[PROF] Tableaux
Fiche 3

Ce document a pour objectif de guider les enseignants pour le cours d'algorithmique. Il est déconseillé de le fournir aux étudiants, car il vise surtout à vous permettre de guider la séance. Vous n'êtes évidemment pas obligé de le suivre à la lettre (c'est même déconseillé, car cela va autant vous perturber vous que la classe : suivez votre chemin de pensée et/ou celui de la classe, et ensuite vérifiez que vous n'avez rien oublié).

La troisième séance vise à faire découvrir les tableaux aux étudiants, ainsi que les chaînes de caractères (qui en découlent).

1 Tableaux

42	14	18	666	1337

— Montrer un exemple de tableau d'entiers très simple
— Faire remarquer que ce tableau ne contient QUE des entiers...
— ...les tableaux des langages bas niveaux/proches de la machine ne contiennent que le même type de données...
— ...les langages plus abstraits (comme Python) autorisent à stocker des types différents dans les tableaux
— En pratique, c'est plus rapide de ne gérer *que* le même type de données
— Indiquer que l'on verra plus tard comment stocker plusieurs types distincts de données dans une liste

— Demander aux étudiants l'index de démarrage du tableau
— Rappeler que *la plupart* des langages font démarrer à 0...
— ...mais que certains langages font démarrer à 1

1	*2*	*3*	*4*	*5*
0	*1*	*2*	*3*	*4*
42	14	18	666	1337

— Faites une double numérotation du tableau (démarrant à 0 et à 1)
— Indiquez la longueur du tableau
— Indiquez le numéro de la dernière case
— Expliquez l'avantage des deux numérotations :
  — démarrer à 0 permet de faire une boucle s'arrêtant *avant* la longueur du tableau ($< len$)
  — démarrer à 1 permet de n'avoir que des valeurs entières/non nulles comme index + la dernière case sera celle de la longueur

— Présenter le format des types : **`int tab[]`**
— Présenter le format des types : **`int tab[5]`**
— Demander aux étudiants comment accéder à la case 3 du tableau
— Indiquer qu'en algorithmique, si on donne un tableau en paramètre, sa taille sera *toujours* indiquée dans un autre paramètre (**`tab`** et **`len`**)

2 Chaînes de caractères

— Demander aux étudiants comment sont stockées les chaînes de caractères

— Expliquer qu'en C, et dans beaucoup de langages, c'est un tableau de caractères

— Indiquer que la dernière case contient '\0'

— Expliquer la différence entre les 2 mondes :

  — dans les années 50-60, on imposait des formulaires de taille prédéfinie (30 caractères pour les noms, ...)
  c'est plus simple à gérer, peu de possibilité de *casser* le système (car tout est données), mais extrêmement peu flexible (l'humain s'adapte à la machine)

  — dans les années 60-70, on cherche un délimiteur de fin de chaîne de caractères
  beaucoup plus flexible et orienté utilisateur (la machine s'adapte à l'humain), dépend des limitations techniques de la machine (que faire si on envoie un roman dans le champ nom ?), le programme cherche une *fin* de ligne, donc la donnée agit sur les chemins d'exécution

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

— Demander la taille de la chaîne et la taille du tableau

— Faire remarquer que la taille de la chaîne donne l'index de la dernière case du tableau

— Indiquer qu'il ne faut pas oublier de prévoir de la place pour le '\0' lorsque l'on recopie des chaînes de carcatères

— Indiquer qu'il y a une différence entre **`char str[]`** et **`char *str`**, mais que l'on utilise le deuxième pour simplifier

— Indiquer que lorsque l'on parle de chaînes de caractères, l'accès à la chaîne sera donné avec **`str`**, mais pas sa taille

— Indiquer que néanmoins, si on parle explicitement de *tableaux* de caractères, alors leur taille sera fournie, car ceux-ci peuvent contenir plusieurs '\0' et ne pas se terminer par l'un d'eux (car il ne s'agit pas d'une *chaîne de caractères* mais d'un *tableau de caractères*)

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)