

**ECOLE NATIONALE
DE L'AVIATION CIVILE**

Langage C

Partie 2

Emmanuel Romagnoli

- La notion de tableau
- Les tableaux à plusieurs dimensions
- Le cas des chaînes de caractères

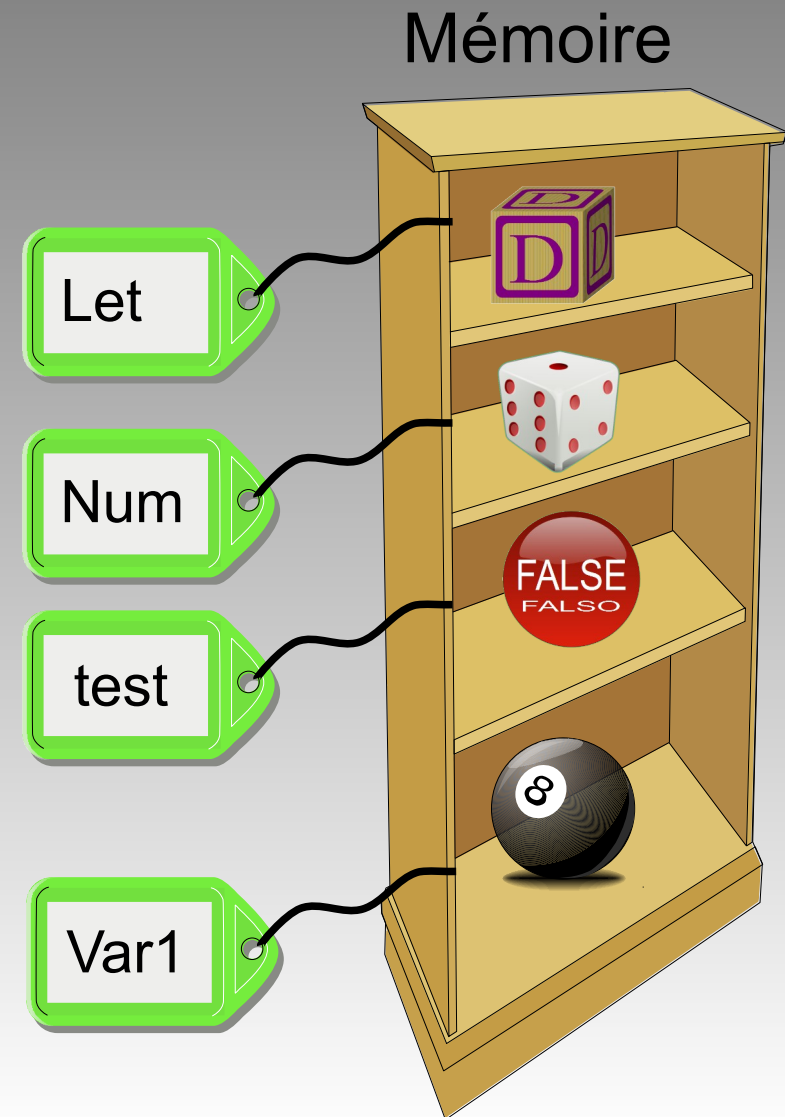
La notion de tableau



Les variables scalaires

Jusqu'à présent, on a manipulé des variables qui ne peuvent contenir qu'une seule valeur.

Un emplacement mémoire est associé à une étiquette.

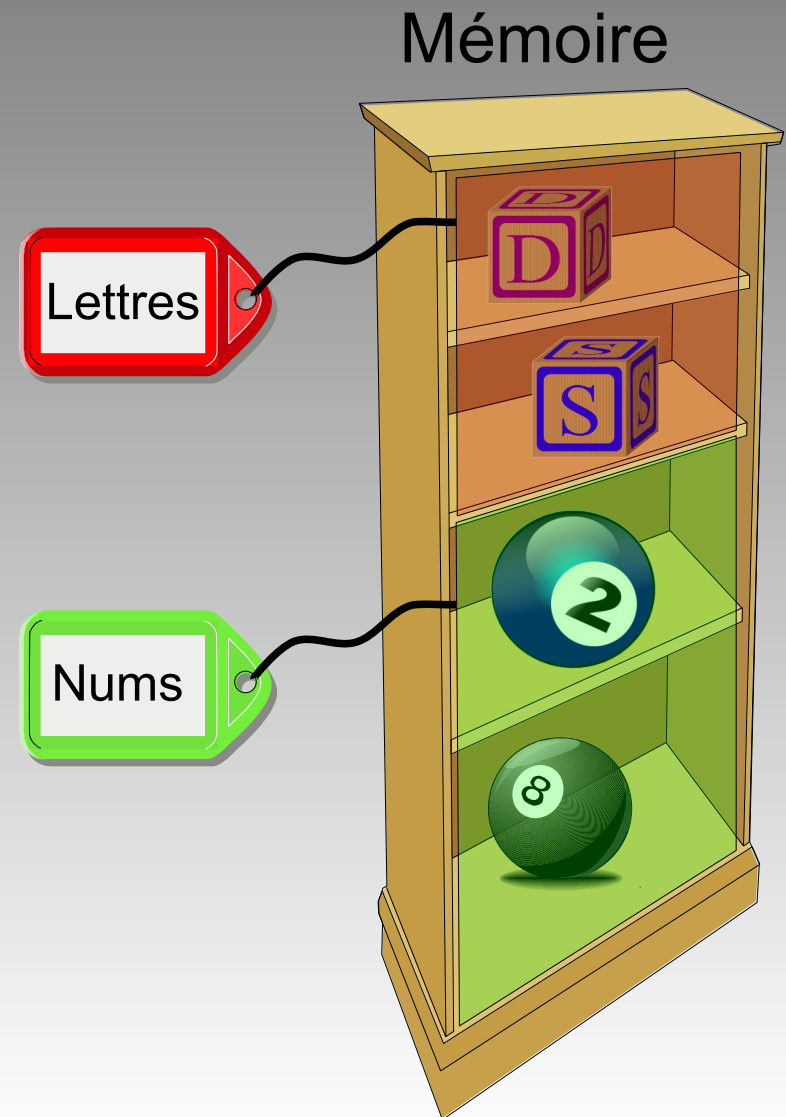


Plusieurs cases derrière une étiquette

On va maintenant associer plusieurs emplacements mémoire à une seule étiquette.

On pourra manipuler des variables de type « tableau » qui pourront stocker plusieurs informations.

Chaque information sera stockée dans une case du tableau correspondant à un emplacement mémoire.

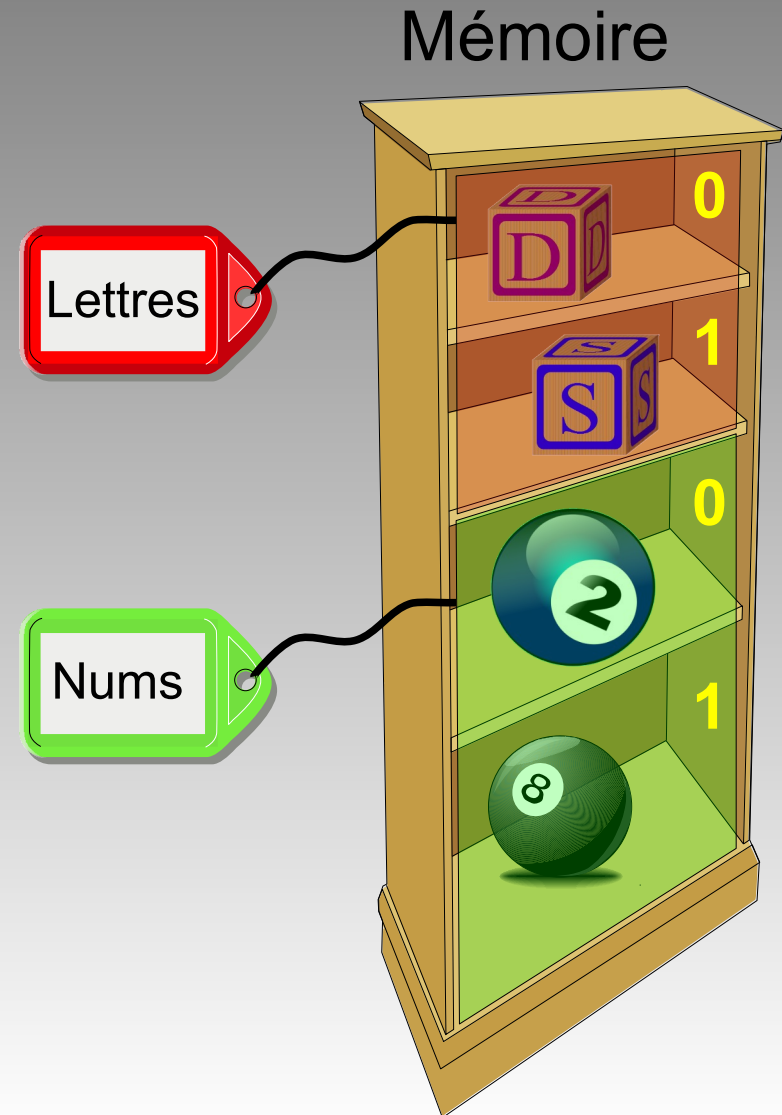


La numérotation des cases du tableau

Chaque case du tableau est identifiée par un numéro et la numérotation commence à partir de 0.

La case n°0 du tableau « Lettres » contient « D » et la case n°1 de ce même tableau contient « S ».

La case n°0 du tableau « Nums » contient « 2 » et la case n°1 de ce même tableau contient « 8 ».



Première méthode

Sa création s'effectue en spécifiant le type d'information stockée dans les cases, suivi du nom du tableau puis la taille indiquée entre les crochets **[]**.

```
type nom [taille];
```

Exemple :

```
int tab [10];
```

Si on oublie de spécifier la taille, on obtient une erreur à la compilation : « array size missing ».

Seconde méthode

On peut créer un tableau tout en initialisant son contenu, en respectant la syntaxe ci-dessous :

```
type nom [] = { val1, val2, ...};
```

Exemple :

```
int tab [] = { 10, 20, 33 };
```

On remarque qu'il n'est plus nécessaire de mettre la taille entre les crochets : le compilateur est capable de la déduire en composant le nombre d'éléments entre les accolades.

Seconde méthode

On peut utiliser la syntaxe précédente afin d'initialiser que certaines cases du tableau comme le montre l'exemple ci-contre.

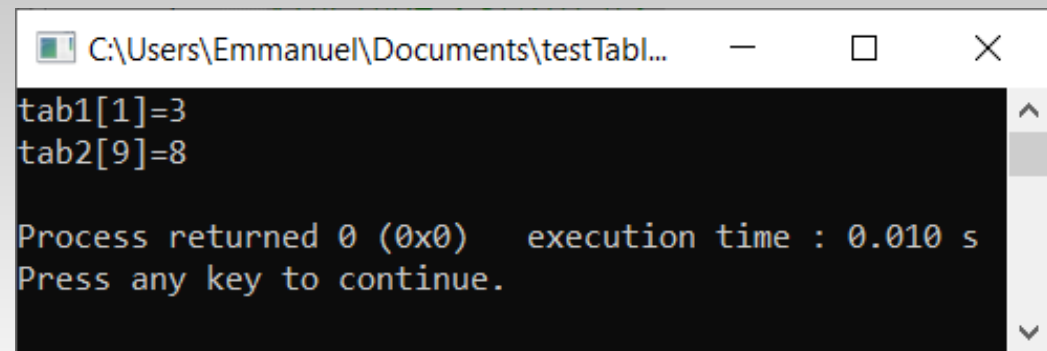
Le compilateur déduit la taille 10 de `tab2`, grâce au plus grand indice mentionné entre les crochets (ici 9).

```
#include <stdio.h>

int main()
{
    int tab1 [3] = { [1] = 3 };
    int tab2 [] = { [9] = 8 };

    printf("tab1[1]=%d\n", tab1[1]);
    printf("tab2[9]=%d\n", tab2[9]);

    return 0;
}
```



```
C:\Users\Emmanuel\Documents\testTabl...
tab1[1]=3
tab2[9]=8

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

Seconde méthode

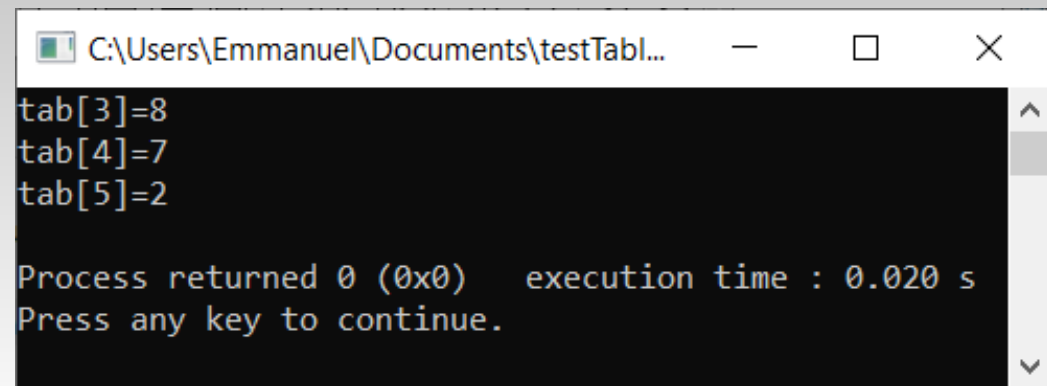
Dans le cas où on précise l'indice d'une case pour en initialiser le contenu, le compilateur est également capable de déterminer la valeur des indices pour les initialisations suivantes, et par voie de conséquence, la taille du tableau.

```
#include <stdio.h>

int main()
{
    int tab [] = { [3] = 8, 7, 2 };

    printf("tab[3]=%d\n", tab[3]);
    printf("tab[4]=%d\n", tab[4]);
    printf("tab[5]=%d\n", tab[5]);

    return 0;
}
```



```
C:\Users\Emmanuel\Documents\testTabl...
tab[3]=8
tab[4]=7
tab[5]=2

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

On ne peut stocker qu'**un seul type d'information** dans un tableau.

On ne peut pas mettre un `int` dans la case n°0, puis un `double` dans la case n°1...

La taille d'un tableau est **figée**, on ne peut pas le modifier après sa création.

Une fois le tableau créé, on peut alors utiliser l'opérateur **[]** pour spécifier l'indice de la case où une valeur doit être affectée

```
int main()
{
    int tab [2];

    tab[0] = 10;
    tab[1] = 20;

    return 0;
}
```

On peut aussi utiliser l'opérateur **[]** pour lire le contenu d'une case d'un tableau.

```
#include <stdio.h>

int main()
{
    int tab [2];

    tab[0] = 10;
    tab[1] = 20;

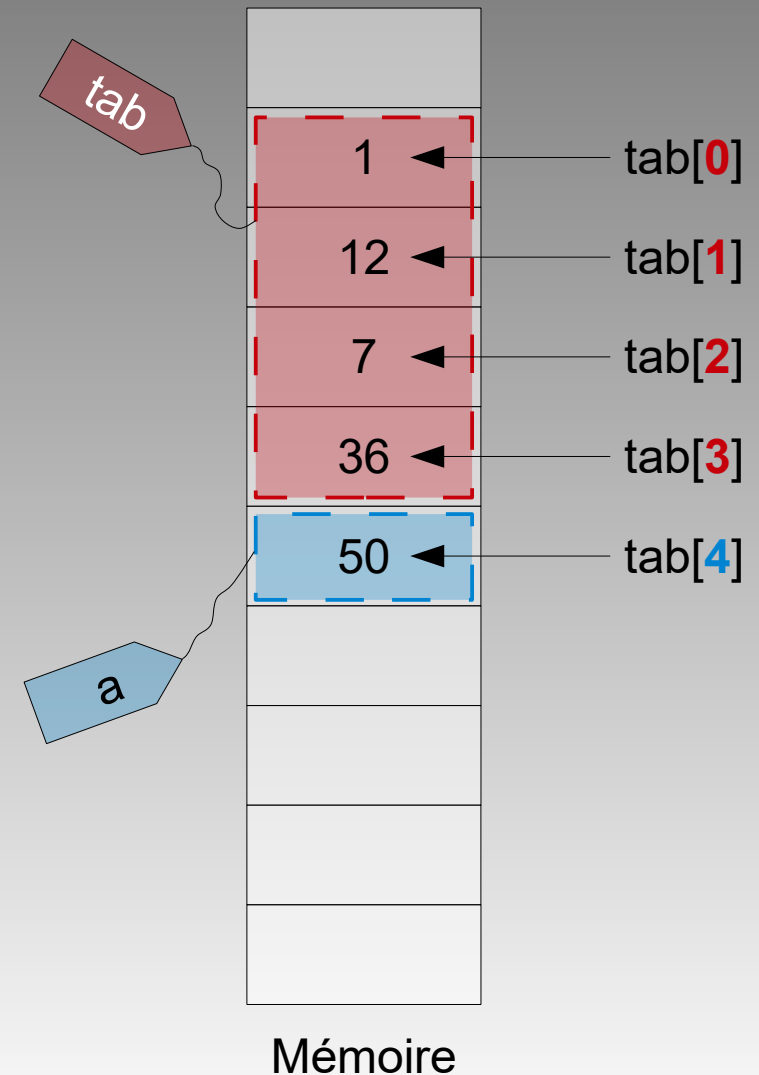
    printf ("La case 0 contient %d\n", tab[0]);
    printf ("La case 1 contient %d\n", tab[1]);

    return 0;
}
```

Le dépassement de capacité

Il est nécessaire de **conserver la taille du tableau** dans une autre variable.

Cette variable servira à contrôler « à la main » qu'un indice ne dépasse pas la capacité du tableau car le langage **C n'effectue aucun contrôle** de l'indice placée entre crochets.

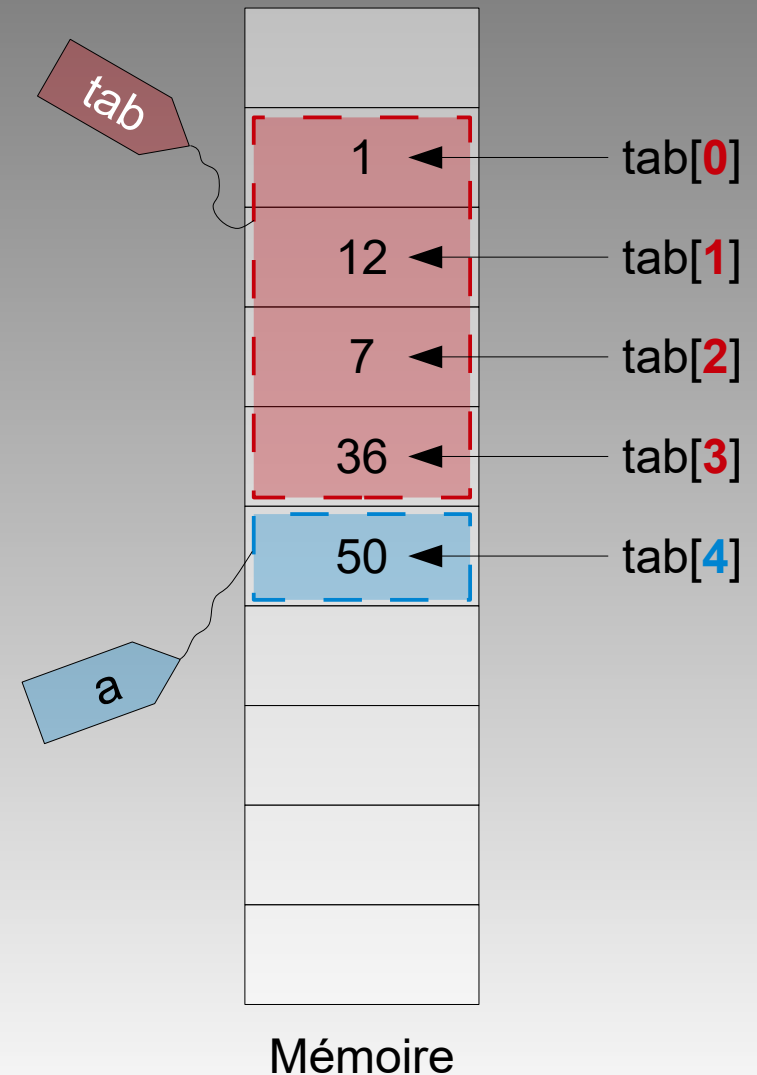


Le dépassement de capacité

Sans contrôle, il est possible d'écraser, une information écrite à côté du tableau et ainsi faire planter un programme.

Par exemple, on crée un tableau de 4 cases et une variable `a` qui est placée automatiquement juste après le tableau.

Si on écrit « `tab[4] = 10;` », on écrase la variable `a`.



Connaître la taille d'un tableau

```
#include <stdio.h>

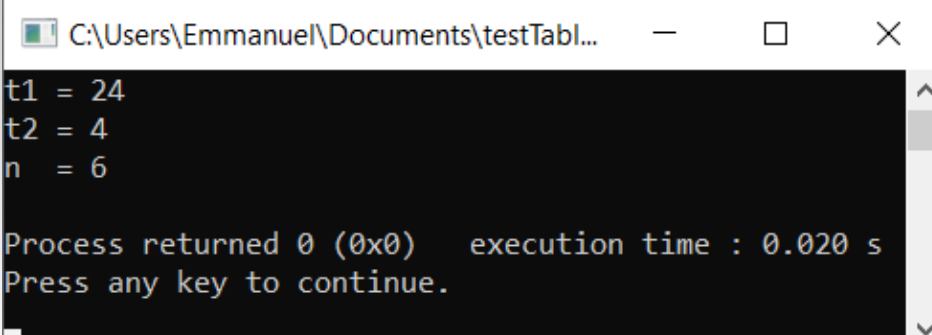
int main()
{
    int  tab [] = { [3] = 8, 7, 2 };

    long t1 = sizeof(tab);
    long t2 = sizeof(tab[0]);
    long n  = t1/t2;

    printf("t1 = %ld\n", t1);
    printf("t2 = %ld\n", t2);
    printf("n  = %ld\n", n );

    return 0;
}
```

On peut connaître la taille, exprimée en octets, d'un tableau grâce à l'opérateur `sizeof`. On peut donc en déduire le nombre de cases, en divisant cette taille par celle, occupée par sa première case.



```
C:\Users\Emmanuel\Documents\testTabl...
t1 = 24
t2 = 4
n  = 6

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```


L'algorithme de départ

On considère l'algorithme ci-dessous

```
ALGORITHME Moyenne

VARIABLE

    tab : Tableau [1, 10] de réels
    cpt : entier
    moy : réel

DEBUT

    POUR cpt DE 1 A 10, PAS 1 FAIRE
        Afficher ("Saisissez la note n°" + cpt + " ")
        Lire (tab [cpt])
    FIN DE POUR

    moy ← 0
    POUR cpt DE 1 A 10, PAS 1 FAIRE
        moy ← moy + tab[cpt]
    FIN DE POUR
    moy ← moy / 10

    Afficher ("La moyenne est : " + moy)

FIN
```

Traduction de l'algorithme en C

Le début de l'algorithme est le suivant :

```
ALGORITHME moyenne  
  
VARIABLE  
  
    tab : Tableau [1, 10] de réels  
    cpt : entier  
    moy : réel
```

Sa traduction en langage C donne :

```
int main (int argc, char** argv)  
{  
    float tab [10];  
    short cpt;  
    float moy;
```

Traduction de l'algorithme en C

Le morceau ci-dessous ...

DEBUT

```
POUR cpt DE 1 A 10, PAS 1 FAIRE  
  Afficher ("Saisissez la note n°" + cpt)  
  Lire (tab [cpt])  
FIN DE POUR
```

... se traduit en C par :

```
for (cpt=0; cpt<10; cpt++)  
{  
  printf ("Saisissez la note numero %d : ");  
  scanf ("%f", &(tab[cpt]) );  
}
```

Traduction de l'algorithme en C

La troisième partie est :

```
moy ← 0

POUR cpt DE 1 A 10, PAS 1 FAIRE
    moy ← moy + tab[cpt]
FIN DE POUR

moy ← moy / 10
```

Ce qui donne en C :

```
moy = 0;

for (cpt=0; cpt<10; cpt++)
    moy += tab[cpt];

moy /= 10 ;
```

Traduction de l'algorithme en C

Le dernier morceau comporte :

```
Afficher ("La moyenne est : " + moy)  
  
FIN
```

On obtient donc en C :

```
printf ("La moyenne est : %.2f\n", moy);  
}
```

Le programme complet

Le programme complet en C est finalement le suivant :

```
#include <stdio.h>

int main (int argc, char** argv)
{
    float tab [10];
    short cpt;
    float moy;

    for (cpt=0; cpt<10; cpt++)
    {
        printf ("Saisissez la note numero %d : ", cpt);
        scanf ("%f", &(tab[cpt]) );
    }

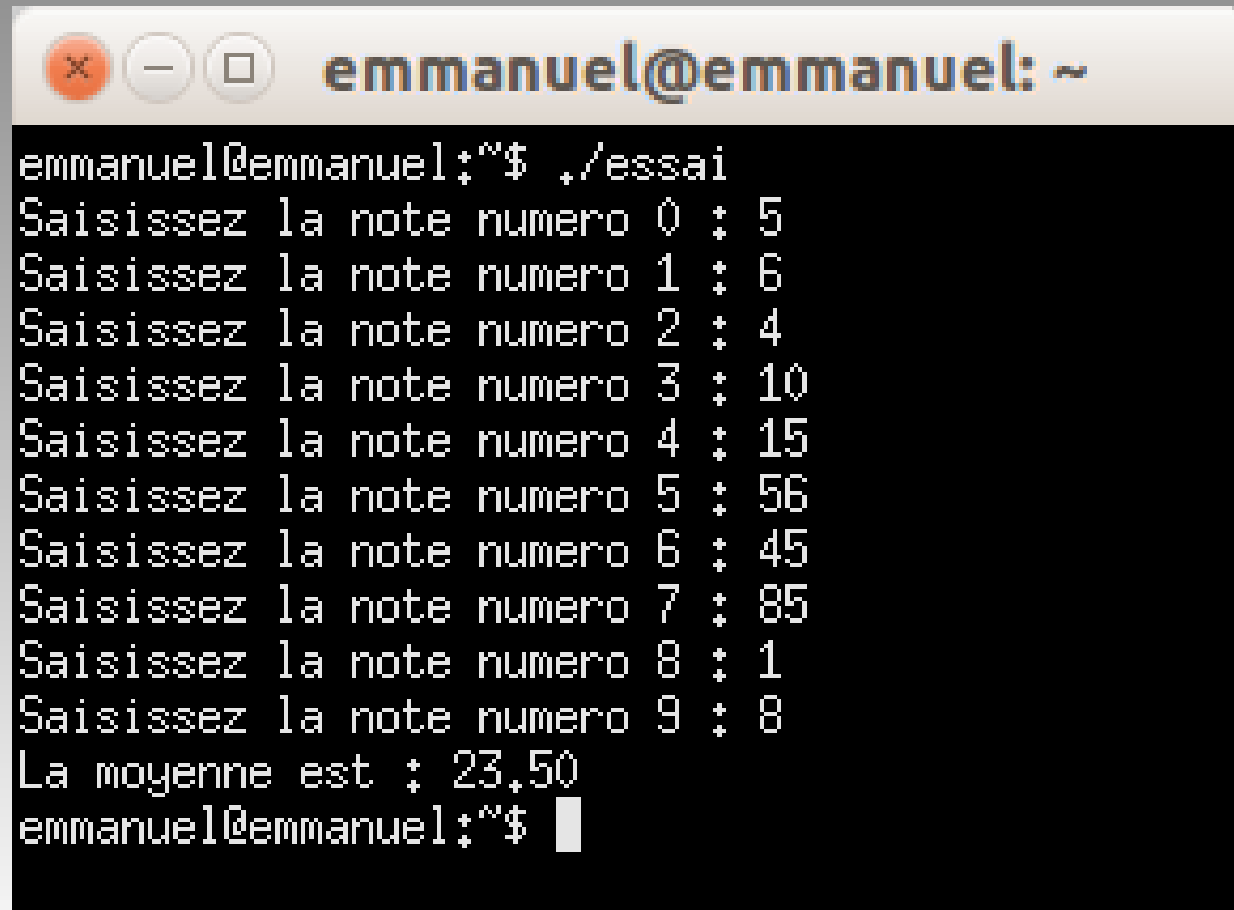
    moy = 0;

    for (cpt=0; cpt<10; cpt++)
        moy += tab[cpt];

    moy /= 10;

    printf ("La moyenne est : %.2f\n", moy);
}
```

On exécute le programme et on obtient le résultat suivant :



```
emmanuel@emmanuel: ~  
emmanuel@emmanuel:~$ ./essai  
Saisissez la note numero 0 : 5  
Saisissez la note numero 1 : 6  
Saisissez la note numero 2 : 4  
Saisissez la note numero 3 : 10  
Saisissez la note numero 4 : 15  
Saisissez la note numero 5 : 56  
Saisissez la note numero 6 : 45  
Saisissez la note numero 7 : 85  
Saisissez la note numero 8 : 1  
Saisissez la note numero 9 : 8  
La moyenne est : 23.50  
emmanuel@emmanuel:~$
```

Maximum et minimum

Écrire un programme en C, appelé « MaxMin » qui effectue les traitements suivants :

- recherche des valeurs minimale et maximale dans un tableau de valeurs ;
- stockage ces valeurs dans des variables appelée `min` et `max` ;
- affichage le contenu de ces variables à l'écran.

Insérer et décaler

Écrire un programme qui effectue les traitements suivants :

- création d'un tableau contenant 10 entiers initialisés à 0 ;
- saisie de 10 entiers, pour chaque saisie, on doit :
 - décaler toutes les valeurs contenues dans les cases de 1 cran « vers le haut » (on perd la valeur contenue dans la case n°9) ;
 - affecter la valeur saisie dans la case n°0.

Variance et écart-type

Modifiez le programme « Moyenne » de manière à y ajouter le calcul de la variance et de l'écart-type σ .

Rappels :

$$\text{Var}(X) = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2 = \sigma^2$$

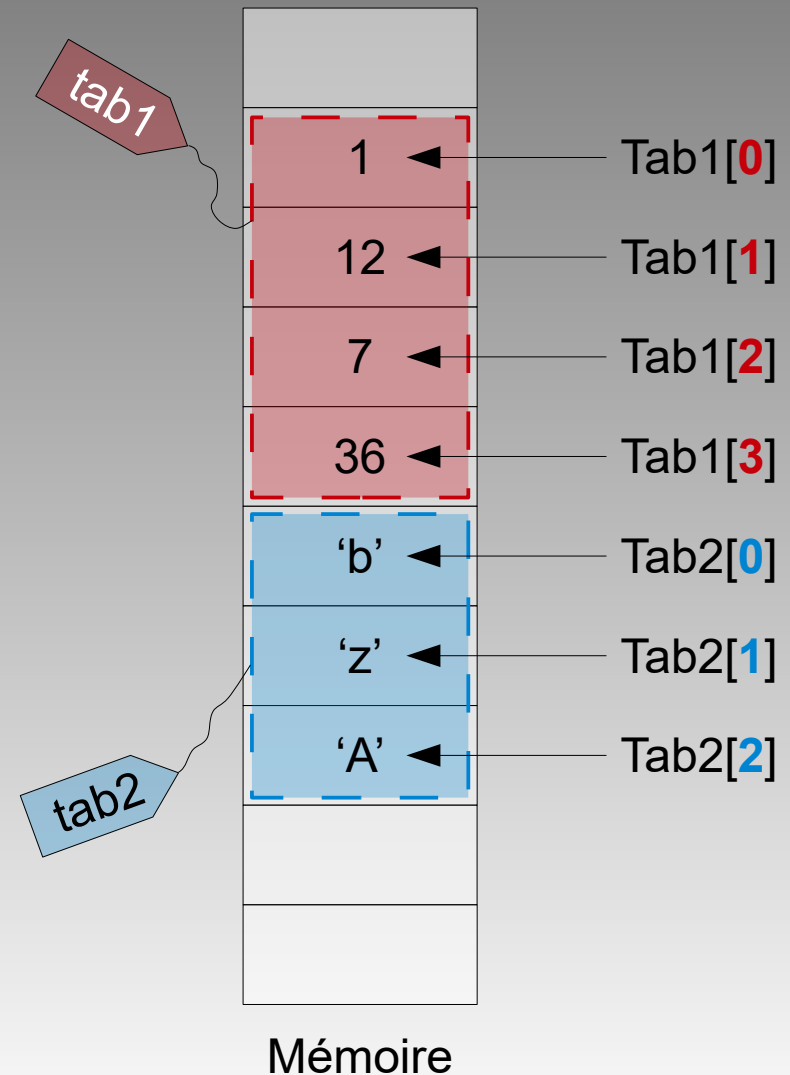
Les tableaux à plusieurs dimensions



Rappel concernant le tableau à 1 dimension

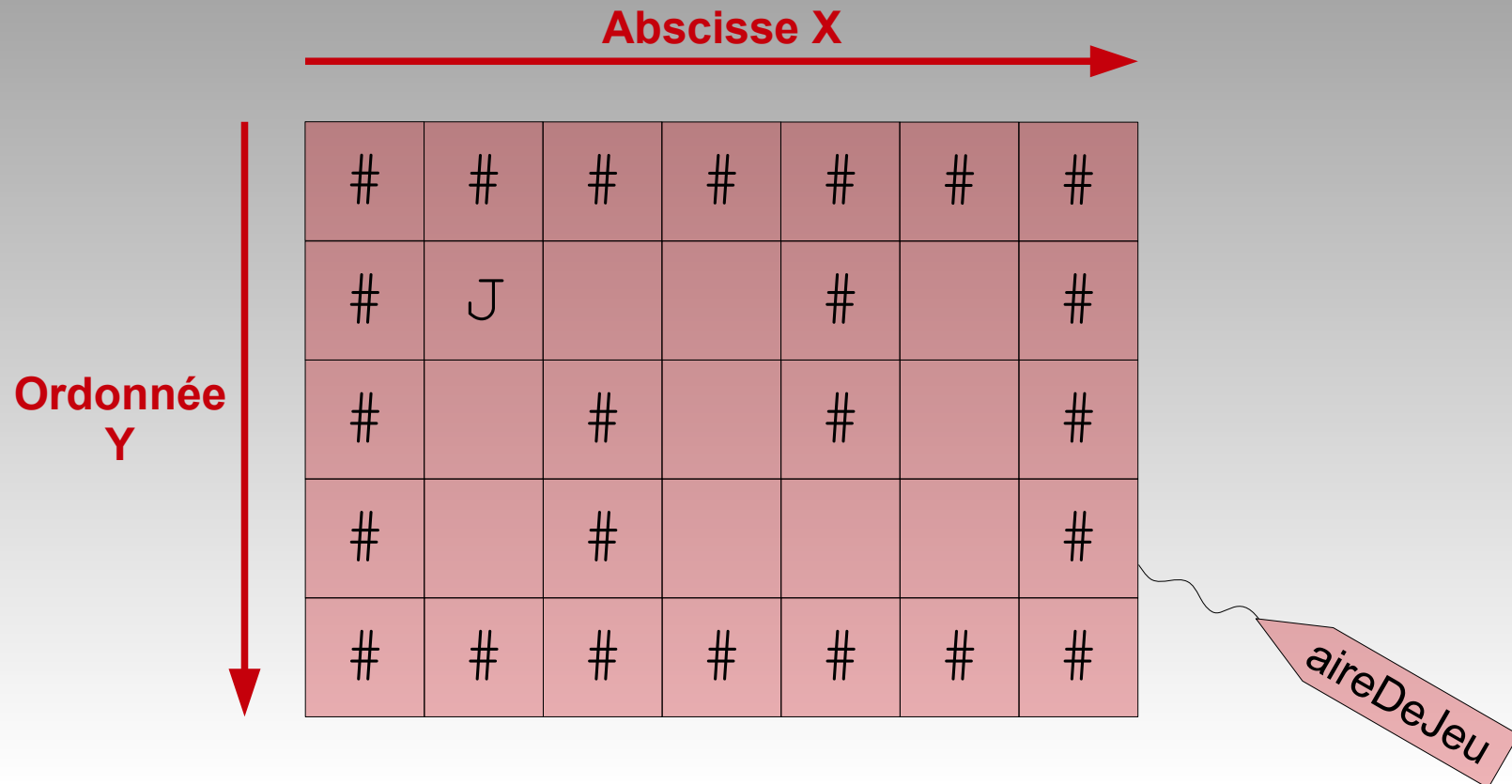
Jusqu'à présent, on a manipulé des tableaux à **1** dimension car on utilisait **1** indice pour repérer la case du tableau.

En mémoire, ce tableau est une suite de positions mémoires qui contiennent toutes le même type d'éléments (par exemple un tableau de nombre entiers).



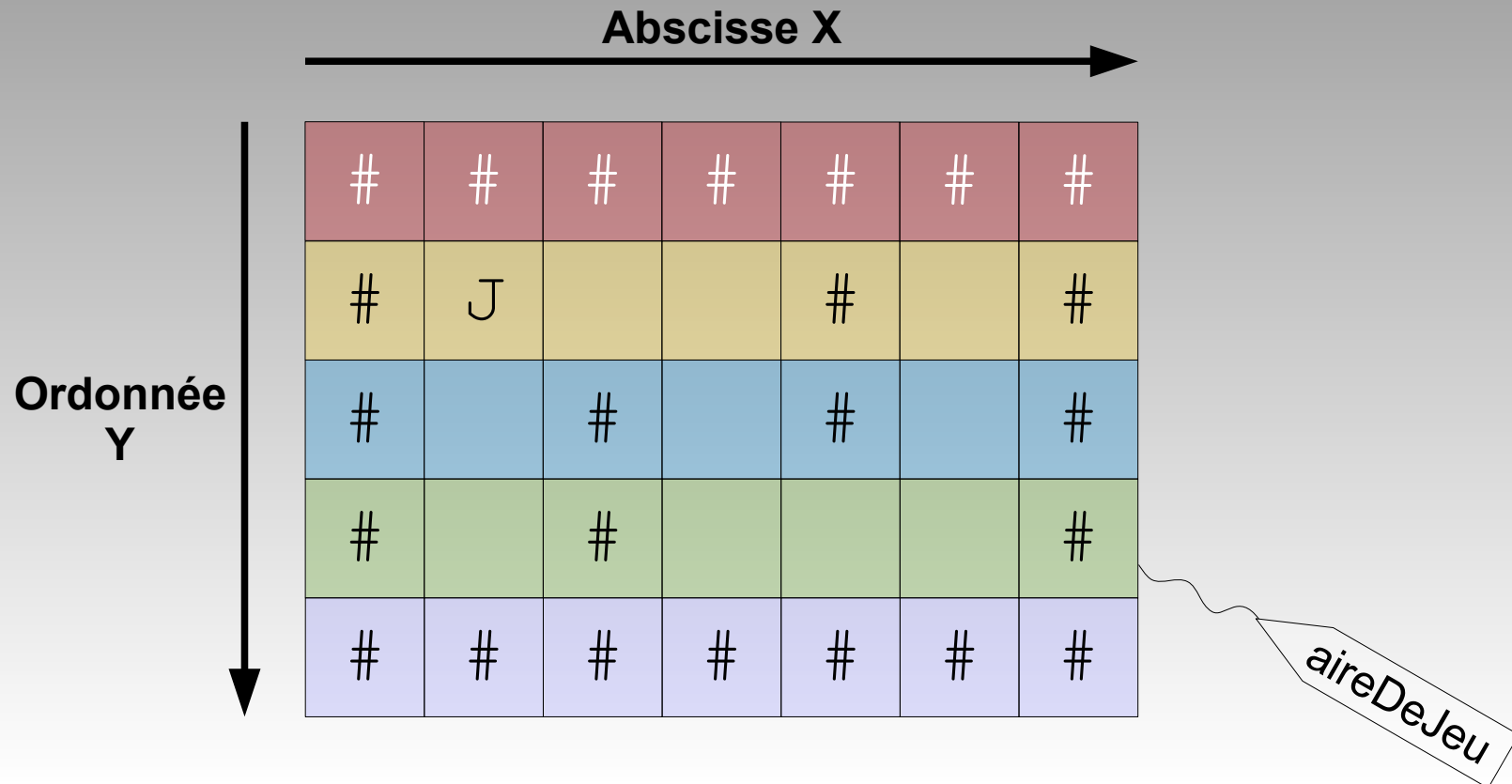
Le tableau à 2 dimensions

On considère maintenant des tableaux à **2** dimensions qui vont permettre de modéliser, par exemple, des aires de jeu, des plaques de métal...



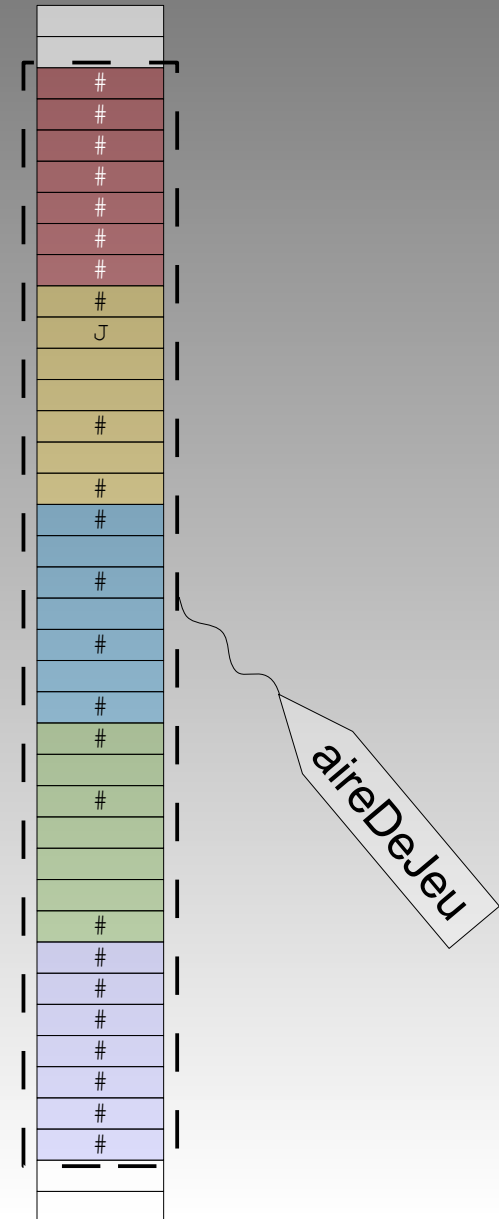
Un tableau de lignes

En réalité, ce tableau à **2** dimensions sera construit comme **un tableau de tableaux** (dans notre exemple, un tableau de lignes qui contiendront des caractères dans chaque case).



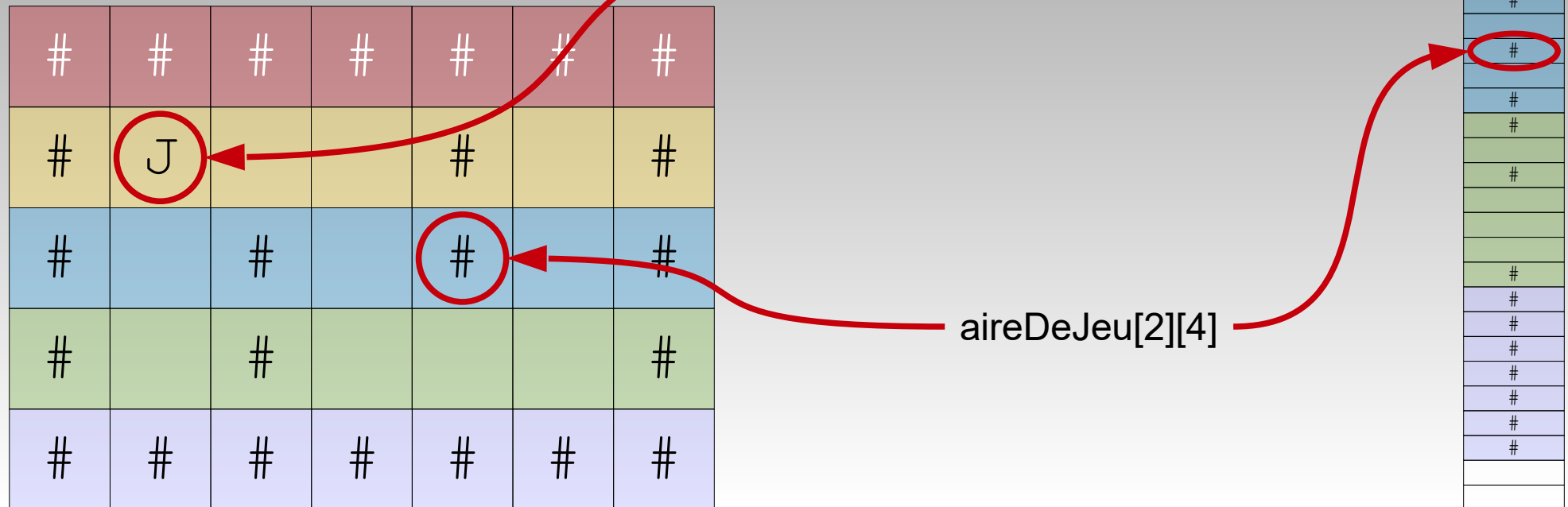
Le stockage en mémoire

En mémoire, on a donc le premier groupe de cases rouges, suivi du deuxième groupe de cases jaunes...



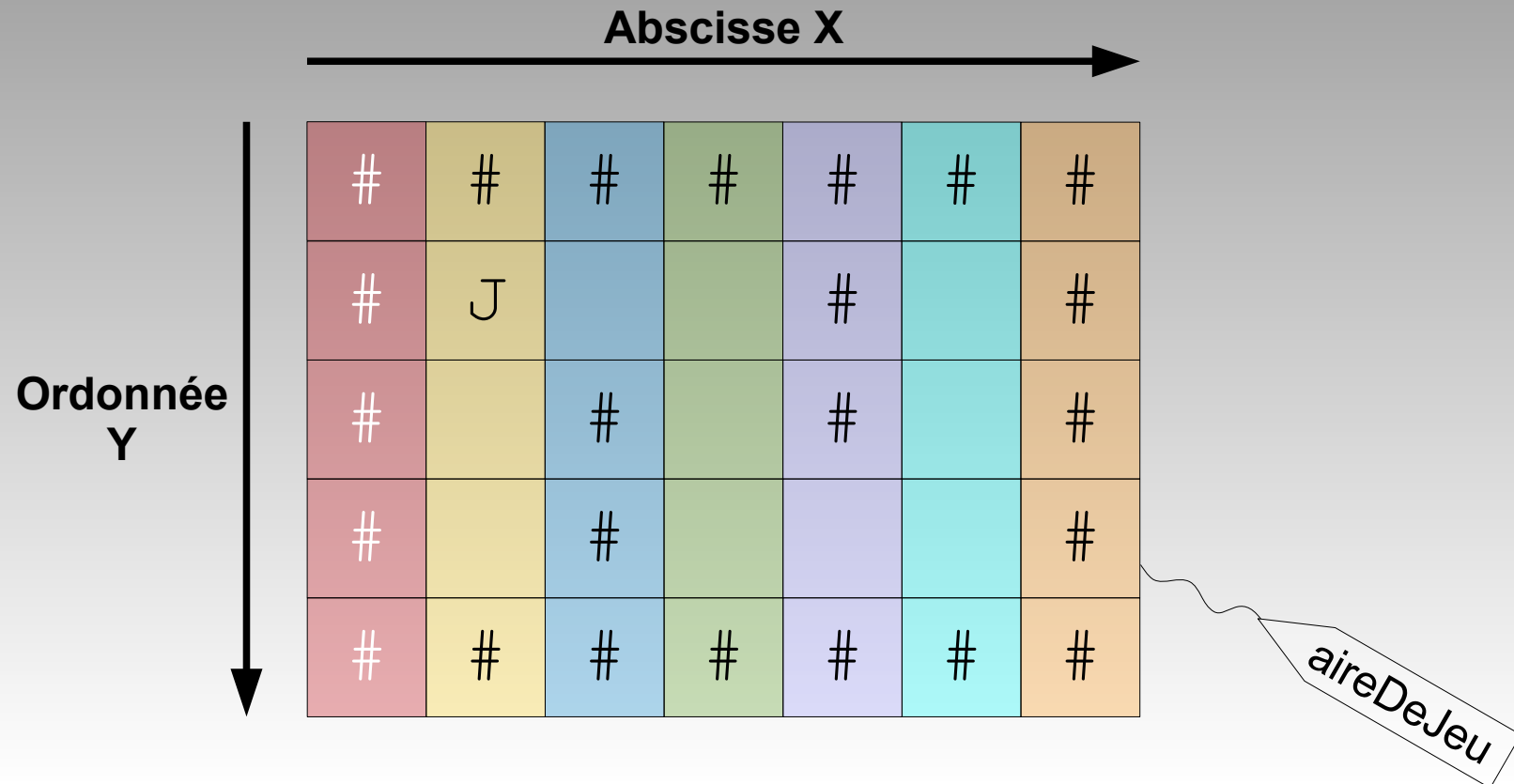
Les deux indices pour désigner une case

Le premier indice (l'ordonnée Y) permet de désigner un groupe de cases et le second indice (l'abscisse X) permet de manipuler une case au sein de ce groupe



Un tableau de colonnes

L'organisation du tableau est laissée à la discrétion du programmeur qui peut opter pour un tableau de colonnes selon ses besoins.



Utilisation des indices dans ces cas

Avec cette nouvelle organisation, le premier indice (l'abscisse X) désigne un groupe de cases et le second indice (l'ordonnée Y) correspond à une case au sein de ce groupe

#	#	#	#	#	#	#
#	J			#		#
#		#		#		#
#		#				#
#	#	#	#	#	#	#

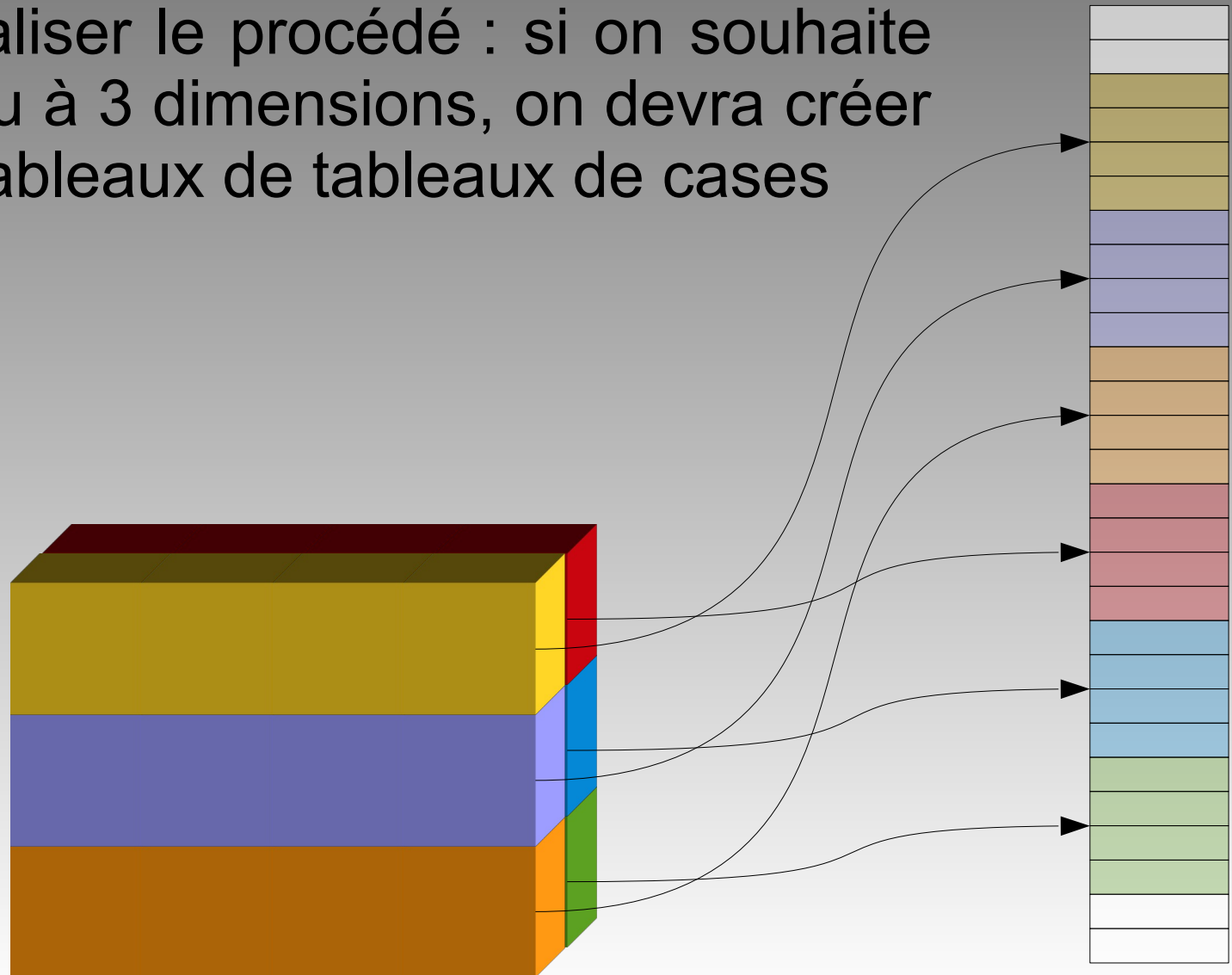
aireDeJeu[1][1]

aireDeJeu[4][2]

#
#
#
#
#
J
#
#
#
#
#
#
#
#
#
#
#
#
#
#
#

Le tableau à 3, 4, 5 ... N dimensions

On peut généraliser le procédé : si on souhaite gérer un tableau à 3 dimensions, on devra créer un tableau de tableaux de tableaux de cases



La création

La création s'effectue en spécifiant le type d'information stockée dans chaque case, suivi du nom du tableau puis plusieurs couples de crochets **[]** (autant de couples que de dimensions). La taille de chaque dimension doit être spécifiée entre les crochets.

```
int main (int argc, char** argv)
{
    int tab1 [10][20];      // tableau d'entiers à 2 dimensions
    double tab2 [3][4][5]; // tableau de double à 3 dimensions
}
```

Le remplissage du tableau à sa création

Le remplissage du tableau peut s'effectuer aussi dès la création en saisissant les différentes valeurs entre accolades.

```
int tab1 [][] = { {12, -2, 40, 24, 0, 0, 0} ,  
                  {12, -2, 40, 24, 0, 0, 0} ,  
                  {12, -2, 40, 24, 0, 0, 0} ,  
                  {12, -2, 40, 24, 0, 0, 0} };  
  
double tab2 [][][] = { { {6.23, 12.7, -40.4564, 10, -68 } ,  
                          {6.23, 12.7, -40.4564, 10, -68 } ,  
                          {6.23, 12.7, -40.4564, 10, -68 } } ,  
  
                        { {6.23, 12.7, -40.4564, 10, -68 } ,  
                          {6.23, 12.7, -40.4564, 10, -68 } ,  
                          {6.23, 12.7, -40.4564, 10, -68 } } ,  
  
                        // ...etc  
  
                        {6.23, 12.7, -40.4564, 10, -68 } } } ;
```

L'opérateur []

... ou de lire une valeur.

```
int i, j, k, l, m;

for (i=0; i<4; i++)
    for (j=0; j<7; j++)
        printf ("La case (%d,%d) du tableau 1 a comme valeur : %d\n",
                i, j, tab1[i][j]);

for (k=0; k<6; k++)
    for (l=0; l<3; l++)
        for (m=0; m<4; m++)
            printf ("La case (%d,%d,%d) du tableau 2 a comme valeur : %.2lf\n",
                    i, j, k, tab2[k][l][m]);
}
```

On remarque l'utilisation de plusieurs boucles imbriquées pour faire évoluer les indices.

L'opérateur []

Une fois le tableau créé, on peut alors utiliser l'opérateur **[]** pour accéder à chaque case afin d'affecter ...

```
int main (int argc, char** argv)
{
    int      tab1  [4][7];
    double   tab2  [6][3][5];

    tab1[0][0]    = 12;
    tab1[1][6]    = -2;
    tab1[2][2]    = 40;
    tab1[3][1]    = 24;
    tab2[0][1][2] =  6.23;
    tab2[1][0][3] = 12.7;
    tab2[2][2][2] = -40.4564;
    tab2[3][0][0] = 10;
    tab2[4][2][1] = -68;
    tab2[5][1][1] = 17.45;    // ...etc
}
```

L'opérateur []

... ou de lire une valeur.

```
int i, j, k, l, m;

for (i=0; i<4; i++)
    for (j=0; j<7; j++)
        printf ("La case (%d,%d) du tableau 1 a comme valeur : %d\n",
                i, j, tab1[i][j]);

for (k=0; k<6; k++)
    for (l=0; l<3; l++)
        for (m=0; m<4; m++)
            printf ("La case (%d,%d,%d) du tableau 2 a comme valeur : %.2lf\n",
                    i, j, k, tab2[k][l][m]);
}
```

On remarque l'utilisation de plusieurs boucles imbriquées pour faire évoluer les indices.

Les frais kilométriques

Écrire un programme en C qui permet à une personne de calculer ses frais kilométriques selon le barème ci-dessous.

Puissance fiscale	Jusqu'à 5000 km	De 5001 à 20000 km	Au-delà de 20000 km
3 CV et moins	$d \times 0.451$	$(d \times 0.270) + 906$	$d \times 0.315$
4 CV	$d \times 0.518$	$(d \times 0.291) + 1136$	$d \times 0.349$
5 CV	$d \times 0.543$	$(d \times 0.305) + 1188$	$d \times 0.364$
6 CV	$d \times 0.568$	$(d \times 0.32) + 1244$	$d \times 0.382$
7 CV et plus	$d \times 0.595$	$(d \times 0.337) + 1288$	$d \times 0.401$

Les calculs matriciels

Écrire un programme qui manipule 4 matrices de 4x4 éléments de type double :

- le contenu des matrices `matA` et `matB` est laissé au libre choix du programmeur (saisie, valeur en dur...) ;
- la matrice `matC` est définie comme étant la somme de `matA` et `matB` ;
- la matrice `matD` est définie comme la multiplication de `matA` et `matB`.

Le programme doit donc :

- calculer le contenu de `matC` et `matD` ;
- afficher de façon tabulée le contenu des 4 matrices dans la console.

Le cas des chaînes de caractères



Un tableau de caractères

Une chaîne de caractères est simplement un tableau de caractères terminé par un symbole "invisible" qui joue le rôle de caractère de fin (le "zéro terminal", noté `\0`).

La chaîne de caractères est écrite entre guillemets (le caractère invisible est alors sous-entendu).

En tant que tableau de caractères, on peut donc accéder aux caractères de façon individuelle.

La manipulation des chaînes de caractères est importante car elles sont utilisées dans divers domaines :

- le compilateur analyse un code source à la recherche de mots-clés ;
- un programme analyse les informations saisies par l'utilisateur pour vérifier qu'elles sont correctes (par exemple, la forme d'une adresse email) ...

Deux manières de créer une chaîne

On peut créer une chaînes de caractères en utilisant deux syntaxes différentes :

```
char   chaine2 [] = "Bonjour !";  
char*  chaine1    = "Bonjour !";
```

Il existe cependant une nuance importante entre les deux :

- la première chaîne peut être modifiée (on peut changer un caractère par un autre) ;
- la second chaîne ne peut être modifiée (une telle tentative entraîne une « erreur de segmentation »)

Le « \0 » terminal

Une chaîne écrite entre guillemets comporte un caractère caché qui joue le rôle de délimiteur de fin de chaîne : le « \0 » dont le code ASCII est simplement 0.

Il faut donc tenir compte de ce caractère lorsqu'on doit réserver de la place en mémoire pour stocker une chaîne.

Ce caractère de fin est utilisé par la plupart des fonctions du module `string`.

Tableau de chaîne de caractères

On peut aussi créer un tableau de chaînes de caractères (ou passer un tableau de chaînes de caractères en paramètre d'une fonction, comme c'est le cas pour `main`).

Exemple :

```
char* tab [] = { "#####",  
                 "#       # #",  
                 "# # # #",  
                 "# #   #",  
                 "#####" } ;
```


La fonction printf et le format %

Pour afficher une chaîne de caractères, on utilise le format « %s » comme le montre l'exemple ci-dessous:

```
printf ("ch1=%s\n", ch1);
```

On peut réserver un espace, exprimé en nombre de caractères en mettant cette information entre le % et le s :

```
printf ("ch1=%40s\n", ch1); // placé à droite  
printf ("ch1=%-40s\n", ch1); // placé à gauche
```

Afficher une partie de la chaîne

On peut aussi n'afficher qu'une partie d'une chaîne en mettant un point, suivi du nombre de caractères à afficher (le tout entre le symbole % et la lettre s) :

```
printf ("ch1=%0.4s\n", ch1);
```

La fonction scanf et le format %s

On utilise le même format pour lire une chaîne de caractères depuis le clavier (la saisie se termine par [Entrée]). Il faut cependant penser à réserver suffisamment de la place en mémoire

Exemple :

```
char ch [30];  
scanf ("%s", ch);
```

Limiter le nombre de caractères

On peut, fort heureusement, imposer une taille limite, en mettant le nombre de caractères entre le % et le s : le `scanf` ignore les caractères écrits en surplus

Exemple :

```
char ch [30];  
scanf ("%29s", ch);
```

Présentation

La bibliothèque `string` regroupe un ensemble de fonctions qui permettent de faciliter le traitement des chaînes de caractères.

Pour pouvoir utiliser ces fonctions, il est nécessaire d'écrire la directive de pré-traitement ci-dessous :

```
#include <string.h>
```

La fonction `strlen`

Cette fonction renvoie la taille d'une chaîne de caractères sans compter le `\0` terminal.

Exemple :

```
char* chaine = "Bonjour !";  
int    longueur = strlen (chaine);
```

Les fonctions `atoi`, `atol` et `atoll`

Cette fonction tente de convertir une chaîne de caractères en un nombre entier de type `int`. Lorsque la fonction ne peut pas effectuer la conversion, elle renvoie 0.

Exemple :

```
printf ("%d\n", atoi("10"));    // renvoie 10
printf ("%d\n", atoi("0x7B")); // renvoie  0
printf ("%d\n", atoi("toto")); // renvoie  0
```

`atol` et `atoll` effectuent le même traitement mais renvoie une information de type `long` ou `long long`.

La fonction atof

Cette fonction tente de convertir une chaîne de caractères en un nombre flottant de type `double`. Si elle n'y parvient pas, elle renvoie 0.0.

Exemple :

```
printf("%f\n", atof("12.45")); // renvoie 12.45
printf("%f\n", atof("1.45E4")); // renvoie 14500
printf("%f\n", atof("toto")); // renvoie 0
```


La fonction strcat

Cette fonction prend en paramètre deux chaînes de caractères. Elle permet de concaténer les deux chaînes en modifiant le contenu du premier paramètre

Exemple :

```
char  ch1 [50] = "Bonjour";  
char* ch2      = " Smurtz";
```

```
printf ("Chaine 1 avant la concatenation = %s\n", ch1);  
printf ("Chaine 2 avant la concatenation = %s\n", ch2);
```

```
strcat (ch1, ch2);
```

```
printf ("Chaine 1 apres la concatenation = %s\n", ch1);  
printf ("Chaine 2 apres la concatenation = %s\n", ch2);
```

La fonction `strncat`

Cette fonction est une variante de la fonction précédente. Elle prend deux chaînes de caractères et un entier `n`. Elle ajoute à la première chaîne, au plus `n` caractères de la seconde chaîne.

Exemple :

```
char  ch1 [50] = "Bonjour";  
char* ch2      = " Smurtz";
```

```
printf ("Chaine 1 avant la concatenation = %s\n", ch1);  
printf ("Chaine 2 avant la concatenation = %s\n", ch2);
```

```
strncat (ch1, ch2, 3);
```

```
printf ("Chaine 1 apres la concatenation = %s\n", ch1);  
printf ("Chaine 2 apres la concatenation = %s\n", ch2);
```

La fonction strcpy

La fonction `strcpy` prend deux chaînes de caractères en paramètre et elle permet de copier la seconde dans la première (remplace les caractères de même rang).

Exemple :

```
char  ch1 [50] = "Bonjour";  
char* ch2      = " Smurtz";
```

```
printf ("Chaine 1 avant la concatenation = %s\n", ch1);  
printf ("Chaine 2 avant la concatenation = %s\n", ch2);
```

```
strcpy (ch1, ch2);
```

```
printf ("Chaine 1 apres la concatenation = %s\n", ch1);  
printf ("Chaine 2 apres la concatenation = %s\n", ch2);
```

La fonction `strncpy`

`strncpy` est une variante qui prend un troisième de type `int`, permettant de spécifier le nombre de caractères qui doivent être au plus copiés

Exemple :

```
char  ch1 [50] = "Bonjour";  
char* ch2      = " Smurtz";
```

```
printf ("Chaine 1 avant la concatenation = %s\n", ch1);  
printf ("Chaine 2 avant la concatenation = %s\n", ch2);
```

```
strncpy (ch1, ch2, 3);
```

```
printf ("Chaine 1 apres la concatenation = %s\n", ch1);  
printf ("Chaine 2 apres la concatenation = %s\n", ch2);
```

La fonction `strcmp`

La fonction `strcmp` prend deux chaînes de caractères en paramètre et elle permet de faire une comparaison lexicographique entre les deux chaînes (on compare caractère par caractère).

Cette fonction renvoie :

- 0 si les deux chaînes sont identiques ;
- une valeur positive si la première chaîne est plus « grande » que la seconde ;
- une valeur négative si la seconde chaîne est plus « grande » que la première.

La fonction strcmp

Exemple :

```
char* ch1 = "ABC";  
char* ch2 = "ABCD";  
char* ch3 = "BBC";
```

```
printf ("strcmp (ch1, ch2) : %d\n", strcmp (ch1, ch2));  
printf ("strcmp (ch1, ch3) : %d\n", strcmp (ch1, ch3));
```

D'autres fonctions intéressantes

Il existe d'autres fonctions intéressantes mais elles seront présentées plus tard, lorsque la notion de pointeurs aura été présentée :

- `strstr` pour trouver une chaîne de caractères dans une autre ;
- `index` et `rindex` pour trouver une occurrence d'un caractère dans une chaîne;
- `strtok` pour découper une chaîne de caractères...

Compter les lettres

Ecrire un programme qui affiche dans un tableau la fréquence des symboles qui composent la chaîne de caractères.

Pour cela, on peut utiliser un tableau d'entiers où :

- l'indice d'une case correspond au code ASCII du symbole
- la valeur de la case correspond à la fréquence

On affiche alors que les symboles ayant une fréquence non nulle.

Remarque : ce travail constitue la première étape pour implémenter le mécanisme de compression de Huffman (qui sera abordé plus tard).

Le tri à bulles



Mélanger de l'eau et de l'huile



On verse de l'eau et de l'huile dans un même récipient et on les mélange.

Mélanger de l'eau et de l'huile

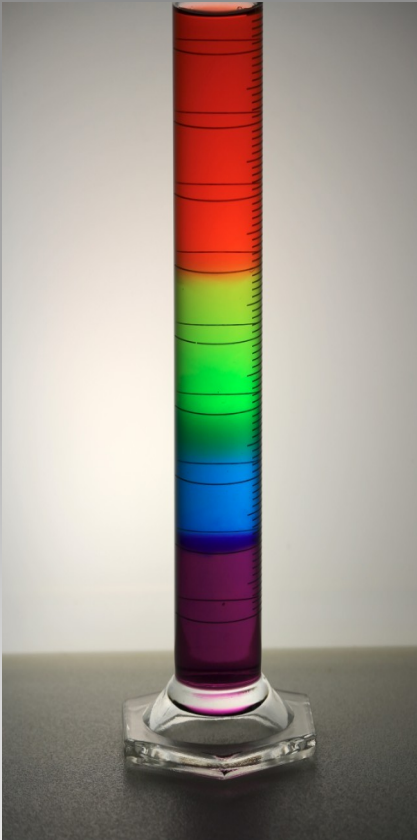


On constate quelques instants après que des bulles d'eau descendent et des bulles d'huile montent car :

- les deux liquides ne sont pas miscibles ;
- l'huile est moins dense que l'eau.

Voir http://emulsion.over-blog.com/pages/PARTIE_I_Realisation_dune_emulsion-2448595.html

Les tours de densités

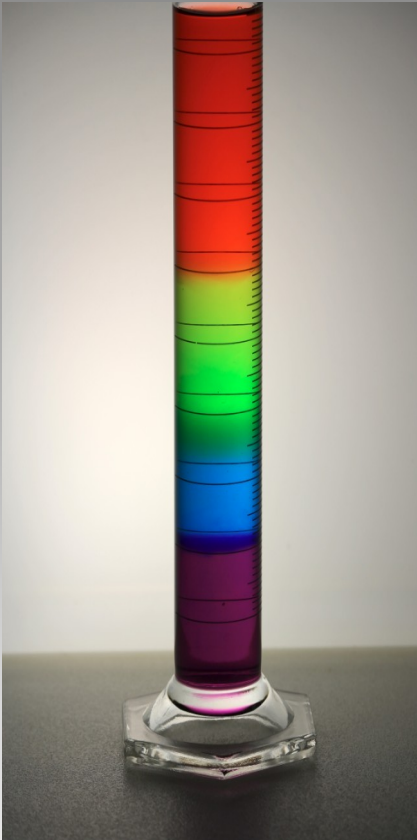


On peut répéter cette expérience avec plusieurs liquide non miscibles qui ont des densités différentes.

On peut alors créer des « tours de densité »

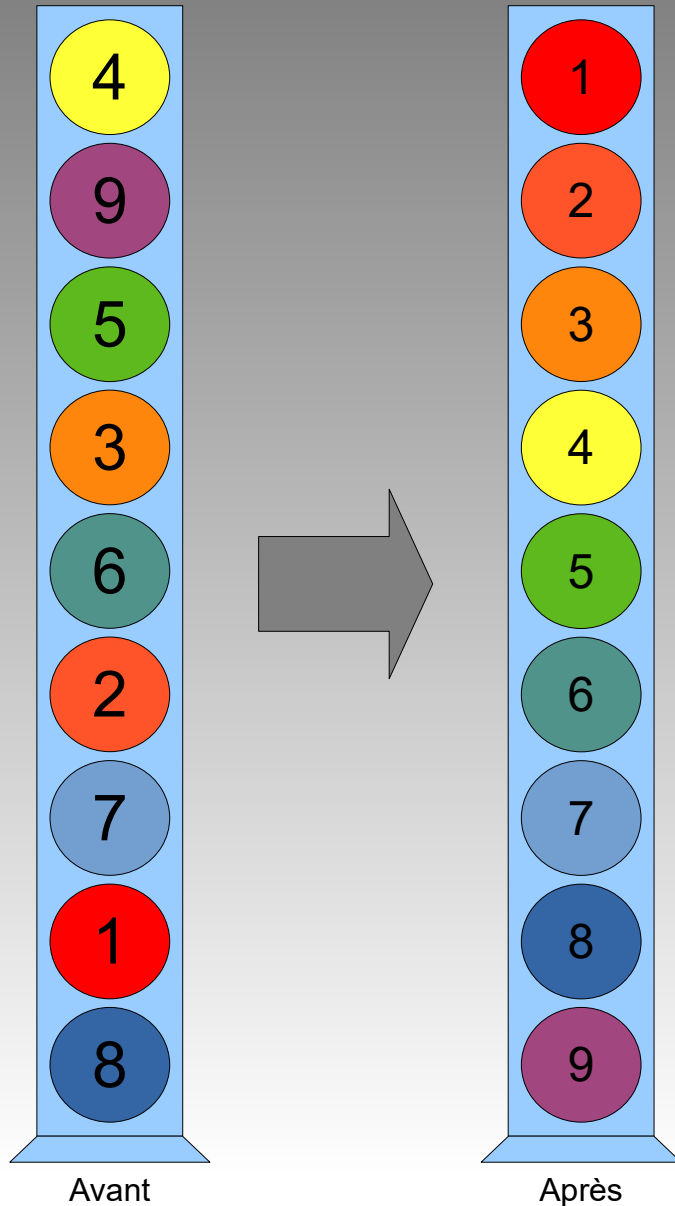
Voir <https://sites.google.com/site/bishopsciencenight/live-demonstrations/liquid-rainbow>

Les tours de densités



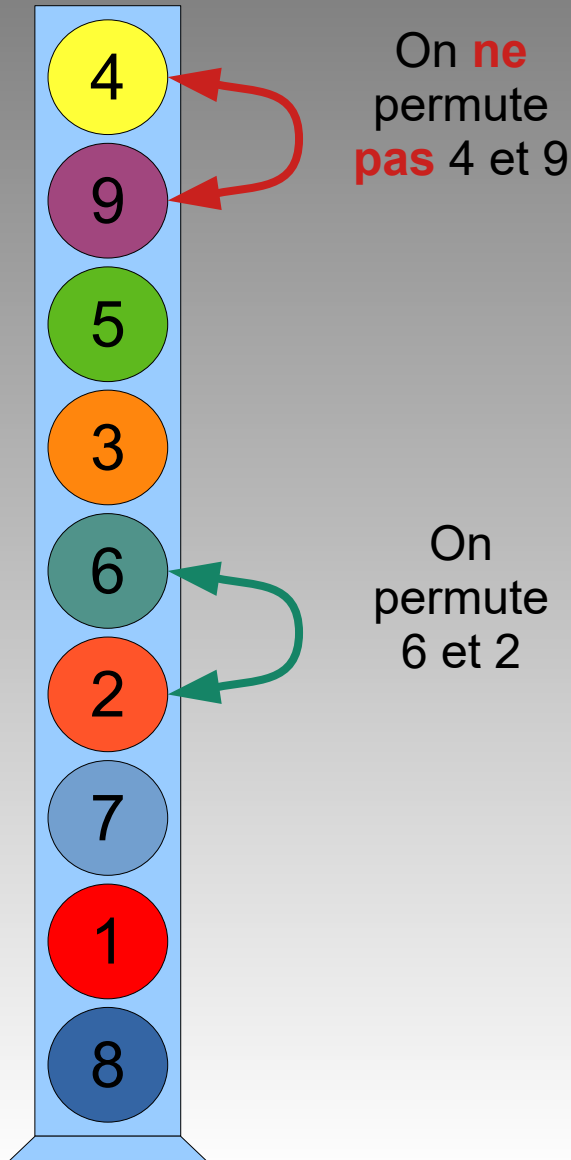
Matériau	Densité
Alcool à friction	0,79
Huile de lampe	0,80
Huile pour bébé	0,83
Huile végétale	0,92
Cube de glace	0,92
Eau	1,00
Lait	1,03
Liquide vaisselle	1,06
Sirop de maïs léger	1,33
Sirop d'érable	1,37
Miel	1,42

Les colonnes de nombres



On peut imaginer le même principe pour une colonne de nombres qui vont se classer en fonction de leur valeur (qui joue le rôle de densité).

La permutation des bulles

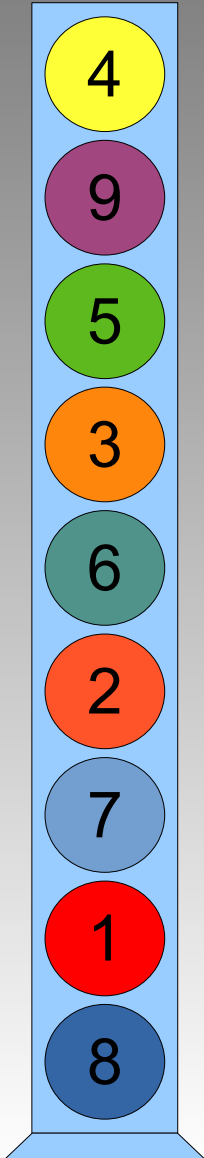


Tous les emplacements de la colonne sont occupées par des bulles.

Par conséquent, leurs déplacements ne peuvent s'effectuer par **permutation** en respectant la contrainte ci-dessous :

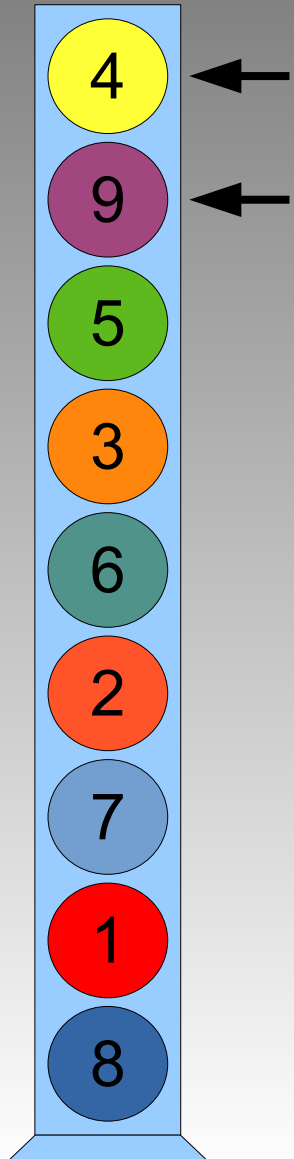
Pour deux bulles données, cette permutation n'est possible que si la bulle située en haut est plus lourde que celle située en bas

Une série de permutation



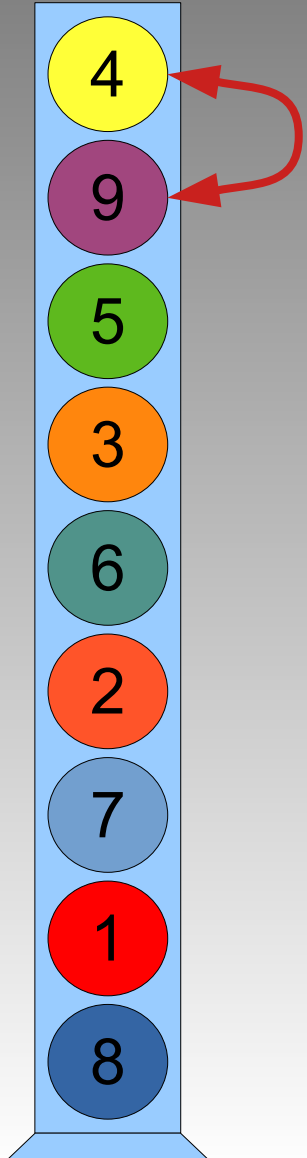
Pour donner l'illusion que les bulles lourdes descendent et que les bulles légères montent, on doit effectuer une série de permutations en partant du haut vers le bas (ou du bas vers le haut).

Une première série de permutation



On commence par la première paire de nombres 4 et 9.

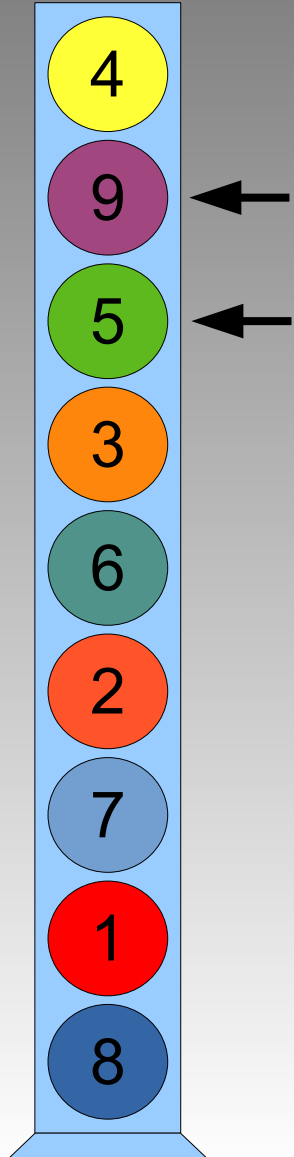
Une première série de permutation



On commence par la première paire de nombres 4 et 9.

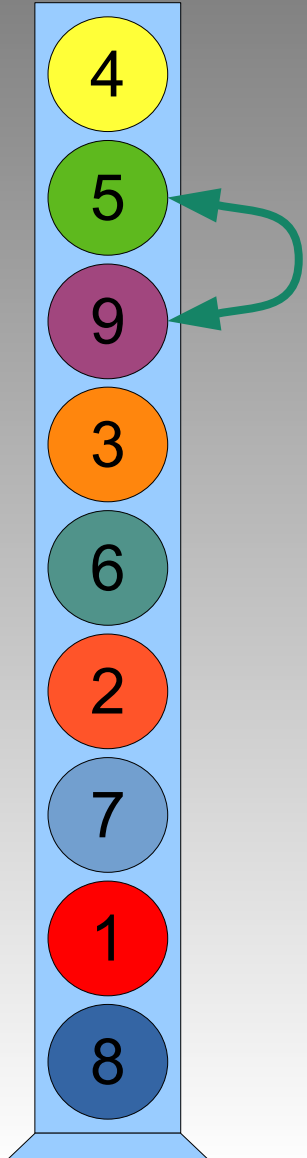
Cette permutation n'a pas lieu car 4, qui est plus « léger », est au dessus de 9.

Une première série de permutation



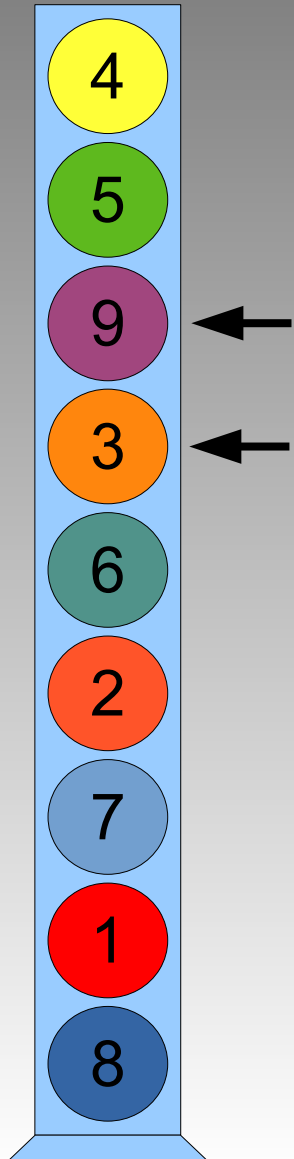
On décale d'un cran vers le bas afin de s'intéresser à la paire de nombres 9 et 5.

Une première série de permutation



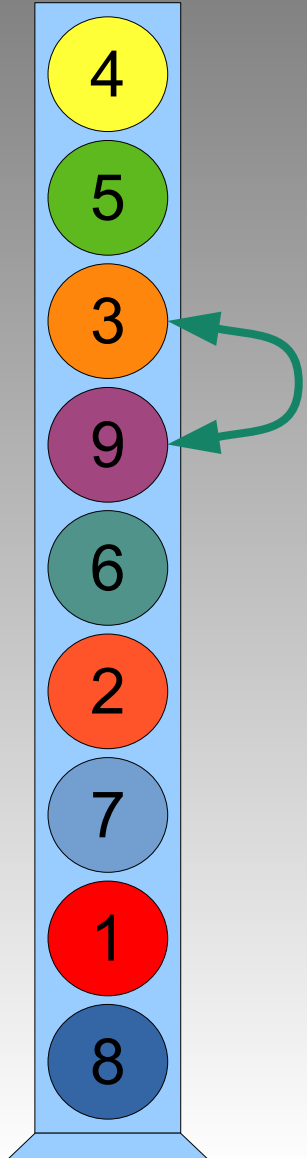
Cette permutation est réalisée car 9, qui est plus « lourd », est au dessus de 5.

Une première série de permutation



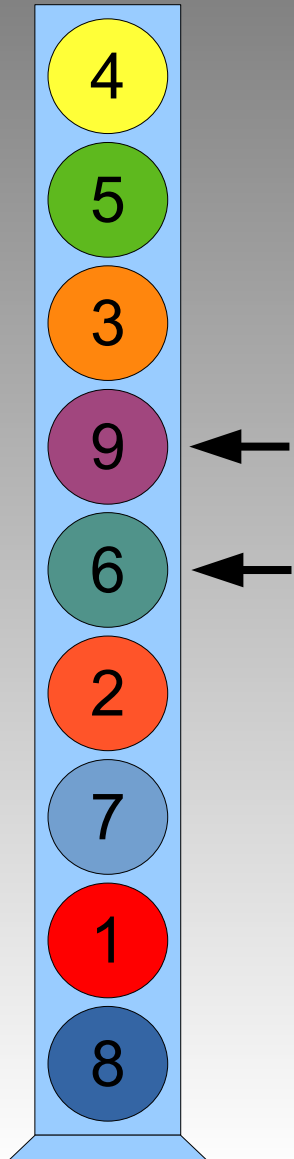
On décale de nouveau d'un cran vers le bas afin de s'intéresser à la paire de nombres 9 et 3.

Une première série de permutation



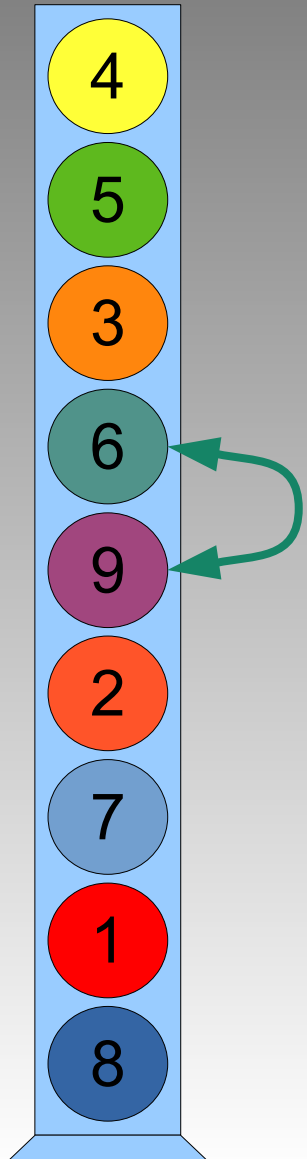
Cette permutation est réalisée car 9, qui est plus « lourd », est au dessus de 3.

Une première série de permutation



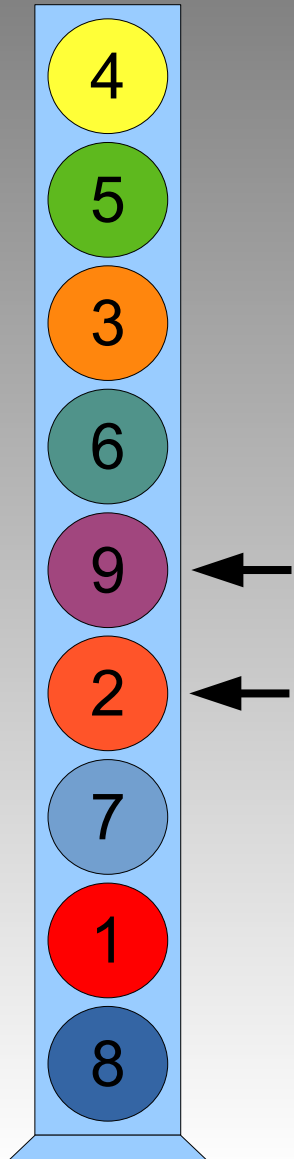
On descend encore d'un cran afin de se focaliser sur le couple de nombres 9 et 6.

Une première série de permutation



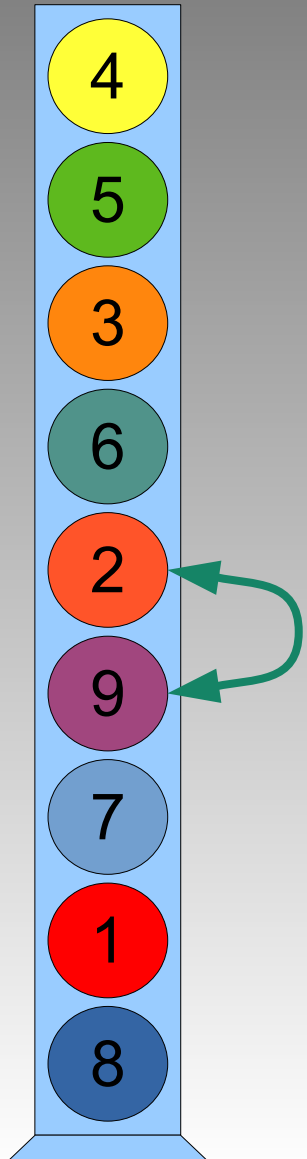
Comme la bulle « 9 » est plus lourde que celle de « 6 », on peut effectuer la permutation.

Une première série de permutation



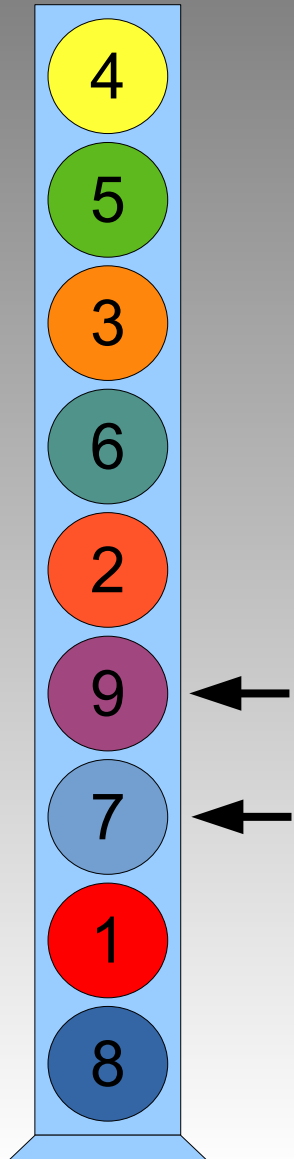
On descend encore d'un cran afin de s'intéresser au couple de nombres 9 et 2.

Une première série de permutation



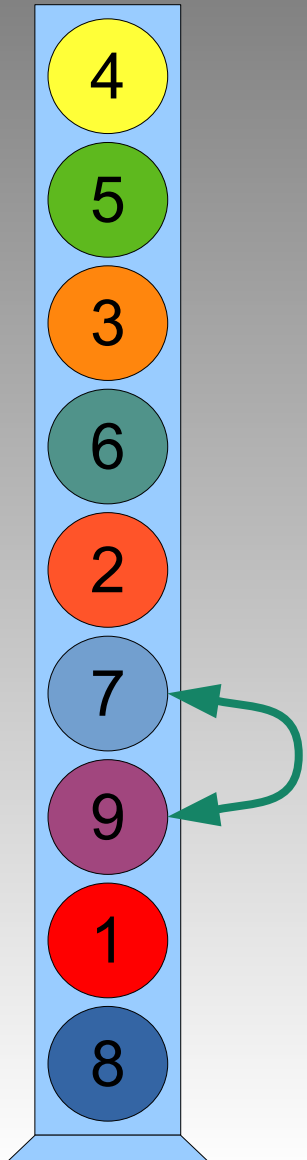
« 9 » étant plus lourd que « 2 », on effectue, cette fois encore, la permutation.

Une première série de permutation



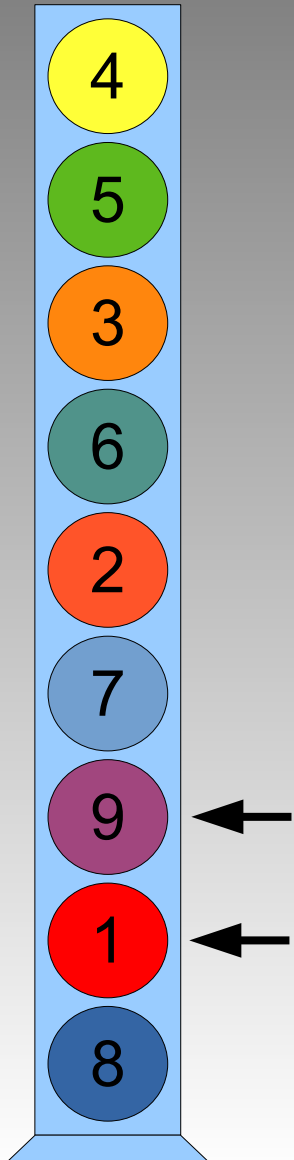
On continue à descendre et on traite la paire de nombres 9 et 7.

Une première série de permutation



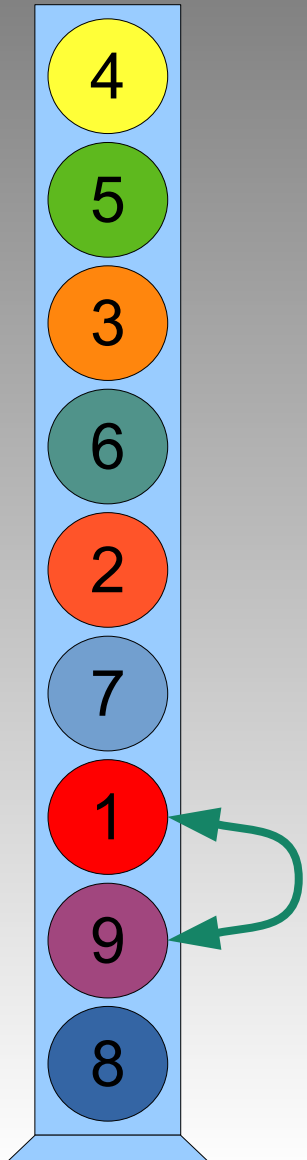
9 étant supérieur à 7, on permute les deux nombres.

Une première série de permutation



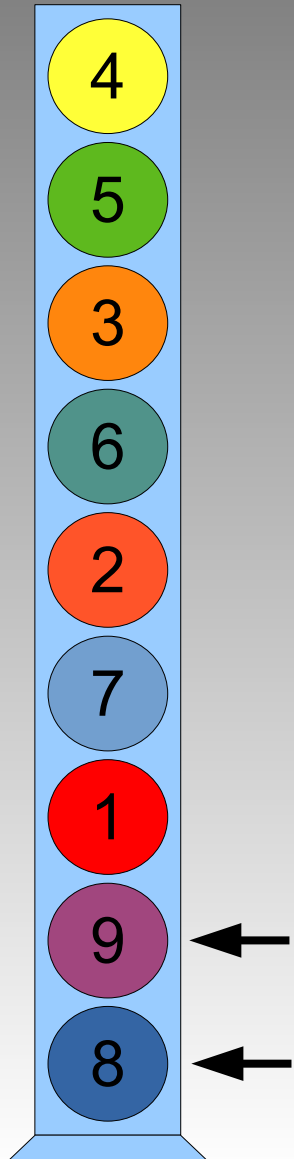
On se focalise sur la paire de nombres 9 et 1.

Une première série de permutation



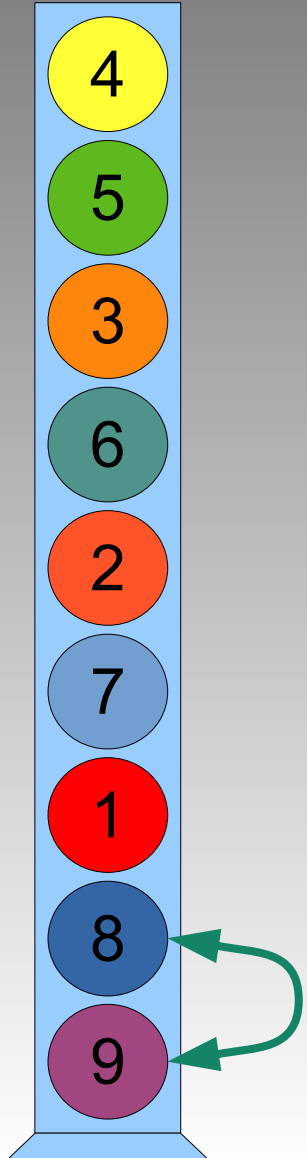
On effectue la permutation car 9 est plus grand que 1.

Une première série de permutation



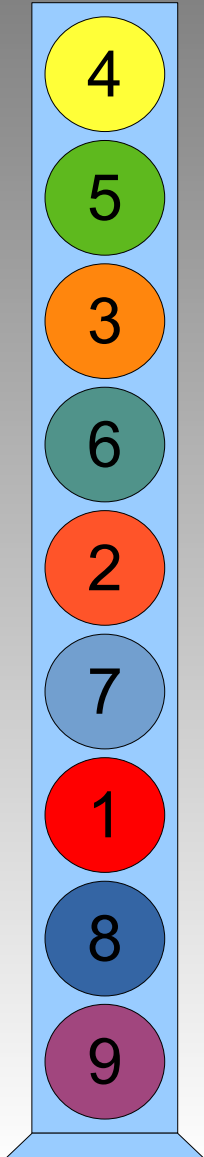
On traite la dernière paire de nombres 9 et 8.

Une première série de permutation



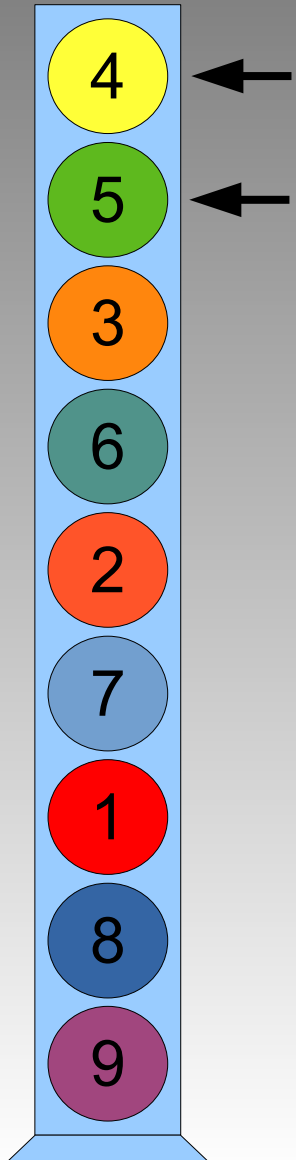
On permute 9 et 8 car 9 est supérieur à 8.

La situation s'est améliorée



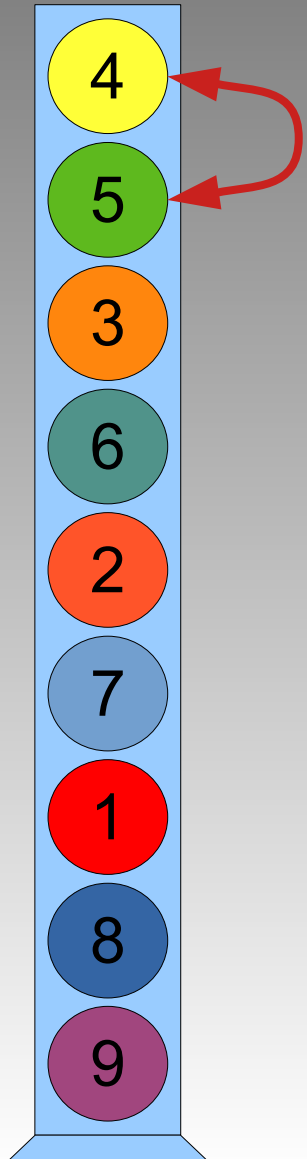
A ce stade, on constate que cette série de permutations a amélioré la situation mais la colonne de nombres n'est pas totalement classée.

La deuxième série de permutations



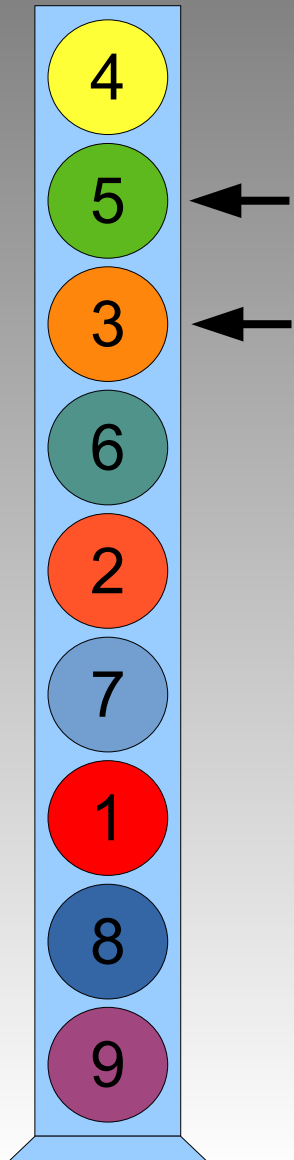
On doit refaire une série de permutations depuis le haut de la colonne : on se focalise donc sur la paire de nombres 4 et 5.

La deuxième série de permutations



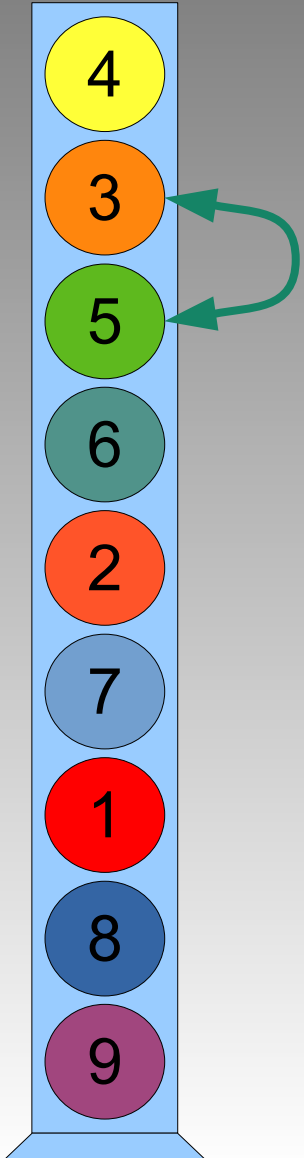
4 étant plus petit (« plus léger ») que 5, la permutation n'a pas lieu.

La deuxième série de permutations



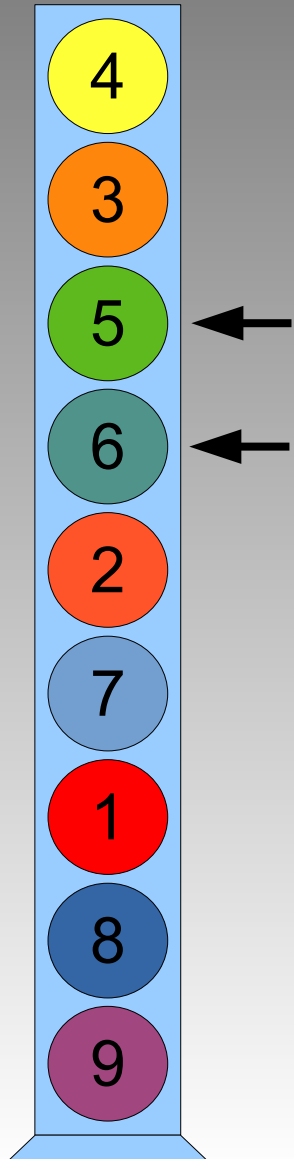
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



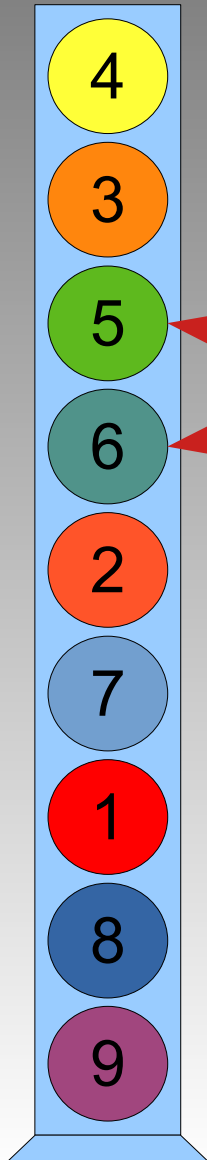
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



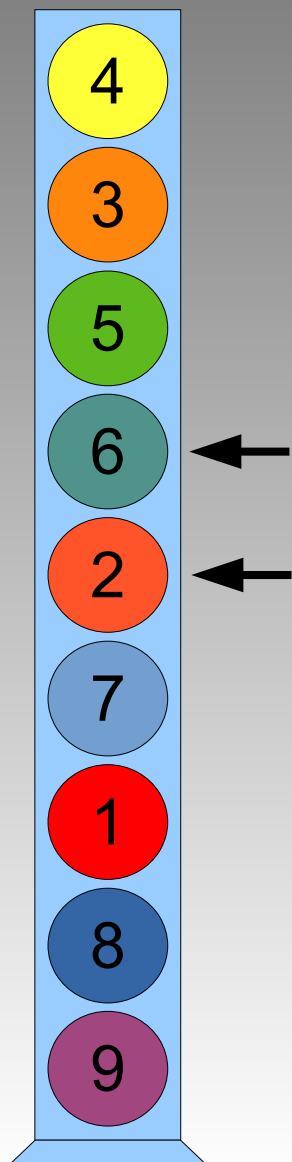
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



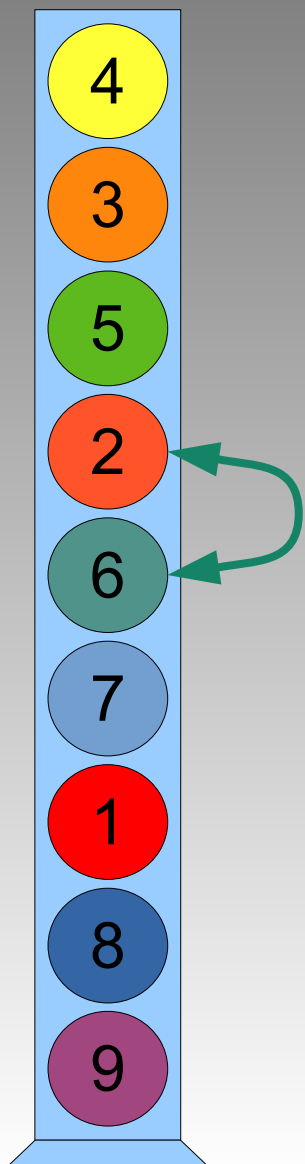
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



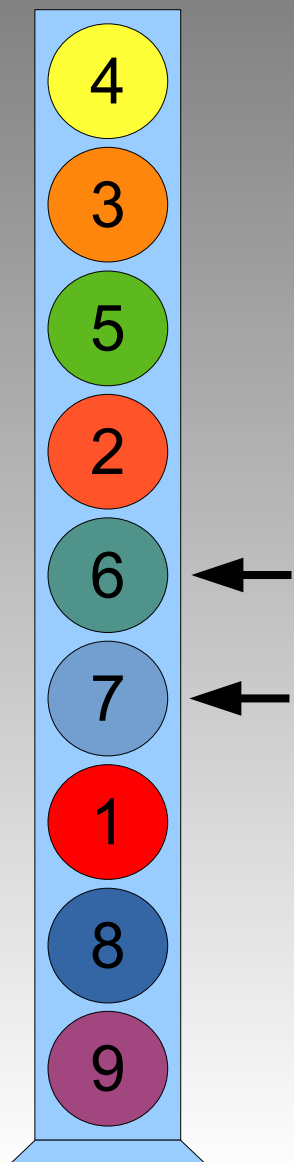
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



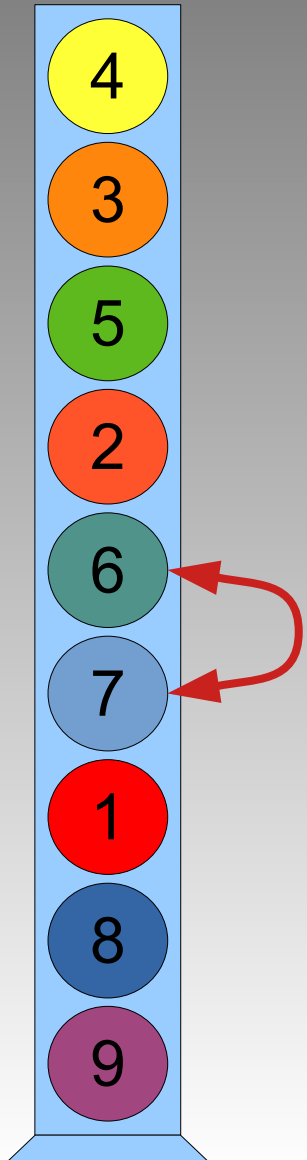
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



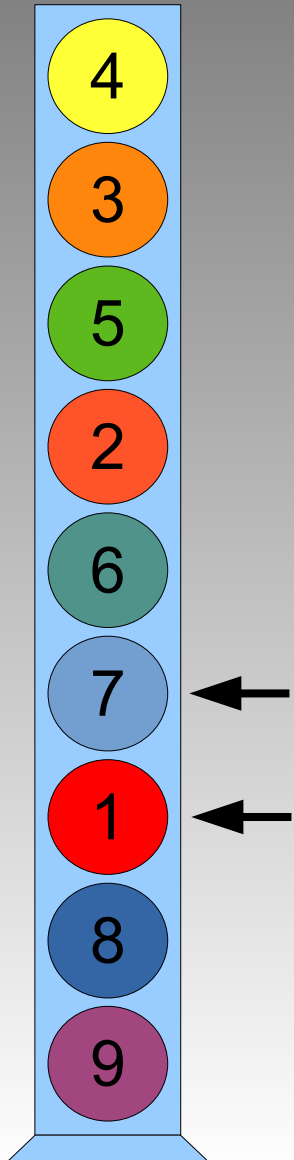
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



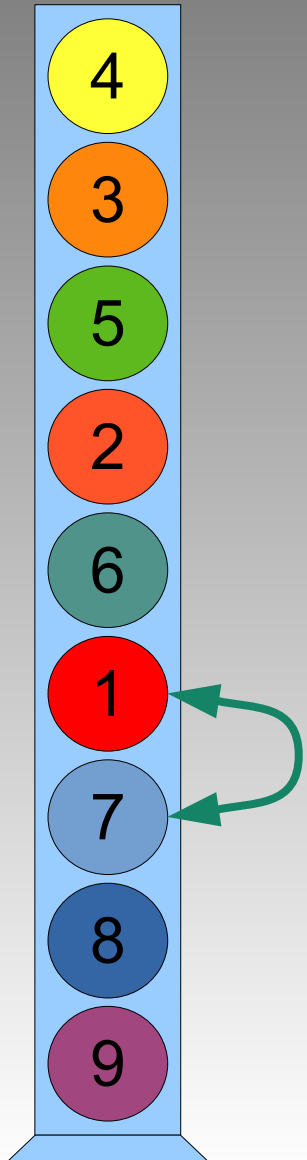
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



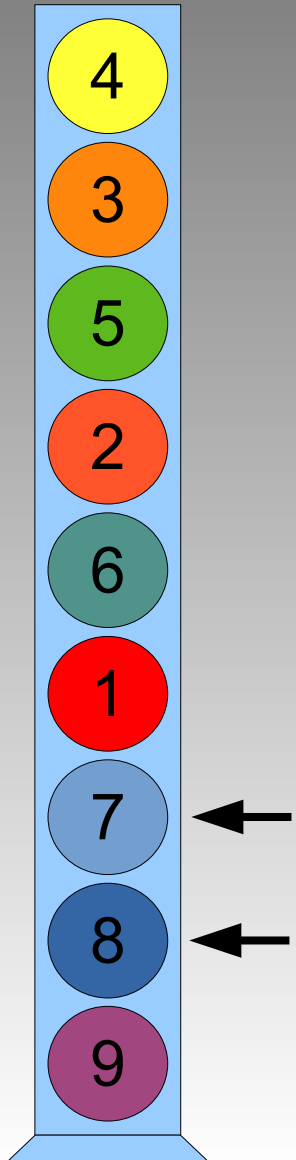
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



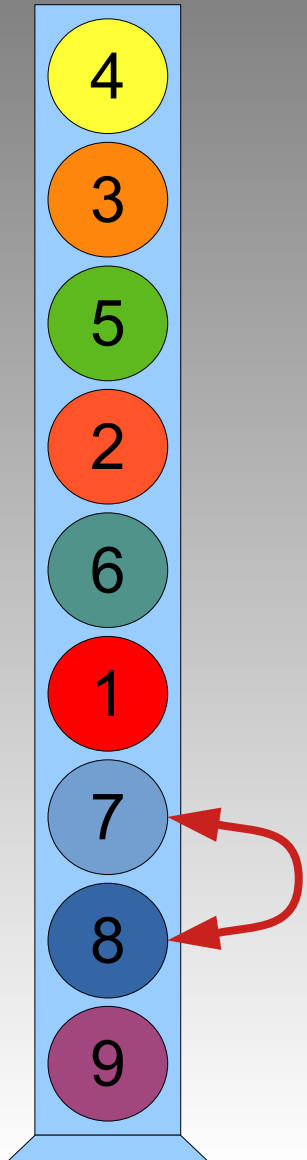
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



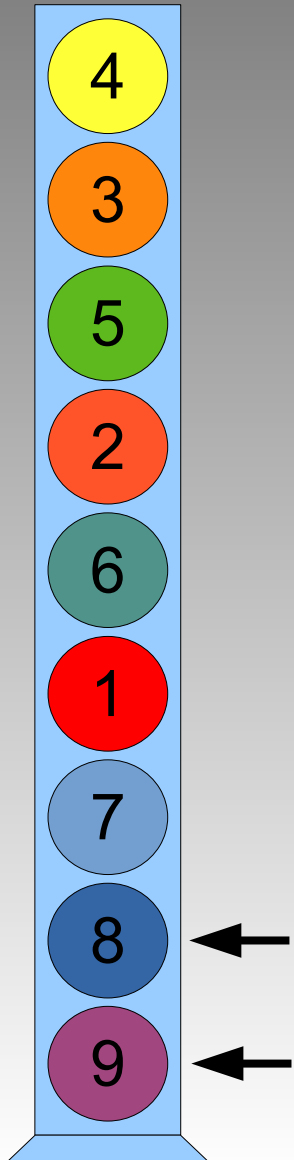
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



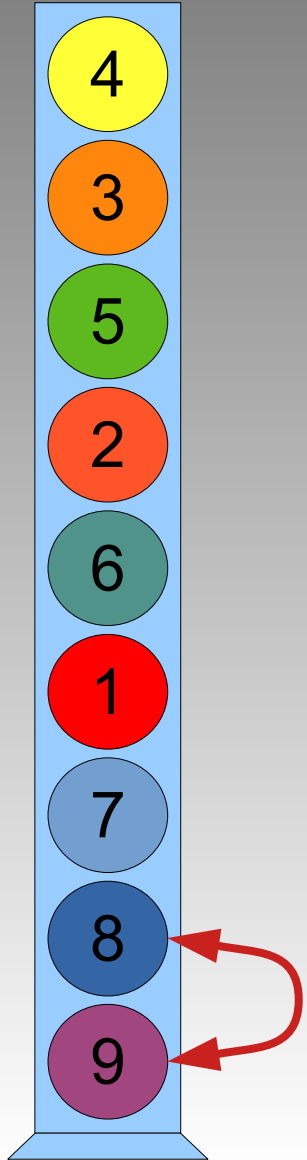
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



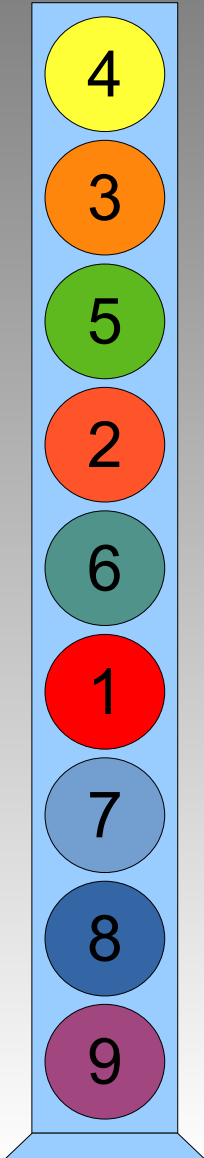
On continue de descendre et on essaye d'effectuer des permutations...

La deuxième série de permutations



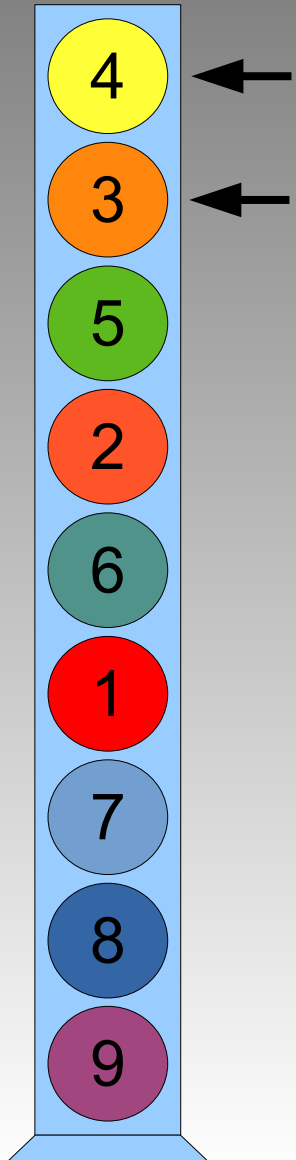
On continue de descendre et on essaye d'effectuer des permutations...

On a, de nouveau, amélioré la situation



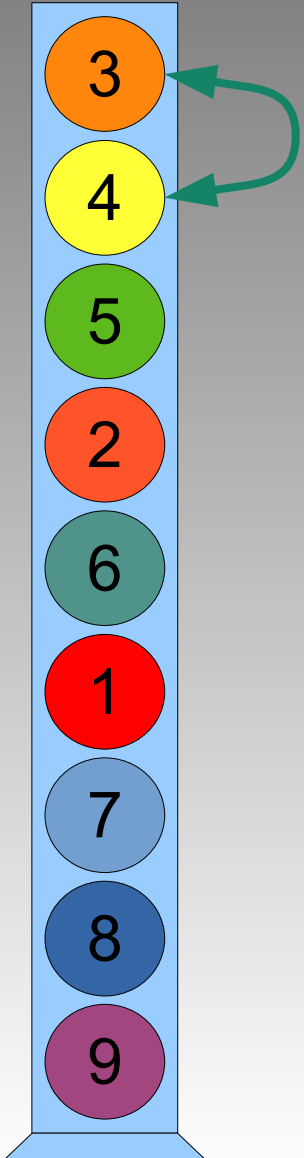
Après cette deuxième série de permutations, la situation s'est, de nouveau, améliorée mais ce n'est pas encore suffisant.

La troisième série de permutations



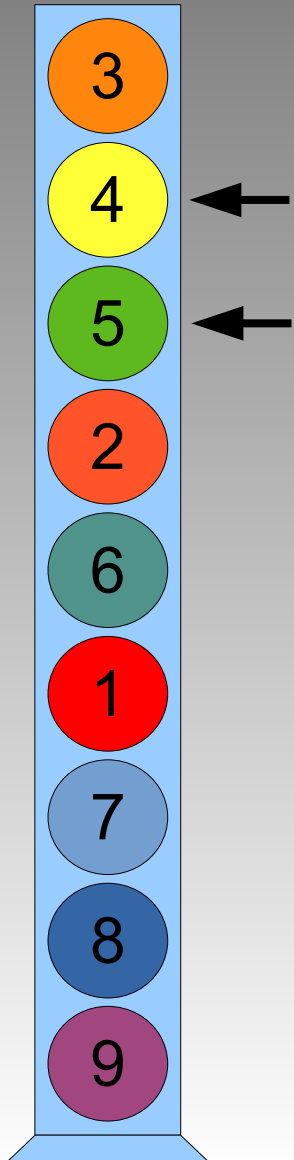
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



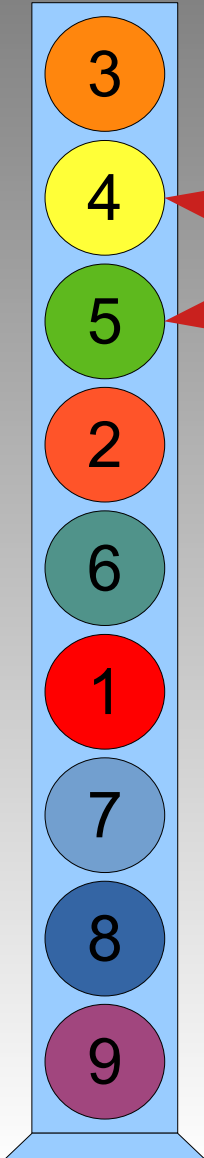
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



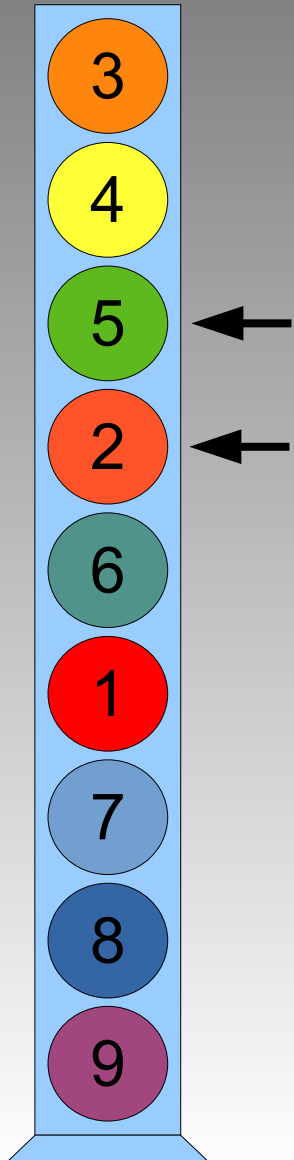
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



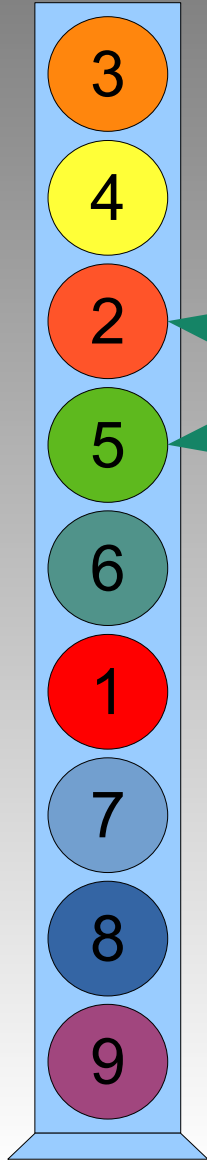
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



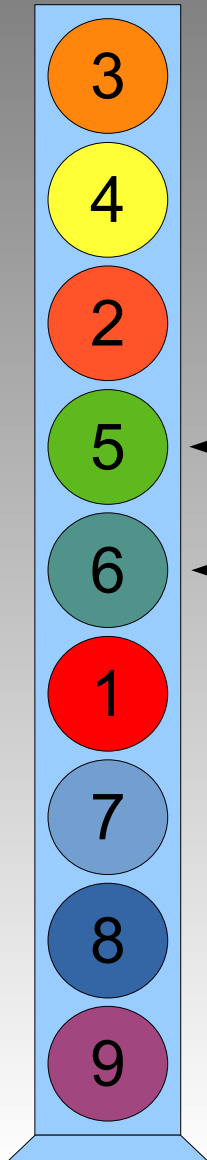
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



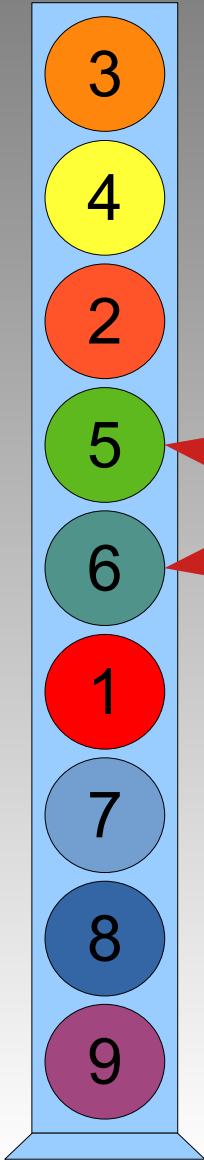
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



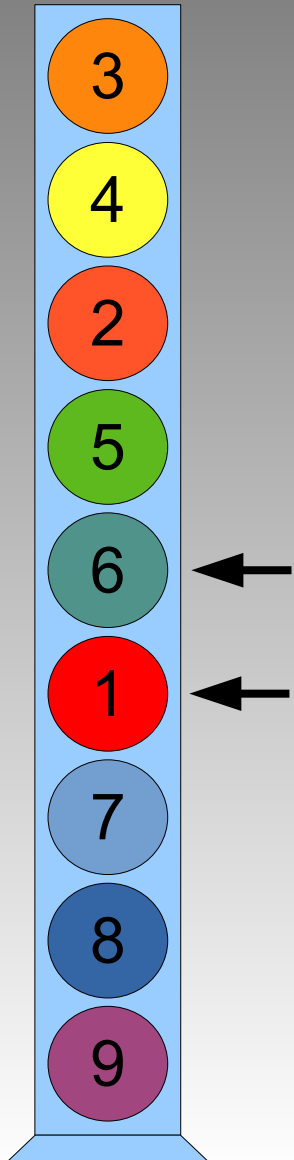
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



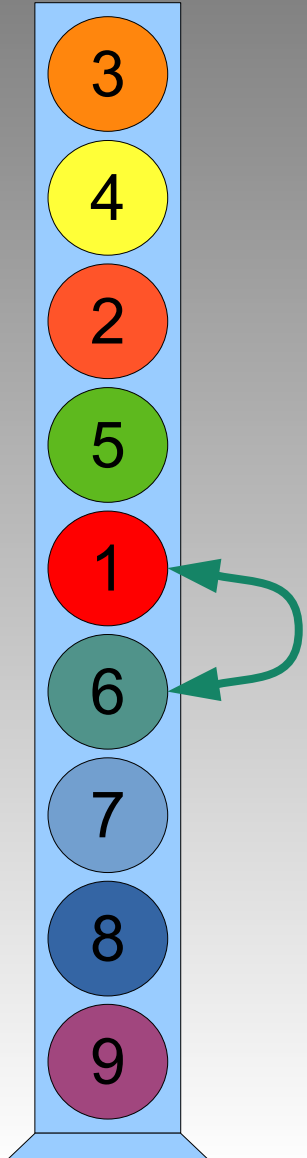
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



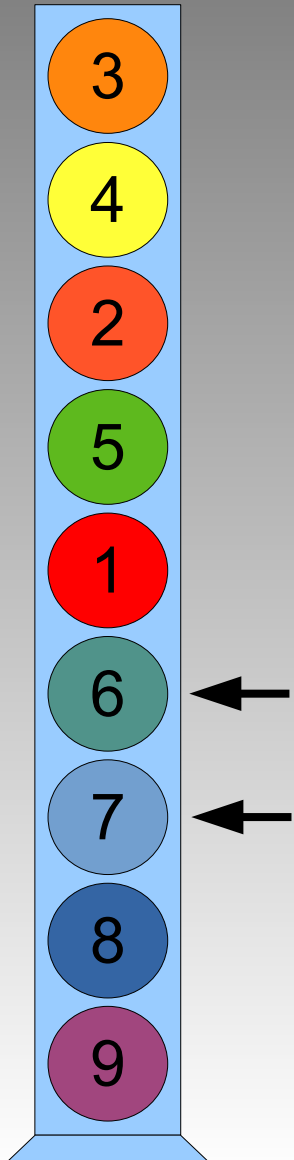
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



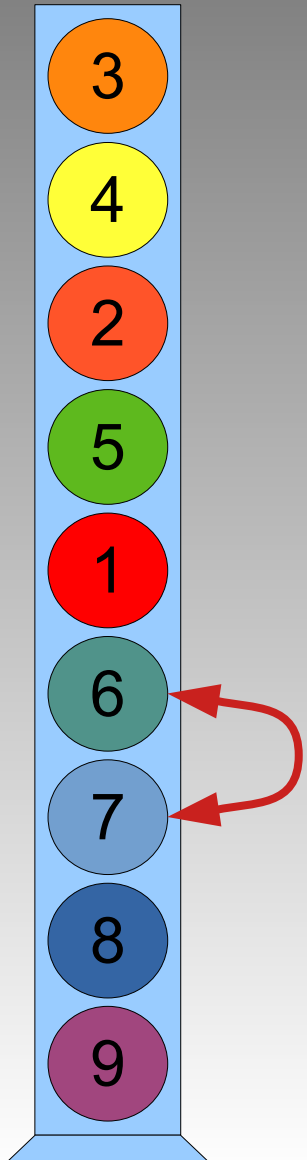
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



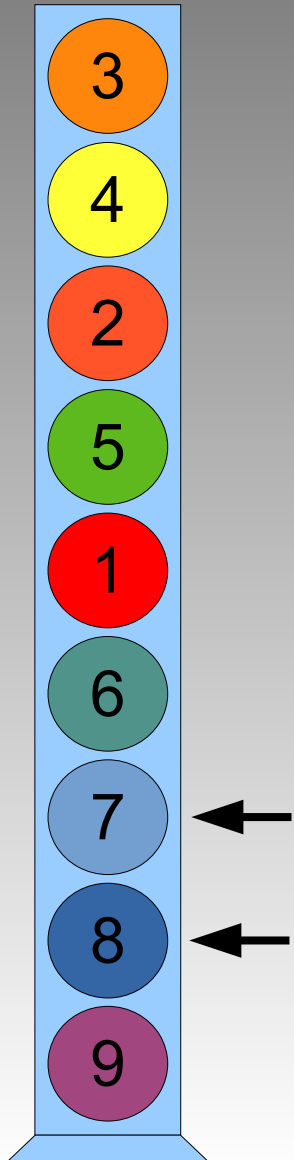
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



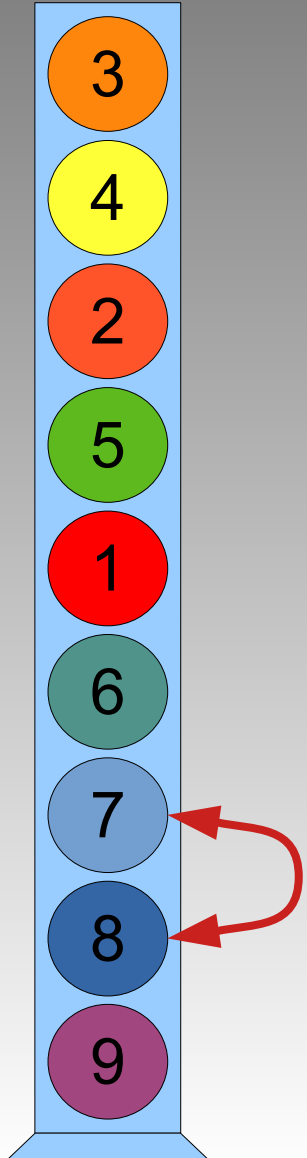
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



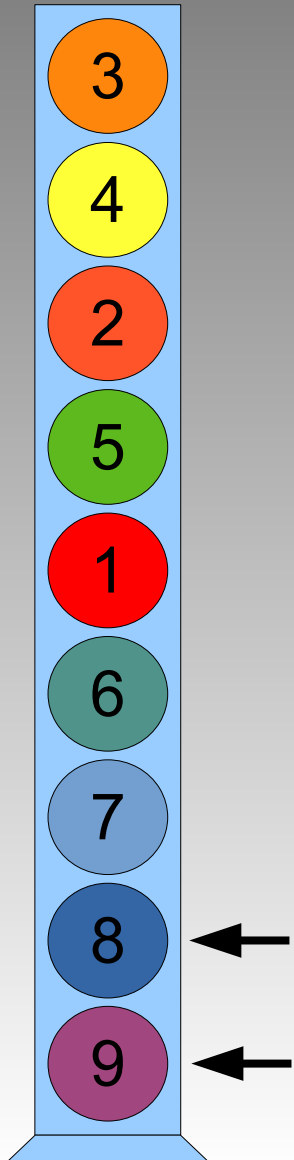
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



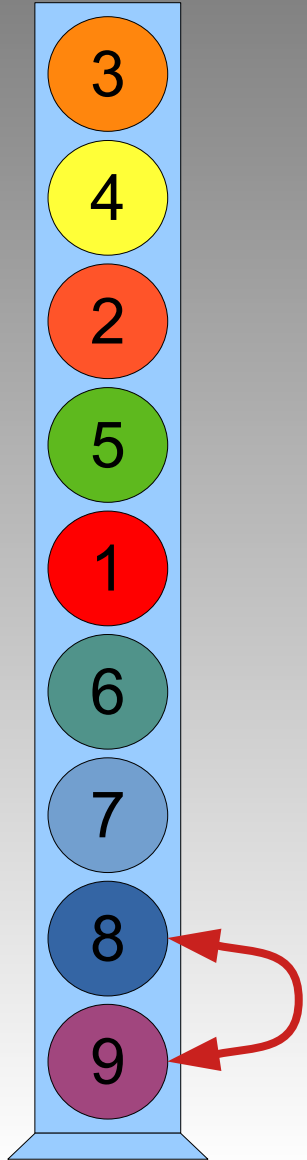
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



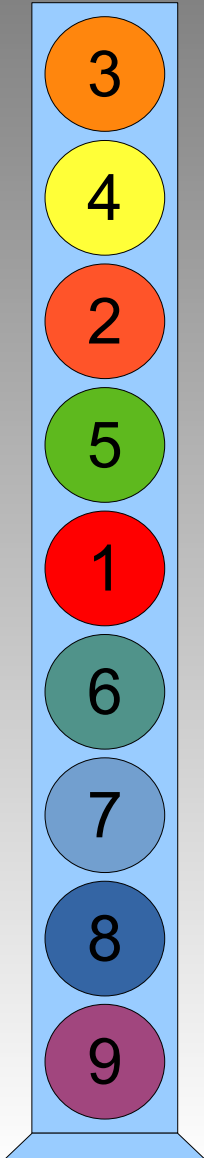
On effectue une 3^{ème} série de permutations.

La troisième série de permutations



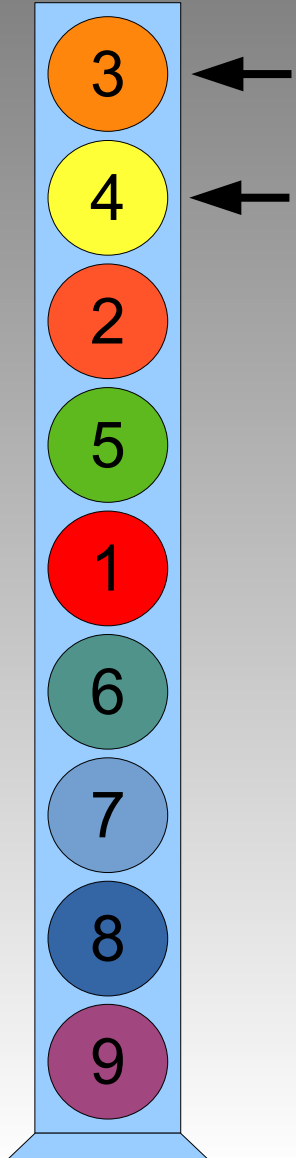
On effectue une 3^{ème} série de permutations.

On se rapproche de l'objectif



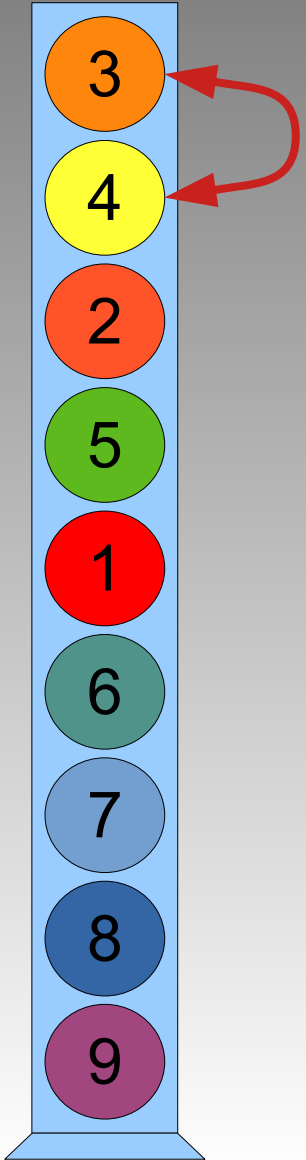
Après cette troisième série de permutations, on s'est rapproché de notre objectif sans l'atteindre.

La quatrième série de permutations



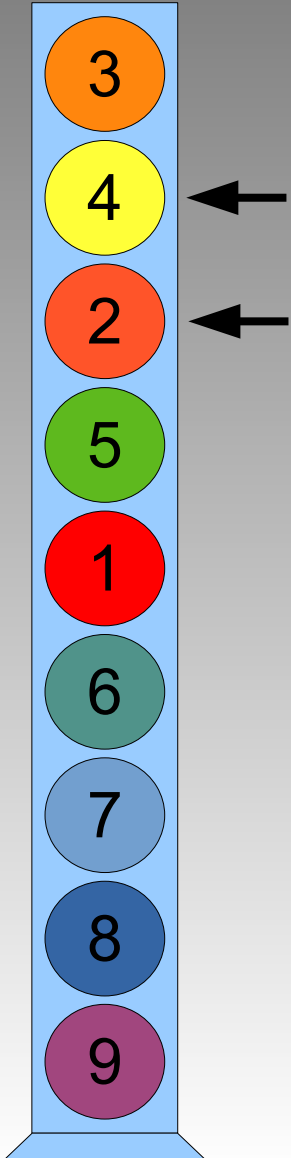
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



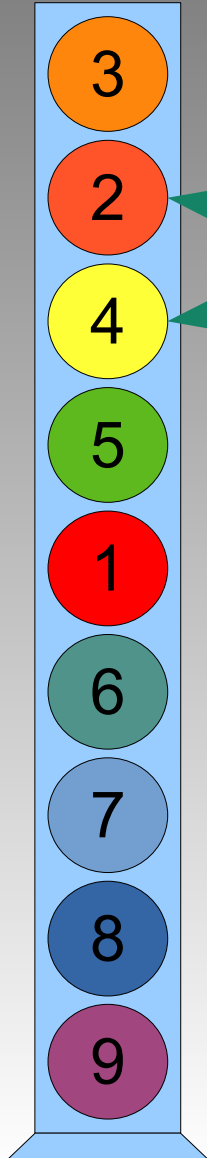
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



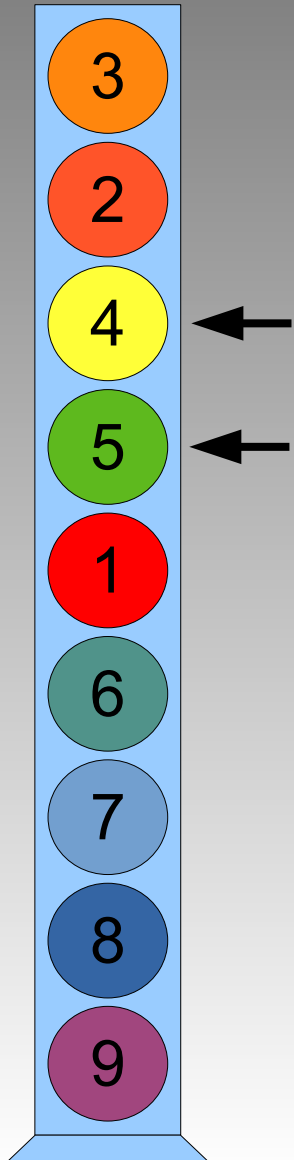
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



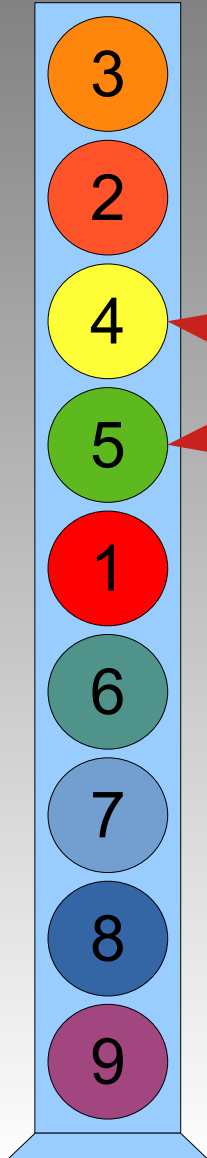
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



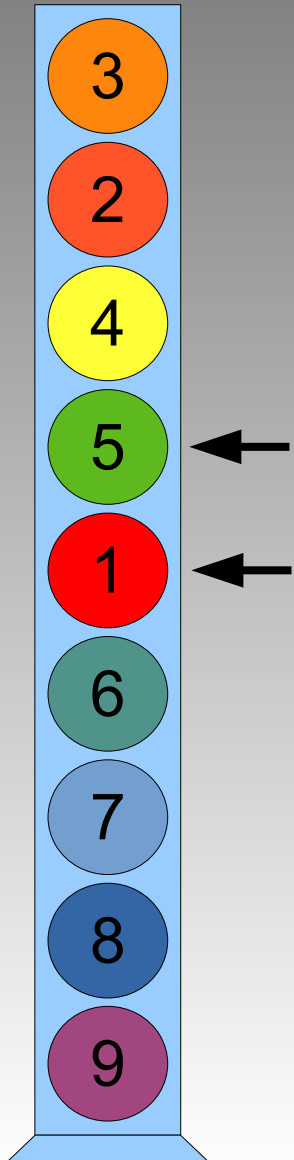
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



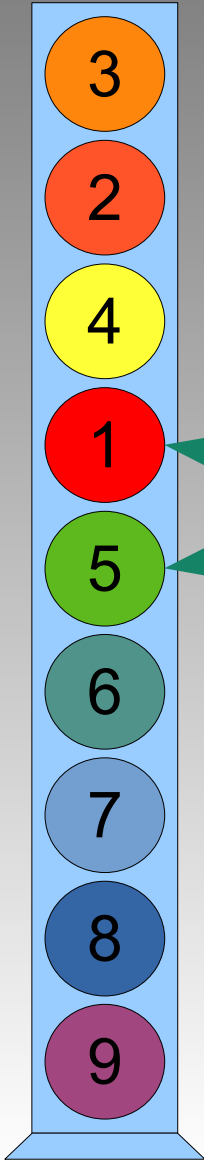
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



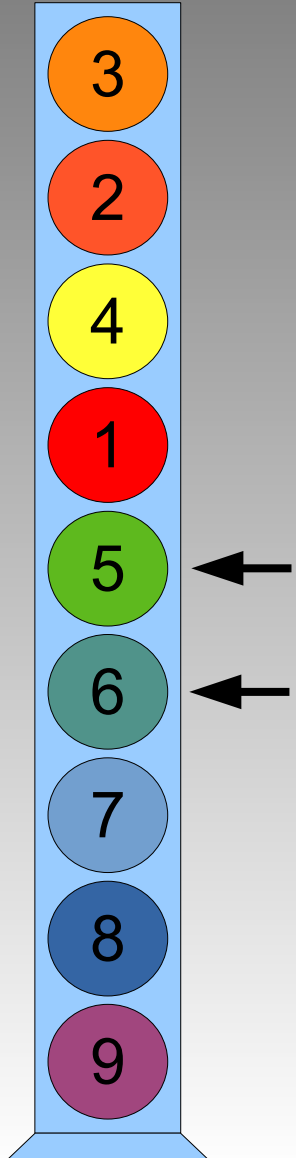
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



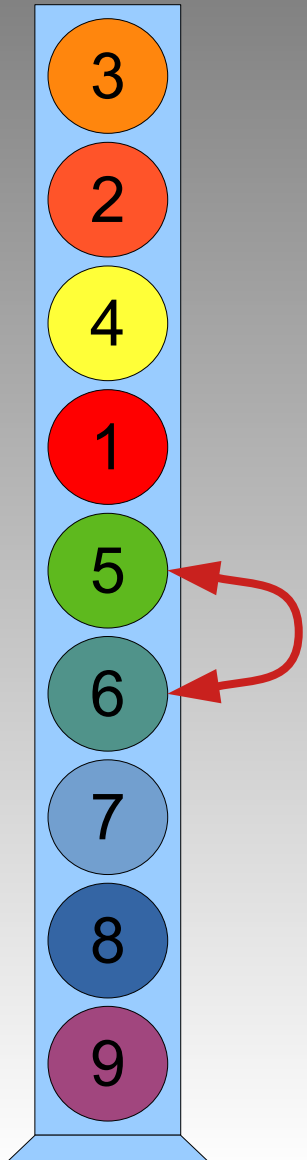
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



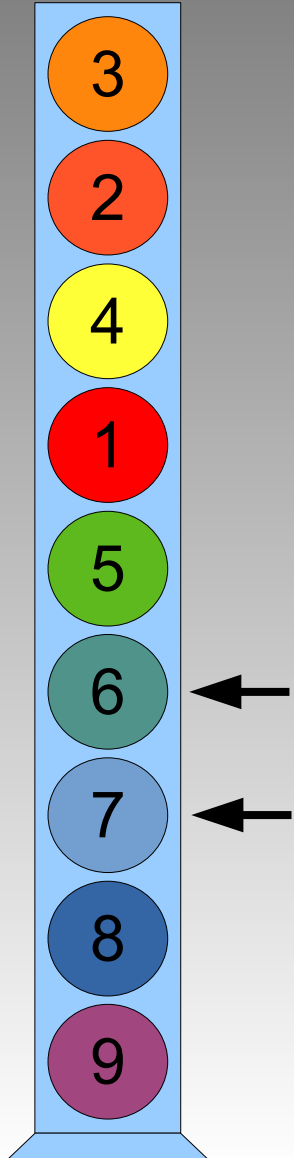
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



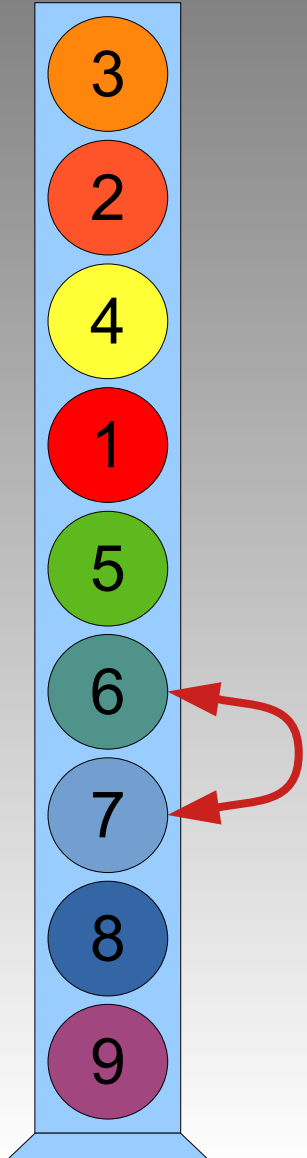
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



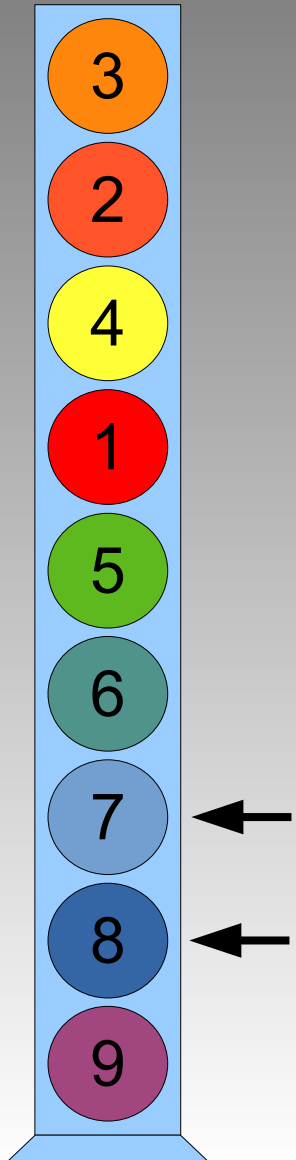
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



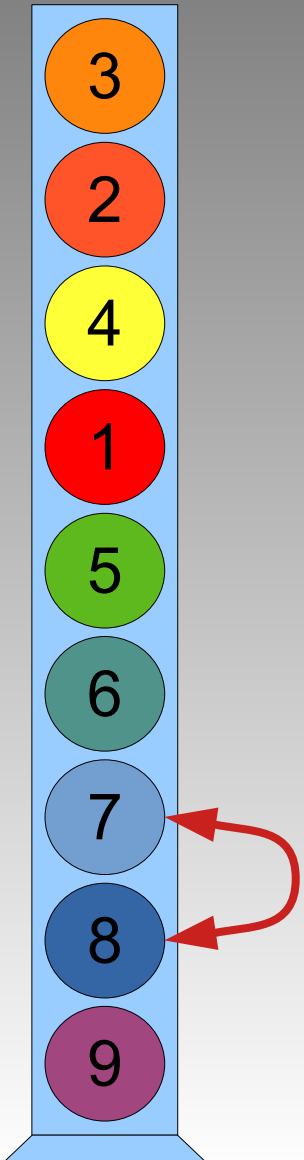
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



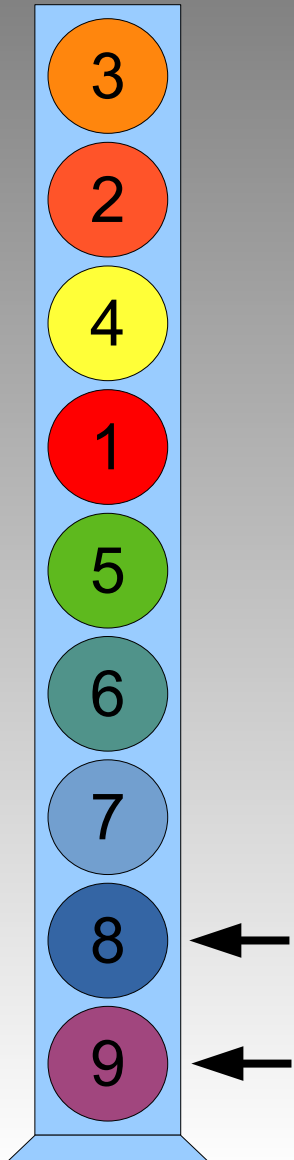
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



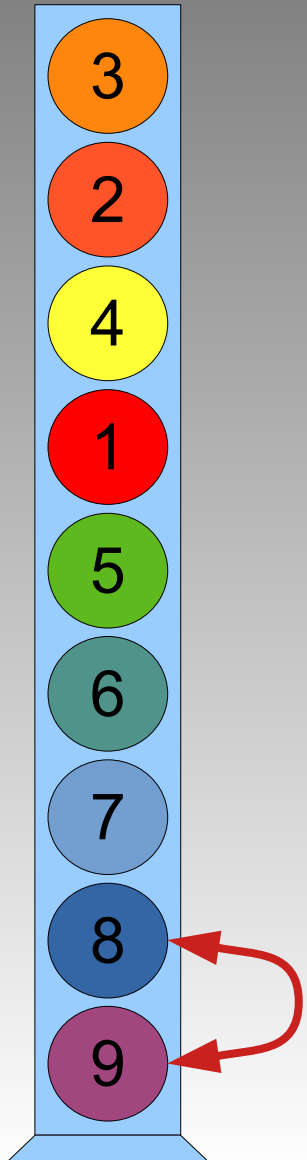
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



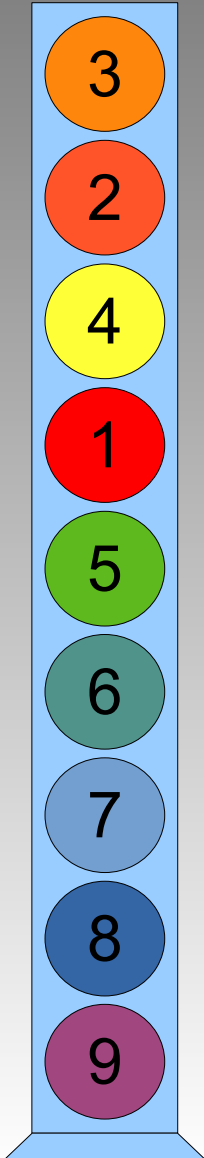
On effectue une 4^{ème} série de permutations.

La quatrième série de permutations



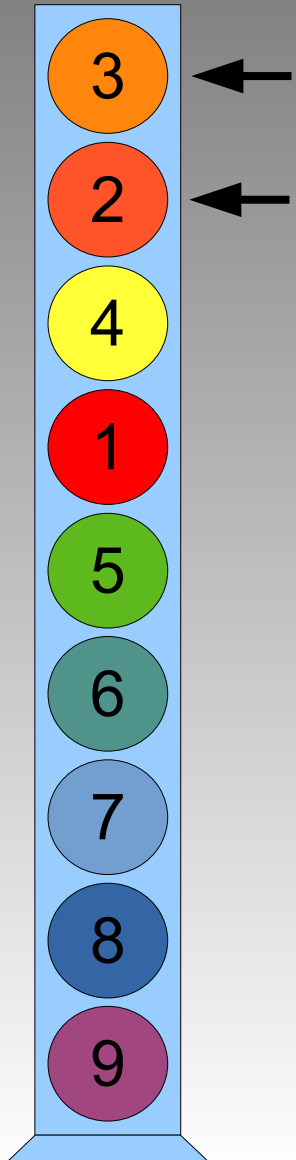
On effectue une 4^{ème} série de permutations.

On se rapproche un peu plus du but



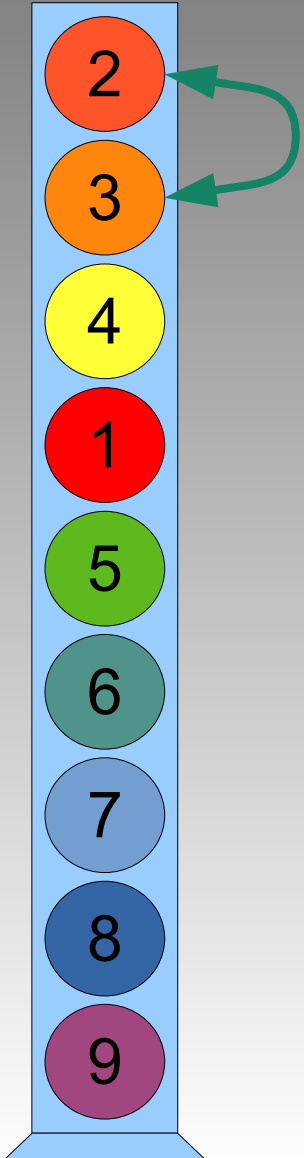
Après cette nouvelle série de permutations, on s'est rapproché un peu plus de l'état ordonné.

La cinquième série de permutations



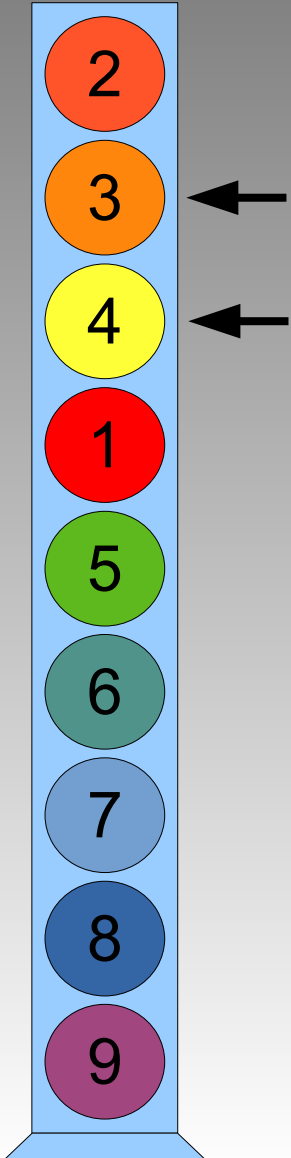
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



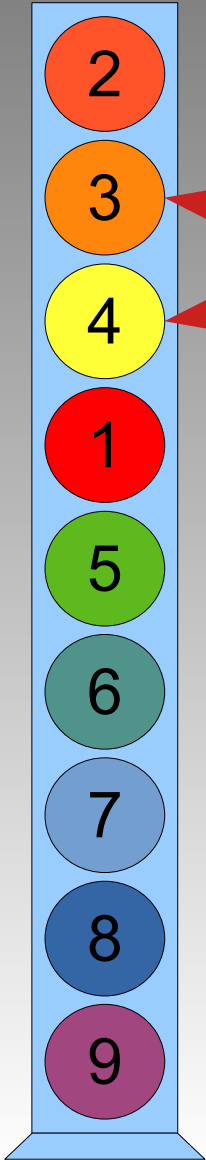
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



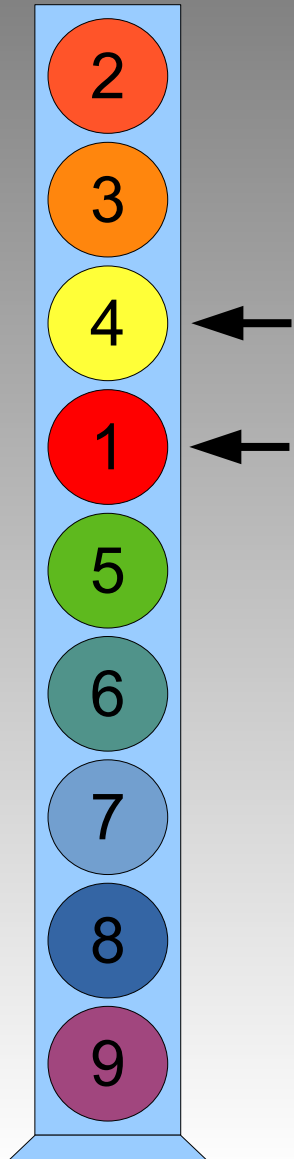
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



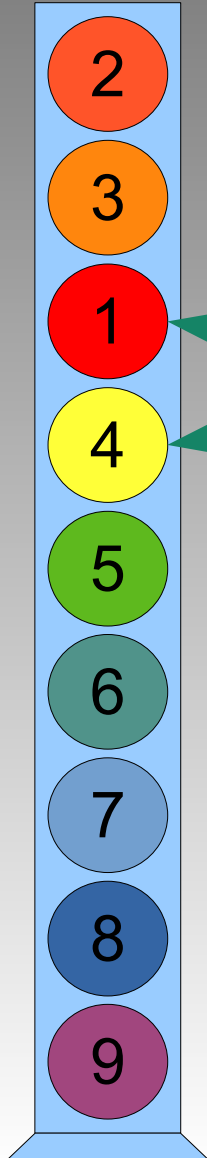
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



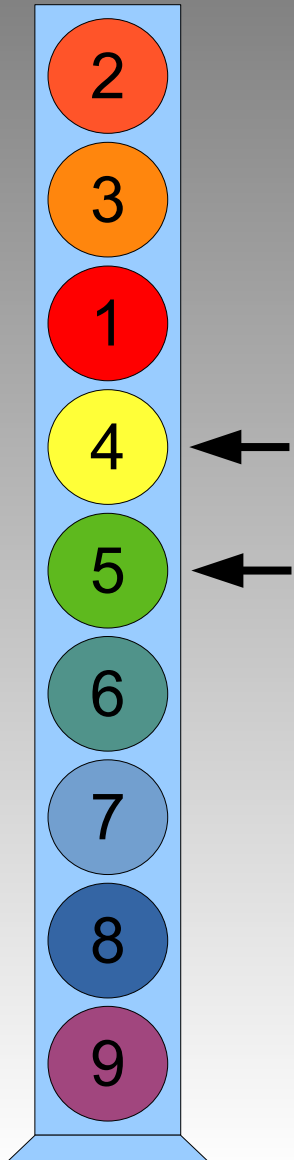
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



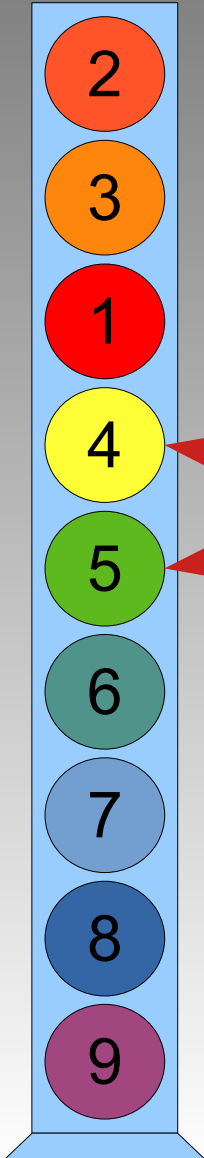
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



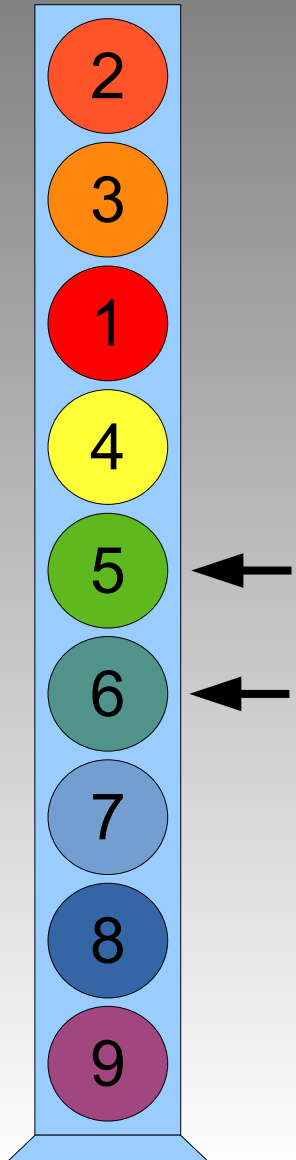
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



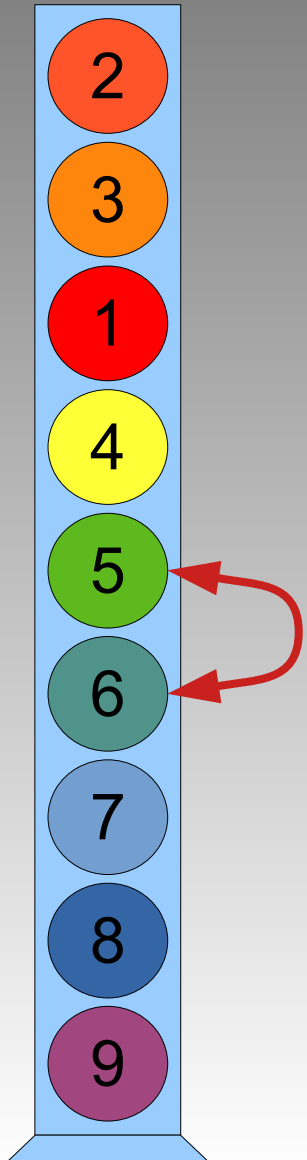
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



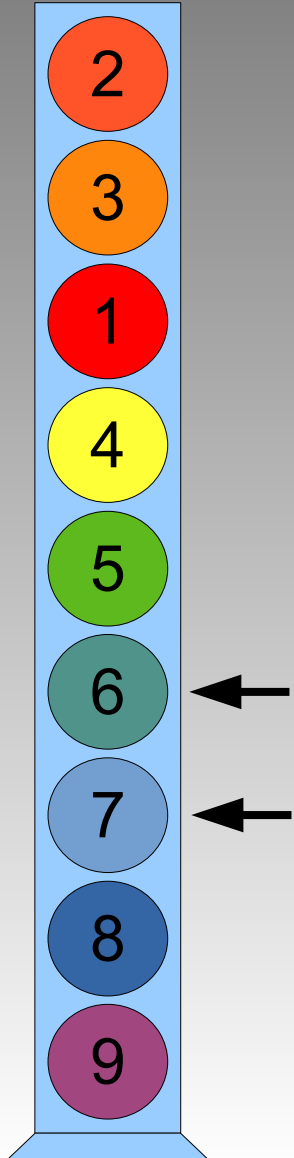
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



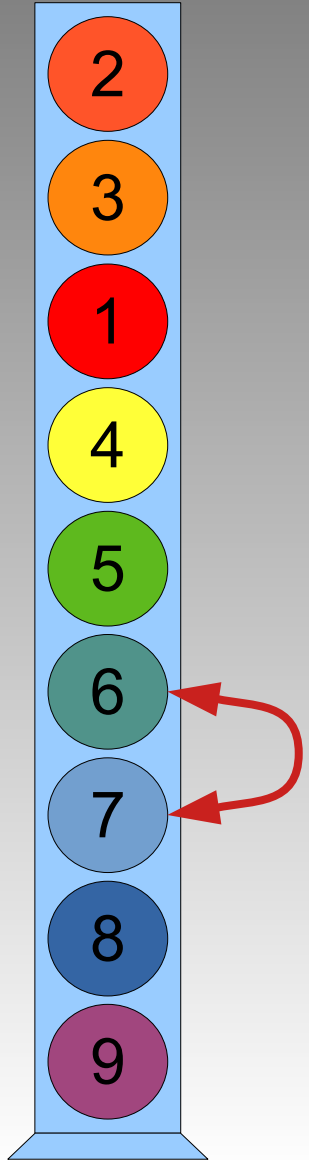
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



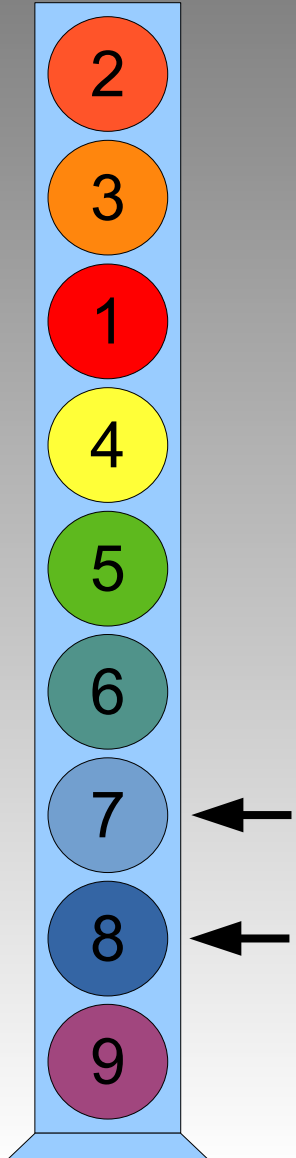
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



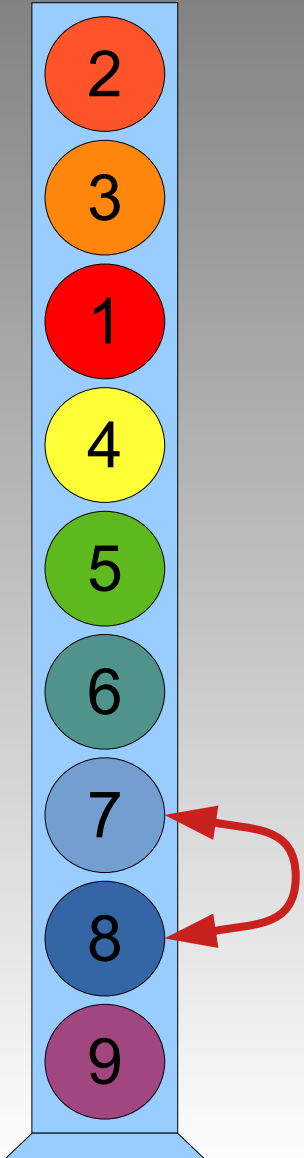
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



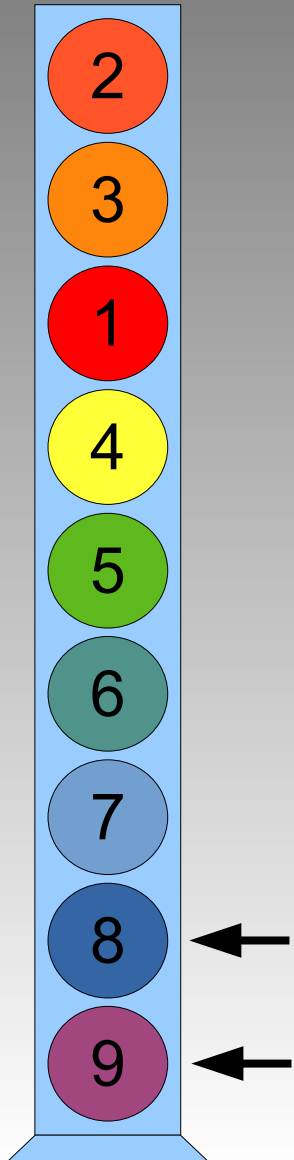
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



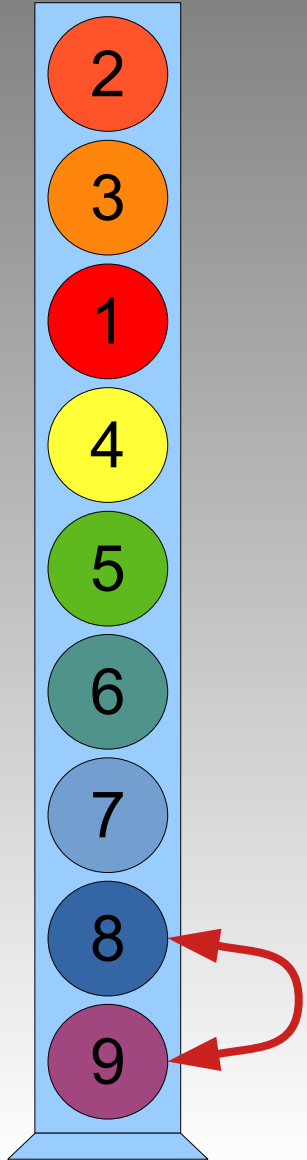
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



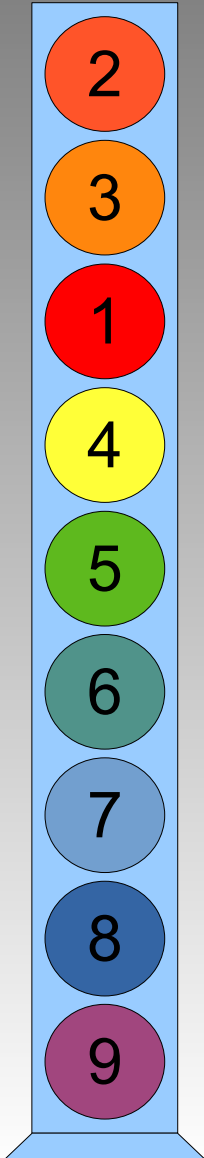
On effectue une 5^{ème} série de permutations

La cinquième série de permutations



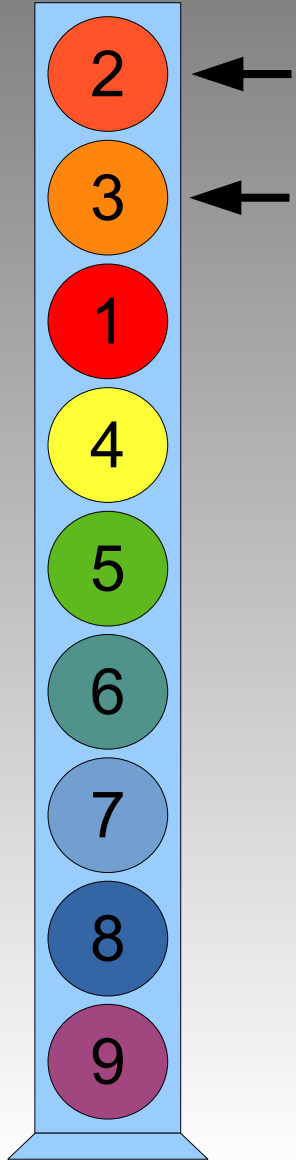
On effectue une 5^{ème} série de permutations

Encore un peu plus proche du but



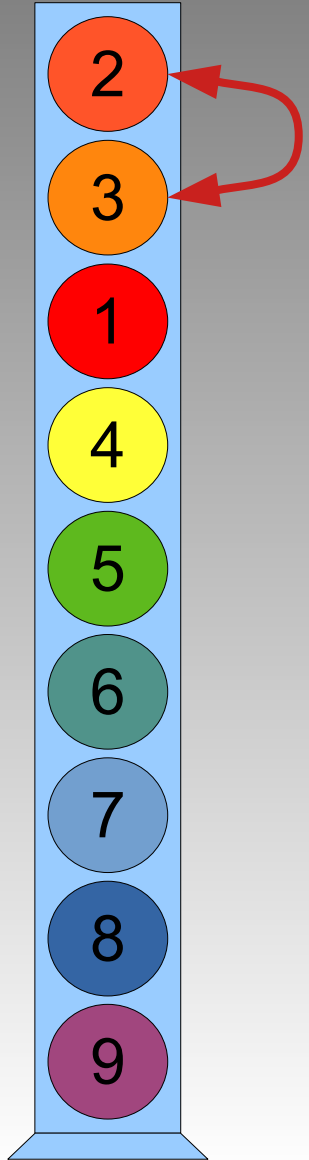
A l'issue de cette cinquième série de permutations, on est proche du but.

La sixième série de permutations



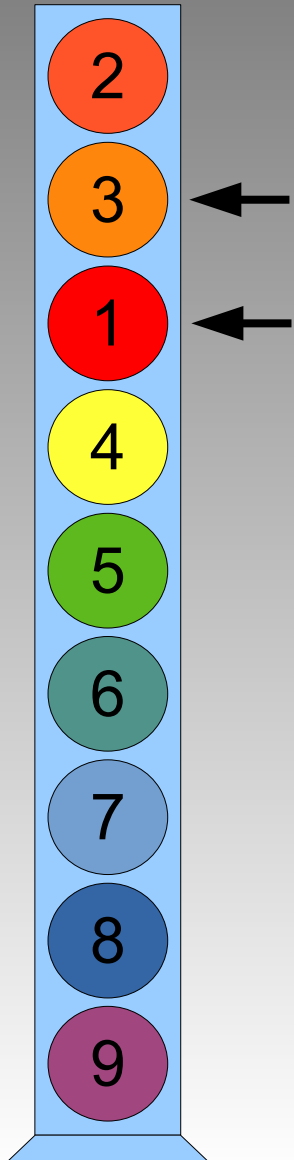
On effectue une 6^{ème} série de permutations

La sixième série de permutations



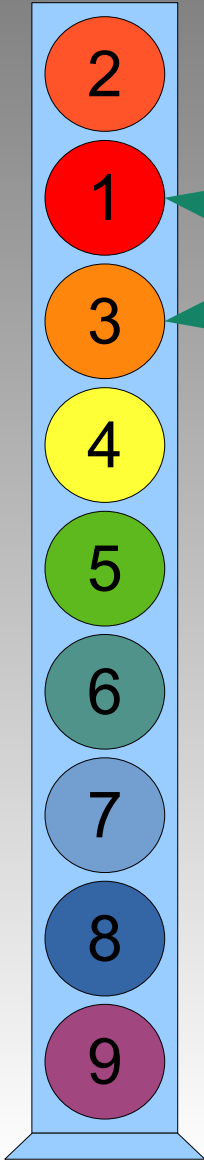
On effectue une 6^{ème} série de permutations

La sixième série de permutations



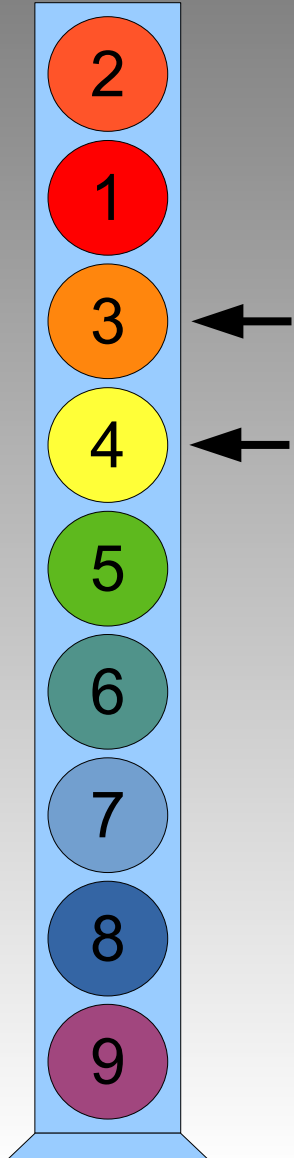
On effectue une 6^{ème} série de permutations

La sixième série de permutations



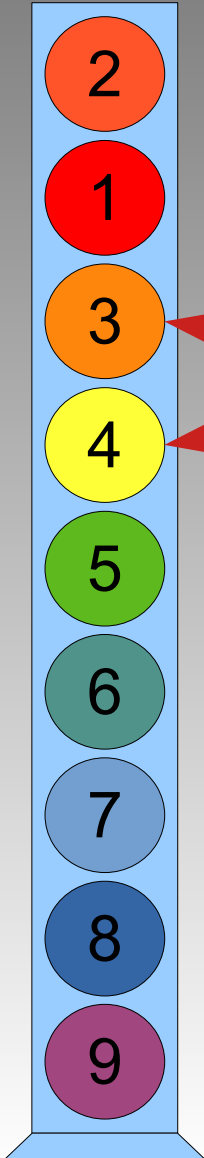
On effectue une 6^{ème} série de permutations

La sixième série de permutations



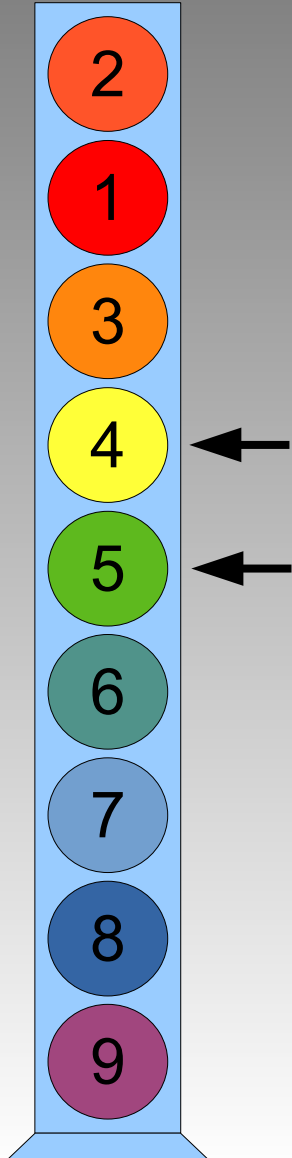
On effectue une 6^{ème} série de permutations

La sixième série de permutations



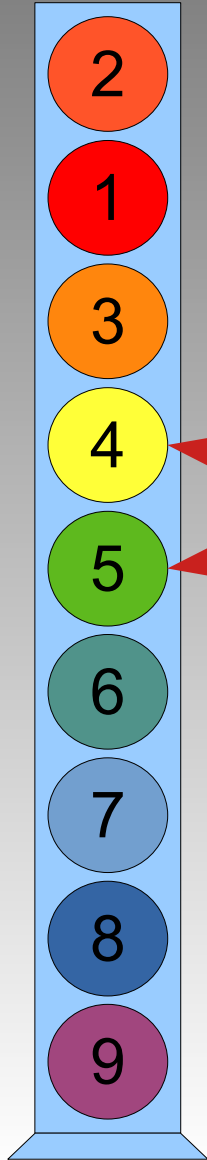
On effectue une 6^{ème} série de permutations

La sixième série de permutations



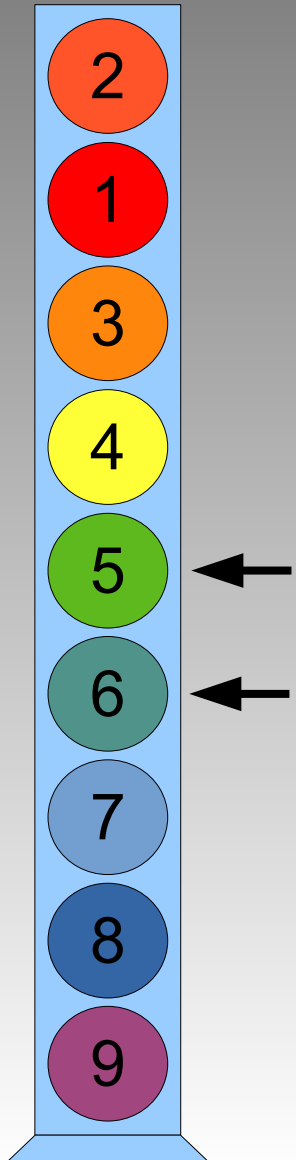
On effectue une 6^{ème} série de permutations

La sixième série de permutations



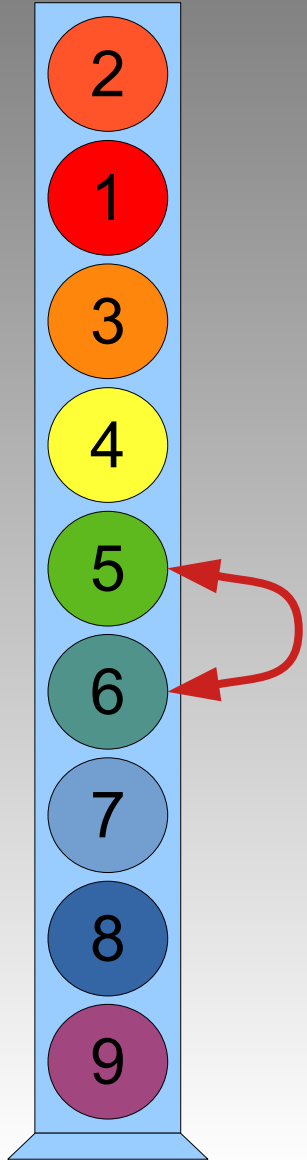
On effectue une 6^{ème} série de permutations

La sixième série de permutations



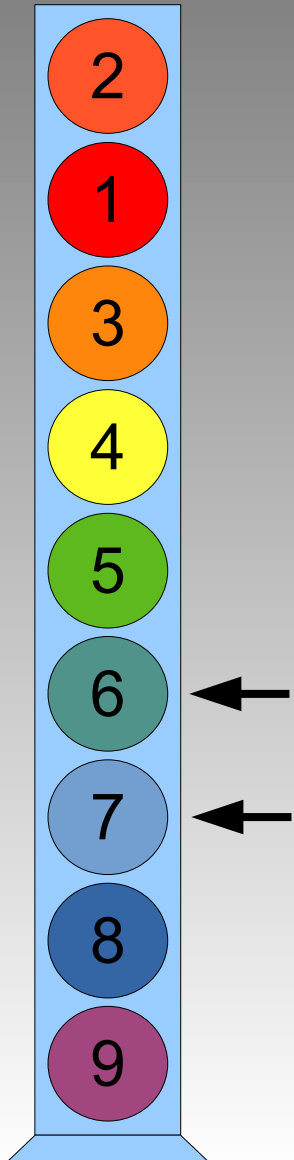
On effectue une 6^{ème} série de permutations

La sixième série de permutations



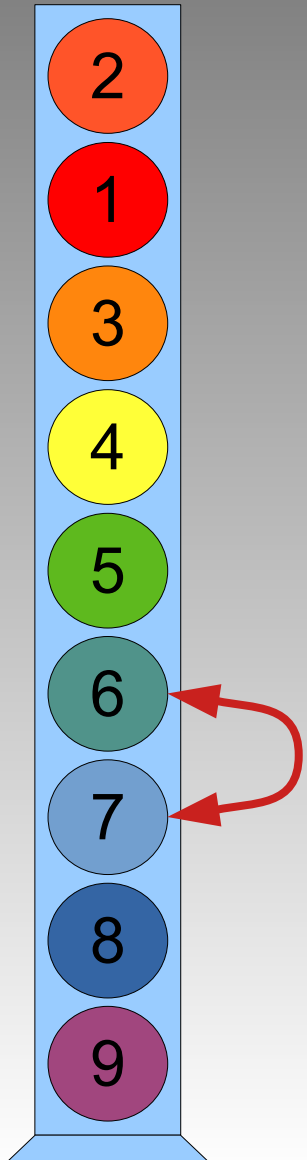
On effectue une 6^{ème} série de permutations

La sixième série de permutations



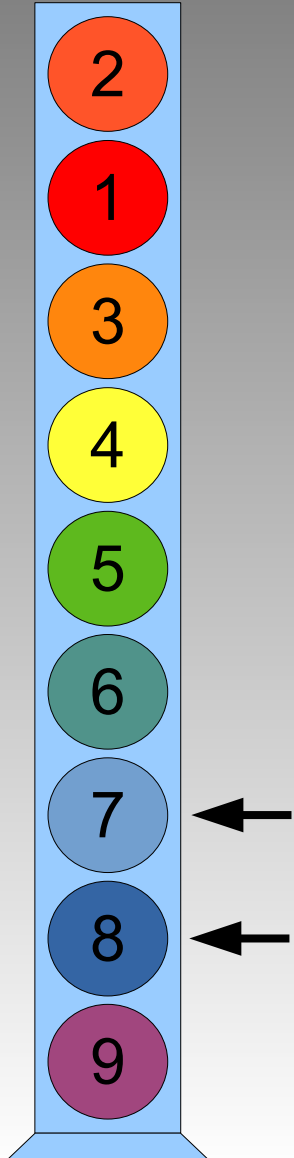
On effectue une 6^{ème} série de permutations

La sixième série de permutations



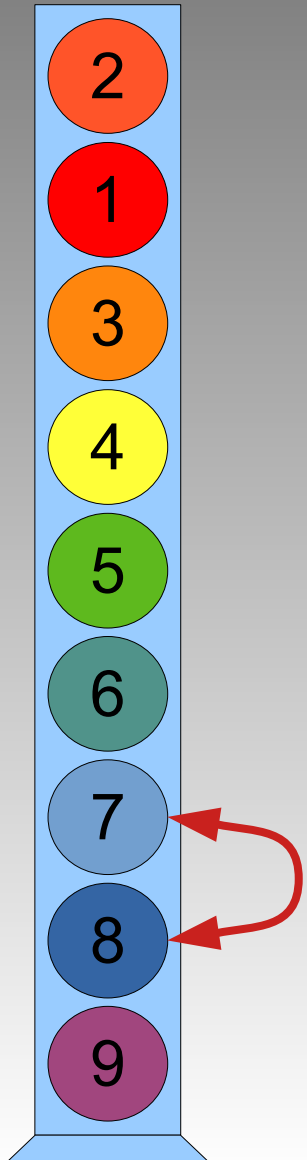
On effectue une 6^{ème} série de permutations

La sixième série de permutations



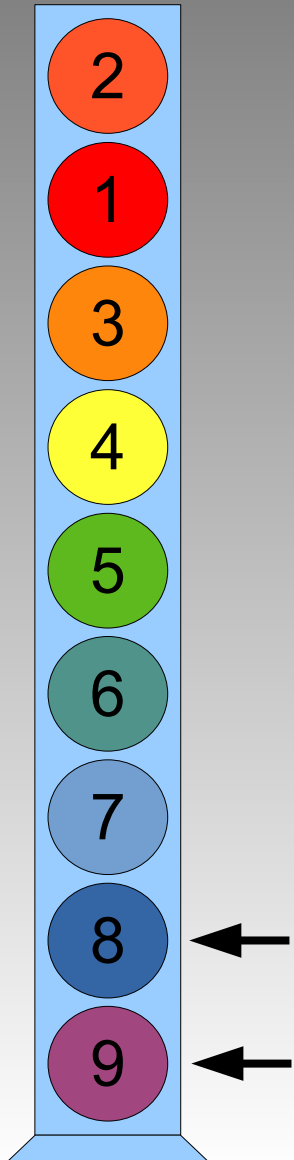
On effectue une 6^{ème} série de permutations

La sixième série de permutations



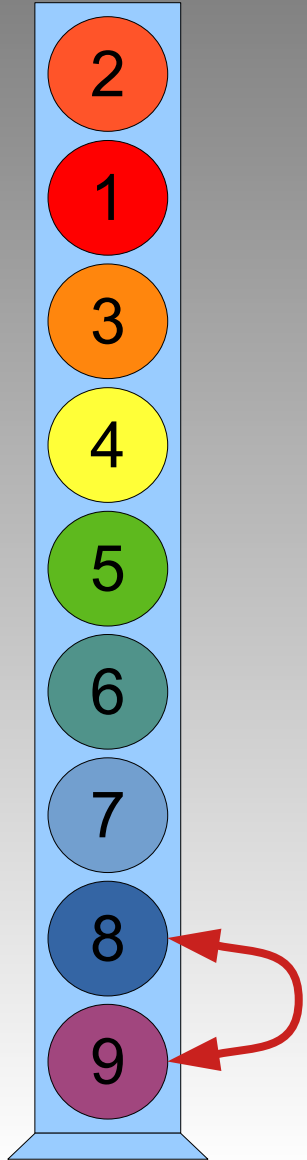
On effectue une 6^{ème} série de permutations

La sixième série de permutations



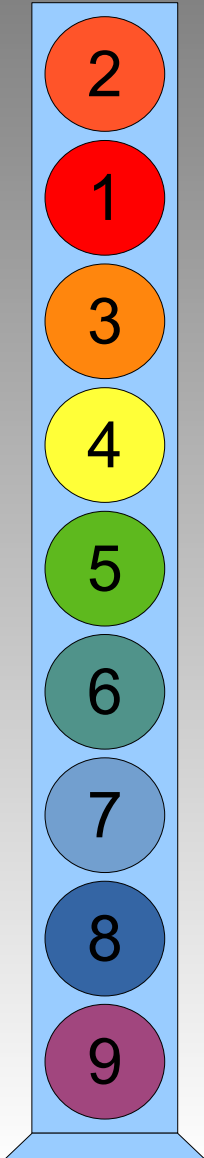
On effectue une 6^{ème} série de permutations

La sixième série de permutations



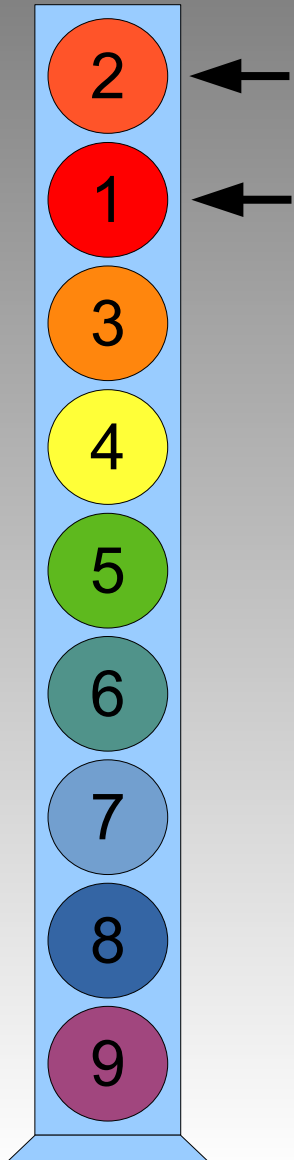
On effectue une 6^{ème} série de permutations

L'objectif est presque atteint



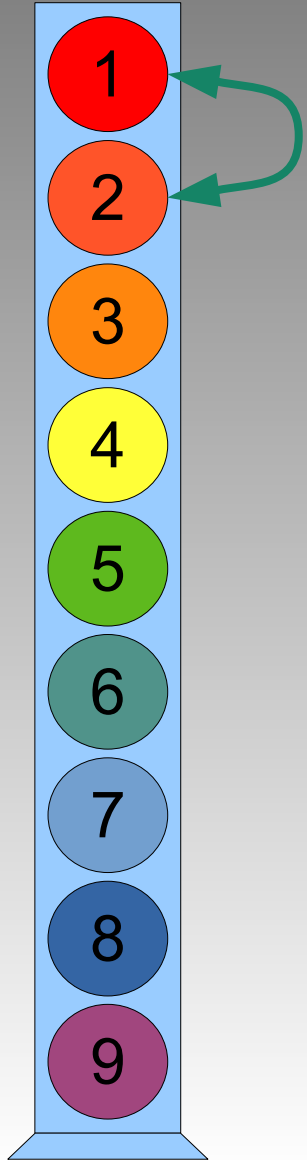
On s'est encore rapproché de l'objectif sans l'atteindre.

La septième série de permutations



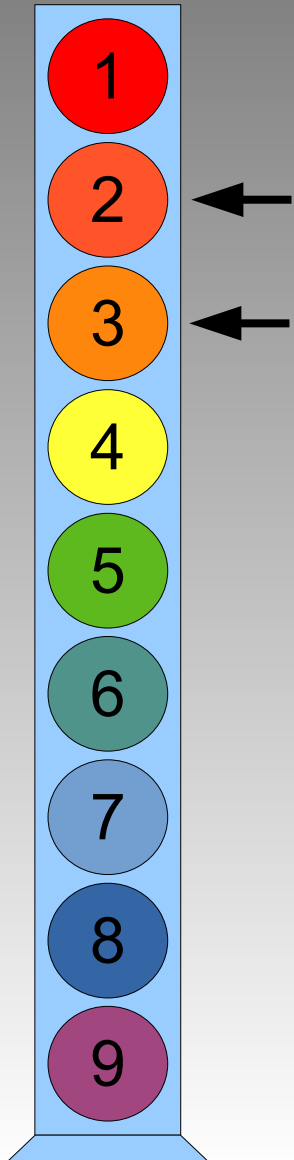
On effectue une 7^{ème} série de permutations

La septième série de permutations



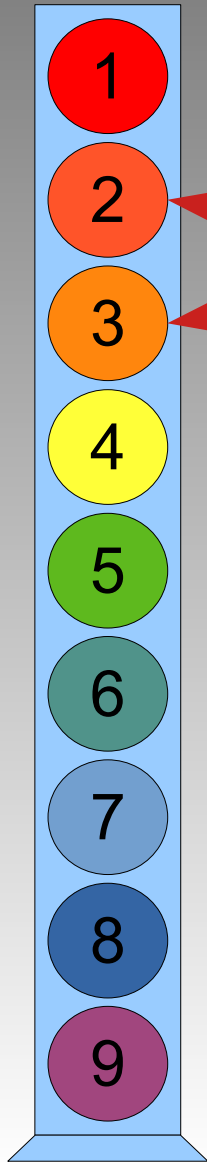
On effectue une 7^{ème} série de permutations

La septième série de permutations



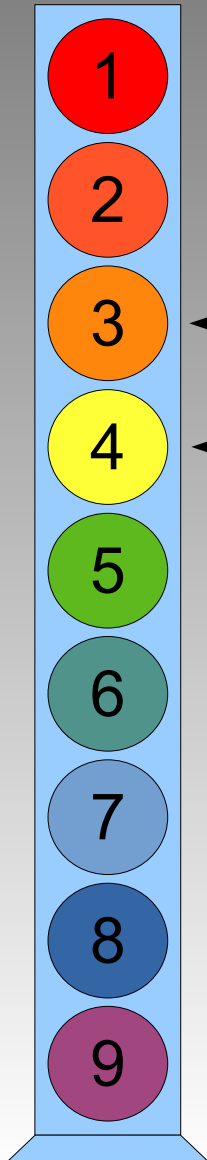
On effectue une 7^{ème} série de permutations

La septième série de permutations



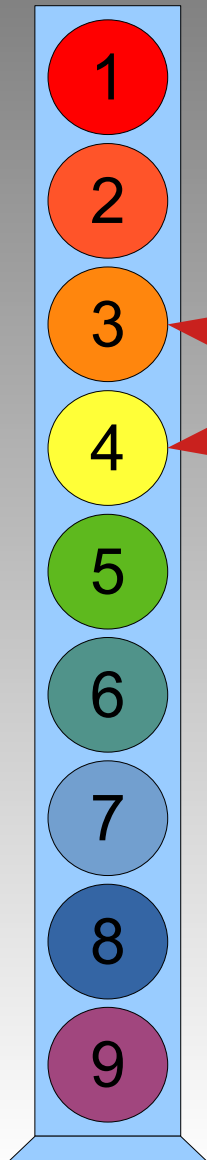
On effectue une 7^{ème} série de permutations

La septième série de permutations



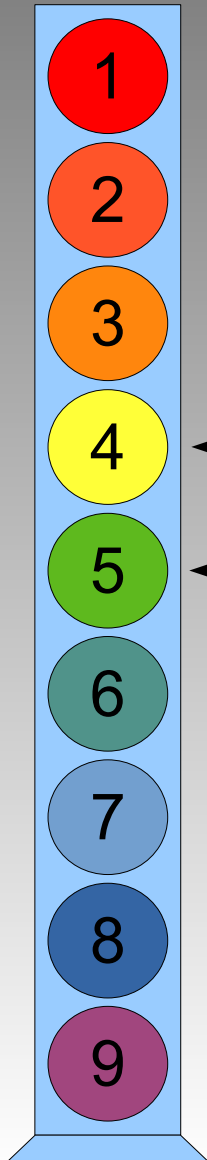
On effectue une 7^{ème} série de permutations

La septième série de permutations



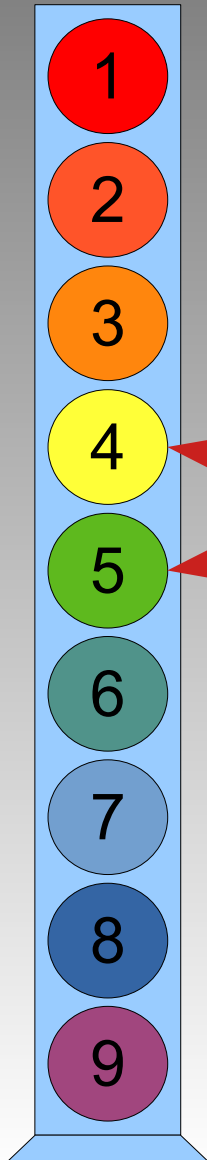
On effectue une 7^{ème} série de permutations

La septième série de permutations



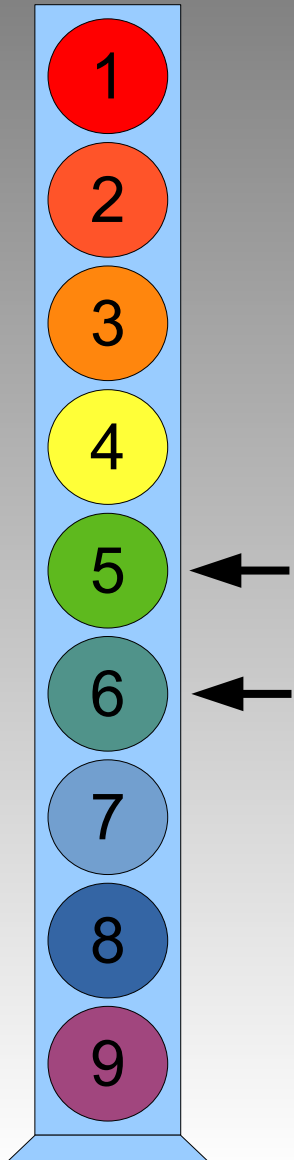
On effectue une 7^{ème} série de permutations

La septième série de permutations



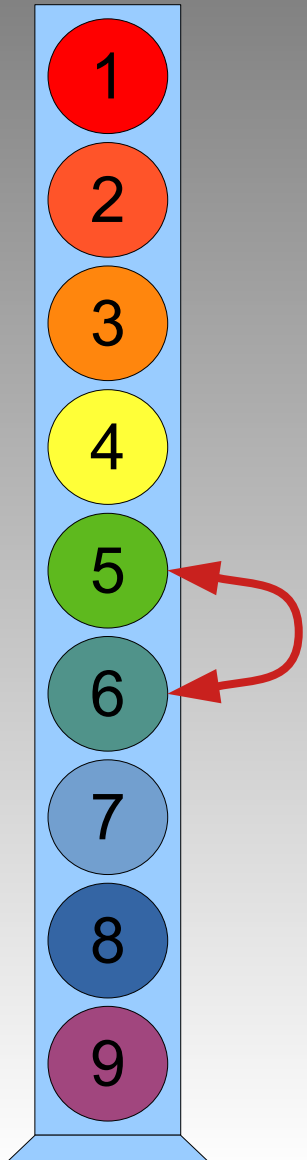
On effectue une 7^{ème} série de permutations

La septième série de permutations



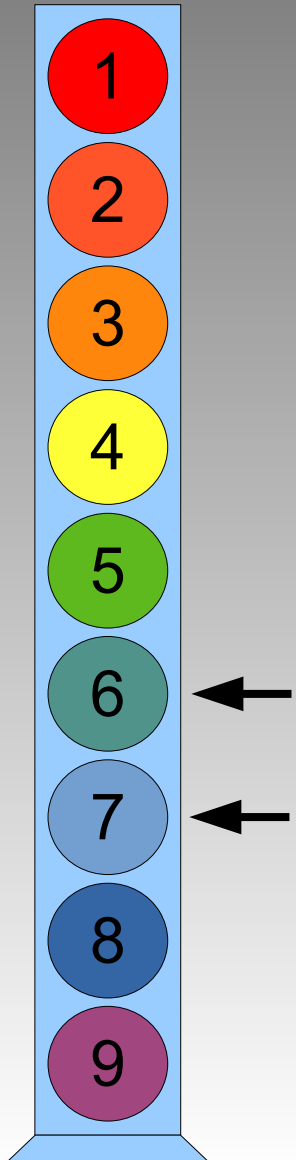
On effectue une 7^{ème} série de permutations

La septième série de permutations



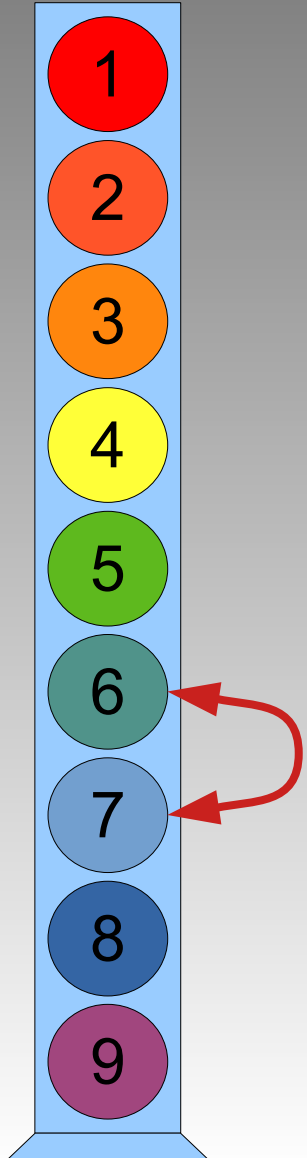
On effectue une 7^{ème} série de permutations

La septième série de permutations



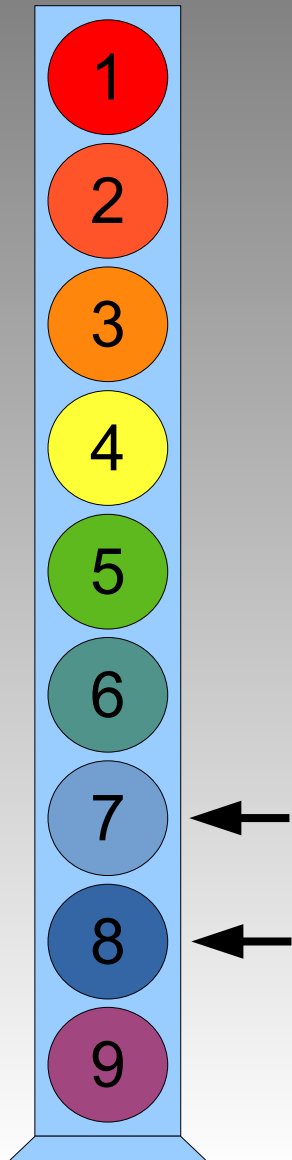
On effectue une 7^{ème} série de permutations

La septième série de permutations



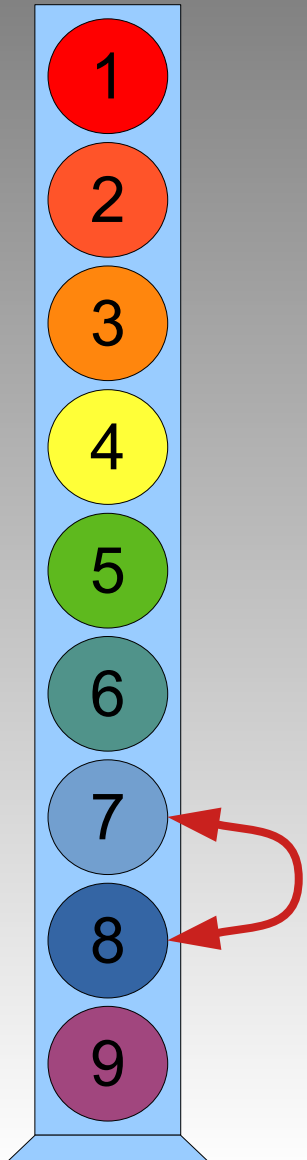
On effectue une 7^{ème} série de permutations

La septième série de permutations



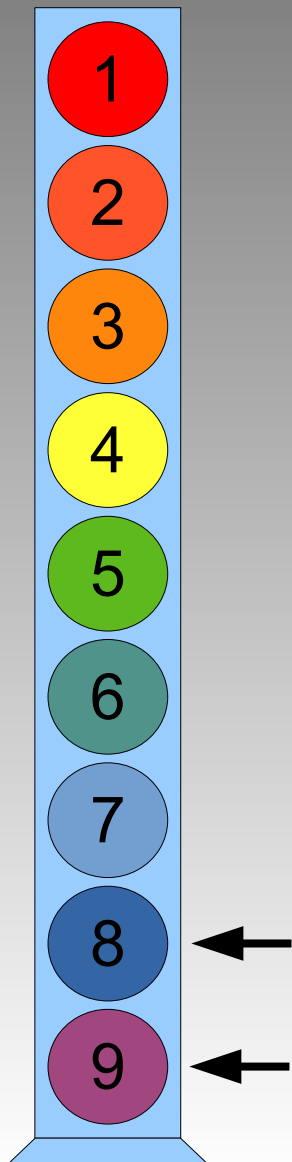
On effectue une 7^{ème} série de permutations

La septième série de permutations



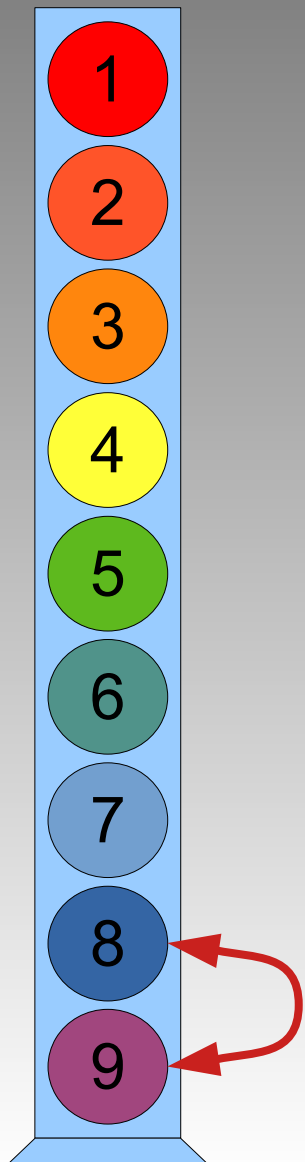
On effectue une 7^{ème} série de permutations

La septième série de permutations



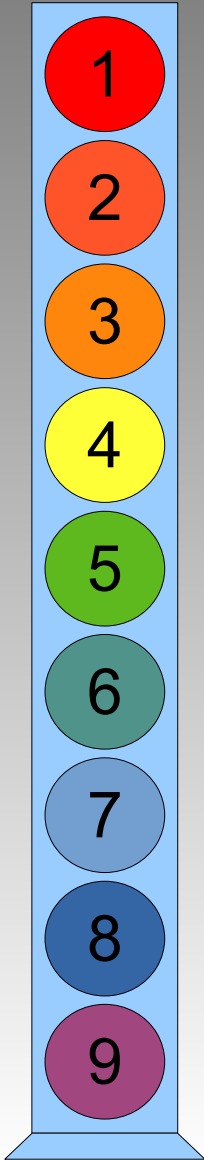
On effectue une 7^{ème} série de permutations

La septième série de permutations



On effectue une 7^{ème} série de permutations

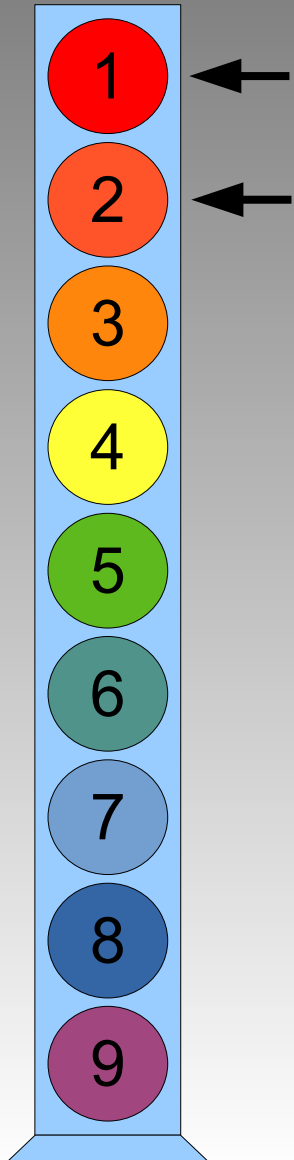
Objectif atteint mais on n'en est pas sûr



A l'issue de cette 7^{ème} série de permutations, la colonne de nombre semble correctement classée.

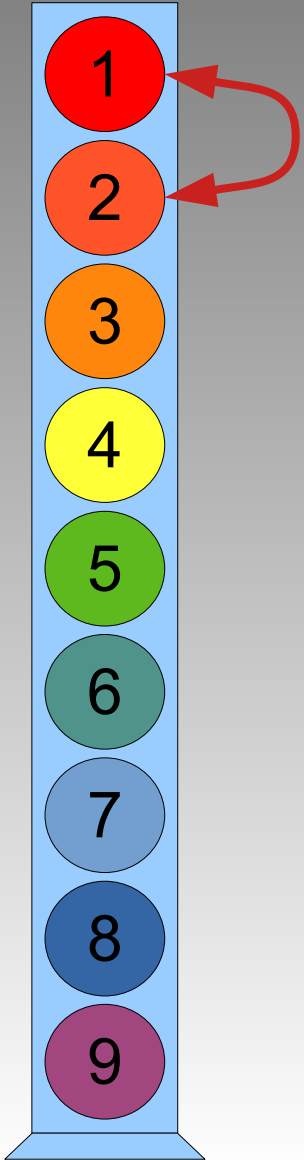
On n'en est cependant pas assuré car on a effectué au moins une permutation au cours de cette dernière.

La huitième série de permutations



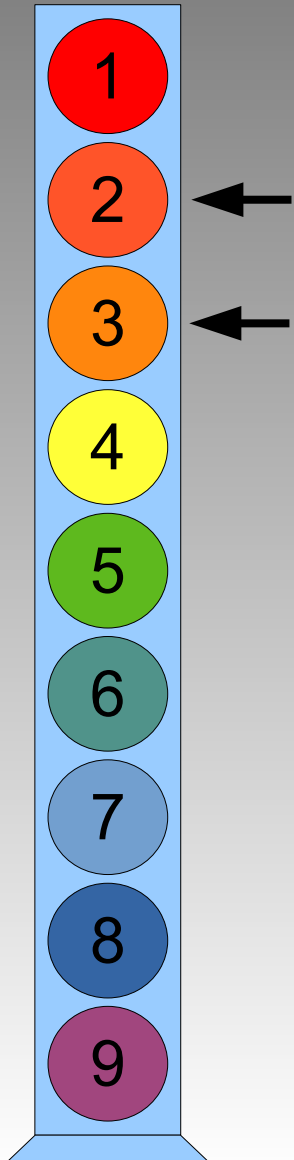
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



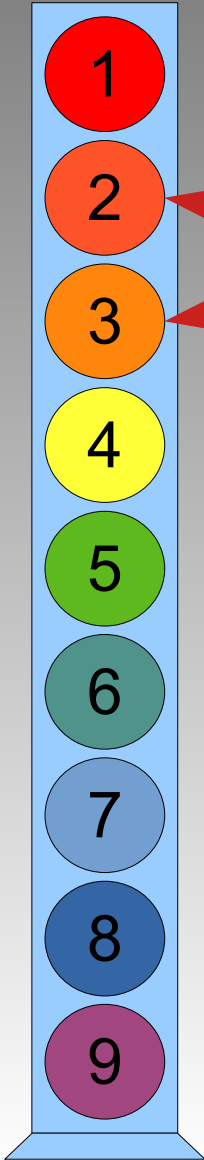
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



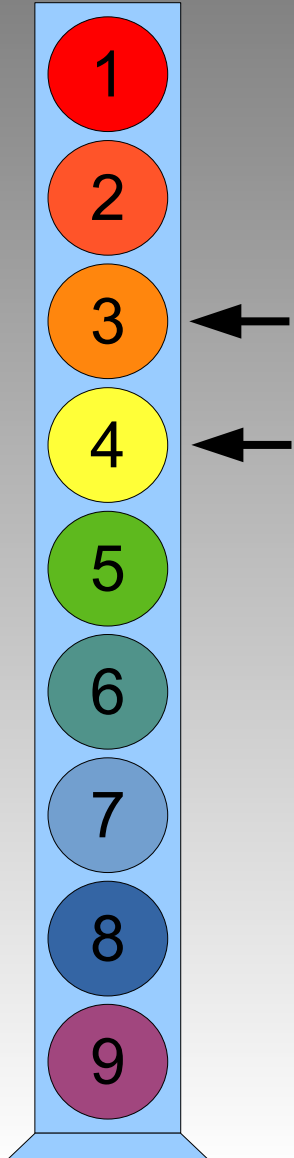
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



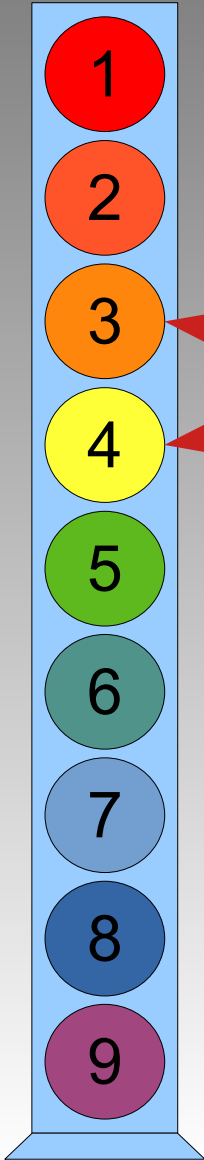
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



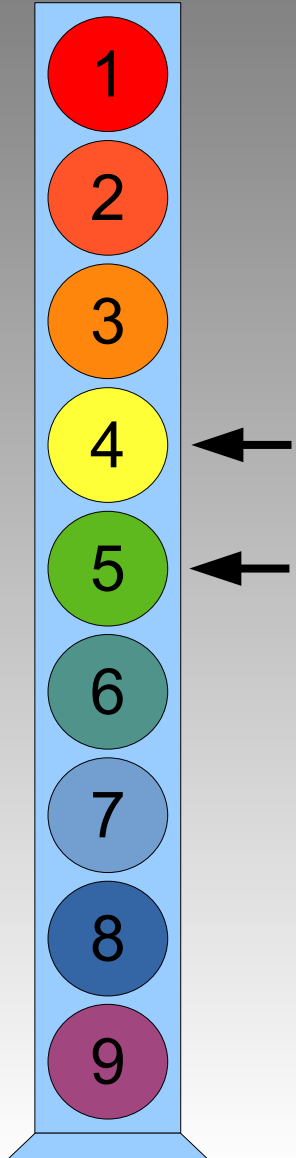
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



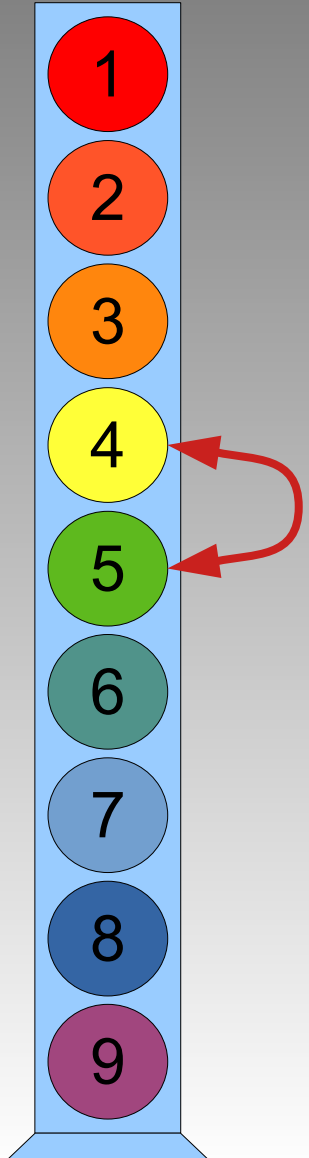
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



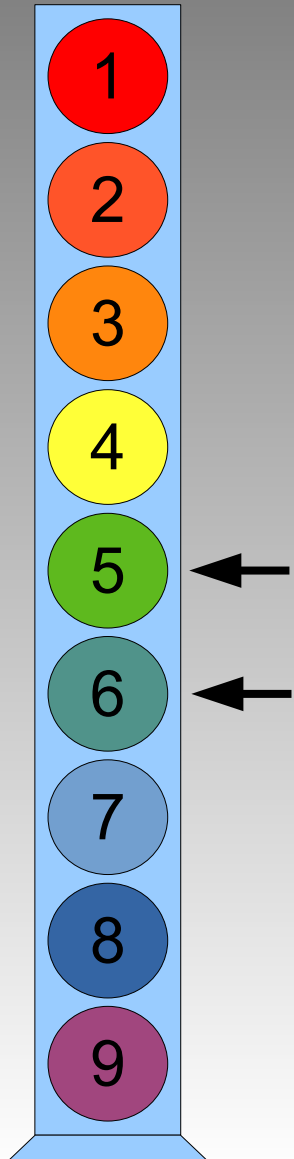
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



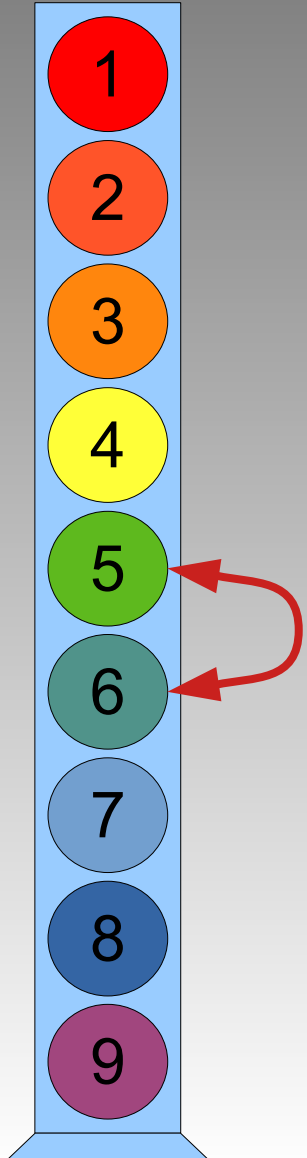
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



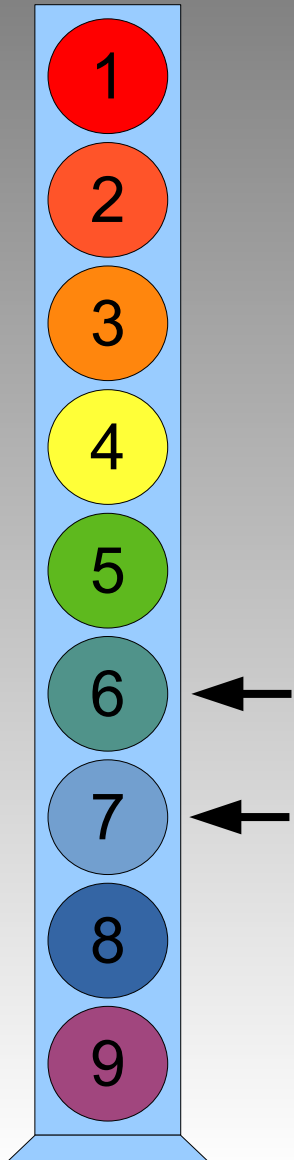
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



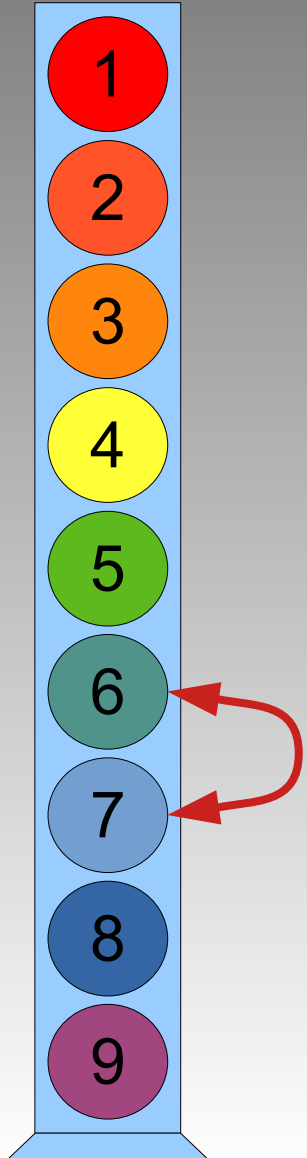
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



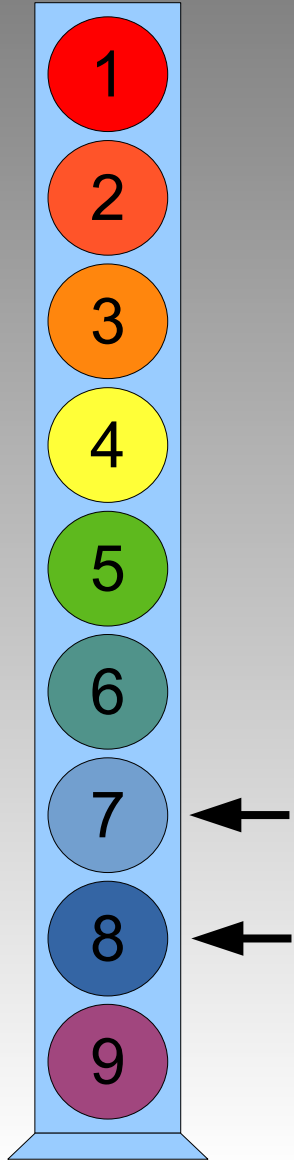
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



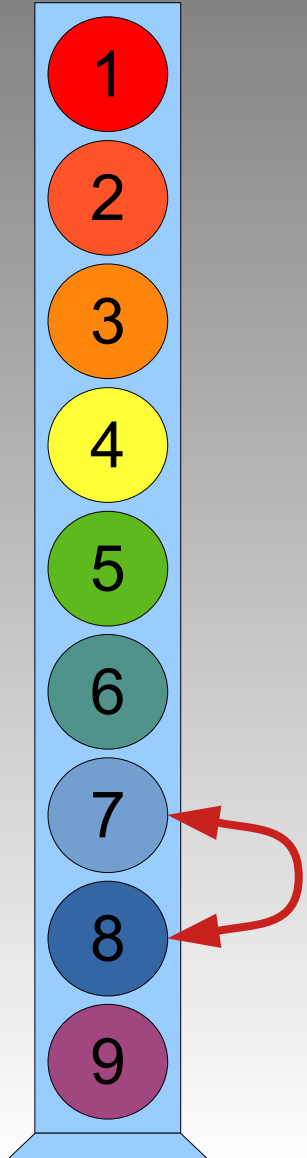
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



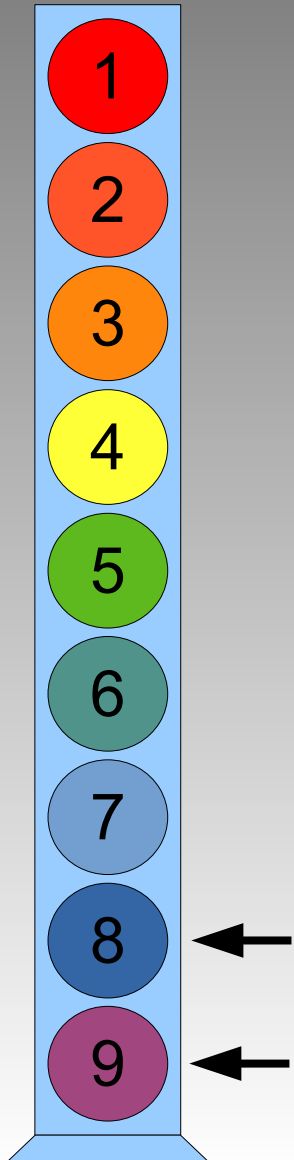
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



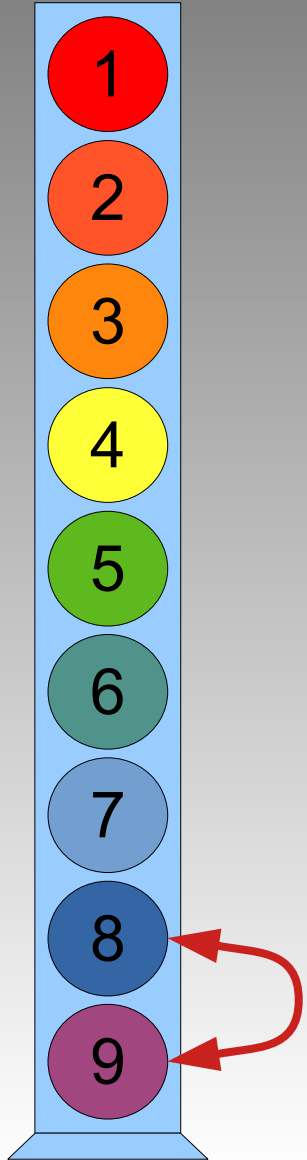
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



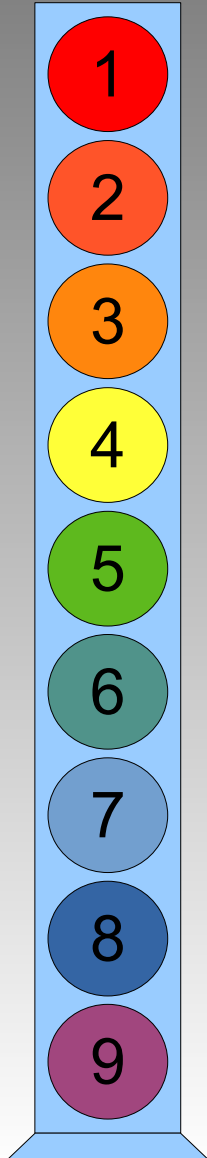
On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

La huitième série de permutations



On effectue une 8^{ème} série de permutations afin de vérifier que la colonne de nombre est bien classée (autrement dit que le signal « permutation effectuée » n'a pas été allumé).

On est sûr d'avoir atteint l'objectif



A l'issue de cette 8^{ème} série de permutations, on constate que le signal « permutation effectuée » ne s'est pas allumé : on est donc sûr que la colonne de nombre est ordonnée

Implémenter, en langage C, le mécanisme du tri à bulles en l'appliquant à un tableau de 9 entiers, compris entre 1 et 9 (à l'instar de l'exemple).

Le programme doit afficher l'état du tableau à chaque étape afin de visualiser le fonctionnement de l'algorithme.