Algorithmique et programmation procédurale - TD 1

Boucles - Corrigé

Exercice 1. Écrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 10 jusqu'à ce que la réponse convienne.

Corrigé

```
Programme BoucleNombre
Variables N : Entier
Début
      Repete
            Ecrire "Entrez un nombre entre 1 et 10"
            Lire N
      Jusquaceque (1 <= N) et (N<=10)
Fin
```

Exercice 2. Écrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

```
Table de 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70
```

Corrigé

```
Programme TableMultiplication
Variables N, i : Entier
Début
      Ecrire "Entrez un nombre : "
      Lire N
      Ecrire "La table de multiplication de ce nombre est : "
      Pour i ← 1 à 10
            Ecrire N, " x ", i, " = ", (N * i)
      FinPour
Fin
```

Exercice 3. Écrire un algorithme qui calcule la somme des carrées des n premiers entiers impairs avec la valeur de n saisie au clavier au début de programme.

Corrigé

```
Programme SommeCarresImpairs
Variables N, i, S : Entier
Début
      Ecrire "Entrez un nombre : "
      Lire N
      S ← 0
      Pour i ← 1 à N
            S ← S + (2*i-1)*(2*i-1)
      FinPour
      Ecrire "La somme des carrées des premiers entiers impairs est ", S
Fin
```

Exercice 4. Écrire un algorithme qui demande un nombre de départ, et qui ensuite calcule la factorielle de ce nombre.

Corrigé

```
Programme Factorielle
Variables N, i, F : Entier
Début
      Ecrire "Entrez un nombre : "
      Lire N
      F ← 1
      Pour i ← 1 à N
            F ← F*i
      FinPour
      Ecrire "La factorielle de ", N " est", F
Fin
```

Exercice 5. On considère le programme suivant :

```
Lire epsilon                                            (1)
n ← 1                                                   (2)
resultat ← 0                                            (3)
Tantque (1/(n*n) > epsilon)                             (4)
      resultat ← resultat + 1/(n*n)                     (5)
      n ← n+1                                           (6)
FinTantque
resultat ← 6 * resultat                                 (7)
Ecrire resultat                                         (8)
```

1. Que fait ce programme (a priori `epsilon` est un nombre réel très petit)?
2. Quelle est sa complexité en nombre d'affectations, de multiplications et de divisions ?

Corrigé

1. résultat = $6*(1 + 1/2^2 + 1/3^2 +\ldots + 1/n_0^2)$
   où : $1/(n_0+1)^2 <$ epsilon $< 1/n_0^2$ … => ce qui se rapproche de π^2
2.

Ligne	Multiplication	Division
1	0	0
2	0	0
3	0	0
4	*1 (condition de boucle)*	*1 (condition de boucle)*
5	*1*	*1*
6	*0*	*0*
7	1	0
8	0	0

Boucle (4) : cette boucle est exécutée [1/sqrt(epsilon)] (la partie entière) fois

Au total :

Nombre d'affectations: 2+2*[1/sqrt(epsilon)]+ 1
Nombre de multiplications : [1/sqrt(epsilon)] + ([1/sqrt(epsilon)]+ 1) + 1
Nombre de divisions : [1/sqrt(epsilon)] + ([1/sqrt(epsilon)]+ 1)

Exercice 6
On considère la fonction suivante écrite en pseudo-code:

```
Fonction ex(n : Entier) : Entier
Variables a, b : Entier
Début
        a ← 2
        b ← 1
        Tantque a < n
                a ← a*2
                b ← b + 1
        FinTantque
        retourner b
Fin
```

1. Que vaut **ex(n)** si **n** est une puissance de 2 ? Que vaut **ex(n)** pour un argument **n** quelconque ?

2. Quelle est la complexité de cette fonction en nombre de comparaisons, de multiplications et d'affectations en fonction de **n**.

Corrigé

1. Si $n = 2^b$, $ex(n) = b = \log_2 n$. Pour un argument n quelconque, $ex(n) = b = [\log_2 n] + 1$ ([ ] : partie entière). (**1 point**)
2. La complexité de cette fonction en nombre de comparaisons, de multiplications et d'affectations en fonction de n.

Ligne	Comparaison	Multiplication	Affectation
1	0	0	1
2	0	0	1
3 Gestion de compteur Corps de boucle	 b 0*(b-1) fois	 0 1*(b-1) fois	 0 (1+1)*(b-1) fois
4	0	1	1
5	0	0	1
6	0	0	0

Au total :

   a. nombre de comparaisons : b
   b. nombre de multiplications : b-1
   c. nombre d'affectations : 2+2(b-1) = 2b

où $b = \log_2 n$ si n est une puissance de 2 et $b = [\log_2 n] + 1$ pour les autres cas.