Tableaux
TD3

Ce document a pour objectif de vous familiariser avec les tableaux et les chaînes de caractères.

Ces deux types de données sont fortement liés selon les langages de développement utilisés. Vous devez connaître les contraintes et spécificités de chacun d'entre eux.

1 Tableaux

Les tableaux en algorithmique, et dans la plupart des langages de programmation, sont simplement des vecteurs. C'est-à-dire qu'il s'agit de tableaux à une seule dimension. La plupart des langages de programmation font démarrer leurs tableaux à la position 0. Ceci implique qu'un tableau de taille 5 (donc qui contient 5 cases) démarre de la case 0 et finit à la case 4 (01234). Dans vos algorithmes, vous manipulerez donc souvent les cases de 0 à *longueur du tableau - 1*, avec comme condition *tant que (iterateur < longueur)* (on s'arrête lorsque l'itérateur atteint la longueur, donc après la dernière case du tableau). Il existe cependant quelques langages où les tableaux démarrent à la case 1, donc finissent à la case *longueur du tableau* : vérifiez toujours quel est l'index (le numéro de case) de la première case dans chacun des langages de programmation que vous utiliserez.

0	*1*	*2*	*3*	*4*
42	14	18	666	1337

Les tableaux contiennent parfois des éléments de types différents (des entiers, des flottants, et des chaînes de caractères), mais il est fréquent qu'ils ne peuvent contenir qu'un seul et unique type à la fois (uniquement des entiers, ou uniquement des flottants, ou uniquement un type précis). Dans les exercices de ce sujet, nos tableaux ne pourront contenir qu'un seul et unique type à chaque fois. Si vous déclarez un tableau d'entiers, alors on ne peut y mettre que des entiers et rien d'autre.

Pour accéder à une case précise d'un tableau, on indique l'index entre crochets. Par exemple, pour un tableau d'entiers stocké dans la variable *tab*, on accède à la première case en écrivant : *tab[0]*. On peut récupérer le contenu de la case pour le mettre dans une variable en écrivant *var = tab[2]*. On peut écrire une valeur dans une case d'un tableau en écrivant *tab[2] = 42*.

L'index peut également être une variable, mais celle-ci doit être un entier (on ne peut pas accéder à une case dont l'index est un flottant ou un caractère). Ainsi, on ne peut ni faire *tab[0,42]* ni *tab['a']* ni *tab["trois"]*, mais si une variable n contient un entier (par exemple : $n = 3$), on peut écrire *tab[n]*.

Toujours concernant les index des cases, si on essaye d'accéder à une case inexistante (par exemple un index négatif comme -1 ou au delà de la taille du tableau), une erreur est renvoyée. Par exemple, pour un tableau de taille 5 (index de 0 à 4), on peut accéder à la case 4 (*tab[4]*) mais pas à la case 5 (*tab[5]*).

2 Chaînes de caractères

Les chaînes de caractères (c'est-à-dire les suites de caractères) sont en réalité des tableaux contenant des caractères. Le format standard des chaînes de caractères implique qu'une chaîne finisse par un unique caractère spécial : *'\0'*. Ainsi, la chaîne de caractères standard "lol" est en réalité un tableau de taille 4 dont la dernière case contient '\0'. Grâce à cette convention, on n'a plus besoin d'embarquer la taille de la chaîne, il suffit juste de chercher un '\0' pour comprendre qu'il n'y a rien après.

0	*1*	*2*	*3*
'l'	'o'	'l'	'\0'

Vous pouvez néanmoins rencontrer des chaînes de caractères non-standards, c'est-à-dire qu'elles ne finissent pas nécessairement par '\0', et peuvent même en contenir dans la chaîne elle-même. Dans ce cas très précis, on vous fournira toujours la taille de la chaîne de caractère en plus du tableau.

0	*1*	*2*	*3*	*4*	*5*	*6*
'A'	'b'	'C'	'\0'	'D'	'e'	'F'

Exercices variés

1) Écrivez maintenant une fonction vérifiant que les éléments d'un tableau forment un palindrome (la longueur est donnée en paramètre). *PalindromeTab(tab, len)*

2) Écrivez une fonction testant si une chaîne de caractères est un palindrome (le '\0' final ne sera bien entendu pas pris en compte dans le palindrome). *PalindromeStr(str)*

3) Écrivez une fonction comparant deux tableaux. Si les tableaux sont les mêmes, alors vous renverrez *vrai*, sinon vous renverrez *faux*. *CompareTab(tab1, tab2, len1, len2)*

4) Écrivez une fonction qui renvoie la valeur la plus grande/petite du tableau. Vous ferez une version itérative et une version récursive pour Min et Max. *MinTabIter(tab, len) MaxTabIter(tab, len) MinTabRec(tab, len) MaxTabRec(tab, len)*

5) Écrivez une fonction qui calcule la somme de tous les éléments d'un tableau. Vous ferez une version itérative et une version récursive. *SommeTabIter(tab, len) SommeTabRec(tab, len)*

Améliorez l'algorithme pour n'utiliser qu'un seul itérateur tout en ajoutant à chaque fois le début et la fin du tableau (n'oubliez pas de vous arrêter là où il faut et de ne pas ajouter trop d'éléments).

6) Écrivez une fonction qui calcule la taille d'une chaîne de caractères (sans compter le '\0' final : "lol" a une taille de 3). Vous ferez une version itérative et une version récursive. *StrlenIter(str) StrlenRec(str)*

7) Écrivez une fonction qui compare deux chaînes de caractères et renvoie *vrai* si elles sont similaires et *faux* si elles diffèrent. Vous ferez une version itérative et une version récursive. *StrcmpIter(str1, str2) StrcmpRec(str1, str2)*

Essayez d'écrire une version itérative qui teste d'abord la longueur des chaînes, puis, une autre version sans ce test.

8) Écrivez une fonction qui recherche un élément dans un tableau. Vous ferez une version itérative et une version récursive. *RechercheEltTabIter(tab, len, elt) RechercheEltTabRec(tab, len, elt)*

9) Écrivez une fonction qui compare deux tableaux. Si les deux tableaux contiennent les mêmes éléments aux mêmes positions, vous renverrez *vrai*, sinon vous renverrez *faux*. Vous ferez une version itérative et une version récursive. *CompareTabIter(tab1, tab2, len1, len2) CompareTabRec(tab1, tab2, len1, len2)*

10) Écrivez une fonction qui teste si les éléments d'un tableau sont tous en ordre croissant. Si tous les éléments sont ordonnés du plus petit au plus grand, alors vous renverrez *vrai*, sinon vous renverrez *faux*. Si les éléments sont tous égaux, alors le résultat sera *vrai*. Vous ferez une version itérative et une version récursive. *TestCroissantTabIter(tab, len) TestCroissantTabRec(tab, len)*

Faites la même chose pour tester la décroissance.

11) Écrivez une fonction qui insère un élément dans un tableau à une position précise, et décale les éléments vers la fin. Le dernier élément qui devrait disparaître du tableau sera renvoyé par la fonction. Par exemple, pour un tableau contenant [A B C D], si l'on y insère 'Z' en position 1, le tableau doit devenir [A Z B C] et la fonction doit renvoyer D. *InsertionTab(tab, len, elt, pos)*

12) Écrivez une fonction qui supprime un élément dans un tableau à une position précise, et décale les éléments vers le début. L'élément supprimé du tableau sera renvoyé par la fonction. De plus, pour éviter que le dernier élément soit dupliqué, vous prendrez un élément en paramètre qui sera inséré à la fin. Par exemple, pour un tableau contenant [A B C D], si l'on supprime l'élément en position 1 tout en ajoutant 'Z', le tableau doit devenir [A C D Z] et la fonction doit renvoyer B. *SuppressionTab(tab, len, pos, elt)*

13) Écrivez une procédure qui inverse la position de tous les éléments. Vous ne devez pas construire de nouveau tableau, mais uniquement modifier en place le tableau (en utilisant des variables temporaires). Par exemple, pour un tableau contenant [A B C D], si l'on inverse la position des éléments, le tableau doit devenir [D C B A]. *InverserTab(tab, len)*

14) Écrivez une fonction qui vérifie sur une chaîne de caractères est bien un préfixe d'une autre chaîne de caractères. Cette fonction renvoie *vrai* si le prefixe est bon et *faux* si ce n'est pas le cas. Par exemple, "abc" est un préfixe à "abcdef", mais pas "bcd" ni "def". Vous ferez une version itérative et une version récursive. *PrefixStrIter(str, prefix) PrefixStrRec(str, prefix)*

15) Écrivez une fonction qui vérifie sur une chaîne de caractères est bien un suffixe d'une autre chaîne de caractères. Cette fonction renvoie *vrai* si le suffixe est bon et *faux* si ce n'est pas le cas. Par exemple, "def" est un suffixe à "abcdef", mais pas "cde" ni "abc". Vous ferez une version itérative et une version récursive. *SuffixStrIter(str, suffix) SuffixStrRec(str, suffix)*

16) Écrivez une fonction qui vérifie sur une chaîne de caractères est contenue dans une autre chaîne de caractères. Cette fonction renvoie *vrai* si la sous-chaîne est contenue dans la chaîne principale et *faux* si ce n'est pas le cas. Par exemple, "abc" est contenue dans "ababc", mais pas "cde" ni "cba". Vous ferez une version itérative et une version récursive. *SubStrIter(str, sub) SubStrRec(str, sub)*

Attention, certains cas sont difficiles à détecter. Dans certains cas, il sera plus difficile de détecter "abc" dans "abcbc" que "abc" dans "ababc". N'oubliez pas de vérifier plusieurs cas complexes tels que : rechercher "abc" dans "abababc" ou "abcbcbc".

Ce document et ses illustrations ont été réalisés par Fabrice BOISSIER en octobre 2022.
La plupart des exercices sont inspirés du cahier d'algo de Nathalie "Junior" BOUQUET et Christophe "Krisboul" BOULLAY.
(dernière mise à jour octobre 2024)