§2. Tableaux statiques

Pour l'instant, données :
- → char
- → int
- → float, double

Plusieurs données ....

Tableaux statiques :
- ⇝ stocker N (fixe, constante définie et connue à l'écriture du code) données (dans des cases) de même types
- ⇝ accès aux données : via l'indice de la case immédiat

≠

tableaux dynamiques : taille décidée "à la volée" en cours d'exécution du code ⟷ §4 pointeurs

I. Syntaxe

① Déclaration des tableaux

↳ au même endroit que la déclaration de variables
||
début du main

Syntaxe : `type nom [ taille ];`
- type : des cases
- taille : constante

ex : tableau T de N caractères

```
int main ()
{
    :
    char T[N];
    :
```

constante :
- ~~#define~~ (à éviter)
- constantes "macro"
- variables constantes

1) Macros : juste après les `#include` .... (pré-compilateur) → inclut le fichier ds le code

⇝ `#define nom_cst val`

↳ recherche/remplace

ex:

```
#include <stdio.h>
#define N 20
int main()
{
    char T[N]; // T : tableau de N caractères
    :
    N = 30;
```

→ #define : recherche/remplacement

⚠ variables → minuscules

macros (#define) → majuscules

2) Rendre des variables constantes :

`const` devant la déclaration → initialisation à la déclaration

ex:

```
int main()
{
    const int n = 20;
    char T[n]; // correct
```

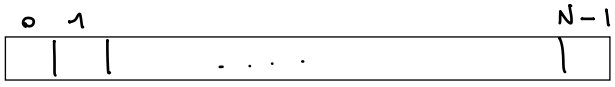
Pour initialiser un tableau à la déclaration :

déclaration : type nom[taille] = { case 1, case 2, ... };

EN GENERAL ⚠

ex: `char T[N] = {'a', 'b', 'c'}; // complétion par des 0 pour les cases non initialisées`

② Affectation / accès



T
taille N

Cases numérotées à partir de 0

Accès à la case i : T[i]

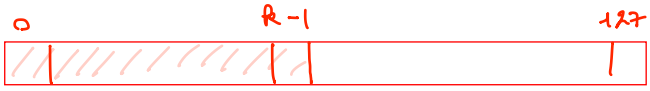
```
x = T[i];
T[i] = 5;
```

⚠ ~~tableau vide~~

tableau ≠ taille allouée / déclarée (constante, fixe / N)

≠ taille utilisée (autre variable)

variable ici : k



N = 128

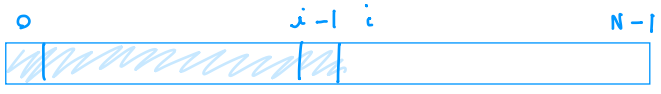
L'utilisateur a rentré (k) données

Ex :
- tableau de taille N = 128
- faire saisir des valeurs entières ≥ 0 à l'utilisateur
  → stockées dans les cases
  ↳ fin de la saisie : valeur < 0 saisie ①
- calculer le min / max des données saisies ②

① Pseudo-code pour la saisie

Variables
- N : constante (128)
- T : tableau taille N
- i : indice case saisie
- x : variable entière dans laquelle on stocke la saisie (buffer)
- m : nombre de val. saisies

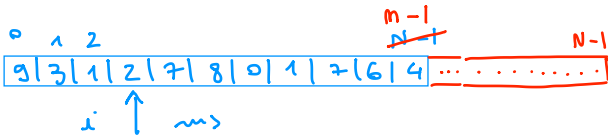

```
i ← 0
afficher ("saisir un entier >= 0")
saisir_entier → x
tantque (x >= 0) && (i <= N-1)        // éviter le débordement : T taille N
                 ou (i < N)
    T[i] ← x
    i ← i+1
    saisir_entier → x
ftq
```



m = i

```
while alternatif :
do {
   ⋮
} while (cond);
```

② Min / max de T
↳ case initialisée



Variables
- N : constante
- T : tableau de taille N / m cases utilisées
- i : parcourt les indices
- tmp : minimum des val. rencontrées jusqu'à la case i-1
  ↓
  meilleur min jusqu'à la case i

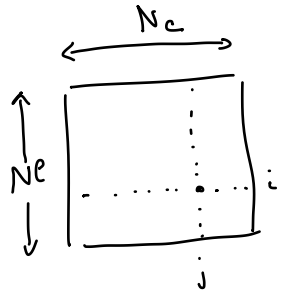
⚠ FAUX

```
variable
tmp ← T[0]
pour i de 0 à N-2
    si (T[i+1] < T[i])
        tmp ← T[i+1]
    fsi
```

Code

```
tmp ← (+∞) T[0]
pour i de 1 à n-1
    si (T[i] < tmp)
        tmp ← T[i]
    fsi
fpour
```

③ Tableaux bi-dimensionnels



Syntaxe déclaration : `type nom [nlignes][ncol];` (constantes ⚠)

accès : `nom[i][j]` ⟶ case ligne i / col j

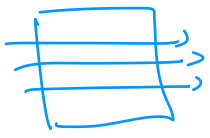
ex :

```
int main()
{
    const int Nl = 3, Nc = 2;
    (*) double M[Nl][Nc];
    ⋮
    M[0][1] = 3.5;
```

initialisation à la déclaration : `{ }` ou `{{ }, … { }}`

Π : $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

(*) ⇝ `double M[Nl][Nc] = {{1,2},{3,4},{5,6}};`



ou remplissage ligne par ligne

⇝ `double M[Nl][Nc] = {1,2,3,4,5,6};`

II. Algorithmique

Chaînes de caractères

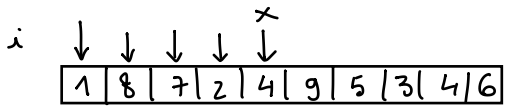
① Min / Max → cf. avant

Complexité
$\mathcal{O}(N)$ → meilleur cas ? X
→ en moyenne
→ pire cas ? X

② Recherche un élément

Tableau T $x \in T$?
élément

On veut que le code renvoie → n° case si x présent
→ -1 si x pas présent



$x = 4$
$x = 0$

```
j ← -1
pour i de 0 à N-1
    si (T[i] == x)
        j ← i
    fsi
fpour
```

efficacité

→ plusieurs fois x ?
renvoie la dernière occurrence

```
j ← -1
i ← 0
tantque ((T[i] ≠ x) && (i < N))
    i ← i+1
ftq
// trouvé ou pas ?
i < N
j ≠ -1
i == N
j == -1
```

```
si (T[i] == x)
    j ← i
fsi
```

meilleur cas : $x = T[0]$ → $\mathcal{O}(1)$
→ moyenne : $\mathcal{O}(N)$
→ pire cas : $x = T[N-1]$ → $\mathcal{O}(N)$

→ meilleur cas : X
→ moyenne : $\mathcal{O}(N)$
→ pire cas : X

```
j ← -1
i ← 0
tantque ((j == -1) && (i < N))
    si (T[i] == x)
        j ← i
    fsi
ftq
```

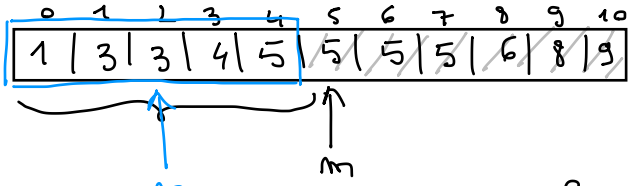
```
si (j == -1)
    → pas trouvé
sinon → trouvé case j
```

Pour aller plus vite :

→ supposer que T est trié (croissant)

Recherche dichotomique

↳ couper en 2 ...



$x = 3$?

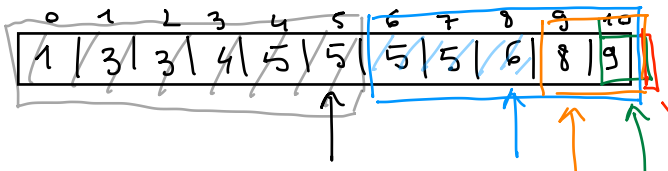
$x = 10$?

m ≠ <

milieu (m) : $\frac{0+10}{2} = 5$

Etape 0

$m = \frac{0+4}{2} = 2$ → trouvé



Etape 0 ⇝ $m = 5$

$x = 10$?

≠

$x > T[5]$

vide

Etape 1

$m = \frac{6+10}{2} = 8$

m

$x \neq T[m]$

>

Etape 2

$m = \left\lfloor \frac{9+10}{2} \right\rfloor = 9$

$x \neq T[m]$

>

Etape 3

m

$x \neq T[m]$

>

Etape 4

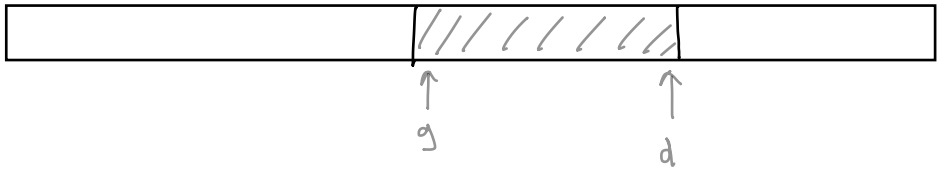
→ sous-tab vide

X

à chaque étape on considère une portion de T

g, d

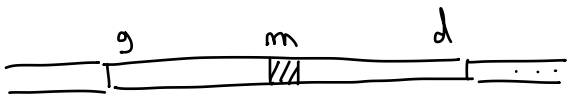
indices 1ère / dernière cases



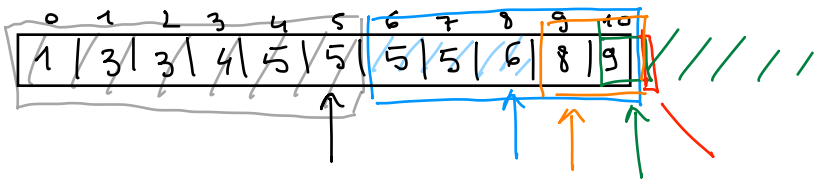
```
g ← 0
d ← N-1
trouvé ← FAUX
tantque ( (trouvé == FAUX) && (g ≤ d) )
    m ← ⌊ (g+d)/2 ⌋
    si (x == T[m])
        trouvé ← VRAI
    sinon
        si (x < T[m]) // chercher à gauche
            d ← m-1          // g ne m-1
        sinon // x > T[m]  ~> chercher à droite
            g ← m+1   <-- (*)
        fsi
    fsi
```

portion g ns d vide

nbre de répétitions ? <-> complexité



dernière étape :



x = 10 ?

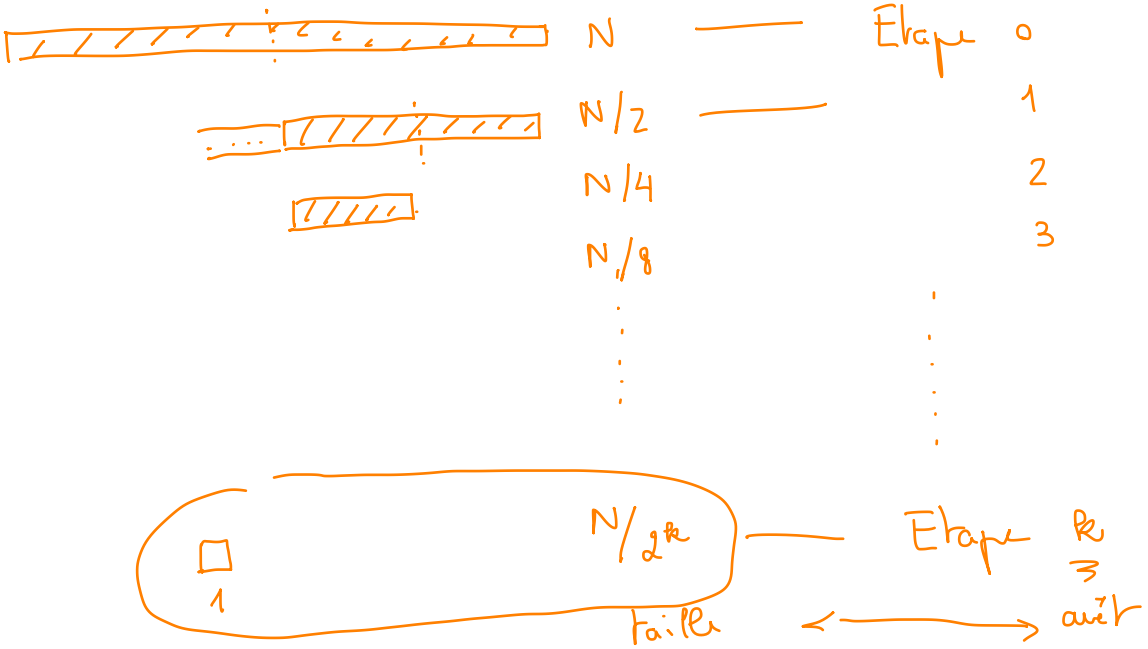
g = d = m (= 10)

g ← m+1 = 11

d = 10

g	ns	d
11		10

sous-tableau vide



Arrêt si : $\frac{N}{2^k} \sim 1 \iff N \sim 2^k$

$\iff \ln N \sim k \ln 2$

$\iff k \sim \frac{\ln N}{\ln 2} = \log_2 N$

$\mathcal{O}(\log_2 N)$

temps

1s → 10^6 op.

compl. \ taille	10^3	10^6	10^9
$\mathcal{O}(N)$	10^3 10^{-3}s	10^6 1s	10^9 10^3s = 16 min 40
$\mathcal{O}(N^2)$	10^6 1s	~~10^{12}~~ 277 h (11) 10^6s	~~10^{18}~~ 10^{12}s ~ 11 Mj
$\mathcal{O}(1) \sim \mathcal{O}(\log_{10} N)$	3	6	9
	3.10^{-6}s	6.10^{-6}s	9.10^{-6}s

③ Codage des chaînes de caractères en C

~~string~~

"abc" → chaîne de caractères

↓

codées par :

→ tableau de caractères

→ avec un '\0' ajouté à la dernière case

a	b	c	\0

Ce codage est connu de printf / scanf

↳ %s

ex:

```
int main()
{
    const int N = 128;
    char T[N];
    scanf("%s", T);
    printf("message lu : %s\n", T);
}
```

T : chaîne de caract ≡ tableau de caract.

≡ pas de & devant un tableau de caractères.

④ Trier un tableau

Algos "simples" / non récursifs

⊖ $O(N^2)$

⊕ meilleurs cas en $O(N)$

- → tri sélection / échange
- → tri par insertion
- → tri bulle (***)

Algos "avancés" / récursif

complexité : $O(N . \log N)$ ⊕

pas du meilleur cas ⊖

- → quicksort
- → tri fusion