Chapitre 12 : Les tableaux automatiques

I. Introduction : manipuler de grands volumes de données.

Jusqu'ici, nos programmes n'ont manipulé qu'une petite quantité de données. Pour chaque donnée que le programme devait stocker, on déclarait une variable. Mais les programmes manipulent en général de grandes quantités de données. Le moindre petit jeu, genre démineur ou Candy Crush utilisent plusieurs centaines de données en mémoire. Les jeux vidéo actuels comme Call of Duty ou GTA stockent des millions à des milliards de données (surtout pour les graphismes).

Bien entendu, si un programme doit stocker et manipuler 1000 données, nous n'allons pas déclarer 1000 variables. Alors comment faire ? Nous allons utiliser des **tableaux**.

L'explosion des données dans les domaines scientifiques au XXème siècle.

Dans les domaines scientifiques, le volume de données atteint des proportions vertigineuses. En astronomie ou dans le spatial, les télescopes modernes produisent chaque jour des téraoctets de données d'observation. En météorologie, les satellites et capteurs collectent en continu des informations atmosphériques mondiales, générant des pétaoctets de données à analyser pour affiner les prévisions. En nanotechnologies ou en sciences des matériaux, les microscopes électroniques et simulations génèrent des images et mesures en très haute résolution, qui représentant des volumes de données gigantesques.

L'ère du Big Data : enjeux de stockage, défis de traitement.

Depuis quelques années, la quantité de données produites chaque jour par nos sociétés ultra-connectées, atteint des volumes colossaux. Les données postées sur les réseaux sociaux ou émises par les capteurs connectés, les transactions en ligne …, sont collectées par des entreprises spécialisées dans les technologies de l'information et de la communication, comme les célèbres « géants du net » que sont les GAFAM (américains) ou BATX (chinois). Ces entreprises

utilisent des algorithmes pour organiser et analyser ces données afin de personnaliser des services, de détecter des tendances, d'essayer de prédire des comportements …
Pour traiter efficacement cette masse d'informations, il faut des structures simples et rapides. Le tableau en constitue la brique de base : il permet de stocker et d'organiser des données de manière séquentielle, en permettant un accès direct et rapide aux éléments.

Les données sont si massives qu'elles ne tiennent souvent pas sur une seule machine. On a donc recours à des systèmes de stockage distribués : les données sont réparties sur plusieurs serveurs reliés en réseau.
Les tableaux, en tant que structure simple et séquentielle, restent la base sur laquelle ces systèmes organisent et manipulent l'information, avant d'être intégrés dans des architectures plus complexes : les tableaux sont les fondations sur lesquelles reposent les traitements des données à grande échelle.

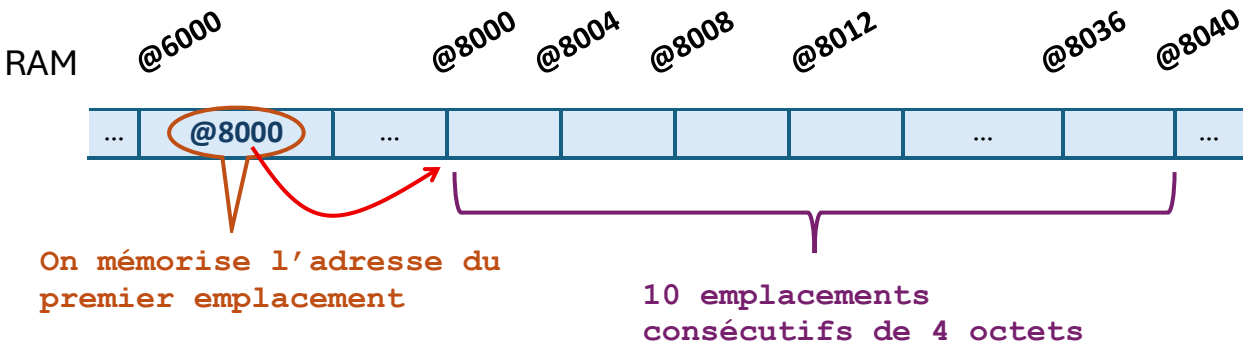
II. Concepts et définitions

Problématique : Comment stocker en mémoire N données sans avoir à déclarer N variables (en utilisant un seul identificateur) ?

Idée :

Si les N emplacement sont **consécutifs** en mémoire et s'ils font tous **la même taille (le même nombre d'octets), alors** il suffit de **mémoriser l'adresse du premier emplacement**, et on pourra accéder à n'importe lequel des N emplacements grâce à un petit calcul sur les adresses.

Exemple : pour stocker 10 entiers. Il faut réserver 10 emplacements consécutifs de 4 octets.



Question : Dans quoi mémoriser l'adresse du premier emplacement ?

Dans un pointeur.

Question : Comment accéder au 4ème emplacement (à l'adresse @8012)?

Aller sur le premier (à l'adresse @8000) et « sauter » 12 octets (taille de 3 emplacements).

Définitions :

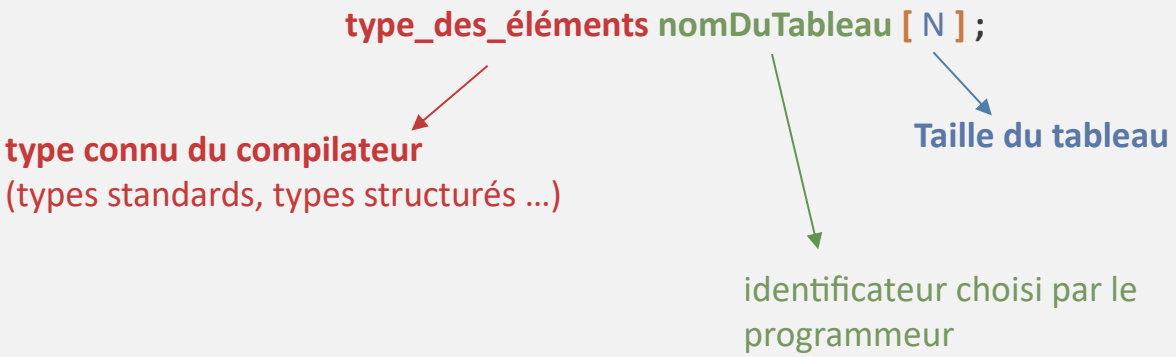
- **On appelle tableau une variable composée de données de même type, stockées de manière contiguë en mémoire (dans des emplacements les uns à la suite des autres).**
- Les emplacements consécutifs sont appelés les **cases du tableau.**

- Les éléments du tableau sont les valeurs contenues dans ses cases.
- **La taille du tableau** est le nombre de cases.
- **L'adresse d'un tableau est l'adresse de sa première case.**

Question : Selon vous, pourquoi les éléments d'un tableau doivent-ils forcément tous être du même type ?

III. Déclaration et initialisation d'un tableau

1. Déclaration d'un tableau automatique



Déclarer un tableau de taille N revient à :

- Réserver N emplacements consécutifs dont la taille est donnée par le type des éléments
- Mémoriser l'adresse du 1er emplacement : **le nom du tableau désigne un pointeur constant qui contient l'adresse de la première case**

Exemples :

float t1[100] ;

Déclaration d'un tableau appelé t1 de 100 éléments de types réels = réservation de 100 emplacements consécutifs de 4 octets chacun.

char mot[20] ;

Déclaration d'un tableau appelé mot de 20 éléments de type caractère = réservation de 20 emplacements consécutifs de 1 octet chacun.

2. Taille d'un tableau automatique

La taille d'un tableau automatique doit être une **constante fixée avant ou lors de sa déclaration.** Elle doit être connue à la compilation. On ne peut pas agrandir ou réduire la taille d'un tableau automatique après sa déclaration.

Pour manipuler des tableaux de taille variable redimensionnable, il faut faire de l'allocation dynamique (→ chapitre 16).

Remarque : Depuis la version C99, il est possible de déclarer des tableaux de taille variable, c'est-à-dire dont la taille est déterminée à l'exécution. Mais cela reste déconseillé car ces tableaux restent non-redimensionnables. Et il faut l'éviter si on veut garantir la portabilité du programme (capacité à être utilisé sur différents environnements).

Utilisation de la directive *#define* pour définir une constante symbolique.

On peut utiliser la directive *#define*, juste après les directives *#include*, pour définir des constantes symboliques utilisées pour fixer la taille des tableaux automatiques.

```c
#include <stdio.h>
#define NCases 10
int main(){
    //déclaration d'un tableau 10 entiers
    int tab[NCases];
    //...
    return 0;
}
```

Lors des pré-traitements, avant la compilation, toutes les apparitions du mot « NCases » seront remplacées par 10 dans le fichier source.

3. Initialisation d'un tableau lors de sa déclaration

Par défaut, les cases d'un tableau ne sont pas initialisées. Leurs valeurs sont indéterminées (elles dépendent des codes binaires restés dans les emplacements après leurs dernières utilisations par les programmes lancés sur l'ordinateur).

On peut initialiser un tableau lors de sa déclaration :

```
//déclaration et initialisation d'un tableau de 5 entiers
//Les valeurs initiales de chaque case sont indiquées
int tab[5]={3,-8,12,1,4};
//déclaration et initialisation de 10 réels.
//Toutes les cases sont initialisées à 0.0
float tab2[10]={0.0};
//déclaration et initialisation de 10 réels.
//La première case est initialisée à 1.0
//La deuxième case est initialisée à 3.0
//Toutes les cases suivantes sont initialisées à 0.0
float tab3[20]={1.0,3.0};
```

Cette manière d'initialiser un tableau n'est possible qu'au moment de sa déclaration. On ne peut pas l'utiliser après.

IV. Accès aux éléments.

1. Accès grâce aux pointeurs

Rappel : le nom du tableau désigne un pointeur constant qui contient l'adresse de la première case.

Les cases d'un tableau étant contiguës en mémoire, l'adresse de la $i^{ème}$ case est l'adresse de la première augmentée de (i-1) fois la taille d'une case.
Pour accéder au contenu d'une case, il faut aller à son adresse par indirection (→ chapitre 11 sur les pointeurs).

Question : Qu'affiche le code ci-dessous ?

```c
#include <stdio.h>
#define N 5
int main(){
    float tab[N]={3.5,12,8.75,-5.5,4};
    printf("%f",*tab);
    return 0;
}
```

Activité 14. Comprendre l'arithmétique des pointeurs.

Le code ci-contre affiche les résultats ci-dessous :

```
630240312727 630240312728
630240312720 630240312724
630240312712 630240312720
```

```c
#include <stdio.h>
#define N 5
int main(){
    char c;
    int x;
    double y;
    char *pc=&c;//pc pointe sur c
    int *p=&x; //p pointe sur x
    double *pp=&y; //pp pointe sur y
    //affichage des adresses
    printf("%lld %lld\n",pc,pc+1);
    printf("%lld %lld\n",p,p+1);
    printf("%lld %lld\n",pp,pp+1);
    return 0;
}
```

Questions : Comment expliquez-vous les valeurs des adresses affichées ? Que se passe-t-il quand on ajoute un entier à un pointeur ? Qu'en déduisez-vous sur l'arithmétique des pointeurs ?

Exercice 25. Manipuler les tableaux grâce aux pointeurs.

Ecrire un programme :

- Déclarer un tableau de N entiers
- En utilisant une boucle et un pointeur pour passer de case en case, remplir le tableau avec des valeurs aléatoires dans l'intervalle [-10,10]
- Afficher le tableau

2. Accès par indice avec l'opérateur []

Soit *t* un tableau de N éléments. Comme vous avez pu le constater lors de l'exercice précédent, pour accéder à la $i^{ème}$ case d'un tableau, il faut aller à l'adresse t+(i-1) :

Accès à la $1^{ère}$ case du tableau : *(t+0)
Accès à la $2^{ème}$ case du tableau : *(t+1)
…
Accès à la $N^{ème}$ case du tableau : *(t+(N-1))

On considère donc que les cases d'un tableau sont numérotées à partir de 0 :

L'**indice** de la première case est 0
L'indice de la deuxième case est 1
L'indice de la dernière case d'un tableau de taille N est N-1

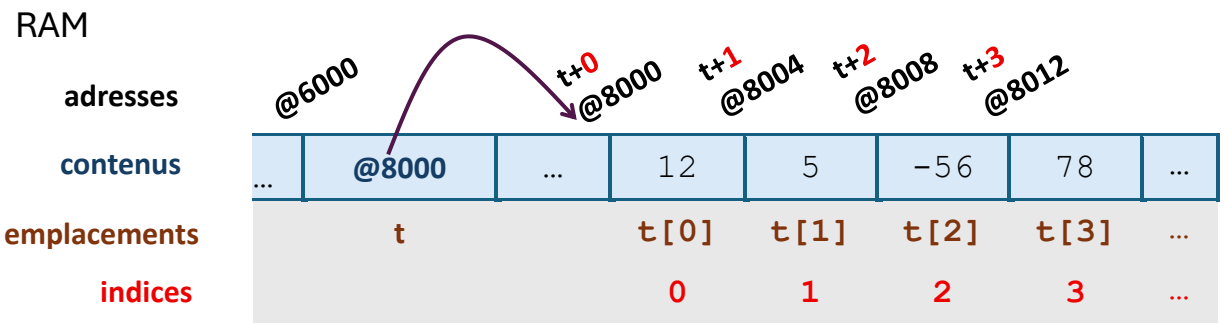
L'opérateur [] permet d'accéder à une case directement par son indice : t[i] permet d'accéder à la case d'indice i.

t[i] est équivalent à *(t+i)

Soit t un tableau de 4 entiers déclarés par l'instruction :

```
int tab[4]={12,5,-56,78} ;
```

Représentation mémoire de t :



V. Manipulation des tableaux

Il n'y a rien de prévu dans la librairie standard du C pour manipuler les tableaux. Vous devez coder vos propres sous-programmes de traitements des tableaux en utilisant des boucles pour en parcourir les cases.

Il n'y a pas de format permettant de lire ou afficher les valeurs d'un tableau avec un seul appel à *scanf* ou à *printf*. Pour afficher un tableau, il faut faire une boucle et afficher chaque case une par une.

Attention au débordement de tableau

Une tentative d'accès à des éléments situés hors du tableau provoque des erreurs à l'exécution. Le comportement peut varier d'une exécution à l'autre. Il peut se produire :

- Un écrasement des données voisines en mémoire provoquant des résultats erronés, des boucles infinies …
- Un plantage du programme si la zone mémoire est interdite d'accès (segmentation fault).
- Une faille de sécurité (buffer overflow) : Si le bug n'est pas détecté ni corrigé, un logiciel malveillant peut l'exploiter pour modifier le comportement du programme (par exemple exécuter du code non prévu) : c'est une porte d'entrée possible pour un piratage.

VI. Les tableaux et les sous-programmes

1. Passage de tableaux en paramètres des sous-programmes

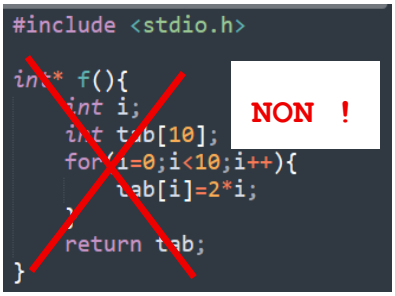
Le passage des tableaux en paramètres des sous-programmes se fait toujours par adresse. On transmet juste l'adresse du tableau (c'est-à-dire l'adresse de sa première case). Les sous-programmes ne font pas de copies des tableaux qu'on leur transmet, ils travaillent directement sur les tableaux de l'appelant.

Mais attention, le C peut être trompeur ! Les 3 prototypes suivants sont strictement équivalents :

void saisirTab(int tab[10]); ⇔ void saisirTab(int tab[]); ⇔ void saisirTab(int *tab);

2. Une fonction ne peut pas retourner un tableau !

Un tableau automatique déclaré dans un sous-programme est une variable locale qui n'existe que le temps de son exécution. L'emplacement du tableau est libéré à la fin du sous-programme. **Une fonction ne peut dons pas retourner un tableau automatique.**



Exercice 26. Faire une librairie de manipulation de tableaux de N entiers.

✓ Définir la constante symbolique N dans le fichier header.

✓ Votre librairie devra contenir au moins les sous-programmes décrits ci-dessous. Vous pourrez la compléter par la suite.

✓ **Pour chaque sous-programme :**

1. **Identifier les données en entrée.**
   Ces données sont-elles modifiées ?

   Non : passage par valeurs possibles
   Oui : passage par adresses

2. **Identifier les résultats en sortie.**

   Un seul résultat : retour de fonction
   Plusieurs résultats : passage d'adresses en paramètres

✓ Déclarer les prototypes dans le fichier header.
Avant chaque prototype, indiquer en commentaires :

- Le rôle du sous-programme
- Les paramètres
- Eventuellement : le retour de fonction

✓ Implémenter les sous-programmes dans le fichier source, sans oublier d'inclure le header.

✓ Liste des sous-programmes :

- Sous-programme qui demande à l'utilisateur de saisir les valeurs d'un tableau avec des valeurs dans un intervalle [a,b]. Le tableau ainsi que les bornes a et b sont reçus en paramètres. Le sous-programme vérifie que l'utilisateur saisit une valeur dans [a,b]. Si une valeur n'est pas correcte, il lui redemande (à blinder avec une boucle).
- Sous-programme qui remplit un tableau avec des valeurs aléatoire dans un intervalle [a,b].
- Sous-programme qui affiche sur une ligne les valeurs d'un tableau reçu en paramètre.
- Sous-programme qui détermine et retransmet à l'appelant le min et le max d'un tableau reçu en paramètre.

  ! rappel : En C, une fonction ne peut retourner qu'une seule valeur. Le sous-programme devra recevoir des adresses auxquelles il va mettre les résultats (pointeurs en paramètres).
- Sous-programme qui recherche si une valeur est présente dans un tableau. La valeur recherchée et le tableau sont reçus en paramètres. La fonction retourne 1 si la valeur est dans le tableau, 0 sinon.
- Fonction qui compare deux tableaux reçus en paramètres et retourne le nombre de cases qui ont des valeurs différentes.

Exercice 27. Jeu de mémoire

On souhaite réaliser un programme permettant à l'utilisateur de tester sa mémoire immédiate. Le programme :

- ✓ demande à l'utilisateur de saisir N nombres entiers entre des bornes a et b fixées dans le programme.
- ✓ affiche ensuite N entiers aléatoires entre a et b.
- ✓ demande à l'utilisateur de ressaisir les N entiers du début dans le même ordre.
- ✓ indique le nombre d'erreurs.

1. Conception

a. Identifier les données principales
b. Faire l'ACD

2. **Réalisation**

   Implémenter le sous-programme principal. Utiliser les sous-programmes de l'exercice précédent pour exécuter les différents traitements.
   Tester le programme.

3. **Améliorer le programme** pour qu'il propose des séries de plusieurs essais avec plusieurs niveaux de difficulté (en modifiant les bornes a et b).

VII. Les tableaux à plusieurs dimensions.

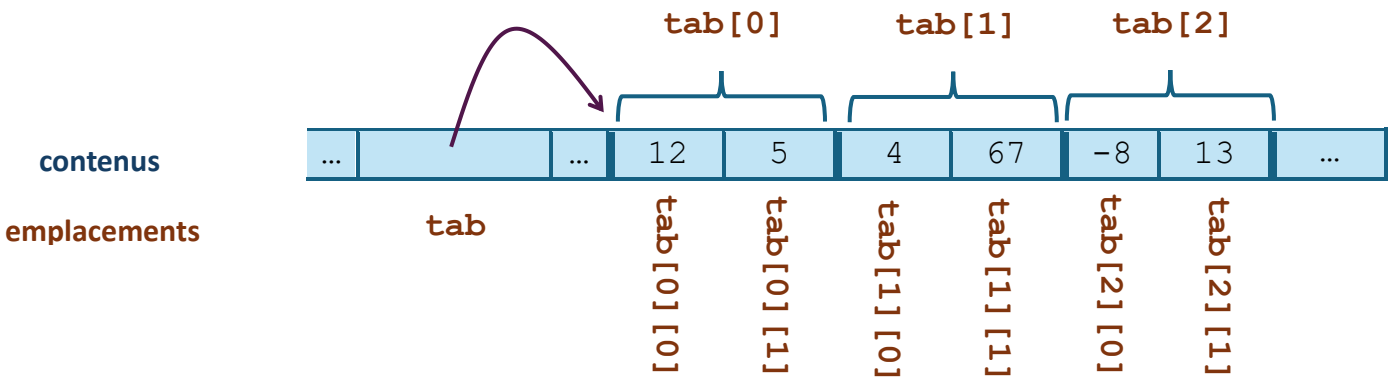
Et si les éléments d'un tableau étaient eux-mêmes des tableaux ?
C'est possible et c'est comme si on manipulait des tableaux à plusieurs dimensions.

Exemple : tableau de 3 cases dont les éléments sont eux-mêmes des tableaux de 2 entiers.

Déclaration avec initialisation :

```
int tab[3][2] = {{12,5},{4,67},{-8,13}} ;
```

Schéma mémoire :



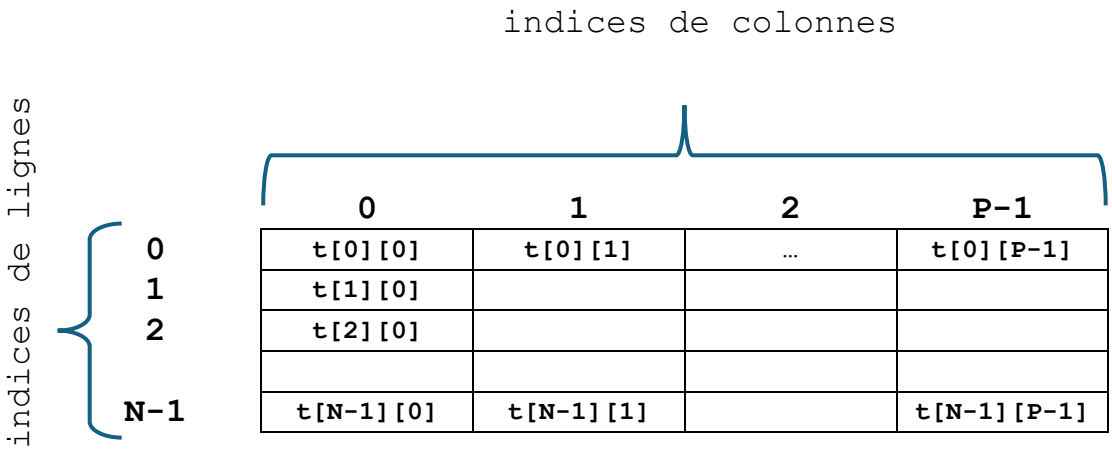
Accès au j^{ème} élément du i^{ème} tableau :

(*(tab+i)) ⇔ tab[i] permet d'accéder au i^{ème} tableau.

(((tab+i))+j) ⇔ **tab[i][j]** permet d'accéder au j^{ème} élément du i^{ème} tableau.

Représentation matricielle d'un tableau à deux dimensions.

Soit t un tableau de tableaux d'entiers : `int t[N][P];`
On peut se représenter t comme un tableau à deux dimensions de N lignes fois P colonnes.



Manipulation des tableaux à plusieurs dimensions

Il faut utiliser des boucles et souvent des boucles imbriquées.

Code exemple :

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//définitions de constantes symboliques
#define N 5
#define P 10
/*sous-programme qui remplit aléatoirement
un tableau d'entiers à deux dimensions N*P
avec des valeurs dans [0,100]*/
void remplirTab2DAlea(int tab[N][P]){
    int i,j;
    //boucle pour parcourir "les lignes"
    for(i=0;i<N;i++){
        //boucle pour parcourir "les colonnes" e la ligne i
        for(j=0;j<P;j++){
            tab[i][j]=rand()%101;
        }
    }

}
/*sous-programme qui affiche
un tableau d'entiers à deux dimensions N*P
sous forme matricielle*/
void afficherTab2D(int tab[N][P]){
    int i,j;
    for(i=0;i<N;i++){
        for(j=0;j<P;j++){
            printf("%3d ",tab[i][j]);
        }
        //passage à la ligne après avoir affiché toute une ligne
        printf("\n");
    }

}
```

```c
int main() {
    int tab[N][P];
    //initialisation pour les tirages aléatoires
    srand(time(NULL));
    //appel des sous-programmes
    remplirTab2DAlea(tab);
    afficherTab2D(tab);
    return 0;
}
```

Exemple de résultat obtenu à l'exécution :

```
 8  74  65  78  53   1  78  84  15  63
60  20  94  26  83  99  85   6  51  58
18   1  60  33   7  83  74   5  45  31
97  17  41  68  82   5  54   8  71  32
80  15   7  97  15   8  13  59  90  33
```

Exercice 28. Jouer avec des matrices.

Les tableaux à deux dimensions sont la structure de base de nombreux jeux de plateaux. Dans les jeux genre Match-3 ou Puzzle Game, les plateaux de jeu sont des grilles de cases contenant un symbole, un chiffre, un « bonbon ». Dans le code, ces grilles sont représentées par des tableaux à deux dimensions, et les actions se traduisent par des opérations sur les cases, les lignes ou les colonnes de ces tableaux.

Exemples :

Action dans le jeu	Traitements
Échanger deux cases	Permuter deux éléments du tableau
Trouver des alignements de 3	Parcourir les lignes/colonnes pour détecter des motifs
Faire "tomber" les éléments	Décaler les valeurs d'une colonne vers le bas

Dans cet exercice, vous allez coder et tester quelques-uns des traitements qu'on retrouve fréquemment dans ce genre de jeux.

1. Ecrire un sous-programme qui remplit une matrice N*P avec des entiers dans l'intervalle [0,4]
2. Prévoir deux sous-programmes d'affichage :
    - Un sous-programme qui affiche la matrice telle qu'elle avec ses valeurs entières
    - Un sous-programme qui affiche la matrice pour qu'elle ressemble à un plateau de jeu : les valeurs sont remplacées par des symboles éventuellement en couleur, ajout de bordures … (utiliser la librairie *affichage_console* fournie au chapitre 11)

3. Ecrire un sous-programme qui reçoit en paramètres les coordonnées d'une case (un numéro de ligne et un numéro de colonne) et qui remplace toutes les cases de la même ligne et de la même colonne par des 0.

4. Ecrire un sous-programme qui décalent toutes les lignes d'une matrice « vers le bas » : Les valeurs de la ligne d'indice i sont copiées dans la ligne d'indice i+1, puis la première ligne est remplie de 0.

5. Imaginer d'autres opérations …

6. Ecrire un *main()* qui teste ces différents sous-programmes.