Informatique PCSI
Prérequis TP 1 : parcours séquentiel

Un parcours séquentiel est utilisé pour la recherche d'un élément ou le comptage d'éléments. Il s'agit de parcourir à l'aide d'une boucle une structure séquentielle comme une chaîne de caractères ou une liste.

1 Boucles

Un programme est composé d'une suite d'instructions, exécutées l'une après l'autre dans l'ordre où elles sont écrites, contenant des définitions de variables et de fonctions, des affectations, des boucles, des instructions conditionnelles, qui utilisent des expressions pouvant être des résultats d'appels de fonctions.

On distingue deux types de boucles :

- les boucles conditionnelles :

```
while condition:
    instructions
```

- les boucles non conditionnelles :

```
for elt in sequence:
    instructions
```

Avec une boucle non conditionnelle, le bloc d'instructions est répété n fois, n étant la longueur de la séquence (une liste, un tuple, une chaîne de caractères). La variable `elt` prend successivement la valeur de l'un des n éléments de la séquence. Cette variable `elt` peut être utilisée dans le bloc d'instructions ou pas.

Exemple :

```
for i in range(n):
    instructions
```

Avec le code qui suit, le bloc `instructions` est exécuté de la même manière :

```
i = 0
while i < n:
    instructions
    i = i + 1
```

Une variable `i` est créée. Cette variable n'est pas détruite après l'exécution de la boucle. Avec la boucle `for`, elle prend successivement les valeurs 0, 1, ..., `n`-1, avec la boucle `while` elle prend les valeurs 0, 1, ..., `n`.

2 Outils

2.1 Compteurs

Lorsqu'on utilise une boucle `while` on peut souhaiter compter le nombre de passages dans la boucle. De manière générale, on peut avoir besoin de compter le nombre d'apparitions d'un certain fait. On utilise alors une variable que l'on peut appeler *compteur*. Cette variable est initialisée à 0 et peut être incrémentée d'une unité à chaque passage dans la boucle.

Exemple avec une boucle conditionnelle sans test

Le paramètre `n` est un entier naturel. On compte le nombre de divisions euclidiennes successives de `n` par 2, jusqu'à arriver à un quotient nul. On obtient donc le nombre de chiffres dans l'écriture binaire de `n` si `n` est non nul.

```
def taille(n):
    cpt = 0
    while n > 0:
        cpt = cpt + 1
        n = n // 2
    return cpt
```

Avec une boucle inconditionnelle et un test

Le paramètre `n` est un entier naturel non nul. Le compteur est incrémenté quand `n` est divisible par `d`. On compte donc le nombre de diviseurs de `n` qui est le résultat renvoyé par la fonction.

```
def diviseurs(n):
    cpt = 0
    for d in range(1, n + 1):
        if n % d == 0:
            cpt = cpt + 1
    return cpt
```

2.2 Accumulateurs

Un accumulateur est semblable à un compteur mais il peut être incrémenté d'une valeur différente de 1 ou décrémenté.

```
def somme_pairs(liste):
    acc = 0
    for x in liste:
        if x % 2 == 0:
            acc = acc + x
    return acc
```

La boucle contient un test. Nous supposons que `liste` est une liste de nombres. La fonction renvoie la somme des nombres pairs contenus dans la liste.

L'exemple suivant qui concerne le calcul d'une moyenne est sans test.

```
def moyenne(liste):
    acc = 0
    for x in liste:
        acc = acc + x
    return acc / len(liste)
```

La liste est supposée non vide. Si n est sa longueur, nous disons que *le coût est linéaire en* n, car nous opérons n additions et n affectations effectuées dans la boucle. Nous supposons, et c'est le cas, que l'obtention de la longueur par `len(liste)` a un coût constant (une seule opération).

3 Recherche de valeurs

3.1 Définition

Une *occurrence* est l'apparition d'un fait, par exemple la présence d'un mot dans un texte. Lorsqu'on cherche dans un conteneur une occurrence d'un élément, on cherche si une place est occupée par cet élément dans le conteneur. Une place est représentée par son indice.

3.2 Recherche d'une occurrence

Il s'agit de rechercher de manière séquentielle la présence d'une valeur dans un tableau. Cela signifie que la valeur cherchée est successivement comparées à toutes les valeurs du tableau. Cette méthode est aussi appelée *méthode par balayage* ou *recherche linéaire* (en anglais, *linear search*). Un tableau peut être ici une liste, un p-uplet ou une chaîne de caractères. On cherche donc une valeur précise dans une liste ou un p-uplet ou un caractère précis dans une chaîne.

L'algorithme s'arrête dès que l'élément est trouvé ou si la fin du tableau est atteinte.

Un algorithme de recherche d'un élément `x` dans un tableau `t` de longueur `n` utilise, de manière générale, une boucle conditionnelle :

```
i = 0
tant que i < n et x différent de t[i]
    i = i + 1
fin tant que
si i < n
    renvoyer i
```

Cet algorithme est l'occasion d'aborder la notion de *coût*, notion qui sera précisée plus tard. Disons que *le coût en temps d'un algorithme est déterminé par le nombre d'opérations élémentaires, (affectations, comparaisons, opérations arithmétiques simples), exécutées par l'algorithme*.

Nous commençons par compter le nombre maximum de comparaisons effectuées.

Si l'élément recherché n'est pas dans le tableau, il est nécessaire de parcourir tout le tableau, donc d'effectuer n itérations, pour examiner chaque élément du tableau avec $2n$ comparaisons (comparaisons entre `i` et `n`, et entre `x` et `t[i]`. Nous avons alors pour la boucle un total de n affectations et n additions, donc de $4n$ opérations. Nous disons que *le coût est linéaire en* n.

Lorsqu'on parcourt une liste ou une chaîne, on parle de parcours séquentiel. Dans la notion de séquence nous avons la notion d'ordre. Les éléments ont chacun une place numérotée par un indice.

Nous pouvons envisager plusieurs variantes qu'il faut maîtriser : soit on souhaite simplement chercher si une valeur est présente, soit on cherche un indice d'occurrence, soit on cherche tous les indices d'occurrence. Quelques exemples seront proposés en exercices.