Guide superflu de programmation en C

Matthieu Herrb

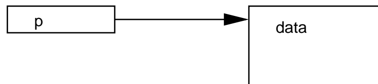
Mars 2007

Bibliographie

- B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall, 2nd edition, 1988.
- B. W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley Professional Computing Series, 1999.
- S. Summit. *C Programming FAQs: Frequently Asked Questions*. Addison-Wesley, 1995. `http://www.eskimo.com/~scs/C-faq/top.html`
- D. Goldberg. *What every computer scientist should know about floating-point arithmetic*. ACM Computing Surveys, 23(1):5–48, March 1991.
- D.E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1973.
- L.W. Cannon, R.A. Elliot, L.W. Kirchhoff, J.H. Miller, J.M. Milner, R.W. Mitze, E.P. Shan, and N.O. Whittington. *Indian Hill C style and coding standards*. Bell Labs.

Pointeurs en C : Rappels

- Principe : une variable contient l'adresse d'une autre variable



- Notation : type "pointeur sur type" : `type *`
  exemple : `char *`, `int *`, `POINT *`, `struct data *`, etc.
- données pointées par `p` : `*p`
- champ d'une structure pointée par `p` :
  `(*p).champ` ⇔ `p->champ`
- adresse d'une variable : `&data`
- type `void *` : pointeur vers type non défini.

Pointeurs en C : Rappels (2)

- `p = &data;` stocke dans `p` l'adresse de `data`
- `*p = data;` copie dans la variable désignée par `p` la valeur de la variable `data`
- Arithmétique sur les pointeurs (sauf `void *`) :
  - addition/soustraction d'un entier : déplace le pointeur de la taille du type désigné. Exemple :

    ```
    int *p; /* p est un pointeur sur entiers */
    p = &i; /* adresse de i */
    p++;    /* adresse de l'entier suivant i */
    ```

  - soustraction de pointeurs vers même type: retourne le nombre d'élement entre les 2 adresses.
  - autres opérations : **interdites**

Pointeurs en C : allocation dynamique

Possibilité d'allouer dynamiquement l'espace mémoire :
le *tas* (heap en anglais).

- Allocation :
```
void *malloc(size_t taille);
void *calloc(size_t nelem, size_t taille);
```
- Libération :
```
void free(void *p);
```

Allocation dynamique: exemple

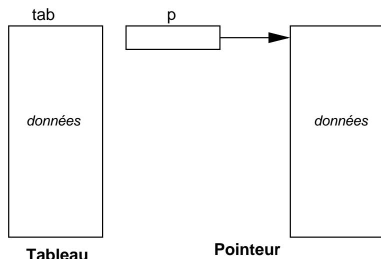
```
typedef struct point {
     double x, y;
} POINT;

POINT *p;
...
p = (POINT *)malloc(sizeof(POINT));
p->x = 0.0;
p->y = 0.0;
...
free(p);
```

Pointeurs et tableaux

- En général, il y a équivalence : la notation `[]` peut servir pour un tableau ou pour une zone désignée par un pointeur.
  `p[i]` ⇔ `*(p+i)`
- `tab` est un tableau : `int tab[10];`
  `p` est un pointeur vers 10 int :
  `int *p = (int *)malloc(10*sizeof(int));`

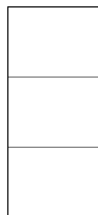
- `tab` ⇔ `&tab[0]`
- `p` ⇔ `&p[0]`
- Mais:
  `sizeof(p)` ≠ `sizeof(tab)` !



Tableaux à 2 dimensions et pointeurs

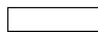
La notation `[][]` a 2 significations :

- `int tab[10][10];`
  `tab[i][j]` ⇔ `*(tab i*10 + j)`
  une multiplication
  + une addition
  + une indirection

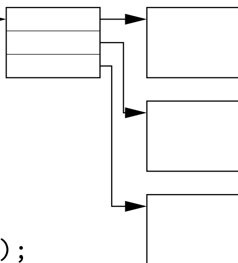


int tab[3][3];

- `int **p;`
  `p[i][j]` ⇔ `*(*(tab + i) + j)`
  2 additions
  + 2 indirections
  → plus efficace !



int **p;



```
p = (int **)malloc(10 * sizeof(int *));
for (i = 0; i < 10; i++) {
    p[i] = (int *)malloc(10 * sizeof(int));
}
```

Chaînes de caractères

- pas de type particulier
- suite de `char` ou `unsigned char` terminée par `\0`.
- peuvent être stockées dans un tableau ou dans la mémoire dynamique.
- syntaxe particulière pour la déclaration :
  `char s1[] = "Ceci est une chaine";` ou
  `char *s2 = "Ceci aussi";`
- Attention aux débordements : toujours allouer une zone assez grande pour stocker le résulat d'une opération
- utiliser `strlcat()`, `strlcpy()`, `snprintf()` à la place de `strcat()`, `strcpy()`, `sprintf()`.

Pièges de l'allocation dynamique (1)

- Débordement (cf chaînes ci-dessus)
- Fuites : perte d'un pointeur vers une zone allouée : plus possible de la libérer.
  Exemple :

```
char *p;

p = (char *)malloc(10);
...
p = (char *)malloc(20);
```

Pièges de l'allocation dynamique (2)

- Référence à une zone libérée

```
p = (char *)malloc (10);
...
free(p);
...
i = *(p + 3);
```

- Référence à une zone non allouée
- Libération d'une zone invalide

Architectures 64 bits

- traditionnellement: **LP32**: `long` et pointeurs stockés sur 32 bits.
- architectures 64 bits: **LP64**: `long` et pointeurs stockés sur 64 bits.

type	**taille 32 bits**	**taille 64 bits**
char	1	1
short	2	2
int	4	4
long	4	8
void *	4	8
long long	8	8
float	4	4
double	8	8

Outils de mise au point

- debogueur niveau source : `gdb`, `ddd`.
- outils de debug mémoire : `purify`, `valgrind`