TD4. Tableau, tableaux dynamiques et boucles

Bases de la Programmation Impérative – Chapitre 2 : iterations
version du 25 décembre 2023

Ce TD a trois objectifs :

- Définir les notions suivantes :
  - **séquence de taille fixe**
  - **tableau**
  - **séquence de taille variable**
  - **tableau dynamique** aussi appelé **vecteur**
  - **list** sans `e`
  - **liste chaînée**
- Prendre en main la structure de contrôle `for` en parcourant des list Python
- Introduire la notion de **complexité de calcul** de façon intuitive

Préambule : éléments de langage

Question 1

Donner une définition du terme **séquence de taille fixe**.

Question 2

Donner une définition du terme **tableau** ainsi qu'une ligne de code, dans n'importe quel langage, permettant de créer un tableau.

Question 3

Donner une définition du terme **séquence de taille variable**.

Question 4

Donner une définition du terme **tableau dynamique** ainsi qu'une ligne de code, dans n'importe quel langage, permettant de créer un tableau dynamique.

Question 5

Donner une définition du terme **list** ainsi qu'une ligne de code Python permettant de créer une `list`.

Question 6

Donner une définition du terme **liste chaînée** et réfléchir à la façon de créer une telle liste.

Exercice 1 : parcours de tableaux

Dans cet exercice nous allons parcourir un tableau. Comme les tableaux n'existent pas en Python, quand nous disons tableau, cela signifie une `list` Python sur laquelle on effectue uniquement des accès indexés.

Question 1

Écrire une fonction qui renvoie la somme des éléments d'un tableau à l'aide d'une boucle `for`. Combien d'additions sont-elles nécessaires ?

Question 2

Écrire une fonction qui réalise la somme des éléments pairs d'un tableau à l'aide d'une boucle `for`. Combien d'additions sont-elles nécessaires ?

Question 3

Écrire une fonction qui réalise la somme des éléments d'indices pairs d'un tableau à l'aide d'une boucle `for`. Combien d'additions sont-elles nécessaires ?

Question 4

Écrire une fonction qui réalise le produit scalaire de deux vecteurs (au sens mathématique et non informatique tel que décrit dans le préambule) stockés dans deux tableaux. Combien d'additions sont-elles nécessaires ?

Exercice 2 : polygones

On stocke un polygone comme un tableau de `Point`. Les points sont donc ordonnés. On souhaite afficher tous les segments composant le polygone. On dispose pour ce faire d'une fonction `affiche_segment(point1, point2)`.

Question 1

Écrire une fonction qui réalise l'affichage du polygone. Combien d'appels à la fonction `affiche_segment` sont nécessaires ?

Exercice 3 : somme de chiffres

Question 1

Écrire une fonction qui réalise la somme de tous les chiffres dans la représentation en base 10 d'un entier. Quel est le nombre de tours de boucle nécessaires ?

Exercice 4 : multiplication binaire (pour aller plus loin)

On s'intéresse à la réalisation manuelle de la multiplication de deux entiers. Il s'agit de réimplémenter l'algorithme vu à l'école primaire, mais en base 2.

Dans cette implémentation on s'interdit d'utiliser les opérateurs `%`, `//` et `*`. On utilisera les opérateurs bit-à-bit `&` et `>>` en lieu et place de `%` et `//`. Comme nous l'avons vu, l'opérateur `&` applique un *ET LOGIQUE* bit-à-bit à ses deux opérandes. L'opérateur `x >> k` décale les bits du nombre `x` de `k` positions vers la droite. Autrement dit, `x >> k` divise `x` par `2^k`.

On utilisera également l'opérateur `x << k` qui décale les bits du nombre `x` de `k` positions vers la droite. Autrement dit, `x << k` multipliée `x` par `2^k`.

Question 1

Écrire une fonction `get_bits(nombre)` qui renvoie une `list` contenant tous les bits d'un entier.

Question 2

En utilisant la fonction `get_bits`, écrire une fonction qui réalise la multiplication en base 2 comme apprise à l'école primaire, de deux nombres. Combien de tours de boucles sont nécessaires ?