "parallelepipede" de stockage des caractéristiques d'un parallélépipède rectangle à faces perpendiculaires aux axes.

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture d'un parallelepipede.
- Calcul et affichage du volume de ce parallelepipede.

Le volume d'un parallelepipede est égal au produit de la longueur de ses cotés.

Plusieurs solutions sont possibles. Il est intéressant de choisir la plus pratique vis à vis des traitements qui seront réalisés.

ManipulationCube.lda

```
{ Type agrege de stockage des informations       }
{ relatives a une position en trois dimensions   }

structure position3D
  x : reel <- 0.0
  y : reel <- 0.0
  z : reel <- 0.0
fin structure

{ Type agrege de stockage des informations       }
{ relatives a un parallelepipede rectangle       }
{ a faces orthogonales aux axes                  }
{ min: sommet ou les x, y et z sont minimum      }
{ max: sommet ou les x, y et z sont maximum      }

structure parallelepipede
  min : position3D
  max : position3D
fin structure

{ Type agrege de stockage des informations       }
{ relatives a un parallelepipede rectangle       }
{ a faces orthogonales aux axes                  }
{ centre: position du centre du parallelepipede  }
{ largeur, hauteur et profondeur:                }
{ longueurs des cotes selon les axes x, y, et z  }

structure parallelepipede2
  centre : position3D
  largeur : reel
  hauteur : reel
  profondeur : reel
fin structure

{ Type agrege de stockage des informations       }
{ relatives a un parallelepipede rectangle       }
{ a faces orthogonales aux axes                  }
{ min: sommet ou les x, y et z sont minimum      }
{ largeur, hauteur et profondeur:                }
{ longueurs des cotes selon les axes x, y, et z  }

structure parallelepipede3
  min : position3D
  largeur : reel
  hauteur : reel
  profondeur : reel
fin structure

{ Type agrege de stockage des informations      }
{ relatives a un parallelepipede rectangle      }
```

```
{ a faces orthogonales aux axes                    }
{ s1: un sommet quelconque du parallelepipede      }
{ s2: le sommet oppose de s1                       }

structure parallelepipede4
  s1 : position3D
  s2 : position3D
fin structure

{ Programme principal                              }

action principale()
   Locales
    cb : parallelepipede
    volume : reel
  Ecran.afficherln("SVP, x, y & z du sommet min?")
  cb.min.x <- Clavier.saisirReel()
  cb.min.y <- Clavier.saisirReel()
  cb.min.z <- Clavier.saisirReel()
  Ecran.afficherln("SVP, x, y & z du sommet max?")
  cb.max.x <- Clavier.saisirReel()
  cb.max.y <- Clavier.saisirReel()
  cb.max.z <- Clavier.saisirReel()
  volume <- (cb.max.x-cb.min.x)*
            (cb.max.y-cb.min.y)*
            (cb.max.z-cb.min.z)
  Ecran.afficherln("Volume: ",volume)
fin action
```

Exemple d'exécution

Exercice n°7

a) Concevoir un type agrégé "date" de stockage d'une date représentée par un numéro de jour, un numéro de mois et un numéro d'année.

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture d'une date.
- Calcul de la date du lendemain.
- Affichage de cette date.

Etablir la date du lendemain à partir d'une date quelconque nécessite une analyse pour savoir si le jour est le dernier du mois au quel cas, il faudra passer au premier du mois suivant. Le passage au mois suivant peut lui-même entraîner le passage à l'année suivante.
Etablir le nombre de jours d'un mois est un traitement relativement complexe car ce nombre est égal à 31 pour les mois de janvier, mars, mai, juillet, aout, octobre et

décembre, à 30 pour les mois d'avril, juin, septembre et novembre, à 28 pour le mois de février des années non bissextiles et à 29 pour le mois de février des années bissextiles.
Une année n'est pas bissextile si son numéro n'est pas divisible par 4. Si son numéro est divisible par 4, une année est bissextile. Cette règle admet comme exceptions les années dont le numéro est divisible par 100 (qui ne sont donc pas bissextiles). Cette règle admet elle-même comme exceptions les années dont le numéro est divisible par 400 (qui sont donc bissextiles).
Exemples: 1983 n'est pas bissextile, 1980 est bissextile, 2100 n'est pas bissextile, 2000 est bissextile.

ManipulationDates.lda

```
{ Type agrege de stockage des informations      }
{ relatives a une date codee sous la forme      }
{ jour, mois, annee                             }

structure date
  j : entier <- 1
  m : entier <- 1
  a : entier <- 1901
fin structure

{ Programme principal                           }

action principale()
   Locales
    dt : date
    nbJours : entier
  Ecran.afficherln("SVP, j, m & a?")
  dt.j <- Clavier.saisirEntier()
  dt.m <- Clavier.saisirEntier()
  dt.a <- Clavier.saisirEntier()
   dans le cas de dt.m
    2 :
       si ( (dt.a modulo 400) = 0 ) ou
         ( ( (dt.a modulo 4) = 0 ) et
           ( (dt.a modulo 100) <> 0 ) ) alors
        nbJours <- 29
         sinon
        nbJours <- 28
       fsi
    4  :
    6  :
    9  :
    11 :
      nbJours <- 30
```

```
      autres cas :
       nbJours <- 31
    fcas
   dt.j <- dt.j+1
    si dt.j > nbJours alors
     dt.j <- 1
     dt.m <- dt.m+1
      si dt.m > 12 alors
       dt.m <- 1
       dt.a <- dt.a+1
      fsi
    fsi
   Ecran.afficherln(dt.j,"/",dt.m,"/",dt.a)
fin action
```

Exemple d'exécution

Exercice n°8

a) Concevoir un type agrégé quadrilatère (polygone à 4 sommets) en trois dimensions.

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture au clavier d'un quadrilatère
- Calcul puis affichage du périmètre de ce quadrilatère

ManipulationQuadrilatere.lda

```
{ Manipulations sur une classe quadrlatere   }

{ Type agrege de stockage des informations   }
{ relatives a une position en 3 dimensions   }

structure position3D
  x : reel <- 0.0
  y : reel <- 0.0
  z : reel <- 0.0
fin structure

{ Type agrege de stockage des informations   }
{ relatives a un quadrilatere en 3D          }

structure quadrilatere3D
   position3D p1
   position3D p2
   position3D p3
   position3D p4
fin structure
```

```
{ Programme principal                                    }

action principale()
   Locales
    perimetre : double
    q : quadrilatere3D
  perimetre <- 0.0
  Ecran.afficherln("SVP, les coordonnees du quadrilatere")
  Ecran.afficherln("Sommet 1")
  q.p1.x <- Clavier.saisirReel()
  q.p1.y <- Clavier.saisirReel()
  q.p1.z <- Clavier.saisirReel()
  Ecran.afficherln("Sommet 2")
  q.p2.x <- Clavier.saisirReel()
  q.p2.y <- Clavier.saisirReel()
  q.p2.z <- Clavier.saisirReel()
  Ecran.afficherln("Sommet 3")
  q.p3.x <- Clavier.saisirReel()
  q.p3.y <- Clavier.saisirReel()
  q.p3.z <- Clavier.saisirReel()
  Ecran.afficherln("Sommet 4")
  q.p4.x <- Clavier.saisirReel()
  q.p4.y <- Clavier.saisirReel()
  q.p4.z <- Clavier.saisirReel()
  perimetre <- perimetre + sqrt(pow(q.p1.x-q.p2.x,2.0)+
                                pow(q.p1.y-q.p2.y,2.0)+
                                pow(q.p1.z-q.p2.z,2.0))
  perimetre <- perimetre + sqrt(pow(q.p2.x-q.p3.x,2.0)+
                                pow(q.p2.y-q.p3.y,2.0)+
                                pow(q.p2.z-q.p3.z,2.0))
  perimetre <- perimetre + sqrt(pow(q.p3.x-q.p4.x,2.0)+
                                pow(q.p3.y-q.p4.y,2.0)+
                                pow(q.p3.z-q.p4.z,2.0))
  perimetre <- perimetre + sqrt(pow(q.p4.x-q.p1.x,2.0)+
                                pow(q.p4.y-q.p1.y,2.0)+
                                pow(q.p4.z-q.p1.z,2.0))
  Ecran.afficherln("Perimetre : ",perimetre)
fin action
```

Exemple d'exécution

Exercice n°9

a) Concevoir un type agrégé "temps" de stockage d'une heure représentée par un nombre d'heures, de minutes et de secondes.

b) Concevoir un algorithme réalisant les traitements suivants:
- Lecture au clavier d'un temps t
- Lecture au clavier d'un nombre de secondes nbs

- Modification du temps t par incrémentation de nbs secondes
- Affichage écran de t

IncrementationTemps.lda

```
{ Type agrege de stockage des informations      }
{ relatives a un temps code sous la forme       }
{ heure, minute, seconde                        }

structure temps
  h : entier  <- 0
  mn : entier <- 0
  s : entier  <- 0
fin structure

{ Programme principal                           }

action principale()
   Locales
    t : temps
    nbs : entier
  Ecran.afficherln("SVP, t (h, mn & s)?")
  t.h <- Clavier.saisirEntier()
  t.mn <- Clavier.saisirEntier()
  t.s <- Clavier.saisirEntier()
  Ecran.afficherln("SVP, increment?")
  nbs <- Clavier.saisirEntier()
   si nbs >= 0 alors
    t.s <- t.s+nbs
     sinon
    t.s <- t.s+nbs+86400*(-nbs/86400)+86400
   fsi
   si t.s >= 60 alors
    t.mn <- t.mn+(t.s/60)
    t.s <- t.s modulo 60
     si t.mn >= 60 alors
      t.h <- (t.h+(t.mn/60))%24
      t.mn <- t.mn modulo 60
     fsi
   fsi
  Ecran.afficherln("[",t.h,":",t.mn,":",t.s,"]")
fin action
```

Exemple d'exécution

Exercice n°10

a) Concevoir un type agrégé "localisation géographique" avec nombres de degrés en entier, de minutes en entier et de secondes en réel.
Exemple: Horloge astronomique de la cathédrale de Besançon: Longitude 6°01'49.08", latitude 47°14'00.96"

b) Concevoir un algorithme permettant de convertir une localisation géographique en un nombre de degrés en réel (lecture au clavier de la localisation géographique, conversion et affichage du résultat de conversion). Concevoir un algorithme permettant de convertir un nombre de degrés en réel en une localisation géographique (lecture au clavier du nombre de degrés réel, conversion et affichage de la localisation géographique).

c) Concevoir un type agrégé "localisation GPS" avec longitude et latitude.

d) Concevoir un algorithme de calcul de la distance "à vol d'oiseau" existant entre deux localisations GPS définies à l'altitude 0.0.
Le rayon terrestre est de 6378.137 km. La formule de calcul est la suivante:

dlo = (lo2-lo1)/2.0;
dla = (la2-la1)/2.0;
$a = \sin(dla)^2 + \cos(la1)*\cos(la2)*\sin(dlo)^2$;
d = 2.0*atan2(sqrt(a), sqrt(1.0-a))*rayonTerrestre;

Où lo1, lo2, la1 et la2 sont les longitudes et latitudes en réel des localisations GPS.

ManipulationCoordonneesGPS.lda

```
{ Manipulations sur une classe                  }
{ coordonnees GPS                               }

{ Type agrege de stockage des informations      }
{ relatives a une localisation codee            }
{ sous la forme degres, minutes, secondes       }

structure localisation
  d : entier <- 0
  mn : entier <- 0
  s : reel <- 0.0
fin structure

{ Type agrege de stockage des informations      }
{ relatives a une localisation GPS              }

structure localisationGPS
  lon : localisation
  lat : localisation
fin structure
```

```
{ Programme principal                          }

action principale()
   constante reel PI <- 3.14159
   Locales
    l1 : localisationGPS
    l2 : localisationGPS
    lo1 : reel
    lo2 : reel
    la1 : reel
    la2 : reel
    dlo : reel
    dla : reel
    a : reel
    distance : reel
   { Lecture clavier des 2 coordonnees GPS     }
  Ecran.afficherln("SVP, P1?")
  Ecran.afficherln("Longitude :")
  l1.lon.d <- Clavier.saisirEntier()
  l1.lon.mn <- Clavier.saisirEntier()
  l1.lon.s <- Clavier.saisirReel()
  Ecran.afficherln("Latitude :")
  l1.lat.d <- Clavier.saisirEntier()
  l1.lat.mn <- Clavier.saisirEntier()
  l1.lat.s <- Clavier.saisirReel()
  Ecran.afficherln("SVP, P2?")
  Ecran.afficherln("Longitude :")
  l2.lon.d <- Clavier.saisirEntier()
  l2.lon.mn <- Clavier.saisirEntier()
  l2.lon.s <- Clavier.saisirReel()
  Ecran.afficherln("Latitude :")
  l2.lat.d <- Clavier.saisirEntier()
  l2.lat.mn <- Clavier.saisirEntier()
  l2.lat.s <- Clavier.saisirReel()
   { Conversion en valeurs reelles             }
   { des longitudes et latitudes               }
  lo1 <- l1.lon.d+((l1.lon.s/60.0)+l1.lon.mn)/60.0
  lo2 <- l2.lon.d+((l2.lon.s/60.0)+l2.lon.mn)/60.0
  la1 <- l1.lat.d+((l1.lat.s/60.0)+l1.lat.mn)/60.0
  la2 <- l2.lat.d+((l2.lat.s/60.0)+l2.lat.mn)/60.0
   { Conversion en radians des valeurs         }
   { en degres                                 }
  lo1 <- lo1*PI/180.0
  lo2 <- lo2*PI/180.0
  la1 <- la1*PI/180.0
  la2 <- la2*PI/180.0
   { Calcul de la distance en kilometres       }
  dlo <- (lo2-lo1)/2.0
  dla <- (la2-la1)/2.0
  a <- (sin(dla)*sin(dla))+cos(la1)*cos(la2)*(sin(dlo)*sin(dlo))
```

```
  distance <- 2.0 * atan2(sqrt(a),sqrt(1.0-a)) * 6378.137
   { Affichage final                                }
  Ecran.afficherln("Distance : ",distance," km")
fin action
```

Exemple d'exécution

Auteur: Nicolas JANEY
UFR Sciences et Techniques
Université de Besançon
16 Route de Gray, 25030 Besançon
nicolas.janey@univ-fcomte.fr