INF1005C **Programmation procédurale**

Travail dirigé No. 1
Rédaction d'algorithme

Objectifs : Apprendre à rédiger correctement un algorithme
Durée : 1 semaine
Remise du travail : Avant 23h30 le 31 janvier 2021.
Travail préparatoire : Leçons 1 à 4 sur Moodle, et lecture des exercices.
Documents à remettre : Les algorithmes complétés.

Pour chaque exercice :

- **a) Décrivez l'algorithme de manière générale, en français,** sans tenir compte des contraintes du langage simple décrit en **b)**. Cette description est le plan que vous suivrez pour écrire la version raffinée en **b)**, elle doit donc être écrite **avant**. Dans cette description, identifiez clairement :
  - ce que l'ordinateur doit afficher à l'usager,
  - ce que l'ordinateur doit lire de l'usager,
  - où sont les conditions,
  - où sont les répétitions (qui n'ont pas à être sous forme « TANT QUE »),
  - ce qui sera dans une fonction (pour les questions que ça concerne).

- **b) Puis** écrivez une version raffinée de l'algorithme exprimée uniquement à l'aide des opérations élémentaires suivantes :

- LIRE
- AFFICHER
- = (affecter)
- TANT QUE *condition* FAIRE …
- SI *condition* ALORS … SINON …

- Opérateurs arithmétiques :
- + (additionner)
- − (soustraire)
- * (multiplier)
- / (diviser)
- % (reste ou modulo)

- Comparaisons : <, >, ≤, ≥, =, ≠
- Opérateurs booléens : et, ou, pas/non
- Fonctions mathématiques : sinus, cosinus, valeur absolue, racine carrée
- FONCTION *nom* (*paramètres*)
- RÉSULTAT *expression*

Les conditions et répétitions doivent être correctement indentées : les instructions dont l'exécution est contrôlée par une condition (SI/SINON) ou répétition (TANT QUE) sont en retrait vers la droite par rapport à cette condition/répétition. Les différents éléments d'une suite ou d'un texte sont référés avec les crochets, ainsi, *valeurs[n]* est l'élément à la position *n* de la suite ou texte. **Les index commencent à zéro, *valeurs[0]* est donc le premier élément/caractère**. « longueur de » permet de savoir combien de valeurs/caractères se trouvent dans une suite/texte. Pour initialiser tous les éléments à une valeur identique, on peut faire par exemple « variable = tableau de 0 » (variable sera un tableau où tous les éléments sont initialisés à zéro). Aussi, les lettres d'un texte sont modifiables, donc mot[0] = « a » permet de remplacer la première lettre d'un mot par la lettre « a ».

Exemples d'utilisation de suite/chaine :

Un mot est entré par l'usager, puis chaque lettre du mot est affichée avec espaces entre :

a) Demander (**affichage**) et **lire** le mot. **Pour chaque** lettre du mot, **afficher** la lettre suivie d'un espace.

b)
```
Afficher « Entrer un mot : »
Lire mot
position = 0
TANT QUE position < longueur de mot FAIRE
        Afficher mot[position] « »
        position = position + 1
```

Lire *N* valeurs dans une suite :

a) Pour (**répétition**) les positions de 0 à *N* exclu, **lire** la valeur et la placer à cette position dans la suite.

b) position = 0
TANT QUE position < *N* FAIRE
    Lire valeur
    suite[position] = valeur
    position = position + 1

Exemple de problème : Écrire un algorithme qui vérifie si un nombre entré par l'usager est premier ou non.

Une solution possible :

a) Demander le nombre à l'usager (**affichage**). **Lire** le nombre *n* de l'usager.
Pour chaque entier entre 2 et la racine carrée de *n*, vérifier (une **condition**) est-ce que cet entier divise *n*.
Si aucun des entiers testés ne divise *n*, **afficher** que le nombre est premier, sinon **afficher** qu'il ne l'est pas.

b) Afficher « Entrer le nombre à vérifier : »
Lire *n*
i = 2
a trouvé un diviseur = Faux
TANT QUE *i* * *i* $\leq$ *n* FAIRE
    SI *n* % *i* == 0 ALORS
        a trouvé un diviseur = Vrai
    i = *i* + 1
SI a trouvé un diviseur ALORS
    Afficher « Le nombre n'est pas premier »
SINON
    Afficher « Le nombre est premier »

Note : *i* * *i* $\leq$ n est équivalente à $i \leq \sqrt{n}$ si *i* et *n* sont positifs, et n'a pas besoin de l'opération racine carrée.

Exemple de problème : Écrire une fonction qui vérifie si un nombre passé en paramètre est premier ou non.

Une solution possible :

a) Fonction avec paramètre *n*.
Pour chaque entier entre 2 et la racine carrée de *n*, vérifier (une **condition**) est-ce que cet entier divise *n*, le **résultat** est Faux si c'est le cas. Dans le cas où aucun diviseur n'est trouvé le **résultat** et Vrai.

b) FONCTION est premier (*n*) :
i = 2
TANT QUE *i* * *i* $\leq$ *n* FAIRE
    SI *n* % *i* == 0 ALORS
        RÉSULTAT Faux
    i = *i* + 1
RÉSULTAT Vrai

Exemple d'utilisation de cette fonction :

Lire *x*
SI est premier (*x*) ALORS
    Afficher « Oui »

1 – Orthogonal : Écrire un algorithme qui détermine si deux vecteurs à deux dimensions sont orthogonaux ou non. Note : utiliser un produit scalaire ; les opérations sur les vecteurs, dont le produit scalaire, ne sont pas des opérations élémentaires disponibles pour l'algorithme raffiné.

Exemple : L'utilisateur entre les composantes des vecteurs (1 ; 0,5) et (-1 ; 2)
L'affichage attendu est : `Les vecteurs sont orthogonaux.`

2 – Pythagore. L'algorithme devra demander et lire 3 nombres. Ensuite, il vérifiera si ces 3 nombres satisfont le théorème de Pythagore, i.e. que le 3e nombre est bien l'hypoténuse pour les 2 autres nombres. Si ce n'est pas le cas, il redemandera et relira 3 nombres jusqu'à ce que le théorème soit respecté.

3 – Suite : Demander à l'utilisateur de saisir un entier ***n*** strictement positif (et lui redemander jusqu'à ce qu'il ait bien entré un nombre strictement positif, mais vous n'avez pas à vérifier que c'est un entier) puis une suite de *n* entiers. Afficher la longueur de la plus grande suite croissante.

Exemple : Pour *n*=10 et la suite 10,**1,3,4,5,9**,5,8,2,1 le programme affiche 5.

4 – Recherche : Écrire une fonction dont le résultat est la position où se trouve le texte « INF » dans une phrase qui lui est passée ; elle doit avoir le résultat -1 dans le cas où il n'y est pas. La position résultante doit être celle où se trouve la première lettre du « INF » dans la phrase, la position zéro étant la première lettre de la phrase.
Note : chaque caractère compte comme une position, incluant les espaces et les ponctuations. La fonction « longueur de (phrase) » permet de connaître le nombre de caractères dans la phrase (voir exemple en p.1).

Exemple : La phrase passée à la fonction est « `J'ai un cours d'INF1005C` », le résultat attendu est 16.

5 – Distance à l'origine en 3D. Demander et lire plusieurs points de coordonnées (*x*, *y* , *z*). À chaque nouveau point entré, indiquer s'il est plus près de l'origine (0, 0, 0) que tous les points précédemment entrés. Définissez une fonction pour obtenir la distance du point à l'origine. Votre programme doit calculer une seule fois la distance de chaque point entré par l'usager.

6 – Racine carrée : Écrire un algorithme pour calculer la racine carrée d'un nombre réel positif *x*. La méthode sera d'utiliser la série définie comme :

$y_0 = x$
$y_{n+1} = (y_n + x / y_n) / 2$

Lorsque *n* tend vers l'infini, cette série converge vers la racine carrée de *x*. L'estimation de l'erreur au terme y_n, par rapport à la véritable racine carrée, sera $e_n = | y_n - y_{n-1} |$ (soit la valeur absolue de la différence entre deux termes qui se suivent dans la série). L'algorithme doit arrêter, et afficher la valeur de y_n, dès que cette erreur estimée est inférieure à *epsilon*.
Note : la valeur absolue n'est pas une opération élémentaire disponible pour l'algorithme raffiné.

Exemple : L'utilisateur entre les valeurs de *x* et *epsilon* comme étant 2 et 0,01.
L'affichage attendu est : `La racine de 2 est approximativement 1,414215686`.

Dans cet exemple, les valeurs des *y* sont : $y_0 = 2$; $y_1 = 1{,}5$; $y_2 = 1{,}41666666$; $y_3 = 1{,}414215686$. La différence entre y_3 et y_2 est de ~0,002, qui est inférieur au *epsilon* de 0,01, d'où l'affichage de la valeur de y_3 comme approximation acceptée de la racine de 2.