IF121: Informatique Fondamentale

Deug MIAS et MASS

Corrigé de l'Examen du 19 Janvier 2001

1 Réponses au QCM

1: addition binaire Quel est le résultat de l'opération en base 2 suivante ?

```
  11010 +
  11110 +
   1011 =
---------
1000011
```

2: complément à deux Quelle est la valeur en base 10 du nombre **11000010** écrit en complément à 2 sur 8 bits ? Bonne réponse: **-62**

3: tables de vérité Complétez la table de vérité suivante:

A	B	A ou (non B)
vrai	vrai	vrai
vrai	faux	vrai
faux	faux	vrai
faux	vrai	faux

4: passage de paramètres Dans un programme supposé correct on rencontre la séquence suivante:

```
....
LireDonnee(d);
EcrireDonnee(d);
```

On suppose écrites les procédures `LireDonnee` et `EcrireDonnee` qui font ce que leur nom indique. Quelle doit être l'en-tête de chacune de ces procédures ?
Ces procédures font la même chose que les commandes *lire* et *écrire* du pseudo-langage, donc a meilleure réponse était `void LireDonnee(int& x)` et `void EcrireDonnee(int y)`, parce que on doit pouvoir modifier x pour y mettre le résultat de la lecture, et on n'a pas besoin de modifier y pour l'afficher. Mais on a accepté aussi la réponse
`void LireDonnee(int& x)` et `void EcrireDonnee(int& y)`

5: Invariants Etant donné le programme

```
programme p
variables
i : un entier
debut p
     i <- 0
tant que i < 5 faire
     i ← i + 3
fin tant que
fin p
```

Lesquelles, parmi les propriétés suivantes, sont des invariants de la boucle *tant que* ?

a) i est impair b) i est pair c) $i < 5$

Réponse: aucune. La condition de boucle n'est jamais un invariant, et ce programme ne préserve pas la parité de i (essayez!).

Exercice 1 (Matrices symétriques et anti-symétriques (4 points)) *Une matrice carrée m est dite* symétrique *si pour tous i et j, on a*

$$m[i][j] = m[j][i]$$

Elle est dite anti-symétrique *si pour tous i et j on a*

$$m[i][j] = -m[j][i]$$

Écrivez en pseudo-langage un programme qui:

1. *lit en entrée une matrice carrée de taille n inférieure ou égale à une constante MAX fixée. L'utilisateur devra fournir la taille n de la matrice, et la matrice ligne par ligne.*
2. *vérifie si la matrice est symétrique ou/et anti-symétrique.*

Solution 1 *Il est très simple de vérifier en même temps la symétrie et l'antisymétrie après la lecture de la matrice. Le programme le plus simple était donc le suivant.*

```
constante MAX=100
type ligne = tableau de MAX entiers
type matrice = tableau de MAX lignes
programme symasym
variables
        i,j,n: entiers
        s,a: booléens
        m: matrice
début symasym
```

```
  // on lit la taille de la matrice, en insistant sur la contrainte n<=MAX
  répéter
       écrire ‘‘Donnez la taille de la matrice (inférieure ou égale a’’ MAX ‘‘).’’
       lire n
  jusqu’à (n<=MAX)
  // on lit la matrice
  pour i <- 1 a MAX faire
    pour j <-1 a MAX faire
      lire m[i][j]
    fin pour
  fin pour
  // test de symétrie et antisymétrie
  // on initialise les booléens s et a à vrai
  s <- vrai
  a <- vrai
  pour i <- 1 a MAX faire
    pour j <-1 a MAX faire
      si (m[i][j] =/= m[j][i]) alors
          s <- faux
      fin si
      si (m[i][j] =/= -m[j][i]) alors
          a <- faux
      fin si
    fin pour
  fin pour
  // affichage du résultat
  selon s,a
        s=vrai,a=vrai: écrire ‘‘Matrice symétrique et antisymétrique’’
        s=vrai,a=faux: écrire ‘‘Matrice symétrique mais non antisymétrique’’
        s=faux,a=vrai: écrire ‘‘Matrice non symétrique mais antisymétrique’’
        s=faux,a=faux: écrire ‘‘Matrice ni symétrique ni antisymétrique’’
  fin selon
fin symasym
```

On pouvait aussi proposer des variantes plus ou moins améliorées:

- *on peut remarquer qu'il suffit d'itérer la boucle sur la partie en haut de la diagonale, diagonale comprise, et écrire donc la boucle comme*

  ```
  pour i <- 1 a MAX faire
    pour j <-i a MAX faire
  ```

- *on peut écrire les deux tests séparément, en remplaçant la boucle* pour *par une boucle* tant que *en s' arrêtant dés que le témoin devient faux.*

Par contre, on a rencontré lors de la correction des artifices ou des erreurs plus ou moins graves, dont voici les plus importants.

Erreurs les plus fréquents

Erreurs mineurs

Utilisation d'entiers à la place des booléens *quelques étudiants n'est pas très familier avec les booléens, et a utilisé à la place des témoins booléens des variables entières:*

- *soit comme booléen, en testant alors l'égalité à 0 ou 1 plutôt que à vrai et faux*
- *soit comme compteur, en comptant le nombre d'entrées dans la matrice qui ne violent pas le test, et en le comparant en sortie de boucle avec le nombre d'entrées testées (généralement n^2)*

Quoique moins élégantes, ces solutions ont été noté avec le maximum des points.

Erreurs graves

Test de symétrie pendant la lecture *quelquesun a pensé de pouvoir effectuer le test en même temps que la lecture de la matrice. Il s'agit là d'une erreur grave, parce qu'au moment où on lit $m[1][10]$ on ne connaît pas encore la valeur de $m[10][1]$ qui sera lue beaucoup plus tard et donc le test $m[1][10] = m[10][1]$ est dépourvu de sens.*
Dans ce cas, on a quand même donné un point si la structure de la boucle de lecture était correcte.

Inversion de la condition de test, ou affectation/affichage aléatoires *Dans certains programmes, on initialise le témoin à faux, et on le mets à vrai dés que la condition est respectée. Cela est aussi une faute grave: pour la plupart des solutions présentée sous cette forme, le résultat revient toujours à vrai, vu que la dernière condition testée est $m[n][n] = m[n][n]$, qui est bien évidemment vrai.*
Dans certains programmes, on modifie les variable témoin, et des fois on affiche même un résultat, systématiquement à chaque itération, ce qui donne des résultats aléatoires ou incohérents.

Oublis du cas de la matrice nulle *Dans certains programmes, on oublie qu'une matrice peut être à la fois symétrique* et *antysymétrique (cas de la matrice nulle).*

Erreurs très graves

Écriture du code de test en dehors des boucles *Dans certains programmes, on retrouve des tests sur $m[i][j] = m[j][i]$ en dehors de toute boucle. Ce code est dépourvu de sens.*

Recopie du sujet *Certains ont tout simplement recopié le sujet. Ils sont clairement hors sujet.*

Exercice 2 (Interclassement (6 points)) *Etant données deux séquences triés d'entiers s_1 et s_2, on peut facilement interclasser leurs éléments pour obtenir une nouvelle séquence triée s_3 qui contient les éléments de s_1 et s_2.*
On souhaite pouvoir fusionner deux séquences d'entiers qui sont fournies en entrée par l'utilisateur, et dont la longueur ne dépasse pas une constante donnée MAX. Pour représenter les séquences, on utilisera une structure ayant deux composantes, un tableau de type tab, pour contenir les entiers, et un entier qui donne le nombres d'éléments significatifs dans le tableau (en commençant par l'indice 1):

```
constante MAX=100
type tab  = tableau de MAX entiers
type seq  = structure
               t: tab
               n: entier
            fin structure
```

1. *Ecrivez, en pseudo-langage, une* fonction est_trié *qui admet en paramètres une séquence s de type seq. Cette fonction retournera le booléen vrai si les éléments de la séquence sont triés en ordre croissant, et faux sinon.*

2. *Ecrivez, en pseudo-langage, une* procédure interclasse *qui interclasse les données de deux séquences s_1 et s_2 dans une séquence s_3. Les séquences s_1 et s_2 sont de type seq, et on vous demande de définir le type de la séquence s_3.*

Faites particulièrement attention à définir correctement les entêtes de vos fonctions et procédures.

Solution 2 *Il s'agit ici de prendre deux séquences* triées *et de les interclasser en obtenant encore une séquence triée. Dans le sujet, il subsistait une légère ambiguïté concernant le fait que les séquences étaient déjà triées, donc dans la correction on a donné le maximum même aux personnes que, plutôt qu'un interclassement, ont tout simplement mis les deux séquences ensemble et écrit un tri sur le résultat après.*

Fonction *est_trié* *Il s'agit d'écrire une simple boucle qui teste que les éléments sont en ordre non décroissant, les tout enrobé en une déclaration de fonction. Notons que l'on ne souhaite pas modifier la séquence passée en paramètre, donc on utilisera un passage par valeur.*

```
fonction esttrié (valeur s: une séquence): booléen
variables
```

```
        b: booléen
        i: entier
début esttrié
  b <- vrai
  pour i <-1 à (s.n)-1 faire
    si s.t[i]>s.t[i+1] alors
       b <- faux
    fin si
  fin pour
  retourner b
fin esttrié
```

Possible améliorations: comme pour l'exercice précédent, on peut s'arrêter dés que la variable témoin est mise à faux

```
fonction esttrié (valeur s: une séquence): booléen
variables
        b: booléen
        i: entier
début esttrié
  b <- vrai
  i <- 1
  tant que (i <= (s.n)-1) et (b=vrai) faire
    si s.t[i]>s.t[i+1] alors
       b <- faux
    fin si
    i<-i+1
  fin tant que
  retourner b
fin esttrié
```

Erreurs les plus fréquents

Erreurs mineurs

Utilisation d'entiers à la place des booléens *quelques étudiants n'est pas très familier avec les booléens, et a utilisé à la place des témoins booléens des variables entières:*

- *soit comme booléen, en testant alors l'égalité à 0 ou 1 plutôt que à vrai et faux*
- *soit comme compteur, en comptant le nombre d'entrées dans le tableau qui ne violent pas le test, et en le comparant en sortie de boucle avec le nombre d'entrées testées (généralement n)*

Quoique moins élégantes, ces solutions ont été noté avec le maximum des points.

Erreurs graves

Lecture de s à l'intérieur de la fonction *quelques étudiants ont mis à l'intérieur de la fonction une boucle de lecture de la séquence s, qui est passée en paramètre! Cela rends la fonction totalement inutile.*

Inversion de la condition de test, ou affectation/affichage aléatoires *Dans certains programmes, on initialise le témoin à faux, et on le mets à vrai dés que la condition est respectée. Cela est aussi une faute grave: pour la plupart des solutions présentée sous cette forme, le résultat est aléatoire et revient à la valeur du test* $s.t[s.n-1] > s.t[s.n]$.
Dans certains programmes, on modifie les variable témoin, et des fois on affiche même un résultat, systématiquement à chaque itération, ce qui donne des résultats aléatoires ou incohérents.

Type de s_3 *On doit maintenant déclarer le type de la séquence résultat* s_3*, qui doit pouvoir contenir les deux séquences* s_1 *et* s_2*. Comme ces deux séquences ont longueur maximale MAX,* s_3 *devra avoir comme longueur maximale* $2 * MAX$*, et sont type sera alors le suivant*

```
type tab2  = tableau de 2*MAX entiers
type seq2  = structure
                t: tab2
                n: entier
             fin structure
```

Erreurs les plus fréquents

Erreurs graves

s_3 **déclaré de type** *seq quelques étudiants ont déclaré* s_3 *de type seq, ce qui est catastrophique pour le programme, vu que l'on aura systématiquement erreur à l'exécution dés que la somme des tailles de* s_1 *et de* s_2 *dépasse MAX.*

Erreurs très graves

déclarations absurdes *Dans plusieurs programmes, on a trouvé des déclaration de type absurdes comme:*

```
type seq2  = structure
                t: tab
                n=2*MAX
             fin structure
```

On vous rappelle qu'il s'agit d'une déclaration *de type, donc on peut seulement dire que n est un entier, pas donner sa valeur (qui d'ailleurs n'a pas de sens, vu que cette valeur doit être déterminée à l'exécution).*

Procédure *interclasse* *Pour interclasser deux séquences triées, on va utiliser 3 indices:*

- *i indique le prochain élément de s1 à examiner*
- *j indique le prochain élément de s2 à examiner*
- *k indique le prochain élément de s3 à remplir*

et on va choisir toujours l'élément le plus petit entre s1.t[i] et s2.t[j] pour le copier dans s3.t[k], en incrémentant ensuite i, j et k en fonction de ce test. Pour que la comparaison soit significative, il faut bien sûr s'assurer que $i <= s1.n$ et $j <= s2.n$, donc si i dépasse $s1.n$, c'est que l'on a fini d'examiner $s1$ et on doit copier ce qui reste de $s2$ dans $s3$. De même, si j dépasse $s2.n$, c'est que l'on a fini d'examiner $s2$ et on doit copier ce qui reste de $s1$ dans $s3$.

```
procédure interclasse (valeur s1,s2: seq, référence s3: seq2)
variables
        i,j,k: entiers
début interclasse
  i<-1
  j<-1
  k<-1
  // on choisit toujours le plus petit élément entre s1 et s2
  // tant que la comparaison est significative ...
  tant que ((i<=s1.n) et (j<=s2.n)) faire
       si s1.t[i]<s2.t[j] alors
          s3.t[k]<-s1.t[i]
          i<-i+1
       sinon
          s3.t[k]<-s2.t[j]
          j<-j+1
       fin si
       k<-k+1
  fin tant que
  // une fois cela fait, il faut s'assurer de traiter ce qui reste de s1 ...
  tant que (i<=s1.n)
       s3.t[k]<-s1.t[i]
       i<-i+1
       k<-k+1
  fin tant que
  // ou ce qui reste de s2 ...
  tant que (j<=s2.n)
       s3.t[k]<-s2.t[j]
       j<-j+1
       k<-k+1
  fin tant que
```

```
  // enfin, on mets dans s3.n la bonne longueur
  s3.n <- s1.n+s2.n
fin interclasse
```

Erreurs les plus fréquents

Erreurs mineurs *Certains ont écrit des programmes divers et variés qui font intervenir des tris ou des insertions dans l'ordre plutôt qu'un vrai interclassément. Quoique peu élégantes, ces solutions ont eu le maximum des points.*

Erreurs graves

oublis du traitement du reste de s1 et s2 *c'est un erreur grave, vu que la séquence s3 ne va pas être remplie complètement*

éléments de s1 et s2 pris à 2 à 2 *certains ont écrit des programmes qui comparent correctement les prochaines éléments x et y de s1 et s2, mais ensuite insèrent dans s3 x* et *y dans l'ordre et non pas seulement le plus petit. Bien sûr, cela est incorrect, comme vous pouvez constater en essayant sur les séquences 1 1 1 1 1 et 2 3 4 5.*

Erreurs très graves

boucles pour imbriqués *on a trouvé des solutions proposant 3 boucles pour imbriquées sur k, i et j, ce qui produit des résultats absurdes*

Exercice 3 (Invariants (4 points)) *Voici un programme écrit en pseudo-langage.*

```
constante MAX=100
type tab = tableau de MAX entiers
programme tabmult
variables
     t       : tab
     i, m : entiers
début tabmult
     lire t
     m <- t[1]
     i <- 2
     tant que (i <= MAX) faire
          si t[i] = 0 alors
             m <- 0
             i <- MAX+1
          sinon
             m <- m * t[i]
             i <- i+1
          fin si
     fin tant que
```

```
        écrire m
    fin tabmult
```

Montrez à l'aide d'un invariant que le programme affiche le produit des éléments du tableau t.

Solution 3 *L'invariant le plus approprié pour cette démonstration est*

$$I \quad = \quad (m = \prod_{k=1}^{k=i-1} t[k]) \text{ et } i \leq MAX + 1$$

Démontrons l'invariant

à l'entrée de la boucle

I est vrai parceque $i = 2 \leq MAX + 1 = 101$ et

$$m = t[1] = \prod_{k=1}^{k=1} t[k] = \prod_{k=1}^{k=i-1} t[k]$$

à chaque itération *supposons $i \leq MAX$ et I vrai au début de l'itération. Nous devons prouver I vrai après l'itération. On doit traiter deux cas, selon que $t[i] = 0$ ou non.*

Cas $t[i] = 0$

dans ce cas, le produit des éléments de t est forcément 0, et le code exécuté est

```
m <- 0
i <- MAX+1
```

donc à la fin de cette itération on doit vérifier I pour les nouvelles valeurs i' et m' de i et m, c'est à dire

$$\begin{aligned} I \quad &= \quad (m' = \prod_{k=1}^{k=i'-1} t[k]) \text{ et } i' \leq MAX + 1 \\ &= \quad (0 = \prod_{k=1}^{k=MAX+1-1} t[k]) \text{ et } MAX + 1 \leq MAX + 1 \\ &= \quad (0 = \prod_{k=1}^{k=MAX} t[k]) \text{ et } MAX + 1 \leq MAX + 1 \end{aligned}$$

qui est encore vrai parce que le produit vaut bien 0.

Cas $t[i] \neq 0$

dans ce cas, le code exécuté est

```
m <- m*t[i]
i <- i+1
```

or, on sait que avant l'exécution de ce code l'invariant I est vrai, donc on a

$$(m = \prod_{k=1}^{k=i-1} t[k]) \text{ et } i \leq MAX + 1$$

il nous suffit de vérifier qu'après l'exécution de ce code I reste vrai avec les nouvelles valeurs i' et m' de i et m; or

$$\begin{aligned} m' &= m * t[i] \\ &= (\prod_{k=1}^{k=i-1} t[k]) * t[i] \\ &= \prod_{k=1}^{k=i} t[k] \\ &= \prod_{k=1}^{k=i'} t[k] \end{aligned}$$

et comme $i \leq MAX$, on a bien $i' = i + 1 \leq MAX + 1$

Utilisons l'invariant pour prouver le résultat *À la sortie de la boucle, on sait que l'invariant est vrai et que la condition de boucle est fausse:*

$$m = \prod_{k=1}^{k=i-1} t[k] \text{ et } i \leq MAX + 1 \text{ et } i \not\leq MAX$$

donc, forcément $i = MAX + 1$ et par conséquence

$$m = \prod_{k=1}^{k=i-1} t[k] = \prod_{k=1}^{k=MAX+1-1} t[k] = \prod_{k=1}^{k=MAX} t[k]$$

et le programme affiche bien le produit des éléments de t.

Erreurs *On a trouvé essentiellement deux types d'erreurs:*

oublis du cas $t[i] = 0$ *certaines discussion du problème ne traitent pas le cas* $t[i] = 0$

hors sujet *un grand nombre de personnes ont paraphrasé en français le fonctionnement du programme, sans identifier aucun invariant et sans donner de preuves faisant intervenir l'invariant. Ils sont hors sujet, indépendamment de la longueur de la paraphrase.*

Exercice 4 (Programmation C-- (2 points)) *Traduisez en* C-- *le programme écrit en pseudo-langage de l'exercice 3.*

Solution 4

```
#include <iostream.h>
const int MAX=100;
typedef int tab[MAX];
void main()
     tab t;
     int i, m;
     for(i=0;i<MAX;i++)cin>>t[i];;
     m=t[0];
     i=1;
     while (i <MAX)
          if (t[i] == 0)m=0;i=MAX;
          else m=m*t[i];i=i+1;

     cout<<m;
```