

Exercice 1

Une première méthode consiste à compter les cubes bleus, les blancs et les rouges. On n'a plus alors qu'à réinitialiser correctement le tableau, avec le bon nombre de cubes de chaque couleur.

Ceci donne donc le code plutôt simple qui suit :

```
typedef enum {BLEU, BLANC, ROUGE} Couleur ;

void compterLesCouleurs(Couleur t[], int taille, int *b, int *w, int *r)
{
    int i;

    *r=*w=*r=0;
    for(i=0;i<taille;i++)
        if (t[i]==BLEU) *b++;
        else if (t[i]==BLANC) *w++;
        else *r++;
}

void trierLesCouleur(Couleur t[], int taille)
{
    int i, rouges, blancs, bleus, nb=0;

    compterLesCouleurs(t, taille, &bleus, &blancs, &rouges);
    for(i=0; i<bleus;i++) { t[nb]=BLEU; nb++; }
    for(i=0; i<blancs;i++) { t[nb]=BLANC; nb++; }
    for(i=0; i<rouges;i++) { t[nb]=ROUGE; nb++; }
}
```

Une autre méthode, plus compliquée, pour obtenir les cubes ordonnés par couleur est la suivante : on parcourt un à un tous les cubes, et on effectue le traitement, en fonction de la couleur du cube courant :

- si le cube est BLEU, on le retire et on le met au début, en première position ; on passe au suivant ;
- si le cube est BLANC, on ne fait rien, on se contente de passer au suivant ;
- si le cube est ROUGE, on le met à la fin, en dernière position ; on passe au suivant.

Mettre un cube au début ou à la fin d'un tableau est en fait une insertion : on décale les cubes pour venir combler le vide laissé par le cube courant, il reste alors une place libre au début ou à la fin selon le cas, dans laquelle on vient ranger le cube. Evidemment, cela aurait été bien plus facile avec des listes. Sur l'exemple de l'énoncé, cela donne (les cubes en gras sont ceux déjà testés et rangés, le premier cube non gras en partant de la gauche est celui testé à chaque étape) :

Blanc	Bleu	Rouge	Bleu	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc
Blanc	Bleu	Rouge	Bleu	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc
Bleu	Blanc	Rouge	Bleu	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc
Bleu	Blanc	Bleu	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc	Rouge
Bleu	Bleu	Blanc	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc	Rouge
Bleu	Bleu	Blanc	Blanc	Bleu	Bleu	Rouge	Bleu	Blanc	Rouge
Bleu	Bleu	Bleu	Blanc	Blanc	Bleu	Rouge	Bleu	Blanc	Rouge
Bleu	Bleu	Bleu	Bleu	Blanc	Blanc	Rouge	Bleu	Blanc	Rouge
Bleu	Bleu	Bleu	Bleu	Blanc	Blanc	Bleu	Blanc	Rouge	Rouge
Bleu	Bleu	Bleu	Bleu	Bleu	Blanc	Blanc	Blanc	Rouge	Rouge
Bleu	Bleu	Bleu	Bleu	Bleu	Blanc	Blanc	Blanc	Rouge	Rouge

On pourrait améliorer la méthode en faisant évoluer les bornes inférieure et supérieure du tableau, afin de réduire les coûts d'insertion, en effet, à chaque étape, l'ensemble de cubes à ordonner est de plus en plus restreint.

Exercice 2

Il apparaît que le nom de répertoire se termine avec le dernier '/'. La base du nom de fichier est elle constituée de la chaîne de caractères entre le dernier '/' et le dernier '.'. Il n'y a bien sûr pas forcément de '/', ni de '.', ou pas forcément un '.' dans la dernière partie du nom : votre méthode doit fonctionner aussi sur des noms tels que "/mnt.dir/tmp0001", "tmp0002", ".", "/" ou encore ".tmp0003".

On écrira alors plusieurs fonctions. Nous avons besoin d'une fonction qui recherche la dernière occurrence d'un caractère donné, appelée avec '/' et la chaîne initiale puis avec '.' et la partie nom de finale du nom de fichier. Cette fonction délivre -1 en cas d'échec. Pour initialiser les différentes structures du résultat, nous aurons besoin de fonctions de copie de chaînes. Nous utiliserons celles offertes par le langage C. Attention à la fonction strncpy, qui ne copie qu'une partie de chaîne, et qui ne termine pas le résultat par le caractère fin de chaîne '\0'.

```
typedef char Nom[128];

typedef struct {
    Nom repertoire, base, suffixe ;
} StrucNomFichier ;

int chercheDernier(Nom nomFichier, char c)
{
    int res=-1, i=0;
    while(nomFichier[i] != '\0') {
        if (nomFichier[i] == c) res=i;
        i++;
    }
    return res;
}

StrucNomFichier decompose (Nom nomFichier)
{
    int pf, ps;
    StrucNomFichier snf ;

    pf = 1+chercheDernier(nomFichier, '/') ;
    ps = chercheDernier (nomFichier+pf, '.') ;
    strncpy(snf.repertoire, nomFichier, pf) ;
    snf.repertoire[pf]='\0';
    if (ps == -1) /* pas de suffixe */ {
        strcpy(snf.base, nomFichier+pf) ;
        snf.suffixe[0]='\0' ;
    }
    else {
        strncpy(snf.base, nomFichier+pf, ps);
        snf.base[ps]='\0';
        strcpy(snf.suffixe, nomFichier+pf+ps);
    }
    return snf ;
}
```

Exercice 3

L'indice k varie de 1 à 25. Il suffit de calculer la progression des indices i et j, qui évoluent ainsi :

i	4	0	1	2	3	2	3	4	0	1	0	1	2	3	4	3	4	0	1	2	1	2	3	4	0
j	2	3	4	0	1	1	2	3	4	0	0	1	2	3	4	4	0	1	2	3	3	4	0	1	2
k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Le résultat de l'exécution de la fonction *zorclub* donne donc :

```
11  18  25  2  9
10  12  19  21 3
4   6   13  20 22
23  5   7   14 16
17  24  1   8   15
```

Bien que ce n'était pas demandé, on pouvait remarquer qu'il s'agit d'un carré magique : les sommes de tous les éléments de chaque ligne, de chaque colonne et des deux diagonales sont identiques (65 sur cet exemple).