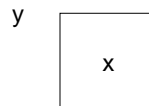


Pointeurs et tableaux

Pointeur

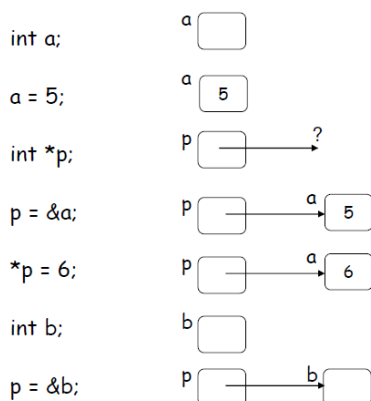
- Tout mot (octet) de la mémoire est identifié par un numéro unique : son adresse mémoire
- On peut donc identifier toute zone mémoire servant au stockage d'une variable par :
 - Son adresse de début
 - Sa taille (qui dépend du type de la variable)
- Notation utilisée :



représente un objet dont l'adresse est y et dont la valeur est x

- Cette valeur peut elle-même être une adresse.

Déclaration de variable

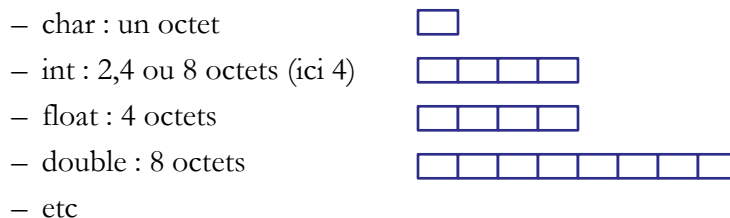


MIAGE - Université de Bordeaux

3

Occupation mémoire

- A toute variable créée est associée une zone de la mémoire, servant à stocker le contenu de cette variable
- La taille de cette zone mémoire dépend du type de la variable considérée :



MIAGE - Université de Bordeaux

4

Opérateur sizeof

- l'opérateur sizeof donne la taille en octets du type ou de la variable passée en paramètre
 - Ce n'est pas une fonction normale
 - Évaluée et remplacée par la constante entière correspondante lors de la compilation

```
printf ("Sur mon système, un int fait %d octets\n",
        sizeof (int)); /* Taille d'un type */
double d;
printf ("Sur mon système, un double fait %d octets\n",
        sizeof (d)); /* Taille d'une variable du type */
```

Tableaux (1)

- Un tableau est une collection de variables de même type que l'on peut accéder individuellement par leur indice dans le tableau
- On déclare la taille du tableau entre « [] »

```
int a[10]; /*définit un vecteur de 10 entiers consécutifs */
```
- On accède à un élément d'un tableau en donnant l'indice de l'élément entre « [] »
- En C, les indices commencent à 0

```
a[3]
```

 désigne la valeur du 4^{ème} élément du tableau a

Tableaux (2)

- On peut initialiser le contenu d'un tableau lors de sa déclaration

- Liste de constantes du type adéquat, entre accolades
- Nombre d'éléments comptés par le compilateur

```
int t[7] = {2,4,7,5,6,3,1};
```

```
int t[]={2,4,7,5,6,3,1}; /*le compilateur comptera*/
```

- La liste peut n'être que partielle :

```
int t[7] = {2,4,7,5}; /*7 places prises, 4 initialisées */
```

MIAGE - Université de Bordeaux

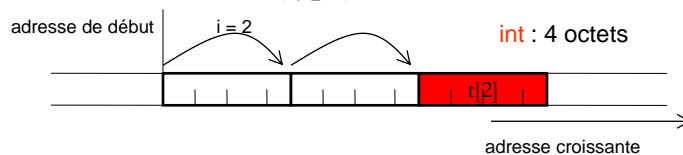
7

Tableaux (3)

- Si t est de type type , $t[i]$ est donc le contenu de la zone mémoire :

- Commencant à l'octet d'adresse $(t + i * \text{sizeof}(\text{type}))$

- De taille $\text{sizeof}(\text{type})$



MIAGE - Université de Bordeaux

8

Tableaux et pointeurs (1)

- Un tableau est un pointeur sur une zone mémoire de variables contiguës de même type
On peut utiliser un nom de tableau comme pointeur

```
int t[7]; /* Tableau de 7 entiers */
int *r; /* Pointeur sur un entier */
r = t; /* t est en fait de type (int *) */
a = t[0]; /* Lecture du premier élément du tableau */
b = *r; /* Comme r pointe sur t, on lit aussi t[0] */
c = *t; /* Comme t est un (int *), ceci marche aussi */
t = r; /* Ne marche pas ! T est une constante ... */
```

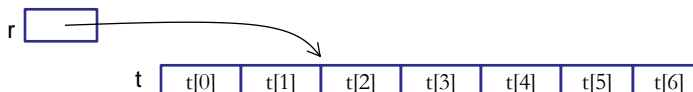
MIAGE - Université de Bordeaux

9

Tableaux et pointeurs (2)

- Tableaux et pointeurs étant de même type, on peut également utiliser la notation « [] » à partir de variables de types pointeurs :
 - « [i] » veut dire : contenu de la ième case de type adéquat, comptée à partir de l'adresse du pointeur

```
int t[7]; /* Tableau de 7 entiers */
int *r; /* Pointeur sur un entier */
r = &t[2]; /* On pointe sur la troisième case de t */
r[1] = 12; /* On met 12 dans r[1], c'est-à-dire t[3] */
```



MIAGE - Université de Bordeaux

10

Arithmétique des pointeurs (1)

- À partir d'un pointeur, on peut créer un pointeur de même type mais pointant à i cases en amont ou en aval

$q = \&p[i];$ /* i indifféremment positif ou négatif */

- On peut simplifier cette expression en définissant une arithmétique des pointeurs :

– $p+i \Leftrightarrow \&p[i]$

– $p-i \Leftrightarrow \&p[-i]$

`int t[7]; /*Tableau de 7 entiers */`

`int *r; /*Pointeur sur un entier */`

`r = t + 2; /*Identique à l'exemple précédent */`

MIAGE - Université de Bordeaux

11

Tableaux multidimensionnels (1)

- Il est possible de déclarer des tableaux multidimensionnels en spécifiant plusieurs paires de « `[]` »

`int t[7][3]; /* Tableau de 7 lignes et 3 colonnes */`

- Les données sont stockées par ligne majeure, c'est-à-dire que dans `t[n][m]`, la case `t[i+1][0]` est stockée après la case `t[i][m-1]`

`int t[2][3] = { {1,2,3}, {4,5,6} };`

t	1	2	3	4	5	6
	t[0,0]	t[0,1]	t[0,2]	t[1,0]	t[1,1]	t[1,2]

MIAGE - Université de Bordeaux

12

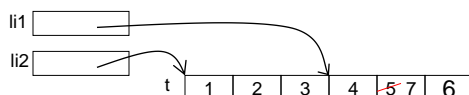
Tableaux multidimensionnels (2)

- Le compilateur effectue les calculs d'adresses de façon à accéder à la bonne case du (sous)-tableau stocké de façon linéaire

```

int t[2][3] = {{1, 2, 3}, {4, 5, 6}};
int * li0;      /* Pointeur sur la première ligne */
int * li1;      /* Pointeur sur la deuxième ligne */
li1 = t[1];     /* t de type(int **), t[1] de type(int *) */
li0 = t[0];     /* Même valeur d'adresse, pas même type ! */
li1[1] = 7;     /* Écriture tout à fait correcte */
t[0][4] = 7;    /* Pas de vérification de débordement! */

```



MIAGE - Université de Bordeaux

13

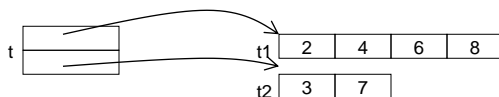
Tableaux multidimensionnels (3)

- On peut émuler des tableaux multidimensionnels au moyen de pointeurs de pointeurs

```

int *t[2];      /* t est de type (int **) */
int t1[4] = {2,4,6,8};
int t2[2] = {3,7};
t[0] = t1;
t[1] = t2;
a = t[0][3];    /* a = 8 */
b = t[1][0];    /* b = 3 */
c = t[1][3];    /* En dehors des limites ! */

```



MIAGE - Université de Bordeaux

14

Chaînes de caractères

- Il n'existe pas de type chaîne spécifique
- Une chaîne de caractères est un tableau unidimensionnel de caractères
 - Le nom de chaîne fait référence à l'adresse de début du premier caractère de la chaîne
- Une chaîne de caractères bien fermée est toujours terminée par un caractère nul `\0`
 - Indique où le contenu utile de la chaîne finit
 - Ne pas oublier de compter réserver sa place.

Chaînes de caractères (2)

- On peut déclarer et initialiser une chaîne de caractères comme on déclare un tableau de caractères


```
char t[] = {'a','b','c','\0'};
```
- Il est cependant préférable d'utiliser les constantes


```
char t[] = "Bonjour\n"; /* Tableau modifiable en mémoire */
char *s = "Salut";      /* Pointeur sur une constante */
t[1] = 'Z';             /* Légal car t est modifiable */
s[1] = 'Z';             /* Segfault car zone constante */
s = t;                  /* Légal car s est une variable */
```


Affichage de chaînes

- Directement dans printf si chaîne simple
 - Code %s pour afficher des chaînes
- La chaîne est affichée jusqu'au premier \0 rencontré

```
char t[] = "Bonjour\n";
char * s = "Salut!";
char * r = "%sEt aussi %s\n";
printf("Bonjour\n"); /* Le compilat. crée une chaîne constante */
printf(t);           /* Attention : pas de "%" dans la chaîne! */
printf("%s", t);      /* Le plus sûr si on n'est pas sûr de t */
printf(r, t, s);      /* Possible mais délicat */
```

MIAGE - Université de Bordeaux

17

Entrée de chaînes

- On utilise scanf
 - Code "%s" aussi
 - Pas besoin du « & » car une chaîne est déjà une référence sur une zone mémoire
 - Limiter la taille avec la syntaxe "%num" pour éviter les débordements de tampons

```
char str[10]; /* Une chaîne de 10 caractères */
int a;
printf("Entrez votre nom :");
scanf("%s", str); /* Pas de « & » car str est une référence */
scanf("%9s", str); /* On limite à 1 de moins (pour \0) */
```

MIAGE - Université de Bordeaux

18

Fonctions de manipulation de chaînes

- Définies dans la bibliothèque <string.h>
 - `strlen()` : renvoie la taille courante d'une chaîne (pas la taille maximale du tableau)
 - `strcpy()` : copie d'une chaîne source vers un tableau de caractères destination
 - `strcat()` : ajout d'une chaîne source à la fin d'une chaîne destination
 - `strchr()` : recherche de la première occurrence d'un caractère dans une chaîne
 - etc