

Cours d'Algorithmique-Programmation 2^e partie (IAP2): programmation impérative et structures de données simples

Compléments sur les tableaux + Pointeurs en C

Sandrine Blazy

ensiiē - 1^{ère} année

7 novembre 2007

**école nationale supérieure d'informatique
pour l'industrie et l'entreprise**

ensiiē

1 Tris par sélection

Matrices

Le type pointeur sur élément

Pointeurs et arguments

argc et argv

Tri par sélection

Principe

On cherche le plus petit élément du vecteur et on l'échange avec le premier élément puis on trie le reste.

Plus précisément et itérativement, à la i -ème étape du tri d'un vecteur de longueur n , on cherche l'indice j du plus petit élément du sous-vecteur d'indice compris entre i et $n - 1$ et on échange cet élément avec l'élément d'indice i . L'échange n'est effectivement réalisé que si ce plus petit élément n'est pas déjà en i .

```
/* Interface
indice_min_sous_vecteur: vecteur → int
post calcule l'indice du plus petit élément du vecteur
*/
```

```
/* Interface
tri_selection: vecteur → vecteur
post trie le tableau dans l'ordre croissant
*/
```

Tri par sélection

d'une liste, en Caml

```
let_ rec indice_min_sous_vecteur l = match l with  
| [] -> failwith "indice_min_sous_vecteur: liste vide"  
| [x] -> x  
| x::l' -> min_x (indice_min_sous_vecteur l');
```

```
let_ rec supprimer e l = match l with  
| [] -> []  
| x::l' -> if x=e then l' else x::(supprimer e l');
```

```
let_ rec tri_selection l = match l with  
| [] -> []  
| _ -> let_ m = indice_min_sous_vecteur l in  
      let_ l' = supprimer m l in m::(tri_selection l');
```

Tri par sélection

d'un tableau, en C

```
int indice_min_sous_vecteur(int t[], int debut, int fin){  
    int imin = debut;  
    int i;  
    for (i = debut+1; i<=fin; i++) {  
        if (t[i] < t[imin]) imin = i;  
    }  
    return imin;  
}
```

Tri par sélection

en C

```
/* Interface tri_selection: int [] -> int -> void
   post trie le tableau dans l'ordre_croissant
*/
void tri_selection(int t[], int n){
    int indice_fin = n-1;
    int i, j;
    int temp;
    for(i = 0; i < n; i++){
        j = indice_min_sous_vecteur(t, i, indice_fin);
        if(j > i){
            temp = t[i];
            t[i] = t[j];
            t[j] = temp;
        }
    }
}
```

iië

Contrairement aux fonctions de tri de listes qui retournent une liste triée, les fonctions de tri de vecteurs agissent par effet de bord : elles modifient le vecteur qui leur est passé en argument et à la fin du traitement, ce vecteur est trié. On dit qu'elles travaillent *en place*.

Comparaison : la version sur les vecteurs est considérablement plus simple. En effet, nous nous restreignons aux éléments autres que le plus petit simplement en incrémentant l'indice i après avoir échangé le plus petit élément avec celui d'indice i alors que pour les listes, il nous faut parcourir à nouveau la liste pour aller supprimer ce plus petit élément.

Tris par sélection

2 Matrices

Le type pointeur sur élément

Pointeurs et arguments

argc et argv

Matrices $n \times p$

Définition

Matrice = tableau de dimension 2.

n lignes, p colonnes \rightarrow un tableau à n lignes, chaque ligne étant un tableau (vecteur) de p éléments.

```
#define NB_LIG 10
#define NB_COL 20

int main (void) {
    int ma_matrice [NB_LIG] [NB_COL];
    for (i=0; i<NB_LIG; i++)
        for (j=0; j<NB_COL; j++)
            ma_matrice [i][j] = i+j;

    return 0;
}
```

Les valeurs sont rangées ligne par ligne.

```
int t[4][5] = {  
    { 0,1,2,3,4},  
    { 10,11,12,13,14},  
    { 20,21,22,23,24},  
    { 30,31,32,33,34},  
};
```

Remarque : on aurait pu omettre 4, mais pas 5.

Matrices

Matrice en argument

```
#define NB_LIG 2
#define NB_COL 3

void affiche (double m[][NB_COL], int nblig, int nbcoll)
{
    int c, l;

    for (l = 0; l < nblig; l++)
    {
        for (c = 0; c < nbcoll; c++)
            printf(" %g", m[l][c]);
        printf("\n");
    }
}
```

```
int main (void)
{
    double mat[NB_LIG][NB_COL] =
        {
            { 11.0, 12.0, 13.0 },
            { 21.0, 22.0, 23.0 }
        };
    affiche(mat, NB_LIG, NB_COL);
    return 0;
}
```

Tris par sélection

Matrices

3 Le type pointeur sur élément

Pointeurs et arguments

argc et argv

- Toute donnée rangée en mémoire a une adresse. Un pointeur est une variable qui a pour valeur une adresse.
Opérateur de prise d'adresse `&` (`& var`).
Opérateur d'indirection `*` (`* adr`).
- Intérêt : gestion dynamique de la mémoire.
- Syntaxe d'une déclaration : `id_type * id_var;`
- Sémantique : alloue en mémoire l'espace suffisant pour contenir une adresse.

- Exemple

```
char x;  
char *p; /* ou char *p, x; */  
...  
p = &x;  
*p = 'w'; /* *p est une variable pointée */  
          /* on a donc x == 'w' */
```

- L'opérateur * est surchargé. Ex. : *i * *j
- On peut pointer sur n'importe quel type (prédéfini, structure, tableau, pointeur, fonction).

- Le type pointeur générique `void *` : capable de pointer vers n'importe quel type.

Attention : pas de contrôle de type

Ex. : `int f (void * i , void * j)`

- Notation pour les pointeurs sur des enregistrements

```
struct personne {  
    ... nom;  
    int age;  
    ...  
} toto;  
struct personne *p;
```

/ (*p).nom ou p->nom pour accéder au champ nom
(et) obligatoires car * prioritaire par rapport à . */*

- Taille d'un pointeur : `sizeof (void *)`, qui est souvent la taille de `int`.
`sizeof (t_ou_exp) == nombre d'octets de t_ou_exp`
 Ex. : `sizeof (short) == 2`,
`sizeof (t[100])`,
`sizeof (t) == 100`
- Coercition : forcer un type ("cast").
 Syntaxe : `(id_type) id_exp`
 Ex. : `(double)5/2 (float)2`
- Compatibilité :
 - les pointeurs sont compatibles entre eux
 - pointeurs et entiers sont compatibles (coercition)
 - coercition obligatoire, sauf pour le type `void *`
 - Si `type * p` ; alors `p == (type *) (void *) p`
 - une valeur est commune à tous les pointeurs : `NULL`

- `NULL = (void *)0`

Ex. `:type * p = NULL;`

→ un pointeur vaut soit `NULL`, soit l'adresse d'une case mémoire

→ `*p` n'a de sens que si `p != NULL`

- Les tableaux sont transmis par adresse.

Tris par sélection

Matrices

Le type pointeur sur élément

4 Pointeurs et arguments

argc et argv

- Transmission par adresse des arguments aux fonctions
- Indiquer dans les interfaces quels arguments ont des effets de bord
 - clause **modifie**
 - valable également pour les variables globales

Pointeurs et arguments

Exemple

```
/* Interface échange: dr int * X dr int * → void
   modifie px, py
   post échange le contenu des deux arguments */
void échange (int * px, int * py){
    int temp = *px;
    ...
    *px = *py;
    *py = temp;} /* *px partout au lieu de px, idem pour py */

int main(void) {
    int x, y;
    ...
    échange (&x, &y);} /* &x au lieu de x dans l'appel, idem pour y */
    ...
```

Tris par sélection

Matrices

Le type pointeur sur élément

Pointeurs et arguments

5 argc et argv

```
int main(int argc, char ** argv)
```

argc : nombre de chaînes de caractères de la ligne de commandes.

argv : pointeur sur un tableau de chaînes de caractères.