
TD 2

Exercice n° 1

Soit un tableau d'entiers de 50 entiers dont les valeurs ont été affectées aléatoirement :
affichez la plus petite, la plus grande et la moyenne des valeurs contenues dans le tableau.

```
#include <stdio.h>

main()
{
    #define MAX_TAB 50

    int tab[MAX_TAB];
    int i, min, max, moyenne;

    /* initialisation du générateur aléatoire */
    srand((unsigned) time(NULL));

    /* rand() retourne une valeur aléatoire entre 0 et RAND_MAX */
    for(i=0; i < MAX_TAB; i++)
        tab[i] = rand() % 100;

    /* affichage des valeurs du tableau */
    printf("\n");
    for(i=0; i < MAX_TAB; i++)
        printf("%d_", tab[i]);
    printf("\n");

    /* calcul de la plus petite, la plus grande et la moyenne */
    moyenne = 0;
    min = max = tab[0];
    for(i = 0; i < MAX_TAB; i++) {
        moyenne += tab[i];
        if (tab[i] < min)
            min = tab[i];

        if (tab[i] > max)
```

```

        max = tab [ i ];
    }

    printf ( "\n la plus grande valeur est %d\n la plus petite est %d\n la moyenne est %d\n" , max , min , moyenne / MAX_TAB );
}

```

Exercice n° 2

Soit un tableau d'entiers de 50 entiers dont les valeurs ont été affectées aléatoirement :

Tri par minimum Le tri par minimum (tri par extraction) est un des algorithmes de tri les plus triviaux. Il consiste en la recherche du plus petit élément que l'on va replacer à la première position (indice 0 dans notre tableau), puis on recherche le second plus petit élément (ou le second plus petit) que l'on va replacer en seconde position, etc., jusqu'à ce que le tableau soit entièrement trié.

Donnez l'algorithme en C de ce tri.

Tri à bulles Le tri à bulles est un algorithme de tri très simple. Il consiste à faire remonter le plus grand élément du tableau (comme une bulle d'air remonte à la surface) en comparant les éléments successifs. C'est-à-dire que l'on va comparer le 1er et le 2e élément du tableau, conserver le plus grand et puis les échanger s'ils sont désordonnés les uns par rapport aux autres. On recommence cette opération jusqu'à la fin du tableau. Ensuite, il ne reste plus qu'à renouveler cela jusqu'à l'avant-dernière place et ainsi de suite... On arrête quand le tableau à trier est de taille 1 ou qu'on n'a pas fait d'échanges au dernier passage.

Donnez l'algorithme en C de ce tri.

Combien d'itérations sont nécessaires pour réaliser ces deux tris ?

```
#include <stdio.h>
```

```

main ()
{
    #define MAX_TAB 50

    int tab [ MAX_TAB ];
    int i , j , min ;

    /* initialisation du générateur aléatoire */
    srand ( ( unsigned ) time ( NULL ) );

    /* rand () retourne une valeur aléatoire entre 0 et RAND_MAX */
    for ( i = 0 ; i < MAX_TAB ; i ++ )
        tab [ i ] = rand () % 100 ;
}

```

```

/* affichage des valeurs du tableau */
printf("\n");
for(i=0; i < MAX_TAB; i++)
    printf("%d_", tab[i]);
printf("\n");

/* tri par minimum */
for(i = 0 ; i < MAX_TAB - 1 ; i++)
{
    min = i;
    for(j = i+1 ; j < MAX_TAB ; j++)
        if(tab[j] < tab[min])
            min = j;
    if(min != i)
    {
        int x;

        x = tab[i];
        tab[i] = tab[min];
        tab[min] = x;
    }
}

/* affichage du tableau trié */
printf("\n");
for(i=0; i < MAX_TAB; i++)
    printf("%d_", tab[i]);
printf("\n");

/* rand() retourne une valeur aléatoire entre 0 et RAND_MAX */
for(i=0; i < MAX_TAB; i++)
    tab[i] = rand() % 100;

/* affichage des valeurs du tableau */
printf("\n");
for(i=0; i < MAX_TAB; i++)
    printf("%d_", tab[i]);
printf("\n");

/* tri à bulles */
i = 0; /* Indice de répétition du tri */
j = 0; /* Variable de boucle */
int tmp = 0; /* Variable de stockage temporaire */

```

```

/* Booléen marquant l'arrêt du tri si le tableau est ordonné */
int en_desordre = 1;
/* Boucle de répétition du tri et le test qui
   arrête le tri dès que le tableau est ordonné */
for(i = 0 ; (i < MAX_TAB) && en_desordre; i++)
{
    /* Supposons le tableau ordonné */
    en_desordre = 0;
    /* Vérification des éléments des places j et j-1 */
    for(j = 1 ; j < MAX_TAB - i ; j++)
    {
        /* Si les 2 éléments sont mal triés */
        if(tab[j] < tab[j-1])
        {
            /* Inversion des 2 éléments */
            tmp = tab[j-1];
            tab[j-1] = tab[j];
            tab[j] = tmp;

            /* Le tableau n'est toujours pas trié
               */
            en_desordre = 1;
        }
    }
}

/* affichage des valeurs du tableau */
printf("\n");
for(i=0; i < MAX_TAB; i++)
    printf("%d_", tab[i]);
printf("\n");
}

```

Exercice n° 3

Le *run-length encoding*, appelé en français le codage par plages, est un algorithme de compression de données en informatique.

Le système s'applique essentiellement à des documents scannés en noir et blanc : au lieu de coder un bit par point, on dispose d'un compteur indiquant combien de points blancs ou noirs se suivent. Comme il est rare de ne pas avoir un grand nombre de pixels noirs ou de pixels blancs qui se suivent, le système a bien pour effet une compression.

Par exemple, considérons un écran de texte noir sur fond blanc. Il sera constitué de longues séquences de pixels blancs pour le fond, et de courtes séquences de pixels noirs pour le texte. Représentons une ligne d'un tel écran, avec B pour les pixels noirs et W pour les pixels blancs :

```
WWWWWWWWWWWWWWBWWWWWWWWWWWWWWWWWWWWBBBWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWW BWWWWWWWWWWWWWWWW
```

Un encodage *RLE* consiste alors à indiquer pour chaque suite de pixels d'une même couleur, le nombre de pixels de cette séquence. Le résultat comporte en général moins de caractères, bien que ce ne soit pas une obligation. On obtient par exemple pour la ligne précédente :

```
12W1B14W3B23W1B11W
```

Nous allons utiliser ce principe pour compresser un tableau d'entiers comprenant un grand nombre d'entrées redondantes. Dans notre système de compression, le premier entier indiquera le nombre d'occurrences de la valeur qui le suit. La valeur 0 sera utilisée pour compléter le tableau.

Exemple :

Le codage de 4444433322110 sera représenté par :

```
5433222110
```

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    #define MAX_TAB 10
```

```
    int tab[MAX_TAB], compression[MAX_TAB];
```

```
    int i, j;
```

```
    int compteur, val;
```

```
    /* initialisation du générateur aléatoire */
```

```
    srand((unsigned) time(NULL));
```

```
    /* rand() retourne une valeur aléatoire entre 0 et RAND_MAX */
```

```
    for(i=0; i < MAX_TAB; i++)
```

```
        tab[i] = rand() % 2; /* ici valeurs aléatoires entre 0 et 1
        */
```

```
    /* affichage des valeurs du tableau */
```

```
    printf("\n");
```

```
    for(i=0; i < MAX_TAB; i++)
```

```
        printf("%d_", tab[i]);
```

```
    printf("\n");
```

```
    /* i pointeur sur le premier tableau
```

```
       j sur le second */
```

```
    i = j = 0;
```

```
    compteur = 1;
```

```

/* compression du tableau */
while (i < MAX_TAB){
    val = tab[i];

    compression[j + 1] = val;
    i++;

    if (tab[i] == val)
        compteur++;
    else {
        compression[j] = compteur;
        compteur = 1;
        j += 2;
    }
}

for (i = j; i < MAX_TAB; i++)
    compression[j] = 0;

/* affichage du tableau compressé */
printf("\n");
for(i=0; i < MAX_TAB; i++)
    printf("%d_", compression[i]);
printf("\n");
}

```