Éléments d'informatique – Cours 5. Tableaux.

Pierre Boudes

12 octobre 2009

La mémoire (rappels)

- 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*.

La mémoire (rappels)

- 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*.
- 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 (rappels)

- 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*.
- 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.

Remarque.

La taille d'un `int` est en principe exactement celle d'un mot mémoire, cest 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.

Mémoire et tableaux

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

Mémoire et tableaux

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

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

Mémoire et tableaux

- 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				...

Mémoire et tableaux

- 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.

Mémoire et tableaux

- 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 dimension (ou taille) doit être connue au moment de la compilation (statique)

Mémoire et tableaux

- 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 dimension (ou 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]`

Mémoire et tableaux

- 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 dimension (ou 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 accèder 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 accèder 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.

Le bogue le plus classique (quand on a de la chance) : `segmentation fault`. Cela signifie que le programme a effectué un accès à une case mémoire qui ne lui était pas réservée.

Premier exemple

Premier exemple

```c
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

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

Second exemple

```c
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] = %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] = %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] = %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] = %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] = %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] = %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

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	
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] = %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] = %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] = %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	Renvoie EXIT_SUCCESS				

Pour aller plus loin

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

Pour aller plus loin

- La dimension 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 dimension 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 (les pointeurs) seront vues en détail au second semestre.
  - 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.

Interrogation (durée 1h)

Soient deux tableaux d'entiers `lignes` et `colonnes`, initialisés à des valeurs de votre choix de l'intervalle $[0, 4]$. Les dimensions 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 `lignes` et `colonnes`, initialisés à des valeurs de votre choix de l'intervalle $[0, 4]$. Les dimensions 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 !