

# 1 Tableaux dynamiques à plusieurs dimensions

L'allocation **dynamique** permet trivialement de créer des tableaux à 1 dimension.

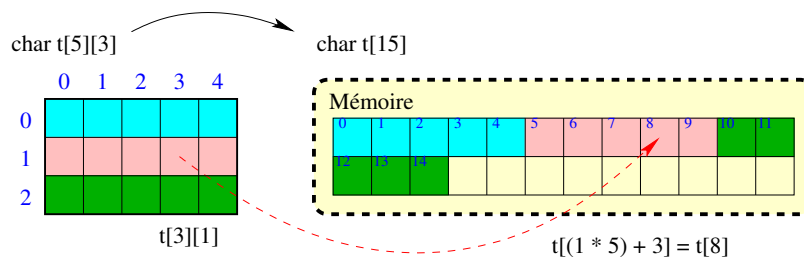
On peut également construire des tableaux à **plusieurs dimensions**.

Il y a **2** manières de procéder. On illustre sur des tableaux à 2 dimensions `char t[DIM1][DIM2]`, mais ça se généralise.

## 1.1 Représentation linéaire

On fait un tableau à **1 dimension** de taille `DIM1 * DIM2` chars.

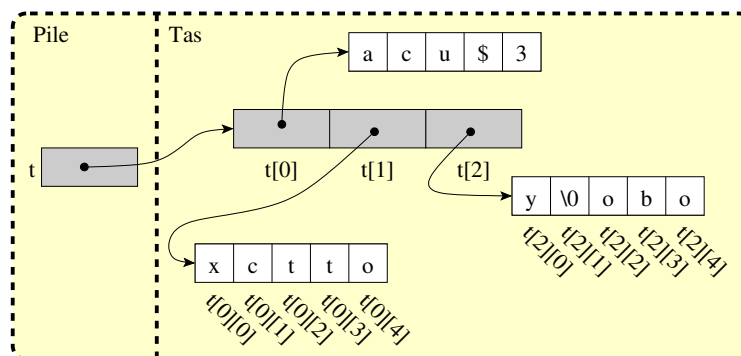
On **linéarise** l'adressage :  $t[x][y] \rightarrow t[y * DIM1 + x]$ .



## 1.2 Représentation par tableau de pointeurs

On fait un tableau à **1 dimension** de `DIM1` pointeurs.

Chaque pointeur pointe sur un tableau de `DIM2` chars.



# 2 Passage d'argument par adresse

Une fonction ne peut **retourner** qu'**une seule** valeur au plus.

On voudrait parfois **plusieurs valeurs** résultat.

En C, les paramètres sont passés par **valeur** :

1. Ils sont **évalués**,
2. ils sont **copiés** sur la pile avant l'appel.

⇒ Impossible de modifier une variable de l'appelant depuis l'appelé.

Pour s'en sortir, dans la fonction **appelante** on va passer l'**adresse** des **variables dans lesquelles stocker** (« retourner ») les résultats.

Dans la fonction **appelée**, l'argument reçu est une **adresse**, donc le paramètre est un **pointeur**.

```
#include <stdio.h>

void f (int *v)
{
    ▶ (*v)++ ;
}

int main ()
{
    int x = 5 ;
    f (&x) ;
    printf ("x = %d\n" , x) ; // On attend 6 comme affichage.
    return (0) ;
}
```

Un tableau étant un **pointeur** sur la **1<sup>ère</sup> case**, il est **forcément** passé par adresse.

### 3 Débugger son programme

Mettre des **printf** dans un programme pour le débbugger est une technique **très efficace**, surtout lorsque le programme ne plante pas brutalement.

En cas de plantage brutal (*core dumped, segmentation fault*), utiliser le debugger **gdb**.

Nécessite d'avoir compilé le programme avec l'option **-g** de **gcc**.

Lancer **gdb** en terminal : **gdb fichier-exécutable**.

Si l'exécutable a besoin d'arguments en ligne de commande, lancer **gdb fichier-exécutable**, puis sous **gdb**, entrer la commande : **set args arg1 arg2 ... argn**.

**Point d'arrêt** : instruction à laquelle stopper l'exécution (arrêt **avant** exécution).

Mettre un point d'arrêt : **break location**

où *location* = *n-ligne* ou *fichier:n-ligne* ou *nom-fonction* ou *fichier:nom-fonction*.

Lancer le programme : **run**.

Exécuter instruction suivante (sans suivre les appels de fonction) : **next**.

Exécuter instruction suivante (en rentrant dans les appels de fonction) : **step**.

Inspecter les variables ou expressions : **print expression-C**.

Charger un fichier **core** : **gdb fichier-exécutable fichier-core**.