VIII.
Variables indexées

1. Les variables indexées (tableaux)

Exemple de la vie courante :

- courbe annuelle des températures (1D) : 365 valeurs
- échiquier (2D) : case E-6
- Appartement (3D) : (n° cage d'escalier, étage, n° appartement)

Représentation en C :

- un seul identificateur
- le nombre d'éléments est fixé à la compilation
- les éléments sont tous de même type (et occupent la même place)

Syntaxe pour définir un tableau :

type identificateur[*expression*$_{const}$] ... [*expression*$_{const}$] ;

Exemples :

```
float temperaturesAn[366] ;
char echiquier[8][8] ;

#define HAUT 960
#define LARG 640
unsigned char ecranIphone3[HAUT][LARG][3] ;
/* valeurs RGB d'un pixel dans 3 char */
```

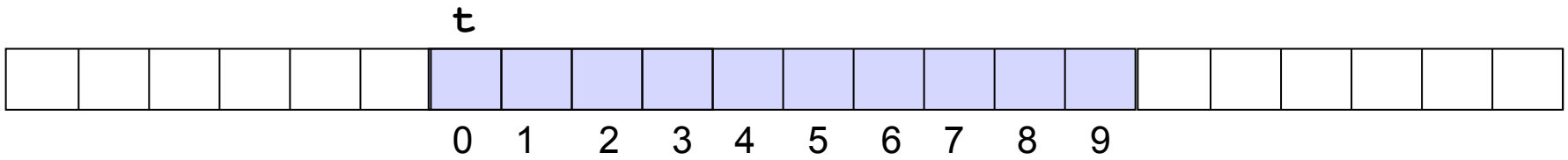
2. Opérateur [ ]

```
int t[10];
```

Représentation interne d'un tableau :

- les cellules sont physiquement consécutives,
- la première est à l'indice 0,
- accès à un élément : *identificateur* **[** *rang* **]**

  `t[0] , t[1] , …, t[9]`
- l'indice (ou rang) indique le nbre de cellules à sauter / début



- le C ne fait aucun contrôle sur la validité des indices…

Exemples :

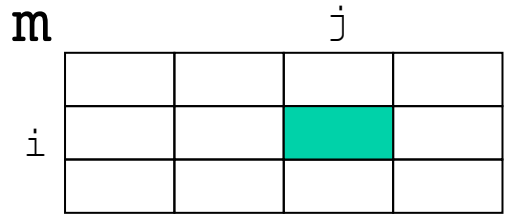
- *Saisie d'un tableau de 10 notes et calcul de la moyenne*

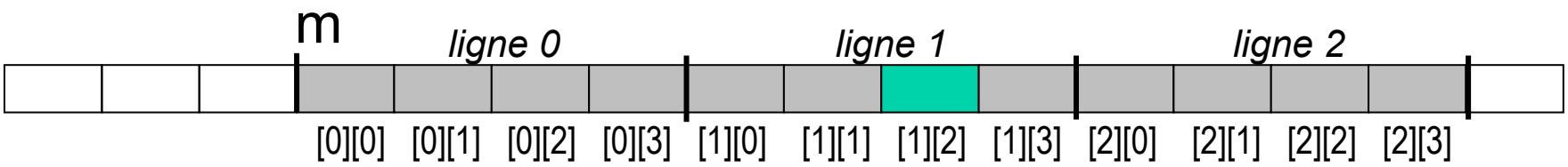
Exemple pour une matrice

```
#define NB_LI 3
#define NB_COL 4
int m[NB_LI][NB_COL];
```

- les cellules sont rangées ligne par ligne dans la mémoire,
- accès à un élément : m[*n° ligne*] [*n° colonne*]

  case (i, j) de la matrice : `m[i][j]`





Exemple :

- *Schéma général de parcours des cases d'une matrice*

3. Initialisation et affectation

Les tableaux peuvent être initialisés

- à la définition
- et par une suite d'expressions constantes

```
int t[5] = {0, 0, 1 } ;
int m[5][2] = {{10,20}, {30,40}, {50,60} } ;
```

ATTENTION : l'affectation globale des tableaux n'est pas permise

➔ boucle de recopie !

```
int t1[3] = {0, 0, 1}, t2[3], i ;
t2 = t1 ;
for (i=0 ; i<3 ; i++)
  t2[i] = t1[i] ;
/*fin-for*/
```

4. Quelques compléments

a) Optimisation des expressions logiques

Rappel : `||` et **`&&`** ont une associativité gauche-droite

Le C optimise les calculs des opérations logiques :

`expression1` **`&&`** *`expresion2`*

- *`expression2`* n'est évaluée que si *`expression1`* est **`vrai`**

`expression1` **`||`** *`expresion2`*

- *`expression2`* n'est évaluée que si *`expression1`* est **`faux`**

Exemple : *Recherche du 1er élément non nul dans un tableau*

Exemple :

```
    #define MAX 10
    int t[MAX], i;
    …
/* on suppose t déjà initialisé */
/* recherchons le premier élément non nul */
    i=0;
    While(t[i]==0 && i<MAX)      :-(
          (i<MAX && t[i]==0)     :-)
      i++ ;
/* fin tant que */
/* quelle est la condition fausse parmi les 2 ?*/
    if (i<MAX)
      traiter t[i]
    else
      pas d'élément non nul
```

b) Construction de type

Construire de nouveaux types avec le mot **`typedef`**

- Syntaxe :
  `typedef` *définition*
- Exemple :

```
typedef double Matrice1[10][20] ;
Matrice1 A, B;
```

- Ou encore :

```
typedef double Ligne[20];
typedef Ligne Matrice2[10];
Matrice2 C;
```

Résumé

- Macros :

```
#define  MAX  100
```

- Définir des tableaux :

```
char t[MAX] ;          // cellules numérotées de 0 à MAX-1
float m[10][20] ;      // matrice 10 lignes et 20 colonnes
```

- Schéma type de parcours d'une matrice :

```
for(i=0 ; i<10; i++)
   for(j=0 ; j<20 ; j++)
   { traiter m[i][j] }
```

- Optimisation des expressions logiques :

expr1 **&&** *expr2* *expr2* n'est évaluée que si *expr1* est **vrai**

expr1 **||** *expr2* *expr2* n'est évaluée que si *expr1* est **faux**

- Définir de nouveaux types :

```
typedef float Matrice[10][20] ;  // canevas
Matrice A, B;          // 2 matrices de float 10 x 20
```