Initiation au C

Cours 3

David Simplot

1. Types et tableaux

- Définition de nouveaux type :

typedef *definition_type nom_type*;

■ *Exemple*

```
typedef int MonInt;
typedef unsigned char Byte;
typedef signed char Boolean;
```

Il s'agit d'un renommage des types. Par exemple, une variable de type `Byte` peut être utilisée en lieu et place d'un `unsigned char`.

■

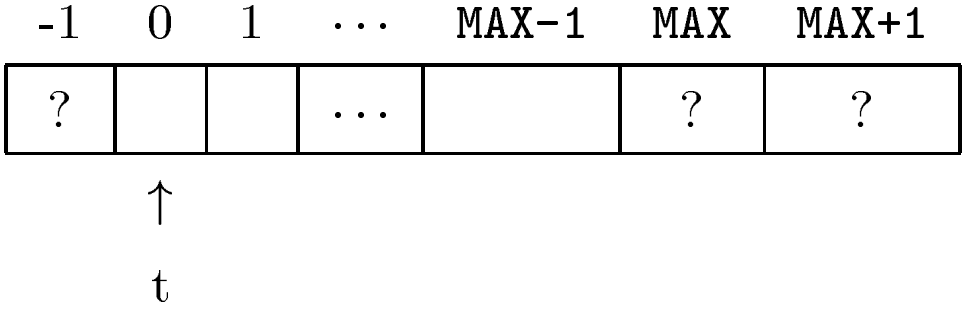
1. Types et tableaux (2)

- Tableaux à une dimension :

■ *Exemple*

```
#define MAX 100
...
int t[MAX];
```

– `t` est un pointeur vers le premier élement du tableaux, son type est `const int *`, il ne peut être modifié ;
– `t[i]` est le `i+1` unième élément du tableau (quel que soit `i`) :



– La **déclaration** d'une **variable** comme tableau **réserve** l'espace mémoire et la variable est un pointeur qui pointe vers cette zone mémoire réservée ;

Une **définition** de **paramètre** ne réserve pas de zone mémoire.

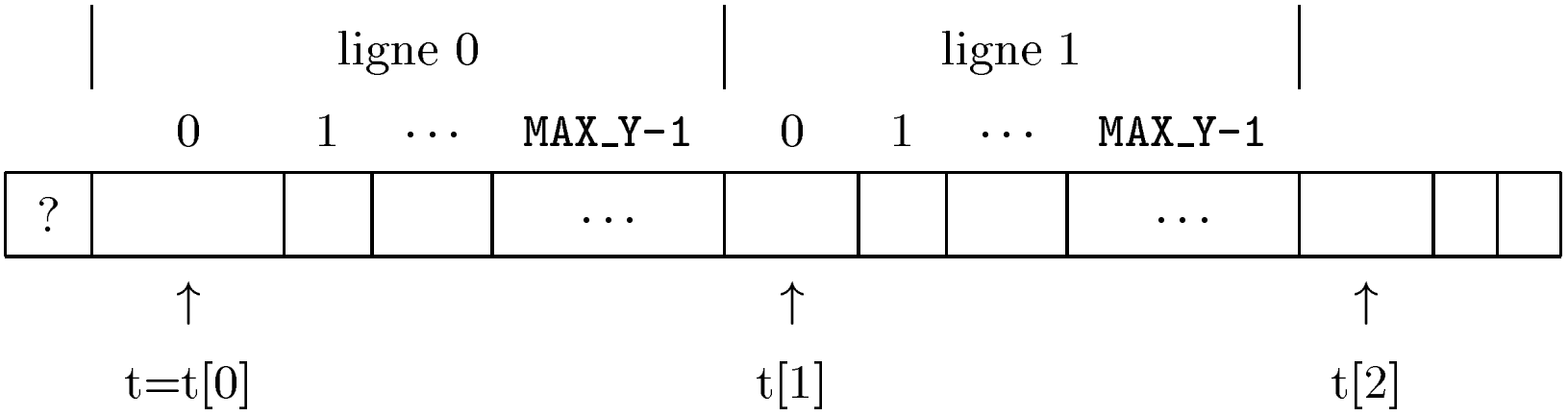
1. Types et tableaux (3)

- Tableaux multidimensionnels :

  ■ *Exemple*

```
#define MAX_X 10
#define MAX_Y 200
...
int t[MAX_X][MAX_Y];
```

  – `t` est un pointeur vers le premier élément du tableau (il pointe vers `t[0][0]`) ;
  – `t[i][j]` est le j+1 unième élément de la i+1 unième ligne de `t` ;
  – `t[i]` est un pointeur (il est considéré comme un tableau de MAX_Y entiers) vers le premier élément de la i+1 unième ligne :


ligne 0
ligne 1
0
1
···
MAX_Y-1
0
1
···
MAX_Y-1
?
···
···
t=t[0]
t[1]
t[2]

1. Types et tableaux (4)

Exercice

Sachant que `t` pointe à l'adresse 1515, vers quelle adresse pointe `t[i]` (où `i` est un entier signé) ? Vers quelle adresse pointe `t+i` ? Quelle est l'adresse de l'élément `t[i][j]` où `i` et `j` sont des entiers signés ?

♣

On voit que `MAX_X` n'intervient pas pour trouver l'adresse de `t[i][j]`. C'est pourquoi dans un **passage de paramètres**, on peut omettre la première dimension...

1. Types et tableaux (5)

- Nouveau types avec des tableaux

■ *Exemple*

```
#define MAX 100
...
typedef int MesTableaux[MAX];
typedef unsigned char Byte;
typedef Byte ArrayByte[MAX][MAX];
```

Toute variable de type `MesTableaux` sera un tableau d'entiers de taille `MAX`.

■

1. Types et tableaux (6)

- Nouveaux types avec des pointeurs

Exemple

```
#define MAX 100
...
typedef int *PtInt;
typedef char *TabString[MAX];
```

2. Structures

- Types composés : les structures

```
struct tag
{
    type1 champ11, champ12...;
    type2 champ21, champ22,...;
    ...
};
```

2. Structures (2)

Exemple

```
#include <stdio.h>

struct coordonnees
{
  int x, y;
};

int main(void)
{
  struct coordonnees a;
...
```

2. Structures (3)

- On accède aux champs grâce à l'opérateur ".".

■ *Exemple*

```
#include <stdio.h>

struct coordonnees
{
  int x, y;
};

int main(void)
{
  struct coordonnees a;
```

```
    a.x = 0;
    a.y = 1;

    return(0);
}
```

2. Structures (4)

- Structure en paramètre in/out :
  - *Exemple*

```
#include <stdio.h>

struct coordonnees
{
  int x, y;
};

void init_coordonnees(struct coordonnees *c)
{
  (*c).x = 0;
  c->y = 0;
```

```
}

int main(void)
{
  struct coordonnees a;

  a.x = 0;
  a.y = 1;

  return(0);
}
```

2. Structures (5)

- Nouveau type utilisant des structures :

■ *Exemple*

```
struct _coordonnees
{
  int x, y;
};
typedef struct _coordonnes Coordonnees;
```

ou

```
typedef struct _coordonnees
{
  int x, y;
} Coordonnees;
```

■

2. Structures (6)

- Contrairement à un tableau, une structure peut être utilisée en valeur de retour d'une fonction :

■ *Exemple*

```
#define MAX 100
typedef int TabInt[MAX];

TabInt mafonction(int x)
{
  TabInt tmp;   int i;
  for ( i=0 ; i<MAX ; i++ )
    tmp[i] = x;
  return(tmp);
}
```

```
int main(void)
{
  TabInt t;

  t = mafonction(0);

  return(0);
}
```

■

2. Structures (7)

- Avec des struct :

```
#define MAX 100
typedef struct {
  int valeur[MAX]; } TabInt;

TabInt mafonction(int x)
{
  TabInt tmp;  int i;
  for ( i=0 ; i<MAX ; i++ )
    tmp.valeur[i] = x;
  return(tmp);
}
```

```
int main(void)
{
  TabInt t;

  t = mafonction(0);

  return(0);
}
```