

# CS318 Introduction à l'algorithmique

Marius.Bilasco@imag.fr

## Devoir 1 - Corrigé

26 septembre 2005

### 1/ Calcul de la durée d'un événement (structures conditionnelles)

Un événement est défini par l'instant de début et l'instant de fin exprimés tous les deux en heures, minutes et secondes. Écrire un algorithme qui calcule la durée d'un événement. Les instants de début et de fin seront saisis par l'utilisateur de cet algorithme. Si l'instant de fin est situé avant l'instant de début, l'algorithme affichera « Dans ce cas, utilisez votre machine à remonter le temps. »

TYPE

```
Temps=structure
    h, m, s:entier;
fin structure;
```

FONCTION conversionEnHMS(secondes:entier):Temps;

{spécifications

en entrée: un nombre de *secondes*

renvoie: une variable structurée dont les champs (h,m,s) correspondent au nombre d'heures, minutes et secondes équivalent au nombre total de *secondes* en entrée  $3600*h+60*m+s=secondes$

}

FONCTION conversionEnS(temps:Temps):entier;

{spécifications:

en entrée: une variable de type temps (h, m, s)

renvoie: le nombre de secondes écoulées entre l'heure 0:0.0 et l'heure h:m.s  $3600*h+60*m+s$

}

PROCEDURE lireTemps([out]temps:Temps);

{spécifications

en entrée: rien

en sortie: les champs de la variable temps sont saisis au clavier par l'utilisateur

}

ALGO durée\_événement

VAR

[in]tempsD, tempsF : Temps;

[out]tempsEcoulé : Temps;

secondesEcoulées: entier

DEBUT

lireTemps(tempsD);

lireTemps(tempsF);

secondesEcoulées<-conversionEnS(tempsF)-conversionEnS(tempsD);

```
si (secondesEcoulees<0) alors
    Afficher "utilisez la machine a remonter le temps!"
sinon
    tempsEcoule<-conversionEnHMS(secondesEcoulees);
    Afficher tempsEcoule.h, ":", tempsEcoule.m, ".", tempsEcoule.s
fin si
```

FIN

FONCTION conversionEnHMS(secondes:entier):Temps c'est

VAR

temps:Temps;

DEBUT

temps.h<- secondes DIV 3600;

{secondes MOD 3600 represente le nombre de secondes qui restent une fois qu'on a retiré les heures. on divise par 60(s) pour obtenir les minutes}

temps.m<-(secondes MOD 3600) DIV 60;

{secondes MOD 60 represente le nombre de secondes qui restent une fois qu'on a retiré les heures et les minutes}

temps.s<- secondes MOD 60;

renvoyer temps;

FIN

FONCTION conversionEnS(temps:Temps):entier c'est

DEBUT

renvoyer 3600\*temps.h+60\*temps.m+temps.s;

FIN

PROCEDURE lireTemps([out]temps:Temps) c'est

DEBUT

Afficher "Saisissez l'heure (0-23)";

Lire temps.h

Afficher "Saisissez les minutes (0-59)";

Lire temps.m

Afficher "Saisissez les secondes (0-59)";

Lire temps.s;

FIN

## 2/ Somme des nombres pairs et somme des carrés des nombres impairs (structures itératives)

Écrire un algorithme qui permette de calculer la somme des nombre pairs et la somme des carrés des nombres impairs d'une suite d'entiers positifs terminée par le marquer -1. Pour la suite 2 3 1 5 2 -1 les résultats sont: somme pairs=  $2+2=4$  et somme carrés impairs =  $3*3 + 1*1 + 5*5 = 35$

ALGO sommes\_version\_astucieuse\_mais\_coûteuse

VAR

[in]valeur:entier;  
[out]somme\_paires, somme\_carrés\_impaires:entier;

DEBUT

somme\_paires<-0; somme\_carrés\_impaires<-0;  
lire valeur;  
tant que not(valeur = -1) faire  
    somme\_paires<-somme\_paires+valeur\*(1 - valeur MOD 2);  
    somme\_carrés\_impaires<- somme\_carrés\_impaires+valeur\*valeur\*(valeur MOD 2);  
fin tant que

FIN

ALGO sommes\_version\_classique\_mais\_efficace

VAR

[in]valeur:entier;  
[out]somme\_paires, somme\_carrés\_impaires:entier;

DEBUT

somme\_paires<-0; somme\_carrés\_impaires<-0;  
lire valeur;  
tant que not(valeur = -1) faire  
    si valeur MOD 2 = 0 alors  
        somme\_paires<-somme\_paires+valeur;  
    sinon  
        somme\_carrés\_impaires<- somme\_carrés\_impaires+valeur\*valeur;  
    fin si  
    lire valeur;  
fin tant que

FIN

### 3/ **Produit de deux matrices** (tableaux)

Écrire l'algorithme qui calcule le produit de 2 matrices  $A*B$ . Les tailles de deux matrices ( $\leq 10$ ) seront saisis par l'utilisateur, ainsi que leurs éléments.

Si les tailles des deux matrices ne permettent pas l'opération afficher un message d'erreur.

Utilisez la formule ci-dessous pour calculer les éléments de la matrice résultat:

$$C = A * B \text{ où } c_{ij} = \sum_{k=0..ncA-1} (a_{ik} * b_{kj}) \text{ ou } ncA \text{ représenté le nombre de colonnes de } A.$$

Exemple pour deux matrices carrées de tailles 2x2

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} * \begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} = \begin{vmatrix} a_{11}*b_{11}+a_{12}*b_{21} & a_{11}*b_{12}+a_{12}*b_{22} \\ a_{21}*b_{11}+a_{22}*b_{21} & a_{21}*b_{12}+a_{22}*b_{22} \end{vmatrix}$$

CONST

MAX\_LIGNES:entier=20; MAX\_COLONNES:entier=20;

TYPE

Matrice=tableau [1..MAX\_LIGNES][1..MAX\_COLONNES] de réels;

PROCEDURE lireEntierBorné([in]linf, lsup:entier;[out]valeur:entier); {définie pour l'exercice 3}

{spécifications

état d'entrée:  $linf \leq lsup$

état de sortie:  $linf \leq valeur \leq lsup$

}

PROCEDURE lireEntierBorné([in]linf, lsup:entier;[out]valeur:entier);

{spécifications

état d'entrée:  $linf \leq lsup$

état de sortie:  $linf \leq valeur \leq lsup$

}

PROCEDURE lireMatrice([in]nbLignes,nbColonnes:entier;[out]m:Matrice);

{spécifications

état d'entrée:  $0 \leq nbLignes \leq MAX\_LIGNES$ ,  $0 \leq nbColonnes \leq MAX\_COLONNES$

état de sortie: la matrice m de taille nbLignes, nbColonnes est saisi par l'utilisateur

}

PROCEDURE afficherMatrice([in]nbLignes,nbColonnes:entier;[in]m:Matrice);

{spécifications

état d'entrée:  $0 \leq nbLignes \leq MAX\_LIGNES$ ,  $0 \leq nbColonnes \leq MAX\_COLONNES$

état de sortie: la matrice m de taille nbLignes, nbColonnes est affichée à l'écran}

PROCEDURE multiplierMatrices([in]nbLignesA,nbColonnesA:entier; [in]a:Matrice;

[in]nbColonnesB:entier; [in]b:Matrice;

[out]nbLignesC,nbColonnesC:entier; [out]c:Matrice);

{spécifications

état d'entrée: a, b matrices de tailles (nbLignesA x nbColonnesA) et (nbColonnesA x nbColonnesB) et nbLignesB=nbColonnesA

état de sortie: c contient le résultat et nbLignesC=nbLignesA et nbColonnesC=nbColonnesB

}

ALGO multiplication\_matrices

VAR

```
[in]a, b:Matrice;  
[in]nbLignesA, nbColonnesA, nbLignesB, nbColonnesB:entier;  
[out]c:Matrice;  
[out]nbLignesC,nbColonnesC:entier;
```

DEBUT

```
Afficher "Saisir le nombre de lignes de la matrice A";  
lireEntierBorné(1,MAX_LIGNES, nbLignesA);  
Afficher "Saisir le nombre de colonnes de la matrice A";  
lireEntierBorné(1,MAX_COLONNES, nbColonnesA);  
Afficher "Saisir le nombre de lignes de la matrice B";  
lireEntierBorné(1,MAX_LIGNES, nbLignesB);  
Afficher "Saisir le nombre de colonnes de la matrice B";  
lireEntierBorné(1,MAX_COLONNES, nbColonnesB);
```

```
si nbColonnesA=nbLignesB alors
```

```
    multiplierMatrices(nbLignesA, nbColonnesA, a, nbColonnesB, b, nbLignesC,  
nbColonnesC, c);  
    afficherMatrice(nbLignesA, nbColonnesA, a);  
    Afficher "*";  
    afficherMatrice(nbLignesB, nbColonnesB,b);  
    Afficher "=";  
    afficherMatrice(nbLignesC, nbColonnesC,b);
```

```
sinon
```

```
    Afficher "Multiplication impossible!"
```

```
fin si
```

FIN

PROCEDURE lireEntierBorné([in]linf, lsup:entier;[out]valeur:entier) c'est

DEBUT

```
faire
```

```
    Afficher "Saisissez un entier entre ", linf, " et ", lsup;
```

```
    Lire valeur;
```

```
tant que valeur>lsup ou valeur < linf;
```

FIN

PROCEDURE lireMatrice([in]nbLignes,nbColonnes:entier;[out]m:Matrice) c'est

VAR

```
i,j:entier;
```

DEBUT

```
pour i de 1 a nbLignes faire
```

```
    pour j de 1 a nbColonnes faire
```

```
        Afficher "Saisissez la valeur de la case (", i, ", ", j, ") :";
```

```
        lire m[i][j];
```

```
    fin pour
```

```
fin pour
```

FIN

PROCEDURE afficherMatrice([in]nbLignes,nbColonnes:entier;[in]m:Matrice) c'est

VAR

```

    i,j:entier;
DEBUT
    pour i de 1 a nbLignes faire
        pour j de 1 a nbColonnes faire
            Afficher m[i][j];
        fin pour
        Afficher "\n" {saut à la ligne}
    fin pour
FIN

PROCEDURE multiplierMatrices([in]nbLignesA,nbColonnesA:entier; [in]a:Matrice;
                             [in]nbColonnesB:entier; [in]b:Matrice;
                             [out]nbLignesC,nbColonnesC:entier; [out]c:Matrice) c'est

VAR
    i, j, k : entier;
    somme: réel;
DEBUT
    nbLignesC<-nbLignesA; nbColonnesC<-nbColonnesB
    pour i de 1 à nbLignesC faire
        pour j de 1 à nbColonneC faire
            somme <-0;
            pour k de 1 à nbColonnesA faire
                somme<-somme+a[i][k]*b[k][j];
            fin pour
            c[i][j]<-somme;
        fin pour
    fin pour
FIN

```

#### 4/ Trouver la personne la plus âgée (types structurés)

Écrire l'algorithme qui trouve le nom de la personne la plus âgée dans une liste dont la définition est indiquée ci-dessous. L'algorithme doit assurer aussi la saisie (des noms et des dates de naissance).

CONST

NB\_MAX\_PERSONNES:entier=20;

TYPE

Date = structure

année, mois, jour: entier;

fin structure

Personne = structure

nom: chaîne de caractères;

date\_naissance: Date;

fin structure

PROCEDURE lirePersonne([out]pers:Personne);

{spécifications

état d'entrée: rien

état de sortie: les informations sur la personne saisies sont rangées dans la variable personne

}

PROCEDURE afficherPersonne([in]pers:Personne);

{spécifications

état d'entrée: rien

état de sortie: les informations sur la personne sont affichés à l'écran

}

FONCTION comparerAges(date\_naissanceA, date\_naissanceB:Date):entier;

{spécifications

en entrée: les dates de naissances de A et B

renvoie: -1 si la personne A est plus âgée, 0 si A et B sont nés le même jour, 1 si B est plus âgée

}

ALGO trouver tous les doyens

VAR

[in]liste : tableau [0...NB\_MAX\_PERSONNES] de Personne;

[in]nbPers : entier; {le nombre de personnes}

[out]positionsDoyens : tableau [0...NB\_MAX\_PERSONNES] d'entier;

[out]nbDoyens : entier;

DEBUT

lireEntierBorné(1,NB\_MAX\_PERSONNES+1,nbPers); {le +1 est justifié par le fait que le tableau commence à 0 et que donc il y a une place en plus}

pour i de 0 à nbPers-1 faire

lirePersonne(liste[i]);

fin pour

{au début on suppose que le doyen est la première personne}

```

positionsDoyens [0]<-0; nbDoyens<-1;
pour i de 1 à nbPers-1 faire
    selon comparerAges(
        liste[positionsDoyens].date_naissance,
        liste[i].date_naissance
    ):
        cas 0 :      {un autre doyen}
                    positionsDoyens[nbDoyens] <- i;
                    nbDoyens <- nbDoyens + 1;
        cas 1:
                    positionsDoyens[0]<-i; {un nouveau doyen}
                    nbDoyens <- 1;      {effacer/oublier les autres}
    fin selon
fin pour

si nbDoyens = 1 alors
    Afficher "La personnes la plus âgée : ";
sinon
    Afficher "Les personnes les plus âgées : ";
fin si
pour i de 0 à nbDoyens-1 faire
    afficherPersonne(liste[positionsDoyens[i]]);
fin pour

```

FIN

PROCEDURE lirePersonne([out]pers:Personne) c'est

DEBUT

```

    lire pers.nom;
    lire pers.date_naissance.jour; {on aurait pu faire une fonction pour lire la date!}
    lire pers.date_naissance.mois;
    lire pers.date_naissance.année;

```

FIN

PROCEDURE afficherPersonne([in]pers:Personne) c'est

DEBUT

```

    Afficher pers.nom, " né(e) le ", pers.date_naissance.jour, "/", pers.date_naissance.mois, "/",
    pers.date_naissance.année;

```

FIN

FONCTION comparerAges(date\_naissanceA, date\_naissanceB:Date):entier;

DEBUT

```

    si date_naissanceA.année<date_naissanceB.année alors
        renvoyer -1;
    fin si
    si date_naissanceA.année>date_naissanceB.année alors
        renvoyer 1;
    fin si
    {si on arrive ici alors date_naissanceA.année=date_naissanceB.année}
    si date_naissanceA.mois<date_naissanceB.mois alors
        renvoyer -1;
    fin si
    si date_naissanceA.mois>date_naissanceB.mois alors
        renvoyer 1;

```

```

    fin si
    {si on arrive ici alors date_naissanceA.année=date_naissanceB.année et
date_naissanceA.mois=date_naissanceB.mois}
    si date_naissanceA.jour<date_naissanceB.jour alors
        renvoyer -1;
    fin si
    si date_naissanceA.jour>date_naissanceB.jour alors
        renvoyer 1;
    fin si
    {si on arrive ici alors date_naissanceA.année=date_naissanceB.année et
date_naissanceA.mois=date_naissanceB.mois et date_naissanceA.jour=date_naissanceB.jour}
    renvoyer 0;
FIN

```

FONCTION comparerAges\_version2(dnA, dnB:Date):entier;  
VAR

```

    res:entier;
DEBUT
    si dnA.année=dnB.année alors
        si (dnA.mois=dnB.mois) alors
            si (dnA.jour=dnB.jour) alors
                res<-0;
            sinon si dnA.jour<dnB.jour alors
                res <- -1;
            sinon
                res <-1;
            fin si
        sinon si dnA.mois<dnB.mois alors
            res <- -1;
        sinon
            res <-1;
        fin si
    sinon si dnA.année<dnB.année alors
        res <- -1;
    sinon
        res <-1;
    fin si

    fin si
    renvoyer res;
FIN

```

FONCTION comparerAges\_version3(dnA, dnB:Date):entier c'est  
{consiste a transformer les dates JJ/MM/AAAA en réels dont la valeur est AAAAMMJJ et a  
comparer les réels ensuite! On utilise de réel pour pouvoir représenter de grandes valeurs. En  
algorithmique les entiers s'arrête à 65535!}

```

VAR
    date1, date2:réel;
    res:entier;
DEBUT
    date1=dnA.année*10000; {date1= AAAA0000}
    date1=date1+dnA.mois*100; {date1= AAAA0000+MM00=AAAAMM00}

```

```
date1=date1+dnA.jour;      {date1= AAAAMM00+JJ=AAAAMMJJ}
date2=dnB.année*10000+dnB.mois*100+dnB.jour;
si (date2<date1) alors {A le plus âgé}
    res <- -1
sinon si date2=date1 alors {même âge}
    res <-0
sinon {B le plus âgé}
    res <-1;
fin si
renvoyer res;
```

FIN