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

Algorithmique et graphes, thèmes du second degré

Contrôle Continu n° 1 (durée 1h00)

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. Réécrivez (et éventuellement rectifiez) l'algorithme suivant de façon à ce qu'il puisse être présenté à des élèves d'une classe de seconde (justifiez vos modifications).

```
Algorithme ploufplouf

variables x, truc, machin, aaa : entiers naturels
début
    Entrer ( x, truc )
    si ( x < truc )
    alors  aaa ← x
           x ← truc
           truc ← aaa
    fin_si
    machin ← x mod truc
    tantque ( machin ≠ 0 )
           x ← truc
           truc ← machin
           machin ← x mod truc
    fin_tantque
    Afficher ( machin )
fin
```

Cet algorithme présente plusieurs défauts : nom de l'algorithme, absence de commentaires judicieux, noms de variables peu parlants. Par ailleurs, il affiche toujours la valeur 0 ! Il doit donc être rectifié :

```
Algorithme calculPGCD

# cet algorithme détermine le PGCD de deux entiers positifs non nuls
# x et y en utilisant l'algorithme d'Euclide.

variables x, y, reste, echange : entiers naturels

début
    # lecture des données
    Entrer ( x, y )

    # on veut x >= y. Dans le cas contraire, on les échange.
    si ( x < y )
    alors  echange ← x
           x ← y
```

```
        y ← echange
    fin_si

    # boucle de calcul (algorithme d'Euclide)
    reste ← x mod y
    tantque ( reste ≠ 0 )
        x ← y
        y ← reste
        reste ← x mod y
    fin_tantque

    # affichage du résultat : y (dernier reste non nul)
    Afficher ( y )
fin
```

Exercice 3. Ecrire un algorithme permettant de savoir si un point est inclus dans un rectangle parallèle aux axes et spécifié par les coordonnées de son point en bas à gauche et son point en haut à droite (par exemple (2, 3) et (5, 8)).

```
Algorithme PointInclus

# Cet algorithme permet de savoir si le point de coordonnées (a, b) est
# inclus dans le rectangle parallèle aux axes de point bas-gauche (x1, y1) et
# haut-droit (x2, y2)
début
    Entrer(x1, y1, x2, y2, a, b)
    si a ≥ x1 et a ≤ y1 et b ≥ y1 et b ≤ y2 alors
        Afficher(« Le point est inclus dans le rectangle»)
    sinon
        Afficher(« le point est en dehors du rectangle »)
    fin_si
fin
```

Exercice 4. Ecrire une <u>action</u> inversant l'ordre des éléments dans un tableau. Par exemple, si la valeur initiale du tableau t est [2, 8, 5, 3], sa valeur finale sera [3, 5, 8, 2]. L'algorithme utilisé ne devra pas utiliser de deuxième tableau.

```
Action Inverser(L : liste)
# Inverse le contenu de la liste L
variables i, nbElements, echange : entier
début
    nbElements ← NombreElements(L)
    pour i de 0 à nbElements div 2 – 1 faire
        echange ← L[i]
        L[i] ← L[nbElements – i – 1]
        L[nbElements – i – 1] ← echange
    fin_pour
fin
```

Exercice 5. Ecrire un algorithme calculant la longueur de la plus longue suite croissante dans un tableau. Par exemple, pour t = [1, 4, 7, 5, 3, 1, 6], la réponse est 3 (suite 1, 4, 7).

```
Algorithme TailleMaxSuiteCroissante(L : liste d'entiers)
# affiche la taille de la plus longue sous-liste de L croissante
variables i, max, courant, nbElements : entiers
début
    max ← 0 # vaut 0 si la liste est vide.
    courant ← 1
    nbElements ← NombreElements(L)
```

```
    pour i de 0 à nbElements – 2 faire
        si L[i] < L[i+1] alors
            courant ← courant + 1
        sinon
            si courant > max alors
                max ← courant
            fin_si
            courant ← 1 # L[i+1] commence une nouvelle suite
        fin_si
    fin_pour
    Afficher(max)
fin
```

Exercice 6. Ecrire un algorithme calculant la longueur de la plus longue suite monotone (croissante ou décroissante) dans un tableau. Par exemple, pour L = [1, 4, 7, 5, 3, 1, 6], la réponse est 4 (suite 7, 5, 3, 1). (Exercice hors barème, et difficile !)

```
Algorithme TailleMaxSuiteMonotone(E L : liste d'entiers, S max : entier)
# affiche la taille de la plus longue sous-liste de L croissante ou décroissante
variables i, courant, égaux, nbElements : entiers
        croissant : booléen
début
    nbElements ← NombreElements(L)
    if nbElements < 2 alors
        retourner nbElements
    max ← 0
    i ← 0
    courant ← 1
    tant que i < n-1 et L[i] = L[i+1] faire
        courant ← courant + 1
        i ← i+1
    fin_tant_que
    si i < n-1 alors
        croissant ← L[i] < L[i+1]
        courant ← courant + 1
        égaux ← 1
        i ← i+1
    sinon
        retourner nbElements # suite constante
    fin_si
    tant que i < nbElements - 1 faire
        si (croissant et L[i] ≤ L[i+1]) ou ( ! croissant et  L[i] ≥ L[i+1]) alors
            courant ← courant + 1
            si L[i] = L[i+1] alors
                egaux ← egaux + 1
            sinon
                égaux ← 1
        sinon
            si courant > max alors
                max ← courant
            fin_si
            courant ← égaux + 1
            croissant ← ! croissant
        fin_si
        i ← i+1
    fin_tant_que
```

```
    si courant > max alors
            max ← courant
    fin_si
    retourner max
fin
```