

Travaux Pratiques thème 2

Les tableaux statiques

Exercice 1 – Initialiser un tableau.

Dans le programme `exo_tab.c`, écrire une procédure permettant d'initialiser un tableau de taille **N** (**N** est une **constante** définie par l'instruction `#define`) avec des entiers aléatoires compris entre 1 et P (P est un entier défini par l'utilisateur, ou, par la suite, initialisé dans le programme).

Exercice 2 – Recherche d'éléments.

Dans la suite du programme `exo_tab.c`, écrire une procédure permettant d'afficher l'indice d'un élément **n** entré au clavier. Si l'élément **n** est présent plusieurs fois, le programme affichera le plus petit indice. Si l'élément n'est pas présent, il affichera : "**n non trouvé**".

Exercice 3 – Suppression des doublons.

Toujours dans la suite du programme précédant, écrire une procédure permettant de ne garder que la première occurrence de chaque entier. Les autres occurrences seront remplacées par 0.

Exemple : le tableau {1,20,1,9,4,20,1,9} deviendra {1,20,0,9,4,0,0,0}.

Exercice 4 – Génération d'un tableau.

Écrire un programme `exercice4.c` permettant de générer un tableau d'entiers **T** de taille **N** dans l'ordre croissant. Le premier élément du tableau est un entier aléatoire compris entre 1 et 10. Chaque autre élément d'indice **i** du tableau est défini par la relation de récurrence $T[i] = T[i-1] + r$ avec **r** un entier aléatoire compris entre 0 et **P** (On prendra **P** assez petit, 2 ou 3).

Exercice 5.

Dans cet exercice, il s'agit de réécrire le programme de l'exercice 3 (*Suppression des doublons*) avec un tableau trié généré comme l'exercice 4.

Dans un programme `exercice5.c`, modifier le programme de façon à ce qu'il ne parcourt qu'une seule fois le tableau (pas de boucles imbriquées). Il faut s'appuyer sur le fait que le tableau est maintenant trié.

Exercice 6 – Échange d'éléments.

Écrire un programme `echange.c` qui prend en entrée un tableau d'entiers **T** et deux entiers **i** et **j**, et qui échange les éléments du tableau d'indices **i** et **j**.

Exercice 7 – Algorithme de Fisher-Yates.

Cet exercice consiste à mélanger les éléments d'un tableau trié. On pourra utiliser le tableau trié généré dans l'exercice 4. La procédure de mélange (appelé *algorithme de Fisher-Yates*) consiste à parcourir le tableau **T** et, pour chaque indice **i**, à échanger l'élément **T[i]** avec un élément tiré au hasard situé après (c'est-à-dire dont l'indice **j** est tel que $j > i$). Le programme s'appellera `melange.c`.

Exercice 8 – Correcteur.

Écrire un programme **correcteur.c** qui génère un tableau d'entiers de taille **N** composés uniquement de 0 et de 1. Les N-1 premiers éléments du tableau sont tirés au hasard et le dernier élément est la somme des N-1 premier modulo 2.

Ce procédé très simple permet de détecter certaines erreurs de transmission. Si on veut transmettre une série de N-1 bits, on transmet (ce qui est peu coûteux en mémoire) N bits en ajoutant le bit de parité (défini précédemment). Le récepteur de la série peut ainsi vérifier si la transmission n'a pas introduit d'erreur. Bien entendu certaines erreurs de transmissions se compensent et cette méthode ne détecte pas toutes les erreurs possibles.

Exercice 9 – Bataille navale simple.

On veut écrire un programme **bataille_navale.c** qui simule le jeu de la bataille navale pour un seul joueur (dans un cas simple où les bateaux sont tous de longueur 1).

Le programme doit :

1. Attribuer **pts** points au joueur (par exemple, **pts=9**).
2. Fabriquer une grille 4×4 et choisir au hasard 5 cases de cette grille pour l'emplacement des bateaux. En pratique on créera un tableau 4 × 4 dont tous les coefficients sont nuls sauf ceux correspondant aux bateaux qui vaudront 1. Dans le cours du jeu, les coefficients correspondants aux cases ayant subi un tir vaudront 2 (et ainsi les coefficients correspondant aux cases ayant subi un tir sur un bateau vaudront 3).
3. Afficher au début une grille vierge (afficher autant de caractères ' ' que de cases du tableau.)
4. Procéder au tir en :
 - Demandant au joueur les coordonnées de son tir ;
 - Vérifiant si la case correspondante contient un bateau ;
 - Affichant la nouvelle grille avec un 'o' dans la case choisie si celle-ci contient un bateau et un 'x' dans le cas contraire ;
 - Retirant un point au joueur si son tir a fait plouf ;
5. Recommencer l'opération précédente tant que tous les bateaux n'ont pas été découverts ou tant que le score n'est pas tombé à zéro.
6. Afficher finalement le score du joueur.

Pour cet exercice, ne pas hésiter à améliorer le programme de toutes les façons qui paraissent intéressantes (affichage des coordonnées sur la grille, rappel du score à chaque étape, bateaux de taille supérieure, etc)

Exercice 10 – Histogramme.

Dans cet exercice, il s'agit d'écrire un programme **histogramme.c** qui permet de représenter un histogramme de notes entières comprises entre 0 et 20 (c'est-à-dire qu'il y a 21 notes possibles).

Par exemple, si dans une classe, il y a 10 étudiants qui ont obtenus les notes suivantes : 2, 3, 4, 3, 0, 0, 2, 3, 3, 2 (c'est-à-dire 2 étudiants ont obtenu la note 0, 0 étudiant la note 1, 3 étudiants la note 2, 4 étudiants la note 3, 1 étudiant la note 4), le tableau représentant l'histogramme sera : [2, 0, 3, 4, 1] et l'affichage de l'histogramme sera :

```
0 | **
1 |
2 | ***
```

```
3 | *****
```

```
4 | *
```

Les notes (entre 0 et 20) seront générées aléatoirement pour un nombre fixé d'étudiants (par exemple 150).

Exercice 11 – Loto.

Dans cet exercice, nous voulons simuler un jeu de loto (**loto.c**). Deux tableaux représentent l'un le tirage du loto, l'autre la grille du joueur. Le premier tableau sera rempli aléatoirement de six entiers compris entre 1 et 49 (comme au loto) tandis que le second sera rempli par l'utilisateur. Dans chacun des tableaux, toutes les cases doivent contenir des valeurs différentes. Une fois les deux tableaux remplis, il faut comparer leur contenu pour déterminer si le joueur a gagné.