Programmation Avancée

Cours 8 : Les tableaux

Simon Forest

25 mars 2021

Différents types de tableaux en C

```c
int tab1[5];

int main(){
  int tab2[5];
  int *tab3 = malloc(5 * sizeof(int));
  // ...
  free(tab3);
  return 0;
}
```

Tableaux statiques / tableaux dynamiques

- **tableaux statiques** : non-redimensionnables après leur création
- **tableaux dynamiques** : redimensionnables après leur création

Les tableaux statiques peuvent être stockés

- soit dans les **données globales** ;
- soit dans la **pile**.

Les tableaux dynamiques sont stockés dans le **tas**.

Tableaux comme données globales

On peut définir un tableau statique dans les données globales de plusieurs façons :

```
int tab1[5];

void f(int i){
  static int tab2[5];
  // ...
}

int main(){
  // ...
}
```

Tableaux comme données globales

Caractéristiques de ces tableaux :

- leurs tailles doivent être connues à la compilation
- existent en une seule copie
- non-redimensionnables (tableaux statiques)
- sont initialisés à 0 par défaut

Tableaux sur la pile

Les tableaux sur la pile sont définis comme des variables locales de fonctions :

```c
void f(unsigned int i){
  int tab1[5];
  int tab2[i * i];
  unsigned int n;
  scanf("%u",&n);
  int tab3[n];
  // ...
}
```

Tableaux sur la pile

Caractéristiques de ces tableaux :

- leurs tailles doivent être connues à la déclaration (flexibilité partielle)
- existent en une seule copie **par appel de fonctions** (plusieurs copies si récursivité)
- non-redimensionnables (tableaux statiques)
- **ne sont pas** initialisés à 0 par défaut

Tableaux sur le tas

Les tableaux sur le tas se font avec des `malloc` :

```
int *tab1;

int foo(){
  tab1 = malloc(5 * sizeof(int));
  int *tab2 = malloc(5 * sizeof(int));
  tab1[0] = 0;
  tab2[0] = 1;
  tab1[1] = tab2[1] = 42;
  // ...
  free(tab1);
  free(tab2);
}
```

Tableaux sur le tas

Caractéristiques de ces tableaux :

- leurs tailles doivent être précisées à l'allocation
- peuvent exister en plusieurs copies (les différentes copies seront stockées dans des structures : listes chaînées, autres tableaux, *etc.*)
- redimensionnables (tableaux dynamiques)
- **ne sont pas** initialisés à 0 par défaut

Initialisation des tableaux statiques

Par défaut, seuls les tableaux globaux sont initialisés à 0 :

```
char tab1[5];
int main(){
  char tab2[5];
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab1[i]);
  printf("\n");
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab2[i]);
}
```

```
0 0 0 0 0
16 122 69 16 -1
```

Initialisation des tableaux statiques

On peut initialiser les cases avec la syntaxe `{...}` :

```
char tab1[5] = {4,3,2,1,0};
int main(){
  char tab2[5] = {0,1,2,3,4};
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab1[i]);
  printf("\n");
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab2[i]);
}
```

```
4 3 2 1 0
0 1 2 3 4
```

Initialisation des tableaux statiques

On peut même initialiser partiellement :

```
char tab1[5] = {4,3};
int main(){
  char tab2[5] = {0,1};
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab1[i]);
  printf("\n");
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab2[i]);
}
```

```
4 3 0 0 0
0 1 0 0 0
```

Les cases avec des valeurs non spécifiées sont mises à 0.

Initialisation des tableaux statiques

Cette syntaxe permet en part. d'initialiser à 0 les tableaux sur la pile :

```
char tab1[5]; // = {} inutile ici
int main(){
  char tab2[5] = {};
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab1[i]);
  printf("\n");
  for(int i = 0; i < 5; i++)
  printf("%hhd ",tab2[i]);
}
```

```
0 0 0 0 0
0 0 0 0 0
```

Initialisation des tableaux statiques

On a une syntaxe spéciale pour les chaînes de caractères.

On peut initialiser totalement un tableau avec une chaîne :

```
char str1[7] = "Bonjour"; // pas  de '\0' à la fin
char str2[8] = "Bonjour"; // avec un '\0' à la fin
```

On peut initialiser partiellement aussi :

```
char str[100] = "Bonjour"; // complétée avec des '\0'
```

On peut laisser `gcc` trouver la bonne taille pour nous :

```
char str[] = "Bonjour"; // avec un '\0' à la fin
```

Initialisation des tableaux dynamiques

Les tableaux dynamiques ne sont pas initialisés avec `malloc` mais peuvent le paraître au début de l'utilisation du tas :

```c
int main(){
  int *tab1 = malloc(5 * sizeof(int));
  for(int i = 0; i < 5; i++) {
    printf("%d ",tab1[i]);
    tab1[i] = -1;
  }
  printf("\n");
  free(tab1);
  tab1 = malloc(5 * sizeof(int));
  for(int i = 0; i < 5; i++)
  printf("%d ",tab1[i]);
}
```

```
0 0 0 0 0
0 0 -1 -1 -1
```

Initialisation des tableaux dynamiques

Les tableaux dynamiques sont par contre initialisés avec `calloc` :

```c
int main(){
  int *tab1 = calloc(5, sizeof(int));
  for(int i = 0; i < 5; i++) {
    printf("%d ",tab1[i]);
    tab1[i] = -1;
  }
  printf("\n");
  free(tab1);
  tab1 = calloc(5, sizeof(int));
  for(int i = 0; i < 5; i++)
  printf("%d ",tab1[i]);
}
```

```
0 0 0 0 0
0 0 0 0 0
```

Initialisation des tableaux dynamiques

On peut dans tous les cas utiliser `memset` ou faire des boucles :

```c
int main(){
  char *tab1 = malloc(5 * sizeof(char));
  memset(tab1,'A',5 * sizeof(char));
  for(int i = 0; i < 5; i++)
    printf("%c ",tab1[i]);
  printf("\n");
  int *tab2 = malloc(5 * sizeof(int));
  for(int i = 0; i < 5; i++)
    tab2[i] = 4 - i;
  for(int i = 0; i < 5; i++)
    printf("%d ",tab2[i]);
}
```

```
A A A A A
4 3 2 1 0
```

Initialisation des tableaux dynamiques

On peut initialiser les `char *` par des chaînes de caractères :

```
char *str = "Bonjour";
```

Initialisation des tableaux dynamiques

On peut initialiser les `char *` par des chaînes de caractères :

```
char *str = "Bonjour";
```

Attention, le contenu des pointeurs initialisés de cette façon n'est pas modifiable :

```
str[0] = 'D'; // Erreur de segmentation
```

Initialisation des tableaux dynamiques

On peut initialiser les `char *` par des chaînes de caractères :

```
char *str = "Bonjour";
```

Attention, le contenu des pointeurs initialisés de cette façon n'est pas modifiable :

```
str[0] = 'D'; // Erreur de segmentation
```

Pour avoir la modifiabilité, faire :

```
char *str = strdup("Bonjour");
```

Tableaux statiques et dynamiques : quelques différences

Les tableaux statiques et dynamiques sont relativement interchangeables en C :

```c
void f(char *arg){
  // ...
}
void g(char arg[3]){
  // ...
}
int main(){
  char *ptr = malloc(5 * sizeof(char));
  char tab[5];
  f(tab); // OK
  g(ptr); // OK
}
```

Cependant, il y a quelques différences.

Tableaux statiques et dynamiques : quelques différences

Les variables associées à des tableaux statiques **ne sont pas modifiables** : elles se comportent comme des variables de type `t * const`.

```
int tab[5];
int *ptr = malloc(...);
tab = ptr; // ERREUR
```

Dans l'autre sens, il n'y a pas de problème.

```
int tab[5];
int *ptr = tab; // OK
```

Tableaux statiques et dynamiques : quelques différences

Les `sizeof` se comportent différemment :

- sur des tableaux statiques : **taille du contenu**
- sur des tableaux dynamiques : **taille du pointeur**

```
int tab[5];
int *ptr = tab;
printf("%ld\n",sizeof(tab));
printf("%ld\n",sizeof(ptr));
```

```
20
8
```

Tableaux statiques et dynamiques : quelques différences

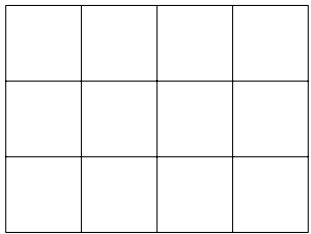
Les `sizeof` se comportent différemment :

- sur des tableaux statiques : **taille du contenu**
- sur des tableaux dynamiques : **taille du pointeur**

```
char msg[] = "Bonjour à vous !";
char *ptr  = "Bonjour à vous !";
printf("%ld\n",sizeof(msg));
printf("%ld\n",sizeof(ptr));
```

```
18
8
```

Tableaux multidimensionnels



On veut souvent manipuler des « tableaux » avec plus d'une dimension :

- matrices
- images
- plateau d'un jeu
- *etc.*

Comment faire pour les représenter en `C` ?

À partir des tableaux unidimensionnels

On peut facilement représenter un tableau 2-dimensionnel avec un tableau simple en numérotant correctement :

8	9	10	11
4	5	6	7
0	1	2	3

```
int l = 4, h = 3;
int tab[l * h];  // ou : int *tab = malloc(l * h * sizeof(int));

int tab_get(int x, int y){ return tab[y * l + x]; }
int tab_set(int x, int y, int v){ tab[y * l + x] = v; }
```

À partir des tableaux unidimensionnels

Cette astuce marche bien sûr pour d'autres dimensions :

```
int l = 4, h = 3, p = 5;
int tab[l * h * p];
// ou : int *tab = malloc(l * h * p * sizeof(int));

int tab_get(int x, int y, int z){
  return tab[z * l * h + y * l + x];
}
int tab_set(int x, int y, int v){
  tab[z * l * h + y * l + x] = v;
}
```

Avec la syntaxe C

Le C offre une syntaxe pour faire des tableaux statiques multidimensionnels :

```c
int l = 4, h = 3, p = 5;
int tab2[h][l];
int tab3[p][h][l];

int x = 2, y = 1, z = 3;
tab2[y][x]    = 3;
tab3[z][y][x] = 5;
```

Avec la syntaxe C

Le C offre une syntaxe pour faire des tableaux statiques multidimensionnels :

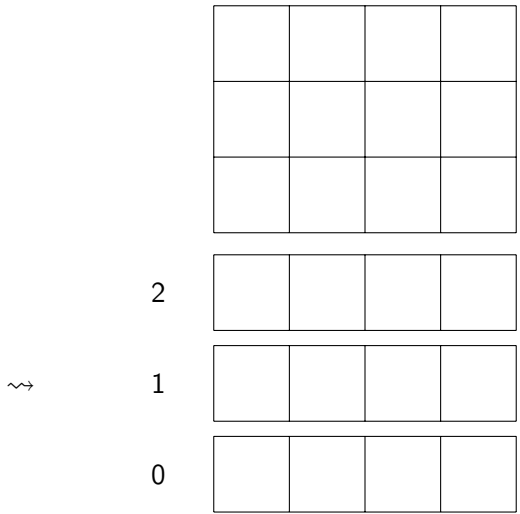
```c
int l = 4, h = 3, p = 5;
int tab2[h][l];
int tab3[p][h][l];

int x = 2, y = 1, z = 3;
tab2[y][x]    = 3;
tab3[z][y][x] = 5;
```

En « interne », c'est bien des tableaux unidimensionnels qui sont utilisés avec la méthode précédente :

```c
tab2[y][x]    = 3;
// revient à: "tab2[y * l + x] = 3"
tab3[z][y][x] = 5;
// revient à: "tab3[z * h * l + y * l + x] = 5"
```

Avec des tableaux de tableaux

On peut voir un tableau 2D comme un tableau de lignes :



Avec des tableaux de tableaux

Rappel : si `t` est un type, alors `t*` est le type des pointeurs ou des **tableaux de** `t`.

- `int` $\rightsquigarrow$ `int*`
- `char` $\rightsquigarrow$ `char*`

En particulier, si `t` est déjà un type de tableau, alors `t*` est un type de tableau de tableau.

- `t = int*` $\rightsquigarrow$ `int**`
- `t = char*` $\rightsquigarrow$ `char**`

Avec des tableaux de tableaux

Supposons que l'on ait un pointeur

```
int **ptr = ...;
```

Avec des tableaux de tableaux

Supposons que l'on ait un pointeur

```
int **ptr = ...;
```

Alors `ptr[0]`, `ptr[1]`, `ptr[2]`, ... sont de type `int*`, donc ce sont **tableaux** que l'on peut indexer aussi :

```
int *tab = ptr[2];
tab[3] = 42;
```

Avec des tableaux de tableaux

Supposons que l'on ait un pointeur

```
int **ptr = ...;
```

Alors `ptr[0]`, `ptr[1]`, `ptr[2]`, ... sont de type `int*`, donc ce sont **tableaux** que l'on peut indexer aussi :

```
int *tab = ptr[2];
tab[3] = 42;
```

On n'a même pas besoin de passer par une variable `tab` intermédiaire :

```
ptr[2][3] = 42;
```

Avec des tableaux de tableaux

Avant de pouvoir accéder aux cases du tableau, il faut faire plusieurs `malloc` :

```
int l = 4, h = 3;
int **ptr;









ptr[2][3] = 42;
```

Avec des tableaux de tableaux

Avant de pouvoir accéder aux cases du tableau, il faut faire plusieurs `malloc` :

```
int l = 4, h = 3;
int **ptr;

ptr = malloc(h * sizeof(int *));




ptr[2][3] = 42;
```

Avec des tableaux de tableaux

Avant de pouvoir accéder aux cases du tableau, il faut faire plusieurs `malloc` :

```
int l = 4, h = 3;
int **ptr;

ptr = malloc(h * sizeof(int *));

for(int lig = 0; lig < h; lig++){
  ptr[lig] = malloc(l * sizeof(int));
}

ptr[2][3] = 42;
```

Avec des tableaux de tableaux

Une fois que l'on a fini d'utiliser le tableau, il faut faire un certain nombre de `free` :

```
int l = 4, h = 3;
int **ptr;

// malloc et utilisation
// [...]

for(int lig = 0; lig < h; lig++){
  free(ptr[lig]);
}

free(ptr);
```

Avec des tableaux de tableaux

Avantage par rapport aux tableaux unidimensionnels (manuels ou syntaxe `C`) : on peut redimensionner facilement.

Avec des tableaux de tableaux

Avantage par rapport aux tableaux unidimensionnels (manuels ou syntaxe C) : on peut redimensionner facilement.

Rajouter des lignes :

```
int **ptr = ...;
// malloc et utilisation avant redimensionnement: [...]
int new_h = ...;
assert(new_h >= h);

ptr = realloc(ptr, new_h * sizeof(int *));

for(int lig = h; lig < new_h; lig++)
  ptr[lig] = malloc(l * sizeof(int));

h = new_h;
```

Avec des tableaux de tableaux

Avantage par rapport aux tableaux unidimensionnels (manuels ou syntaxe C) : on peut redimensionner facilement.

Rajouter ou enlever des colonnes :

```
int **ptr = ...;
// malloc et utilisation avant redimensionnement: [...]
int new_l = ...;

for(int lig = 0; lig < h; lig++)
  ptr[lig] = realloc(ptr[lig], new_l * sizeof(int));

l = new_l;
```