Types de données, tableaux et tris

Introduction

Du point de vue du programme, une variable a trois attributs : un **identificateur**, un **type**, une **valeur**. Du point de vue du compilateur, la variable a en plus une **adresse**, cette notion fondamentale sera vue ultérieurement. Par exemple :

```
int nombre;
```

déclare une variable de type entier, d'identificateur nombre et de valeur non définie.

Comme nous allons le voir, le **type** détermine la taille de la place mémoire réservée pour la variable et le genre d'opérations qui peuvent être effectuées sur cette variable.

Nous manipulerons ensuite un groupe de données sous la forme de tableaux et on s'intéressera aux problèmes de tris.

A. Types de données

Jusqu'à présent, nous n'avons utilisé que des variables entières de type int. Il existe d'autres types de données, mais il faut garder à l'esprit qu'on peut les répartir en deux catégories :

- **variables entières** (char, int, long)
- **variables "flottantes"** (float, double)

A.I. Types entiers

Ils sont au nombre de trois et ce qui les distingue dans un premier temps, c'est la taille qu'ils occupent en mémoire. Par défaut, les types sont signés, c'est-à-dire que les nombres peuvent être positifs ou négatifs (en Code Complément à 2 ou CC2, cf. cours d'électronique numérique). Si on souhaite uniquement des valeurs positives, par exemple, dans le cas du traitement d'images (pixels d'intensité uniquement positive), on peut spécifier le mot-clé **unsigned**.

Type	Char	short int	(long) int
Taille en mémoire (cas des ordinateurs 32bits)	1 octet	2 octets	4 octets
Intervalle des valeurs signées (CC2)	[-128 ;127]	$[-2^{15} ; 2^{15} - 1]$	$[-2^{31} ; 2^{31} - 1]$
Intervalle des valeurs non signées (unsigned)	[0 ;255]	$[0 ; 2^{16} - 1]$	$[0 ; 2^{32} - 1]$
Affichage	**%c pour le caractère (%d pour valeur ASCII)**	**%d**	**%d**

Tip 1 : Dans le cas « unsigned », l'affichage se fait en utilisant « %u ».

Cas particulier du type char

Même si c'est avant tout un entier, on le réserve fréquemment pour la **représentation de caractères**, car un caractère est représenté par un **code ASCII** qui est un nombre compris entre 0 et 127. (il existe une table étendue dont les codes s'étendent de 128 à 255).

Do it 1: Code ASCII
➢ **Donnez le code ASCII (valeur décimale) de la phrase suivante : J'aime l'ENSEA !**
➢ **Donnez le code ASCII (valeur hexadécimale) de la phrase suivante : J'aime l'ENSEA !**

Do it 2 : Conversion minuscule-MAJUSCULE.	
➢ Ecrire une fonction permettant de convertir une minuscule en majuscule. Le prototype est le suivant : ➢ `char minToMAJ(char c) ;`	Aide : regarder la table ASCII
➢ Ecrire une fonction principale utilisant minToMAJ. Attention, LES FICHIERS SONT SEPARES !!!	
➢ Ecrire une fonction permettant de convertir une majuscule en minuscule. Le prototype est le suivant : `char MAJTomin(char c) ;`	
➢ Ecrire une fonction qui convertit, selon son argument d'entrée, une minuscule en majuscule et une majuscule en minuscule. Utiliser les fonctions précédentes. `char convertCasse(char c) ;`	
➢ Tester ces 3 fonctions.	

Tip 2 : Les initialisations suivantes sont équivalentes : char i = 65 ; ⇔ char i = 0x41 ; ⇔ char i = 'A' ;

A.II. Types flottants et conversions de types

Les types **float**, **double** et **long double** représentent les nombres réels, dits en virgule flottante, stockés respectivement en simple, double et haute précision.

Type	Float	double	long double
Taille en mémoire	4 octets	8 octets	10 octets
Affichage	%f	%lf	%lf

Conversion explicite ou transtypage :

La conversion explicite permet d'augmenter (ou de diminuer) la précision lors d'un calcul.

Question 1 : Quels seraient les valeurs affichées à l'écran si on demande l'affichage des variables suivantes ?

```
int x = 5.5;
int y = 4/5;
float z = 4/5;
```

Pour appliquer la conversion explicite, il suffit de préciser avant le nombre ou la variable le type désiré entre parenthèses. Par exemple, nous avons :

```
float z = (float)4/(float)5;

int test(int);  prototype de la fonction test
double nb=5.7;  nb est une variable de type double
test((int) nb); conversion explicite du type de nb
```

Question 2 : Comment écrire en C les expressions de la question 1 pour obtenir le résultat voulu. Faites rapidement un programme pour vérifier vos propositions.

Conversion implicite :

Lorsque plusieurs types sont impliqués dans un calcul, le résultat tient compte de la plus grande précision.

Question 3 : A votre avis, que donne : `float z = (float)4/5;` ? Faites rapidement un programme pour vérifier le résultat.

B. Tableaux statiques

Un tableau est un regroupement de variables de même type, de même nature et ce sous le même identificateur (nom du tableau). Par exemple, les notes des élèves (même nature) peuvent être incluses dans un tableau de variables réelles (même type, un type *float* suffirait sauf pour M. Seigneurbieux). Pour accéder aux différentes cases du tableau, on utilise des indices qui différencient chaque case (et donc chaque note).

La déclaration d'un tableau se fait de la façon suivante :

```
type nom_tableau[D1][D2]…[DN]
```

où D1,…DN sont des entiers correspondant à la taille des N dimensions du tableau.

Voici deux exemples de déclaration de tableaux :

```
int tableau1d[10];
char tableau2d[10][15] ;
```

Pour un tableau à une dimension nommé `tableau1d` de taille 10, les indices vont de 0 à 9, les cases vont donc de `tableau1d[0]` à `tableau1d[9]`.

Tip 3 : l'utilisation des boucles avec les tableaux est presque inévitable et vivement recommandée.

Tip 4 : la définition d'une étiquette (`#define T 100`) permet lors d'un changement de taille de ne modifier que l'étiquette. Elle est donc obligatoire, sinon vivement recommandée !

B.I. Tableaux et mémoire

En mémoire, les cases d'un tableau correspondent à des adresses placées **« successivement ».** La place occupée par chaque case dépend de la déclaration du tableau. Par exemple, dans le cas d'un tableau d'entiers (*int tableau[10];)*, chaque case occupera 4 octects. Si ce tableau avait été déclaré avec l'expression suivante : *char tableau[10];*, chaque case occuperait 1 octet.

Do it 2 : Visualiser un tableau en mémoire	
➢ Dans une fonction principale, déclarer un tableau d'entier, de nom tab et de taille 10.	
➢ Initialiser le tableau aux 10 premiers entiers naturels.	
➢ Sauvegarder et compiler.	
➢ Lancer DDD. Mettre un point d'arrêt et lancer le programme.	
➢ Visualiser le tableau sous forme de variable.	Double cliquer sur la variable directement dans le code.
➢ Visualiser le tableau avec les adresses mémoires	Dans le menu « Data », cliquer sur « Memory ». Dans le menu qui s'affiche : - Remplir la case vide située après « Examine » par 10. - Choisir « decimal » pour le format - Choisir « words (4) » pour la taille - Remplir la case vide située après « from » par le nom de votre tableau (tab).
➢ Suivre l'initialisation du tableau en mode pas à pas.	

Question 3 : Effectuer plusieurs affichages de la mémoire en jouant sur le format (décimal, hexa,…) et sur la taille (bytes, words…). Comprendre :

- le lien qui lie chaque case du tableau (adresses successives)
- la représentation des données selon le format choisi
- la représentation des données selon la taille choisie

B.II. Exercises using 1D-arrays

Do it 3: Static 1D-arrays and functions

- Create a new project (files : main.c, functions.c an functions.h)
- In the functions header, define a label name SIZE equal to 10.
- In the main function, create a 1D array (named tab1D) of float elements using SIZE.

- Create a function that initializes the array with zeroes. The prototype should be:

```
void TAB_init(float tab[], int dim);
```

Where `tab` is the array to display and `dim` the size of the array

- Create a display function that shows all the numbers of the array with a precision of 2 digits (%.2f). The prototype should be:

```
void TAB_display(float tab[], int dim);
```

Where `tab` is the array to display and `dim` the size of the array

- Test both functions with the array previously created.

- Create a function that fills the array with increasing numbers from 1 to SIZE. The prototype should be:

```
void TAB_fill_increase(float tab[], int dim);
```

Where `tab` is the array to display and `dim` the size of the array

- Test the function with the array previously created by displaying "Array filled by increasing numbers" and the array.

- Create a function that fills the array with random numbers within 1 to 100. The prototype should be:

```
void TAB_fill_random(float tab[], int size);
```

Where `tab` is the array to display and `size` the size of the array

- Test the function with the array previously created by displaying "Array filled by random numbers" and the array.

- Create a function that copies the array to another one. The prototype should be:

```
void TAB_copy(float tab_dest[], float tab_source[], int dim);
```

Where `tab_source` is the array to copy, `tab_dest` is the destination array and `dim` the size of the array

- In the main function, create another 1D array of float elements using SIZE.
- Test the function with both arrays previously created by displaying "Copy an array to another" and both arrays.

- Create a function that sums all the elements of the array. The prototype should be:

```
float TAB_sum(float tab[], int dim);
```

Where `tab` is the array containing the elements to sum and `dim` the size of the array. The function returns the sum value.

- Test the function with the first array by displaying "The sum of the array is" and the result.

- Create a function that computes the mean all the elements of the array. The prototype should be:

```
float TAB_mean(float tab[], int dim);
```

Where `tab` is the array containing the elements to average and `dim` the size of the array. The function shall use the TAB_sum function and returns the mean value.

- Test the function with the first array by displaying "The mean of the array is" and the result.

Do it 4: A guided sorting algorithm

- Keep using the previous project.
- Create a function that finds the minimum value of all the elements of the array. The prototype should be:

```
float TAB_min(float tab[], int dim);
```

Where `tab` is the array in which the min value is searched and `dim` the size of the array. The function returns the minimum value.

- Test the function with the first array by displaying "The minimum value of the array is" and the result.
- Create a function that finds the index of the minimum value of all the elements of the array. The prototype should be:

```
int TAB_index_min(float tab[], int dim);
```

Where `tab` is the array in which the index of the min value is searched and `dim` the size of the array. The function returns the index of the minimum value.

- Test the function with the first array by displaying "The index of the minimum value of the array is" and the result.
- Create a function that finds the index of the minimum value of elements within a range given by a start and a stop indices. The prototype should be:

```
int TAB_index_min_range(float tab[], int dim, int start_index, int stop_index);
```

Where `tab` is the array in which the index is searched, `dim` the size of the array, `start_index` the index of the element to begin with and `stop_index` the index of the element to finish with. The function returns the index of the minimum value within the range given.

- Test the function with the first array by displaying "The index of the minimum value within the range 3 and 6 is" and the result.
- Create a function that switches two elements given their indices. The prototype should be:

```
void TAB_switch(float tab[], int dim, int index1, int index2);
```

Where `tab` is the array in which the switching occurs, `dim` the size of the array, `index1` the index of the first element and `index2` the index of the second element.

- Test the function with the first array by displaying "After switching the 2nd and 5th elements, " and the result.
- Create a function that sorts by ascending values. You shall use both previous functions. The prototype should be:

```
void TAB_sort_asc(float tab[], int dim);
```

Where `tab` is the array in which the switching occurs, `dim` the size of the array.

- Test the function with the first array by displaying "Sorting by ascending " and the result.
- Create a function that sorts by descending values. The prototype should be:

```
void TAB_sort_desc(float tab[], int size);
```

Where `tab` is the array in which the switching occurs, `size` the size of the array.

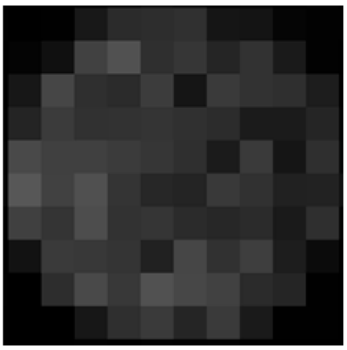
- Test the function with the first array by displaying "Sorting by ascending " and the result.

Do it 5: From 1D-array to 2D array and vice-versa

- Use the previous project.
- In the main function, create a 1D-array which size is NBCOL*NBLIG and a 2D-array which size is NBLIG and NBCOL over the two directions.
- Write a function that converts a 1D-array to a 2D-array. Let's fill the 2D-array line by line. Test.
- Write a function that converts a 2D-array to a 1D-array. Let's read the 2D-array column by column. Test.

C. Introduction to image processing: using 2D-arrays

An **image** can be represented with a 2D-array (NBLIG,NBCOL) that contains integer values. This image will then have NBLIG lines and NBCOL columns. Each array element is assimilated to a **pixel**.



56	48	56	85	112	156	111	111	83	63
48	80	121	139	129	119	117	132	108	70
58	88	131	127	122	131	115	127	127	128
69	101	114	118	135	101	121	112	129	121
82	169	142	166	175	117	86	135	102	92
80	164	160	138	135	122	109	114	105	99
86	145	138	106	90	129	123	113	85	140
79	137	163	206	149	171	99	147	109	118
52	54	194	202	152	166	133	149	159	136
57	49	123	102	105	128	125	139	140	93

Figure 1: Image and pixels

Question 4: How many pixels are there in the image?
Question 5: In C language, which are the indices representing the first and last line, respectively column?

The intensity of the pixel is the integer value of the array element. When speaking of 8-bit images, intensities are within the range 0 and $2^8 - 1$, *i.e.* 255.

Question 6: What is the range for a binary image (1 bit)? And for a 3-bit image?

The histogram of an image is a graphical representation that gives for each intensity the number of pixels whose value corresponds.

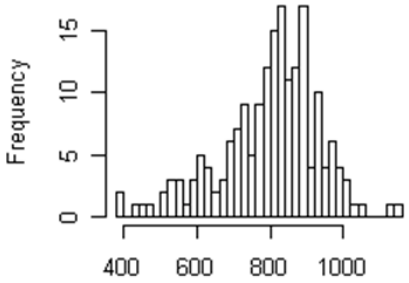


Figure 2: Histogram of an image

In border pixels are not considered, each pixel P has 8 neighbours in an **8-connexity neighbourhood**.

X	X	X
X	P(i,j)	X
X	X	X

Figure 3: 8-connexity neighbourhood

Question 7: Considering that the current pixel is located i^{th} line, j^{th} column, what are the indices of the neighbours of P?

Do it 6: Creating and initializing 3-bit images

- **Create a new project (files : main.c, functions.c an functions.h)**
- **In the functions header, define two labels names NBLIG and NBCOL respectively equal to 20 and 15.**
- **In the main function, create a 2D array of char elements using NBLIG and NBCOL. This array represents our**

image.

- Write a function to initialize the image. The prototype should be:

```
void init(char image[NBLIG][NBCOL],char value);
```

Where `image` represents the image to initialize and `value` represents the value that the image is initialized with.

- Write a function to display the image. The prototype should be:

```
void display(char image[NBLIG][NBCOL]);
```

Where `image` represents the image to display.

- Test

Do it 7: Copying images

- Write a function to copy an image. The prototype should be:

```
Void copy(char image[NBLIG][NBCOL], char copy[NBLIG][NBCOL],);
```

Where image represents the image to be copied and copy represents the copied image.

- In the main function, create another 2D array of char elements using NBLIG and NBCOL. This array represents our copy.
- Test.

Do it 8: Drawing inside images

- Write a function to draw a rectangle inside the image. The prototype should be:

```
int drawRect(char image[NBLIG][NBCOL],int x, int y, int h, int w, char val);
```

Where :

- image represents the image in which a rectangle has to be drawn
- x and y are the coordinate of the top left corner
- h and w are the size of the rectangle
- val is the intensity of the rectangle

The function returns 0 if an issue occurred when drawing the rectangle (issues to be determined) and 1 if successful.

- Test.

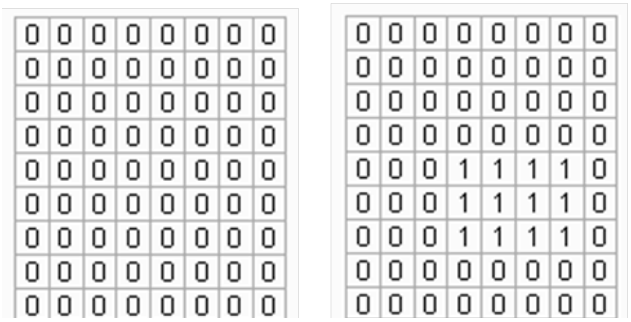


Figure 4: Drawing a rectangle inside an empty array (x=3, y=4, w=4, h=3)

Do it 9: Computing histogram

- Write a function to compute the histogram of the image. The prototype should be:

```
void drawHist(char image[NBLIG][NBCOL);
```

Where :

- image represents the image in which the histogram is computed

The function just shows the intensity alongside with its frequency. For example:

0:5
1:3
2:8
...

- Test.

In the following part, we will use a reference image that we are going to process in order to have a new one. Each function will have two arguments : the reference array (reference image), and another array

which has the same size as the reference one in order to contain the values after processing (processed image).

The P symbol represents a pixel in the reference image and the P' symbol represents a pixel in the processed image. In order to avoid borderlines issues, the processing operations will be only consider for indices starting from 1 to NBLIG-1 or NBCOL-1.

Do it 10: Creating a reference image and the processed array

- In the main function, create a reference image (first initialized with zeroes) which contains five rectangles:
  - One rectangle centred, whose width = 8, height = 6 and intensity = 8
  - Four squares whose sides = 4 with top left points located at (1,1), (NBCOL-5,1), (1,NBLIG-5) , (NBCOL-5,NBLIG-5) and intensities respectively equal to 1, 3, 5 and 7.
- In the main function, create an empty array whose goal is to contain the values after processing (processed image).
- Display the reference image.

Do it 11: Mean filtering

- Write a function to compute locally (8-connexity) a mean and then replace the original value by it. For each pixel P, the mean is computed using P and its 8 neighbours to have P'. In image processing, this operation is equivalent to apply to the reference image the following structural element:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Below is given an example of mean processing of a pixel:

$$\begin{matrix}2 & 2 & 3\\ 4 & \mathbf{5} & 6\\ 7 & 3 & 1\end{matrix} \longrightarrow 4$$

P=5 is the current pixel to be processed. The mean value computed is 3.66667 rounded to 4 since an image contains only integer values.

The prototype should be:

```
void FILTER_mean(char reference[NBLIG][NBCOL], char filtered[NBLIG][NBCOL]);
```

Where :

- `reference` represents the image to be processed
- `filtered` contains the values computed

- Display the filtered image.

Question 8 : According to you, is the mean filtering operation a low-pass or high-pass filter ?

Do it 12: Laplace filtering

- Write a function to compute locally (8-connexity) a value using the following structural element:

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Below is given an example of Laplace processing of a pixel:

$$\begin{matrix}2 & 2 & 3\\ 4 & \mathbf{2} & 6\\ 7 & 3 & 1\end{matrix} \longrightarrow 12$$

P=2 is the current pixel to be processed. The value computed is 1x2 + 1x2 + 1x3 + 1x4 - 8x2 + 1x6 + 1x7 + 1x3 + 1x1=12. Be careful that a 3-bit image only contains value from 0 to 8. P' is the equal to 8 (max value) and not 12.

The prototype should be: `void FILTER_laplace(char reference[NBLIG][NBCOL], char filtered[NBLIG][NBCOL]);` **Where :** - `reference` **represents the image to be processed** - `filtered` **contains the values computed**
➢ **Display the filtered image.**

Question 9 : According to you, is the Laplace filtering operation a low-pass or high-pass filter ?

In the following Do it, we are going to introduce morphological operations processing.

Do it 13: Morphological operations
➢ **In the main function, create a reference binary image (first initialized with zeroes) which contains one square centred whose sides = 6 and intensity = 1.**
➢ **In the main function, create an empty array whose goal is to contain the values after processing (processed image).**
➢ **Write a function that computes locally (8-connexity) the minimum. The prototype should be:** `void MORPH_erode(char reference[NBLIG][NBCOL], char filtered[NBLIG][NBCOL]);`
➢ **Test it and apply it successively four times starting from the reference image (you have to use the copy function between each erosion processing in order to be able to apply it successively). Observations?**
➢ **Write a function that computes locally (8-connexity) the maximum. The prototype should be:** `void MORPH_dilate(char reference[NBLIG][NBCOL], char filtered[NBLIG][NBCOL]);`
➢ **Test it and apply it successively four times starting from the reference image (you have to use the copy function between each dilatation processing in order to be able to apply it successively). Observations?**

Conclusion

A la fin de ce TD, vous devez maîtriser les différents types de données, ainsi que l'utilisation des tableaux statiques. Il vous faut aussi avoir compris ce qui se passe dans la mémoire lorsqu'on utilise des variables individuelles ou des tableaux.