I2 *Algorithmique et programmation structurée*
Janvier 2019

TD no1	**Analyse descendante, pseudo-code**

Objectif de la séance :
Écrire un programme sous forme de fonctions et procédures en pseudo-code.

1 Écriture de pseudo-code

Exercices

Écrire l'algorithme pour :

1. **en fonction d'un entier n fourni en entrée, afficher la suite de nombre jusqu'à 0 (exclu), que le chiffre soit positif ou négatif. Exemples :**
   Si $n = 5$, le programme affiche 5 4 3 2 1
   Si $n = -3$, le programme affiche $-3\ -2\ -1$.
   Si $n = 0$, le programme n'affiche rien.

Solution

1. **Nom:** suite
   Role: ...
   Entrée: n : **Entier**
   Sortie: -
   Entrée/Sortie: -
   Déclaration: i : **Entier**
   debut
   si n $<$ 0 **alors**
   pour i ←n **à** -1 **faire**
   ecrire(i, ' ')
   finpour
   sinon
   pour i ←n **à** 1 **pas de** -1 **faire**
   ecrire(i, ' ')
   finpour
   finsi
   fin

2 Analyse de texte

L'objectif du programme à concevoir est de retrouver au sein d'un texte toutes les apparitions d'un mot donné et d'afficher à quelle(s) position(s) il apparaît.

Le programme demandera d'abord à l'utilisateur de saisir un texte. Ce texte peut contenir plusieurs phrases. Ensuite, le programme demande de saisir un mot dont le

nombre de caractères est compris entre 3 et 10 lettres, minuscules ou majuscules. Si le mot n'est pas valide, le programme redemande autant que de besoin d'entrer un nouveau mot.

Le programme devra rechercher toutes les occurrences de ce mot dans le texte et afficher à l'écran leur(s) position(s) dans la chaîne de caractère et le nombre d'occurences trouvées.

2.1 Conception globale

A partir des spécifications données ci-dessus, vous devez réaliser la conception globale du programme, selon le paradigme de la programmation structurée, en définissant une analyse descendante puis les signatures des fonctions et procédures à écrire.

Remarque no1 : Une chaîne de caractère est représentée numériquement par un tableau de caractères. Ainsi pour désigner le caractère situé à la position i d'une chaîne str, on écrit $str[i]$. Le premier indice d'une chaîne est 1.

Remarque no2 : Vous pouvez vous appuyer pour l'écriture de ce programme sur des fonctions et procédures existants dans tout langage de programmation impérative pour la manipulation de chaînes de caractères. Il s'agit de :

— *fonction longueur(str : Chaine de caracteres) : Naturel* qui retourne le nombre de caractères de la chaîne de caractères donnée en paramètre d'entrée ;

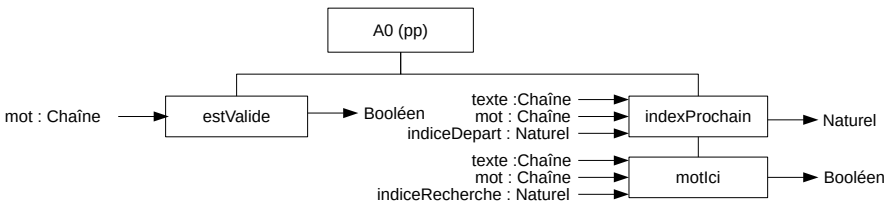
Ces fonctions et procédures existent. Vous n'avez donc pas à écrire leur pseudo-code. Par contre il est nécessaire de les faire apparaître dans l'analyse descendante pour définir les sous-programme où elles seront utilisées.

Pour effectuer cette analyse descendante, nous procéderons suivant les quatre étapes ci-dessous :

Exercices

1. **Définissez de manière informelle les tâches à faire accomplir par le programme, et éventuellement des sous-tâches au sein de chaque tâche.**
2. **Rédigez sous la forme d'un diagramme d'analyse descendante cette décomposition fonctionnelle.**
3. **Identifiez les entrées et les sorties de chaque sous-programme.**
4. **Ecrivez les signatures et descriptions des sous-fonctions.**

Solution



1.

2.2 Conception détaillée

Exercices

1. Écrivez la conception détaillée des fonctions

2. Écrivez la conception détaillée d'un programme principal

Solution

1. **fonction** estValide (mot :**Chaine de caracteres**) **: Booleen**

```
Déclaration  i : Naturel ; valide : Booleen
debut
    valide ← True ;
    si (longueur(mot)<3) ou (longueur(mot)>10) alors
        valide ← False
    sinon
        pour i ←1 à longueur(mot) faire
            si not (mot[i] in ['a'..'z'] + ['A'..'Z'] + ['','','','.']) alors
                valide ← False
            finsi
        finpour
    finsi
    retourner valide
fin
```

2. **fonction** motIci (texte, mot :**Chaine de caracteres** ; indiceRecherche :**Naturel**) **: Booleen**

```
Déclaration  estIci :Booleen ; i : Naturel ;
debut
    estIci ← True
    pour i ←1 à longueur(mot) faire
        si texte[indexRecherche + i - 1] <> mot[i] alors
            estIci ← False
        finsi
    finpour
    retourner estIci
fin
```

3. **fonction** IndexProchain (texte, mot :**Chaine de caracteres** ; indiceDepart :**Naturel**) **: Naturel**

```
Déclaration  i : Naturel ;
debut
    i ← indiceDepart
    tant que (i <= (longueur(texte)-longueur(mot))) et non (motIci(texte, mot, i)
    faire
        i ← i+1
```

```
    fintantque
    si motIci(texte, mot, i) alors
        retourner (  i)
    sinon
        retourner (  longueur(texte))
    finsi ;
fin
```

4.
```
Déclaration: leTexte, leMot : Chaine de caracteres ; i, nbOccurences : Naturel
debut
    nbOccurences ← 0
    lire(leTexte)
    repeter
        lire(leMot)
    jusqu'a ce que estValide(leMot)
    i ← 1
    tant que i <= (longueur(leTexte)-longueur(leMot)) faire
        i ←  indexProchain(leTexte, leMot, i)
        si i < longueur(leTexte) alors
            nbOccurences ← nbOccurences+1
            ecrire(leMot, 'trouvé en position ', i)
        finsi
        i ← i+1
    fintantque
fin
```