Master Sciences, Technologies, Santé

Mention Mathématiques, spécialité Enseignement des mathématiques

Algorithmique et graphes, thèmes du second degré

Feuille TD n°1 – Exercices d'algorithmique

… éléments de correction …

Exercice 1. Résolution d'une équation du 1er degré

Écrire un algorithme permettant de résoudre une équation à coefficients réels de la forme `ax + b = 0` (a et b seront entrés au clavier).

Réponse. *L'algorithme est le suivant :*

```
Algorithme equationPremierDegré
# cet algorithme résout une équation de la forme ax + b = 0
# a et b sont entrés au clavier.
variables   a, b : réels
début
       # lecture données
  Entrer ( a, b )
       # résolution de l'équation
            # cas où a = 0
  si ( a = 0 )
  alors     si ( b = 0 )
            alors Afficher ( "Tous les réels sont solution" )
            sinon Afficher ( "Pas de solution" )
            # cas où a ≠ 0
  sinon     Afficher ( -b / a )
  fin_si
fin
```

Exercice 2. Minimum de trois nombres

Écrire un algorithme permettant d'afficher le plus petit de trois nombres entrés au clavier.

Réponse. *Un algorithme possible est le suivant : si a est plus petit que b, il suffit de comparer a à c… sinon, il faut comparer b à c :*

```
Algorithme minimumTroisNombres
# cet algorithme permet d'afficher le plus petit de trois nombres
# entrés au clavier.
variables   a, b, c : entiers naturels
début
       # lecture données
  Entrer ( a, b, c )
       # comparaisons
  si ( a < b )
  alors     si ( a < c )
            alors Afficher ( a )
            sinon Afficher ( c )
            fin_si
  sinon     si ( b < c )
```

```
            alors Afficher ( b )
            sinon Afficher ( c )
            fin_si
   fin_si
fin
```

Exercice 3. Durée d'un vol d'avion

Écrire un algorithme permettant de calculer la durée d'un vol d'avion, connaissant l'horaire de départ (heures et minutes) et l'horaire d'arrivée (heures et minutes), sans convertir les horaires en minutes. On suppose que le vol dure moins de 24 heures.

Réponse. *On effectue le calcul sans tenir compte des valeurs… et on ajuste le cas échéant. L'algorithme est alors le suivant :*

```
Algorithme duréeVolAvionSansConversion
# cet algorithme permet de calculer la durée d'un vol d'avion
# sans convertir les horaires de départ et d'arrivée en minutes.
variables   h_depart, m_depart, h_arrivee, m_arrivee : entiers naturels
            h_duree, m_duree : entiers naturels
début
        # lecture données
   Entrer ( h_depart, m_depart, h_arrivee, m_arrivee )
        # calcul de la durée
   h_duree ← h_arrivee – h_depart
   m_duree ← m_arrivee – m_depart
        # on rectifie les minutes si nécessaire
   si ( m_durée < 0 )
   alors    m_duree ← 60 + m_duree
            h_duree = h_duree - 1
   fin_si
        # on rectifie les heures si nécessaire
   si ( h_duree < 0 )
   alors    h_duree = 24 + h_duree
   fin_si
        # affichage résultat
   Afficher ( h_duree, m_duree )
fin
```

Exercice 4. Lecture d'algorithme

Que fait l'algorithme suivant ?

```
Algorithme mystèreBoucle2
# 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 ← 1
   tantque ( b ≠ 0 )
   faire    si ( ( b mod 2 ) = 1 )
            alors    c ← c * a
            fin_si
            a ← a * a
            b ← b div 2
   fin_tantque
        # affichage résultat
   Afficher ( c )
```

```
fin
```

Réponse. *Cet algorithme calcule la valeur de* `a` *élevé à la puissance* `b` *(exponentiation rapide). Cet algorithme utilise le même principe que le précédent : cette fois, c'est la décomposition binaire de* `b` *qui est utilisée. La variable* `a`*, elle, prend successivement les valeurs* a, a^2, a^4, a^8, *etc. Lorsqu'un tour de boucle correspond à un 1 dans la décomposition binaire de* `b`*, la variable* `c` *« cumule par produit » la puissance correspondante de* `a`*.*

Ainsi, pour $b = 21 = 2^4 + 2^2 + 2^0$*, la variable* `c` *vaudra finalement* $a * a^4 * a^{16} = a^{1+4+16} = a^{21} = a^b$*. Cet algorithme calcule donc la valeur de* `a` *à la puissance* `b`*, ce que l'on peut aisément vérifier en le faisant tourner sur un (petit) exemple…*

Exercice 5. Afficher les diviseurs d'un entier

Écrire un algorithme permettant d'afficher les diviseurs d'un entier naturel par ordre croissant.

Réponse. *La boucle* `Pour` *vient encore à notre rescousse ici (notons qu'il est inutile cependant de parcourir l'intervalle* `[n/2 + 1, n]`*). Attention, si* `n = 0`*, tous les entiers divisent* `n` *!...*

L'algorithme est le suivant :

```
Algorithme diviseursOrdreCroissant

# cet algorithme permet d'afficher les diviseurs d'un entier naturel
# par ordre croissant

variables   n, diviseur : entiers naturels

début

       # lecture des données
   Entrer ( n )

       # cas où n est nul
   si ( n = 0 )
   alors Afficher ( "Tous les entiers non nuls sont diviseurs de 0" )

       # cas général
   sinon
            # boucle de parcours, si diviseur divise n, on l'affiche
       pour diviseur de 1 à n div 2
       faire    si ( n mod diviseur = 0 )
                alors    Afficher ( diviseur )
                fin_si
       fin_pour
       Afficher ( n )
   fin_si

fin
```

Exercice 6. Nombre premier

Écrire un algorithme permettant de déterminer si un entier naturel entré au clavier est premier.

Réponse. *Il suffit de chercher un diviseur de* `n` *dans l'intervalle* `[2,RacineCarrée(n)]`*. Dès qu'un tel diviseur est trouvé, le nombre* `n` *n'est pas premier. On utilisera donc la structure* `tantque` *qui permet d'arrêter la recherche dès qu'un diviseur est découvert.*

L'algorithme est le suivant :

```
Algorithme nombrePremier

# cet algorithme permet de déterminer si un entier naturel entré au
clavier # est premier

variables   n, diviseur : entiers naturels
            rac : réel

début

       # lecture des données
   Entrer ( n )

       # initialisations
```

```
  rac ← RacineCarrée ( n )            # fonction racine carrée
  diviseur ← 2
      # boucle de recherche d'un diviseur
  tantque ( ( n mod diviseur ≠ 0 ) et ( diviseur ≤ rac ) )
  faire
          diviseur ← diviseur + 1
  fin_tantque
      # affichage résultat
  si ( diviseur > rac )
  alors    Afficher ( "Le nombre est premier" )
  sinon    Afficher ( "Le nombre n'est pas premier" )
  fin_si
fin
```

Exercice 7. Nombres premiers jumeaux inférieurs à 1000

Deux nombres premiers sont jumeaux si leur différence vaut 2 (par exemple, 5 et 7 sont deux nombres premiers jumeaux). Écrire un algorithme permettant d'afficher tous les couples de nombres premiers jumeaux inférieurs à 1000.

Réponse. *Il suffit de modifier l'algorithme précédent, en mémorisant le dernier nombre premier trouvé. On obtient alors :*

```
Algorithme nombresPremiersJumeaux
# cet algorithme permet d'afficher la liste de tous les nombres premiers
# inférieurs à 100
variables   i, n, nprec, diviseur : entiers naturels
            rac : réel
début
      # initialisation
  nprec ← 2
      # boucle principale
  pour i de 3 à 999
  faire
          # initialisations
      rac ← RacineCarrée ( n )            # fonction racine carrée
      diviseur ← 2
          # boucle de recherche d'un diviseur
      tantque ( ( n mod diviseur ≠ 0 ) et ( diviseur ≤ rac ) )
      faire
              diviseur ← diviseur + 1
      fin_tantque
          # si n premier
      si ( diviseur ≤ ≤rac )
      alors        # jumeau avec nprec ?
              si ( n – nprec = 2 )
              alors    Afficher ( nprec, n )
              fin_si
                  # on met à jour nprec
              nprec ← n
      fin_si
  fin_pour
fin
```

Exercice 8. Calcul du n^ième^ nombre de Fibonnacci

Écrire un algorithme permettant de calculer le nombre de Fibonacci `F(n)` : `F(0) = 0`, `F(1) = 1`, et `F(n) = F(n-1) + F(n-2)`.

Réponse. *On mémorise les valeurs de `F(n)` et `F(n-1)`, que l'on maintient à jour au fur et à mesure de l'avancée du calcul…*

L'algorithme est le suivant :

```
Algorithme nombreFibonacci
# cet algorithme permet de calculer le nombre de Fibonacci F(n)
variables   n, i, fibo, moins1 : entiers naturels
début
        # lecture des données
  Entrer ( n )
        # initialisations
  fibo ← 1
  moins1 ← 0
        # test si cas simple ( n = 0 )
  si ( n = 0 )
  alors    Afficher ( 0 )
  sinon         # boucle de calcul pour le cas général
           pour i de 2 à n
           faire    fibo ← fibo + moins1
                    moins1 ← fibo - moins1
           fin_pour
                # affichage résultat
           Afficher ( fibo )
  fin_si
fin
```

Remarquons ici les opérations du corps de boucle `pour` *: nous nous sommes permis d'utiliser un artifice évitant l'utilisation d'une variable supplémentaire. Si nous avons effectué par exemple 3 tours de boucles, nous avons* `moins1 = 2` *et* `fibo = 3` *(on a calculé* `F(4)`*).*

Au 4ème tour, nous faisons `fibo ← 3 + 2 = 5`*, puis* `moins1 ← 5 - 2 = 3`*, ce qui est bien correct.*