



Éléments d'informatique – Cours 5. Tableaux.

Pierre Boudes

12 octobre 2011



This work is licensed under the *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License*.



printf/scanf (1)

Mémoire et tableaux

La mémoire, les variables, les octets

Tableaux

Exemples et trace

Pour aller plus loin

Démos

En TP



printf/scanf (1)

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction (autant de paramètres supplémentaires que de %). Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.



printf/scanf (1)

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction (autant de paramètres supplémentaires que de %). Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.
- Exemples :
 - `printf("Bonjour\n")` affiche Bonjour et un saut de ligne



printf/scanf (1)

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction (autant de paramètres supplémentaires que de %). Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.
- Exemples :
 - `printf("Bonjour\n")` affiche Bonjour et un saut de ligne
 - `printf("i vaut %d\n", i)` affiche i vaut suivi de la valeur décimale de i (et d'un saut de ligne)



printf/scanf (1)

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction (autant de paramètres supplémentaires que de %). Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.
- Exemples :
 - `printf("Bonjour\n")` affiche Bonjour et un saut de ligne
 - `printf("i vaut %d\n", i)` affiche i vaut suivi de la valeur décimale de i (et d'un saut de ligne)
 - `printf("(%d, %d)\n", 31, -4)` affiche (31, -4) et un saut de ligne. Remarquez qu'il y a deux paramètres en plus



printf/scanf (1)

- Pour afficher un texte à l'écran, nous utilisons la fonction **printf** (*print formatted*).
- Chaque % dans le texte à afficher est substitué par la valeur formatée d'un **paramètre supplémentaire** de la fonction (autant de paramètres supplémentaires que de %). Le caractère suivant le symbole % détaille la conversion à utiliser. La conversion %d met une valeur au format **entier décimal**.
- Exemples :
 - `printf("Bonjour\n")` affiche Bonjour et un saut de ligne
 - `printf("i vaut %d\n", i)` affiche i vaut suivi de la valeur décimale de i (et d'un saut de ligne)
 - `printf("(%d, %d)\n", 31, -4)` affiche (31, -4) et un saut de ligne. Remarquez qu'il y a deux paramètres en plus
- Réciproquement pour faire entrer dans le programme une donnée saisie par l'utilisateur, nous utiliserons **scanf**.
- Exemple : `scanf("%d", &x)`



La mémoire, les octets

- La mémoire vive, ou mémoire de travail est un dispositif électronique dans lequel sont stockées les données en cours de traitement. Les données y sont codées en binaire (comme dans le reste de l'ordinateur), à l'aide de bits (0 ou 1) regroupés en **octets** (groupes de 8 bits).



La mémoire, les octets

- La mémoire vive, ou mémoire de travail est un dispositif électronique dans lequel sont stockées les données en cours de traitement. Les données y sont codées en binaire (comme dans le reste de l'ordinateur), à l'aide de bits (0 ou 1) regroupés en **octets** (groupes de 8 bits).
- Du point de vue logiciel la mémoire se présente comme une succession d'octets, numérotés par les entiers à partir de 0. La mémoire est ainsi un grand tableau, dont chaque case (ou cellule) renferme un octets. Les numéros sont les adresses des cases.



La mémoire, les octets

- La mémoire vive, ou mémoire de travail est un dispositif électronique dans lequel sont stockées les données en cours de traitement. Les données y sont codées en binaire (comme dans le reste de l'ordinateur), à l'aide de bits (0 ou 1) regroupés en **octets** (groupes de 8 bits).
- Du point de vue logiciel la mémoire se présente comme une succession d'octets, numérotés par les entiers à partir de 0. La mémoire est ainsi un grand tableau, dont chaque case (ou cellule) renferme un octets. Les numéros sont les adresses des cases.

Adresses :	0	1	2	...
octets (valeurs) :	01000110	11010111	00000001	...



Mémoire et variables (rappels) ✎

- Déclarer une variable a pour effet de réserver de la mémoire et de lui donner un usage particulier pour la suite du programme :



Mémoire et variables (rappels)

- Déclarer une variable a pour effet de réserver de la mémoire et de lui donner un usage particulier pour la suite du programme :
 - La déclaration `int toto;` aura pour effet de réserver l'espace mémoire nécessaire au stockage d'un entier.



Mémoire et variables (rappels)

- Déclarer une variable a pour effet de réserver de la mémoire et de lui donner un usage particulier pour la suite du programme :
 - La déclaration `int toto;` aura pour effet de réserver l'espace mémoire nécessaire au stockage d'un entier.
 - Dans la suite du programme, l'adresse de cet espace mémoire sera utilisée partout où il est fait référence à cette variable (identificateur `toto`).



Mémoire et variables (rappels)

- Déclarer une variable a pour effet de réserver de la mémoire et de lui donner un usage particulier pour la suite du programme :
 - La déclaration `int toto;` aura pour effet de réserver l'espace mémoire nécessaire au stockage d'un entier.
 - Dans la suite du programme, l'adresse de cet espace mémoire sera utilisée partout où il est fait référence à cette variable (identificateur `toto`).
 - C'est le codage machine des entiers en binaire qui sera employé pour manipuler cette donnée.



Mémoire et variables (rappels)

- Déclarer une variable a pour effet de réserver de la mémoire et de lui donner un usage particulier pour la suite du programme :
 - La déclaration `int toto`; aura pour effet de réserver l'espace mémoire nécessaire au stockage d'un entier.
 - Dans la suite du programme, l'adresse de cet espace mémoire sera utilisée partout où il est fait référence à cette variable (identificateur `toto`).
 - C'est le codage machine des entiers en binaire qui sera employé pour manipuler cette donnée.

Remarque.

La taille d'un `int` est en principe exactement celle d'un mot mémoire, c'est à dire 4 ou 8 octets. Nous verrons au cours suivant d'autres types de données, leurs tailles et codages. Quoi qu'il en soit, le compilateur prend en charge ces aspects et nous aurons rarement à nous en soucier en programmant.



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3];
```



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3];
```

Adresses :	...	344				348				352				...
Valeur :	...	1 ... 0			0...1			1...1			...			



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3];
```

Adresses :	...	344				348				352				...
Valeur :	...	1 ... 0			0 ... 1			1 ... 1			...			

- C'est ce qu'on appelle un tableau, statique, unidimensionnel.



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3];
```

Adresses :	...	344				348				352				...
Valeur :	...	1 ... 0			0 ... 1			1 ... 1			...			

- C'est ce qu'on appelle un tableau, statique, unidimensionnel.
 - Sa taille doit être connue au moment de la compilation (statique)



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3] ;
```

Adresses :	...	344				348				352				...
Valeur :	...	1 ... 0			0...1			1...1			...			
Identificateur :	...	toto[0]			toto[1]			toto[2]			...			

- C'est ce qu'on appelle un tableau, statique, unidimensionnel.
 - Sa taille doit être connue au moment de la compilation (statique)
 - Les cases sont accessibles, comme s'il s'agissait de variables, à l'aide des identificateurs `toto[0]`, `toto[1]`, `toto[2]`



Tableaux et mémoire

- En C, on peut réserver plusieurs espaces mémoires contigus pour des données de même type en une seule déclaration :

```
int toto [3] ;
```

Adresses :	...	344				348				352				...
Valeur :	...	1...0			0...1			1...1			...			
Identificateur :	...	toto[0]			toto[1]			toto[2]			...			

- C'est ce qu'on appelle un tableau, statique, unidimensionnel.
 - Sa taille doit être connue au moment de la compilation (statique)
 - Les cases sont accessibles, comme s'il s'agissait de variables, à l'aide des identificateurs `toto[0]`, `toto[1]`, `toto[2]`
 - La numérotation commence à zéro. Si n est le nombre de cases du tableau la dernière case est donc numérotée $n - 1$.



Attention !

Il ne faut jamais accéder à une case au delà de la numérotation : `toto[3]`, `toto[-1]`, etc. Le compilateur ne vous préviendra pas de votre erreur, mais le programme va boguer.



Attention !

Il ne faut jamais accéder à une case au delà de la numérotation : `toto[3]`, `toto[-1]`, etc. Le compilateur ne vous préviendra pas de votre erreur, mais le programme va boguer.

L'erreur d'exécution `segmentation fault` signifie que le programme a effectué un accès à une case mémoire qui ne lui était pas réservée (mais il faut beaucoup s'écarter des bons indices du tableau).



Premier exemple



Premier exemple

```
int main()
{
    /*Declaration et initialisation de variables*/
    int tableau[3];

    tableau[0] = 3;
    tableau[1] = 5;
    tableau[2] = tableau[0] + tableau[1];

    return EXIT_SUCCESS;
}
```



Premier exemple

```
int main()  
{  
    /*Declaration et initialisation de variables*/  
    int tableau[3] = {3,5,8};  
  
    tableau[0] = 3; ← inutile  
    tableau[1] = 5; ← inutile  
    tableau[2] = tableau[0] + tableau[1]; ← inutile  
  
    return EXIT_SUCCESS;  
}
```



Second exemple

```
int main()
{
    /* Declaration et initialisation de variables */
    int tab[3] = {3,5,8};
    int i; /* var. de boucle */
```



Second exemple

```
int main()
{
    /* Declaration et initialisation de variables */
    int tab[3] = {3,5,8};
    int i; /* var. de boucle */

    for (i = 0; i < 3; i = i + 1) /* pour chaque case */
    {
        printf("tab[%d]=%d\n", i, tab[i]);
    }
    return EXIT_SUCCESS;
}
```



Trace du second exemple

```
1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }
```



Trace du second exemple

```
1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }
```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
-------	--------	--------	--------	---	--------------



Trace du second exemple

```
1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }
```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	



Trace du second exemple

```
1 int main()
2 {
3     /* Declaration et initialisation de variables */
4     int tab[3] = {3,5,8};
5     int i; /* var. de boucle */
6
7     for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8     {
9         printf("tab[%d]=\t%d\n", i, tab[i]);
10    }
11    return EXIT_SUCCESS;
12 }
```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	



Trace du second exemple

```
1 int main()
2 {
3     /* Declaration et initialisation de variables */
4     int tab[3] = {3,5,8};
5     int i; /* var. de boucle */
6
7     for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8     {
9         printf("tab[%d]=\t%d\n", i, tab[i]);
10    }
11    return EXIT_SUCCESS;
12 }
```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\u00d\u00d\u00d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	
9					tab[1] = 5



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	
9					tab[1] = 5
10				2	



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	
9					tab[1] = 5
10				2	
9					tab[2] = 8



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	
9					tab[1] = 5
10				2	
9					tab[2] = 8
10				3	



Trace du second exemple

```

1  int main()
2  {
3      /* Declaration et initialisation de variables */
4      int tab[3] = {3,5,8};
5      int i; /* var. de boucle */
6
7      for (i = 0; i < 3; i = i + 1) /* pour chaque case */
8      {
9          printf("tab[%d]=\t%d\n", i, tab[i]);
10     }
11     return EXIT_SUCCESS;
12 }

```

ligne	tab[0]	tab[1]	tab[2]	i	sortie écran
initialisation	3	5	8	?	
7				0	
9					tab[0] = 3
10				1	
9					tab[1] = 5
10				2	
9					tab[2] = 8
10				3	
11	Renvoi EXIT_SUCCESS				



Pour aller plus loin ✎

- La taille d'un tableau statique gagne à être fixée par une constante symbolique (`#define N 3`).



Pour aller plus loin

- La taille d'un tableau statique gagne à être fixée par une constante symbolique (`#define N 3`).
- L'identificateur du tableau (*ie* `tab` dans la déclaration `int tab[3];`) est lui même une variable. Sa valeur est l'adresse de la première case du tableau `tab[0]`.



Pour aller plus loin ~~✎~~

- La taille d'un tableau statique gagne à être fixée par une constante symbolique (`#define N 3`).
- L'identificateur du tableau (*ie* `tab` dans la déclaration `int tab[3];`) est lui même une variable. Sa valeur est l'adresse de la première case du tableau `tab[0]`.
 - Les variables dont la valeur est une adresse s'appellent des **pointeurs** ;
 - La notation *esperluette*, `&x`, donne accès à l'adresse d'une variable ;
 - La notation *étoile*, `*tab`, ne s'applique qu'à une adresse, elle donne alors accès à la valeur contenue à cette adresse.
 - Les expressions `tab[i]` et `*(tab + i)` sont identiques en C.

○
○
○
○○
○

Démos



Interrogation (durée 1h)

Soient deux tableaux d'entiers ligne et colonne, initialisés à des valeurs de votre choix de l'intervalle $[0, 4]$. Les tailles de ces deux tableaux seront fixées par des constantes symboliques, respectivement N et M .

Écrire un programme qui :

- affiche la somme de chaque case du tableau ligne avec chaque case du tableau colonne ;
- compte le nombre de fois où cette somme vaut 5 et affiche le résultat à l'écran.



Interrogation (durée 1h)

Soient deux tableaux d'entiers ligne et colonne, initialisés à des valeurs de votre choix de l'intervalle $[0, 4]$. Les tailles de ces deux tableaux seront fixées par des constantes symboliques, respectivement N et M .

Écrire un programme qui :

- affiche la somme de chaque case du tableau ligne avec chaque case du tableau colonne ;
- compte le nombre de fois où cette somme vaut 5 et affiche le résultat à l'écran.

Les interrogations de TP seront un peu plus faciles !