Master Sciences, Technologies, Santé
Mention Mathématiques, spécialité Enseignement des mathématiques

Algorithmique et graphes, thèmes du second degré

Contrôle Continu n° 1 (durée 1h00)
Quelques éléments de correction…

Exercice 1. Qu'est-ce qu'un algorithme ? Quelles sont les qualités que doit posséder un langage de description d'algorithmes ?

Un algorithme décrit un enchaînement d'opérations permettant, en un temps fini, de résoudre toutes les instances d'un problème donné. Partant d'une instance du problème (données d'entrée), il fournit un résultat correspondant à la solution du problème sur cette instance. Un algorithme doit être universel, c'est-à-dire indépendant du langage de programmation qui sera utilisé pour l'implémenter.

L'expression d'un algorithme nécessite un langage clair (compréhensible), structuré (pour décrire des enchaînements d'opérations), non ambigu (pas de double interprétation possible).

Exercice 2. Qu'affiche l'algorithme suivant lorsque l'utilisateur entre les valeurs 21 et 6 pour a et b (justifiez votre réponse) ? Que calcule cet algorithme ?

```
Algorithme mystèreBoucle
# c'est à vous de trouver ce que fait cet algorithme…
variables   a, b, c : entiers naturels
début
       # lecture des données
  Entrer ( a, b )
       # initialisation et calculs
  c ← 0
  tantque ( a ≠ 0 )
  faire     si ( ( a mod 2 ) ≠ 0 )
            alors     c ← c + b
            fin_si
            a ← a div 2
            b ← b * 2
  fin_tantque
       # affichage résultat
  Afficher ( c )
fin
```

*On vérifie aisément que cet algorithme calcule a * b (en se basant sur la décomposition de l'entier a en binaire) et donc, sur cet exemple, 21 * 6 = 126. Pour justifier cette réponse, il suffit de produire un tableau montrant l'évolution des variables au cours de l'algorithme.*

Exercice 3. Écrire un algorithme permettant de calculer l'intersection de deux intervalles d'entiers [a, b] et [c, d] donnés (par exemple, l'intersection des intervalles [1,6] et [3,11] est l'intervalle [3,6], l'intersection des intervalles [1,6] et [9,11] est vide, l'intersection des intervalles [1,6] et [6,11] est l'intervalle [6,6]).

Un algorithme possible est le suivant (on suppose que les intervalles sont donnés correctement, i.e. a <= b et c <= d).

```
Algorithme intersectionIntervalles
# cet algorithme détermine calcule l'intersection de deux intervalles
# de la forme [a,b] et [c,d]
variables   a, b, c, d, gauche, droit : entiers naturels
début
       # lecture des données
  Entrer ( a, b, c, d )
       # calcul de gauche et droit, bornes a priori
       # de l'intervalle résultat
  si (a < c)
  alors  gauche ← c
  sinon  gauche ← a
  fin_si
  si (b < d)
  alors droit ← b
  sinon droit ← d
  fin_si
       # affichage du résultat
       # si gauche > droit, l'intersection est vide
  si (gauche > droit)
  alors Afficher ("intersection vide")
  sinon Afficher ("[", gauche, ",", droit, "]")
  fin_si
fin
```

Exercice 4. Écrire un algorithme permettant d'afficher les diviseurs d'un entier naturel lu au clavier par ordre décroissant (rappelons que « a div b » et « a mod b » permettent respectivement de récupérer le quotient et le reste de la division euclidienne de a par b).

```
Algorithme diviseursDecroissant
# cet algorithme affiche par ordre décroissant les diviseurs
# d'un entier n donné
variables   n, diviseur : entiers naturels
début
       # lecture des données
  Entrer ( n )
       # cas particulier
  si (n = 0)
  alors Afficher ("tous les entiers non nuls divisent 0…")
  sinon
            # on affiche n
       Afficher ( n )
            # boucle de parcours et affichage des autres diviseurs
       pour diviseur de n div 2 à 1 par pas de -1
       faire
            si (n mod diviseur = 0)
            alors  Afficher( diviseur )
            fin_si
       fin_pour
  fin_si
fin
```

Exercice 5. Écrire un algorithme permettant de compter le nombre d'occurrences (d'apparitions) d'un élément donné dans une liste d'entiers (on s'autorisera à écrire « Entrer (L) » pour récupérer le contenu d'une liste en une seule opération ; la fonction « NombreEléments (L) » permet de récupérer le nombre d'éléments de la liste L ; rappelons que le premier élément d'une liste a pour rang 0).

```
Algorithme nombreOccurrencesDansListe
# cet algorithme permet de déterminer le nombre d'occurrences de
```

```
# élément dans liste
variables   liste : liste d'entiers naturels
            i, élément, nbOccurrences : entiers naturels
début
        # initialisations
   Entrer ( liste )
   Entrer ( élément )
   nbEléments ← NombreEléments ( liste )
   nbOccurrences ← 0
        # parcours de la liste et comptage
   pour i de 0 à nbEléments – 1
   faire    si ( élément = liste [ i ] )
            alors    nbOccurrences ← nbOccurrences + 1
            fin_si
   fin_pour
        # retour résultat
   Retourner ( nbOccurrences )
fin
```

Exercice 6. Écrire un algorithme permettant de compter le nombre d'occurrences (d'apparitions) d'un élément donné dans une liste d'entiers triée.

Cette fois, on n'utilise plus la boucle Pour : on peut arrêter le parcours dès qu'un élément plus grand est trouvé…

```
Algorithme nombreOccurrencesDansListeTriée
# cet algorithme permet de déterminer le nombre d'occurrences de
# élément dans liste (liste triée par ordre croissant)
variables   liste : liste d'entiers naturels
            plusGrand : booléen
            i, élément, nbOccurrences : entiers naturels
début
        # initialisations
   Entrer ( liste )
   Entrer ( élément )
   nbEléments ← NombreEléments ( liste )
   nbOccurrences ← 0
        # parcours de la liste et comptage, avec arrêt si plusGrand = vrai
   i ← 0
   plusGrand ← faux
   tantque (i < nbEléments) et (plusGrand = faux)
   faire    si ( élément < liste [ i ] )
            alors    plusGrand ← vrai
            sinon    si ( élément = liste [ i ] )
                     alors lnbOccurrences ← nbOccurrences + 1
                     fin_si
                     i ← i + 1
            fin_si
   fin_pour
        # retour résultat
   Retourner ( nbOccurrences )
fin
```