algo-exo-constructions-d-algorithmes-de-procedure

- Objet : Comment élaborer l'algorithme une procédure ?
- Niveau requis :
  débutant, avisé
- Commentaires : *Contexte d'utilisation du sujet du tuto.*
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.

Exo 1 : procédure "compter le nombre de caractères dans une chaîne"

Soit une chaîne de caractères terminée par un caractère '.'.
Donnez l'algorithme d'un programme qui compte la longueur de cette chaîne (“.” non compris).

1) Définir la procédure (la moulinette)

chaine de caractères —→| calcul des caractères de la chaîne |—> nombre de lettres de la phrase

2) Définition des données

```
CONSTANTE
point = '.' // caractère d'arrêt
nbremax = 100 //  nombre max de caractères
TYPE
chaîne tableau[taille] de caractères

VARIABLES
taille_phrase : entier // taille réelle de la chaîne de caractère entrée''
i : entier // indice de parcours de la phrase
nombre_lettre // Nombre d'occurrence de la lettre cherchée
```

3) Jeu d'essai

chainedecaractère	nombredelettre
“.”	0
“a.”	1
“ab.”	2

(C'est la notice d'utilisation de la procédure pour le programme qui utilisera cette procédure.)

L'interface de procédure (procédure décrite en pseudo-langage)

```
procédure compterchainedecaractère (entrée chainedecaractère : chaine
                                     sortie  nombredecaractère : entier)

//compterchainedecaractère : compte le nombre de caractère de la chaîne
//chainedecaractère : suite de plusieurs caractères éventuellement espace
d'espace et terminée par un point
//nombredecaractère : nombre de caractère qui ont été" rentrés
```

> chainedecaractère : nom du paramètre
> chaine : c'est le type tableau créé pour le paramètre “chainedecaractère ”
>
> nombredecaractère : c'est le nom de la variable de sortie qui aura pour valeur le résultat (elle est de type prédéfini “entier”.)

5) programme de test

```
CONSTANTES
  point = '.' // caractère d'arrêt

  nbremax = 100 //  nombre max de caractères

TYPES
  chaine = tableau [n] caractere

VARIABLES
  chainedecaractere : chaine
  nombredelettre : entier

PROCEDURES
procédure compterchainedecaractère (entrée chainedecaractère : chaine ;
                                     sortie  nombredecaractère :
entier)

    //compterchainedecaractère : compte le nombre de caractère de la chaîne
    //chainedecaractère : suite de plusieurs caractères éventuellement
espace d'espace et terminée par un point
    //nombredecaractère : nombre de caractère qui ont été" rentrés

  début

    écrire ('entrez une phrase')
    lire (chainedecaractère)

    compterchainedecaractère (chainedecaractère, nombredelettre)

    écrire ('le nombre de caractère est', nombredelettre)

  fin
```

6) Algorithme de la procédure compterchainedecaractère

```
procédure compterchainedecaractère (ENTREE chainedecaractère : chaine ;
                                    SORTIE nombredecaractère : entier)

     // entrée et sortie sont des mots réservés

//compterchainedecaractère : compte le nombre de caractère de la chaîne
//chainedecaractère : suite de plusieurs caractères éventuellement espace
d'espace et terminée par un point
    //nombredecaractère : nombre de caractère qui ont été" rentrés

i : entier //indice

    début

     i := 1

       tantque chainedecaractere [i] <> point faire

            i := i + 1

        fintantque
        nombredecaractère := i-1
    fin
```



Ne pas faire “parler la procédure” : elle ne doit rien faire de plus que ce qu'elle permet fait dans tout programme qui l'utilisera.

Exo 2 : procédure "compter occurrences de deux caractères successifs"

Rechercher si deux lettres choisies par l'utilisateur sont dans une phrase entrée par l'utilisateur.

1) Schéma de la procédure (la moulinette)

```
                    schéma procédure
                   |                |
 phrase ---------->|cherche si deux |------> si caractère1 ET caractère2
sont dans la phrase et successifs
                   |                |        alors terminateur
                   |lettres succ    |        deuxlettres = VRAIE
```

```
caractère 1 ------->|  dans          |
caractère 2 ------> |  cette phrase  |
```

2) Définition des données

```
CONSTANTES\\
terminateur = '.' // caractère d'arrêt\\

TYPES\\
chainetableau tableau[MAX] caractère\\

VARIABLES\\
phrase : caractère\\
car1   : caractère\\
car2   : caractère\\
nbr    : entier\\
```

3) Jeu d'essai

phrase	terminateur	caractère1	caractère2	nombre
blablobli.	'.'	“l”	“r”	0
bonjour.	'.'	“b”	“.”	0
blablo.	'.'	“b”	“l”	2
blablobli.	'.'	“b”	“l”	3

4) Interface (définir la procédure)

vérifier si une phrase contient deux lettres

PROCEDURE

```
procédure DeuxLettresID ( entrée phrase : chainetableau ;\\
                          sortie nbreoccurrence : entier )\\

    // CompterDeuxLettres           : procédure qui recherche deux lettres
successives et définies extérieurement dans une phrase terminée par un
point\\
    // phrase                       : phrase saisie par l'utilisateur\\
    //nbreoccurrence                : décompte du nombre d'occurrence de mot
contenant deux lettres successives\\
```

chainetableau : c'est le type créé

5) Programme de test

```
CONSTANTES
```

```
max = 100

TYPES
chaine tableau[MAX] caractère

VARIABLES
phrase         : chaine
lettre 1       : caractère
lettre 2       : caractère
nbreoccurrence : entier


    début

        écrire ("entrez une phrase terminée par un point")

        lire (phrase)

        écrire ("entrez une première lettre")

        lire (lettre 1)

        écrire ("entrez une deuxième lettre")

        lire (lettre 2)

      DeuxLettresID ( entrée phrase : chainetableau , sortie nbreoccurrence
: entier )

       écrire ("le nombre d'occurrences des lettres choisies dans la phrase
donnée est", nbreoccurrence")

    fin
```

6) Procédure compter deux lettres

```
CompterDeuxLettres( lettre phrase chaine
                    entrée
                    car1   : caractère
                    car2   : caractère
                    FIN    : caractère
                    sortie occurrence : entier

VARIABLE

    i : entier
    occurrence : 0

 début
```

```
    i := 1
    occurrence := 0

    tantque (phrase [i] <> FIN)

        si (phrase[i] = car1)

            si (phrase [i + 1] = car2

             occurrence + 1

            finsi

        finsi

      i := i + 1

    fintantque

 fin
```

Exo 3 : procédure "rechercher un palindrome"

Procédure pour déterminer si une chaîne de caractère terminée par un point est un palindrome.

1) Moulinette (schéma de la procédure)

chaîne. —> | recherche palindrome | —> palindr : booléen (Vrai ou FAUX)

```
i (indice ->)
  1   2   3   4   5 | 6 |... | MAX
-----------------------------------
| L | A | V | A | L | . |
        j (indice <-|.| )
```

2) Jeu d'essai

'.'	c'est un palindrome
'a.'	c'est un palindrome
'aa.'	c'est un palindrome
'aba.'	c'est un palindrome
'acb'	ce n'est pas un palindrome
'aacba.'	ce n'est pas un palindrome
'aacab	ce n'est pas un palindrome

3) Définition des données du problème

```
CONSTANTES
taille =80 //nombre max de caractères de la chaîne
carterm ='.' // caractère terminaleur de la chaîne

TYPE
chaine =tableau[taille] de caractères // type crée pour contenir une liste

VARIABLES
phrase : chaine // phrase où une symétrie est recherchée
i      : entier //
j      : entier // indice de parcours de la phrase depuis la fin.
```

4)Interface (notice) de la procédure

```
procédure cherchePalinPoint(entrée         texte : chaine,
                            entrée sortie  ind1  : entier, ind2 : entier)
          // Cette procédure permet de dire si une chaine de caractère
terminée par un point est un palindrome.
          // ind1 : c'est l'indice de parcours du texte par le début
          // ind2 : c'est l'indice de parcours du texte par la fin
```

5)Programme de test

```
CONSTANTES
constante MAX =80 //nombre max de caractères de la chaîne (Pour pouvoir
créer le type chaine, car un tableau est toujours de taille fixe.
constante FIN ='.' // caractère terminaleur de la chaîne (Un tableau peut ne
pas être tout rempli)

TYPES
Type chaine =tableau[MAX] de caractères // type crée pour contenir une liste

VARIABLES
variable possPalin : chaine // phrase où une symétrie est recherchée
variable indiceDeb : entier //
variable indiceFin : entier // indice de parcours de la phrase depuis la
fin.

PROCEDURES
procédure cherchePalinPoint(entrée         texte : chaine,
                            entrée sortie  ind1  : entier, ind2 : entier
                            sortie result : booléen)
          // Cette procédure permet de dire si une chaine de caractère
terminée par un point est un palindrome.
          // ind1 : c'est l'indice de parcours du texte par le début
          // ind2 : c'est l'indice de parcours du texte par la fin
```

```
Début
     // Saisie de la phrase :
     ECRIRE ('Entrez votre phrase et n'oubliez pas de la terminer par un
point.'
     LIRE(phrase)
     indiceDeb := possPalin[1]
     indiceFin := possPalin[MAX-1]
     result    := booléen

    // Recherche d'un palindrome fini par un point :
    cherchePalinPoint(entrée possPalin:chaine , entrée/sortie ind1:entier,
entrée/sortie ind2:entier, sortie result:bool)
    SI result = VRAI
        ECRIRE('La phrase', possPalin 'est un palindrome.')
    SINON
        ECRIRE('La phrase', possPalin 'n'est pas un palindrome.')
    FINSI
Fin
```

6)Algorithme de la procédure Palindrome terminé par un point

```
procédure cherchePalinPoint(entrée         texte : chaine,
                            entrée sortie  ind1  : entier, ind2 : entier
                            sortie result: booléen)
          // Cette procédure permet de dire si une chaine de caractère
terminée par un point est un palindrome.
          // ind1 : c'est l'indice de parcours du texte par le début
          // ind2 : c'est l'indice de parcours du texte par la fin

CONSTANTE
MAX : 80

TYPE chaine = tableau[MAX] de caractères

VARIABLES

i    : entier
j    : entier
palindrome : booléen

TERM : '.' : caractère


 début
  i := chaine[1]
  j := chaine[TERM - 1]


    SI chaine[i] < chaine[j] ALORS
```

```
    palindrome := VRAI
      TQ (i < j) ET (chaine[i] = chaine[j])
     i := i + 1
     j := j - 1
      FinTQ
    SINON
         palindrome := FAUX
    FINSI

 fin
```

Trier un tableau par remontée des bulles

1) Définition du problème

```
                                  |    |
                liste chiffre---> |    |----> liste de chiffre
                 (désordonnée)    |    |       (ordonnée)
```

2) Jeu d'essai

```
Entrée                        sortie
-----------------------------------
rien                          rien
5                              0
5293741                        123479
```

3) Définition des données du programme TriBulle

```
CONSTANTES
constante MAX = 80

TYPES
table =Tableau[MAX] de entiers

VARIABLES
variable chiffresDesordo : table    //Table d'entiers désordonnés
variable chiffresOrdonnés: table    // Table d'entiers ordonnés
variable nbrchiffre      : entier   // Nombres de chiffres
variable taille          : entiers  // Taille de la table de chiffres
désordonnés en entrée
                                    // et ordonnée en sortie
```

Programme de test de la procédure TriBulle

```
CONSTANTES
constante MAX = 80

TYPES
table =Tableau[MAX] de entiers

VARIABLES
variable chiffresDesordo : table    //Table d'entiers désordonnés
variable chiffresOrdonnés: table    // Tables d'entiers ordonnés
variable nbrchiffre      : entier   // Nombres de chiffres
variable taille          : entiers  // Taille de la table de chiffres
désordonnés en entrée
                                    // et ordonnée en sortie
PROCEDURES
procédure remplir ( entrée        taille_tableau_donné : entier
                    entrée/sortie  ind                 : entier
                    entrée/sortie  chiffre_donnés      : entier
                    sortie         liste_désordonnée   : table
    // La procédure remplir permet de récupérer les données saisies par
l'utilisateur.
    // chiffres_donnés : compteur de chiffres entrés
    // taille_tableau_donné : c'est  la taille choisie par l'utilisateur de
la table.
    // chiffre_donnés       : c'est pour arrêter la saisie d'un nouvel
élément
    //                           quand il y a autant que chiffres_donnés.
    // ind : c'est l'indice de parcours de la table.
    // liste_désordonnée   : c'est la table de chiffre toute remplie qui a
été saisie.

procédure procédure tribulle( entrée         taille          : entier
                    entrée/sortie  chiffresDesordo : table
                    entrée/sortie  chiffreOrdonnés : table)
    // La procédure du tri des bulles permet de ranger une chaîne de
chiffres
    //      désordonnés en ordre croissant.
    // chiffresDesordo : C'est la liste de chiffres désordonnés rentrée par
    //                     l'utilisateur ainsi que la liste de chiffres
triés en sortie.
    // taille : C'est le nombre de chiffres rentrés par l'utilisateur.
    // invert :  C'est vrai quand il y a eu inversion.

procédure Procédure affichage_tableau_ordonné ( entrée table_Rangée : table
                                                entrée taille_table :
entier)
    // La procédure permet l'affichage du tableau trié
    // table_Rangée est la table de chiffres ordonnés.
```

```
    // table est le type crée (un tableau)
    // taille_table définie  la taille de la table.


Début
  saisie(chiffresDesordo, nbrchiffre, taille) // appel de la procédure
saisie
  triBulle(taille, chiffresDesordo)           // appelle de la procédure
triBulle
  affichage(taille, chiffreOrdonnés)          // appelle de la procédure
affichage

Fin
```

5) Interface de la procédure TrieBulle

```
Procédure tribulle ( entrée         taille          : entier
                     entrée/sortie  chiffresDesordo : table
                     entrée/sortie  chiffreOrdonnés : table)
// La procédure du tri des bulles permet de ranger une chaîne de chiffres
//désordonnés en ordre croissant.
// chiffresDesordo : C'est la liste de chiffres désordonnés rentrée par
   l'utilisateur ainsi que la liste de chiffres triés en sortie.
// taille : C'est le nombre de chiffres rentrés par l'utilisateur.
```

6) Algorithme de la procédure TrieBulle

```
procédure tribulle( entrée         taille          : entier
                    entrée/sortie  chiffresDesordo : table
                    entrée/sortie  chiffreOrdonnés : table)
// La procédure du tri des bulles permet de ranger une chaîne de chiffres
//désordonnés en ordre croissant.
// chiffresDesordo : C'est la liste de chiffres désordonnés rentrée par
   l'utilisateur ainsi que la liste de chiffres triés en sortie.
// taille : C'est le nombre de chiffres rentrés par l'utilisateur.

VARIABLES
variable i              : entier   // indice de parcours du tableau.
variable cible          : entier  // variable permettant l'inversion de 2
entiers dans le tableau.
variable comptinversion : entier // Comptage des inversions
variable invert         : booléen // Permet de déterminé s'il y a ou non
inversion

 Début
    // parcours jusqu'à qu'il n'y est plus aucune inversion.
    i := 1
    invert := FAUX
```

```
    Répéter
      Tantque ( i < taille ) faire
     SI table[i] > table[ i + 1 ] ALORS // si l'entier de la case n°1 du
tableau supérieur à
                                        // celui de la case n°2 du même
tableau alors
        cible = table[1]             //
        table[1] = table[i + 1]      //  inversion
        table[i+1] = cible           //
        invert := VRAI

     Finsi
      i := i + 1                          // incrémentation pour passer
aux entiers suivants.
      Fintantque
    Jusquà NON invert

 Fin
```

6 bis) Interfaces et programmes des procédures utilisées par le programme principal

I] Procédure "remplir et afficher tableau désordonné"

```
//exo: tri par remontée des bulles
CONSTANTES
constante MAX = 80  // Taille maximal de la table de chiffres

Procédure remplir ( entrée        taille_tableau_donné : entier
                    entrée/sortie  ind                 : entier
                    entrée/sortie  chiffre_donnés      : entier
                    sortie         liste_désordonnée   : table
// La procédure remplir permet de récupérer les données saisies par
l'utilisateur.
// chiffres_donnés : compteur de chiffres entrés
// taille_tableau_donné : c'est  la taille choisie par l'utilisateur de la
table.
// chiffre_donnés       : c'est pour arrêter la saisie d'un nouvel élément
                          quand il y a autant que chiffres_donnés.
// ind : c'est l'indice de parcours de la table.
// liste_désordonnée   : c'est la table de chiffre toute remplie qui a été
saisie.


Type table = tableau[MAX] d'entiers

Variables
taille_finale    : entier    // nombre de tous les chiffres rentrés par
```

```
l'utilisateur.
chiffre_donnés   : entier  // les chiffres rentrés par l'utilisateur pour
mettre
                                //quelque chose dans le tableau
                                //(l'utilisateur ne rentre pas forcément 80
chiffres)
ind               : entier // indice de parcours de table
liste_désordonnée : table

Début

    // l'utilisateur rentre la taille de la table d'entier et ses valeurs.

  Répéter
     Ecrire ( 'Quel sera le nombre de nombres à trier?' ) // taille du
tableau à trier
                                                                    //( sup
à 0 ,
                                                                    //  non
égal à 1,
                                                                    //
stric inf à 80)
     Lire ( taille_finale )
  Jusquà ( taille_finale >= 0 )  et ( taille_finale < = MAX )
  Ecrire ('Saisissez le chiffre n°)', ind )
    // remplir le tableau

  chiffre_donnés := 0 // initialisation l'utilisateur n'a pas encore donné
ses chiffres à trier
  ind =1              // initialisation de l'indice de parcours

  Tantque  (chiffre_donnés <= taille_finale) Faire
     Ecrire ('Veuillez donner un entier')       // on demande ensuite les
entiers à l'utilisateur
                                                  // pour chaque case du
tableau.
     Lire (liste_désordonnée[ind])
     chiffre_donnés := chiffre_donnés + 1      // Incrémentation pour mettre
les autres
                                                  //entiers dans le tableau.
         ind := ind + 1
  Fintantque
  ecrire('Voici la liste des chiffres que vous avez
entré',(liste_désordonnée[i])

Fin
```

II] procédure "afficher tableau ordonné"

```
Procédure affichage_tableau_ordonné ( entrée table_Rangée : table
```

```
                                        entrée taille_table : entier)

// La procédure permet l'affichage du tableau trié
// table_Rangée est la table de chiffres ordonnés.
// table est le type crée (un tableau)
// taille_table définie  la taille de la table.

Type listeChiffre

Variable taille_table : entier  // compte le nombre de chiffres rentré par
l'utilisateur.

  Début
    Ecrire ( "le tableau rangé est ",table_Rangée[taille_table])
  Fin
```

Recherche dichotomique

VOIR les tris :http://axiomcafe.fr/tri-dans-un-tableau

```
// recherche dichotomique

GENERALITE

méthode de recherche par dichotomie : exemple

i  |    a    |    b    |    c    |    d    |    e    | ...| j

i = début
j = fin
milieu = tab(i[1] + i[n]) / 2
    si cible > milieu alors
       cible est à chercher sur tab[milieu][j]
    si cible < milieu  alors
       cible est à chercher sur tab[i][j]

//////////////////////////

1) Analyse du problème :
   deux programmes : 1) crée la table de prénom
                     2) chercher un prenom dans la table




       entrée            Faire_table         sortie
prénom cherché
```

```
------------------------------------------------------------------------------
----------------------------
                        |  enregistre       |                              |
les prénoms             |  et numérote      |-> une table                  |
résultat : prénom dans la table
                        |  les prénoms      |   de prenoms numérotés       |
ou
                        |   entrés          |    (max 10)                  |
pas dans la table
------------------------------------------------------------------------------
-----------------------------

////////////////////////////////////////////////////////////////////
PROGRAMME crée la table de prénom et enregister le prenom recherché
////////////////////////////////////////////////////////////////////
On sait que la taille de la table de prénom est égale au nombre de prénoms
entrés

CONSTANTES

nombre_de_prenom = 10 // taille max de la table
taille_prenom = 15 // taille max d'un prenom

TYPE

prenom =tableau[nombre_de_prenom] de caractère
table =tableau[nombre_de_prenom] de prenom

VARIABLES

i                : entier // compteur de prénoms entrée
nombre_de_prenom : entier // Nombre de prenoms entrés par l'utilisateur
table_prenom     : table de prenom

prenom           : prenom  // prénom cherché
resultat         : entier


début

    i := 0 // Pas encore de prénom entrés

    écrire("combien de prenom ?")
    lire(nombre_de_prenom)

    TANTQUE ( i < nombre_de_prenom ) FAIRE

            i := i + 1
            écrire(Entrez un prénom, svp.)
            lire( table_prenom[i] )
    FINTANTQUE
```

```
   // enregistrer le prénom à chercher

   ecrire(Pour quel nom voulez-vous l'indice ?)
   lire(prenom)

   //appel procédure cherche_prenom
   chercher_prenom(table , nombre_de_prenom, prenom , resultat)

   SI resultat = 0 ALORS
      écrire ("Le prénom n'est pas dans la liste")
   SINON
      écrire ("Le prénom est dans la table au n° ", resultat)
fin

////////////////////////////////////////////////////////////////////
Programme chercher le prénom
////////////////////////////////////////////////////////////////////


VARIABLES

indice_début : entier // Début de table de prénom
indice_fin   : entier // fin de la table de prénom
indice_moyen : entier // milieu de la table de prénom


TYPE

prenom =tableau[nombre_de_prenom] de caractère
table =tableau[nombre_de_prenom] de prenom


procédure chercher_prenom(entrée tableEnregistrée :table
                          entrée taille :entier
                          entrée prenom_cherché :prenom
                          sortie indice : entier)
                     // taille c'est place du prénom par rapport à la taille
totale de la table de prénom
                     // cible c'est le n° du prenom dans la table de prénom
 début
     indice_début := 1
     indice_fin   := taille
     cible        := (indice_début + indice_fin) DIV 2


      TANTQUE (indice_début < indice_fin) ET (table [cible] <>
prenom_cherché) FAIRE

           SI tableEnregistrée[cible] > prenom_cherché ALORS
               indice_fin = cible - 1 //du milieu à la fin de la table de
```

```
prénom
           SINON
                indice_début := cible + 1 //du début jusqu'au milieu de la
table de prénom

           FINSI

           cible := (indice_début + indice_fin) DIV 2
       FINTANTQUE


       SI (indice_début > indice_fin ) OU ( table [cible] <> prenom_cherché)
ALORS

          cible := 0
       FINSI

 fin
```

Rechercher un mot dans une phrase

1) schéma du problème

```
SCHEMA :
phrase ------------------------------>       |        |
                                              |       |
mot à chercher----------------------->       |         | ---> mot dans phrase
oui ou non
                                              |       |
Longueur du mot à chercher---------->       |        |
```

Jeu d'essai

phrase	Mot_cherché	longueur_Mot_Cherché	Trouvé
.	coucou	6	VRAI
.	' '	0	VRAI
coucou le chat	le	2	VRAI
coucou le chat	couii	5	FAUX
le	chat	4	FAUX

PROCEDURE TROUVER UN MOT

1) Algorithme de principe

```
DEBUT
  Avoir une phrase
  Avoir un mot à chercher
  Avoir la longueur de ce mot cherché
    REPETER
        PRENDRE un mot dans la phrase
        SI  mot de la phrase = mot cherché ALORS
             comparer les deux mots
        FinSi
    JUSQU'A avoir trouvé le mot cherché OU avoir parcouru toute la phrase
FIN
```

2)PROGRAMME DE TEST

```
CONSTANTES

TERM = '.'
ESPACE = ' '

MaxPhrase = 80
MaxMot    = 15


TYPES

phrase_tableau =Tableau [MaxPhrase] de caractères
mot_tableau =Tableau [MaxMot] de caractères

VARIABLES

phraseSaisie : phrase_tableau
motSaisi      : mot_tableau
lgMotCherché : entier
MotTrouvé : booléen


PROCEDURES

TrouverMot(entrée      phrase : phrase_tableau
                     entrée      mot     : mot_tableau
                     entrée     lgMotCherché : entier
             sortie         Trouvé           : booléen

Début

    ECRIRE ('Donnez une phrase de moins de', MaxPhrase, 'caractères et un
```

```
mot
        de',  MaxMot, 'n'oubliez pas le point.')
    LIRE(phraseSaisie)
    ECRIRE('Donnez un mot à cherché de moins de', MaxMot)
    LIRE(motSaisi )
    ECRIRE('Donner la longueur du mot à cherché.')
    LIRE(lgMotCherché)

    SI MotTrouvé :=  TrouverMot(phraseSaisie, MotSaisi, lgMotCherché,
trouvé)
                        ALORS
               ECRIRE('Le mot, motSaisi 'est dans dans la phrase.')
               SINON
                ECRIRE('Le mot, motSaisi 'n'est pas dans dans la phrase.')
            FINSI
Fin
```

3)PROCEDURES TROUVER UN MOT

```
rouverMot( entrée     phrase       : phrase_tableau
                    entrée      mot           : mot_tableau
            entrée      longueur   : entier
            sortie       trouvé       : booléen )

VARIABLES :
position : entier // Indice de parcours de la phrase
longueur : entier  // Longueur du mot analysé dans la phrase
trouvé1 : booléen  // Vrai si la mot est trouvé

PROCEDURES
prendreMot( Entrée             phrase             : phrase_tableau
                     Entrée             mot             : mot_tableau
             Entrée/Sortie   indice             : entier
                     Sortie              longueurMot  :   entier
             Sortie              trouvé1           : booléen  )
      // Cette procédure regarde s'il y a un mot suivant à partir de la
position de l'indice i placé
      // après    le dernier mot analysé dans la phrase où l'on cherche un
mot.
     // Phrase : c'est la phrase dans laquelle on cherche un mot
    // mot : c'est le mot qu'on cherche à trouver dans la phrase
   // indice : c'est l'indice de parcours de la phrase à partir duquel on
commence la recherche
  // longueurMot : c'est la longueur du mot trouvé est que l'indice qui le
traverse va permettre d'en
  //                             la taille.
//// Le programme est page suivante.

comparerMot( Entrée             phrase             : phrase_tableau
                        Entrée           position          : entier
```

```
                Entrée            longueurMot   : entier
                Entrée            mot    : mot_tableau
                Sortie             trouvé2           : booléen  )
    // Cette procédure vérifie que le mot trouvé dans la phrase est de
taille égale au mot cherché.
           // Phrase : c'est la phrase dans laquelle on cherche un mot.
         //  longueurMot : c'est la longueur unique pour les deux mots.
       // mot : c'est le mot cherché
     // trouvé2 : c'est le résultat de la comparaison entre les deux mots.
//// Le programme est page suivante.

Début
position := 0
longueur:= 0
trouvéA : =  prendreMot(phrase, position, longueur)
TantQ   trouvéA:= VRAI
    SI      longueur = longueurMot         ALORS
        trouvéB:= comparerMot( phrase, position,  longueurMot, mot)
    FINSI
FIN
```

PROCEDURES prendreMot

```
prendreMot( Entrée            phrase            : phrase_tableau
                    Entrée            mot             : mot_tableau
            Entrée/Sortie   indice            : entier
                    Sortie             longueurMot  :   entier
            Sortie             trouvé1           : booléen  )
      // Cette procédure regarde s'il y a un mot suivant à partir de la
position de l'indice i placé
      // après    le dernier mot analysé dans la phrase où l'on cherche un
mot.
     // Phrase : c'est la phrase dans laquelle on cherche un mot
    // mot : c'est le mot qu'on cherche à trouver dans la phrase
   // indice : c'est l'indice de parcours de la phrase à partir duquel on
commence la recherche
  // longueurMot : c'est la longueur du mot trouvé est que l'indice qui le
traverse va permettre d'en
  //                           la taille.

PROGRAMME  prendreMot

Début
    TQ texte[indice] = espace FAIRE
        indice := indice + 1
    FinTQ
    lg := 0
    Si (texte[indice] <> espace) ET (texte[indice] <> TERM) ALORS
        indice := indice + 1
```

```
        lg     := lg + 1
        trouvé2:= VRAI
    Sinon
         trouvé2:= FAUX
    FinTQ
Fin
```

comparerMot

```
comparerMot( Entrée             phrase             : phrase_tableau
                          Entrée            position          : entier
                 Entrée            longueurMot   : entier
                 Entrée            mot    : mot_tableau
                 Sortie             trouvé2           : booléen  )
    // Cette procédure vérifie que le mot trouvé dans la phrase est de
taille égale au mot cherché.
           // Phrase : c'est la phrase dans laquelle on cherche un mot.
         //  longueurMot : c'est la longueur unique pour les deux mots.
       // mot : c'est le mot cherché
     // trouvé2 : c'est le résultat de la comparaison entre les deux mots.
//// Le programme est page suivante.
VARIABLES


k : entier // début du mot repéré
i : entier // début du mot cherché
reponse : booléen // Vrai si tout le mot cherché est identique


    Début

        TQ ( k <= long ) ET (phrase[i] = mot[k] FAIRE
            k := k + 1
            i := i + 1
        FinTQ

        SI ( k > long ) ALORS       // cette condition peut s'écrire
            reponse := VRAI     //  reponse := (k>long)
            reponse := FAUX
        FinSi


    Fin
```

From:
http://debian-facile.org/ - **Documentation - Wiki**

Permanent link:
http://debian-facile.org/utilisateurs:hypathie:tutos:algo-exo-constructions-d-algorithmes-de-procedure

Last update: **06/12/2014 17:08**