Licence d'informatique
Module de Pratique du C

Travaux dirigés

Manipulation de tableaux

Philippe MARQUET

Octobre 2004

Les exercices proposés illustrent l'utilisation simple de tableaux. En particulier, les chaînes de caractères peuvent être vues comme des tableaux.

Nous reviendrons sur la représentation mémoire des tableaux et leur lien avec les pointeurs dan un prochain TD.

(• Un tableau ce sont
- des valeurs contiguës en mémoire
- de même type
- indicées à partir de 0, avec l'opérateur `[]`

•)

Exercice 1 (Tableau : initialisation et itération)
Quel résultat fournit le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>

static int te[5];
static int tf[5] = {63, 64};

int
main ()
{
    int ta[5] = {17, 15, 13, 12, 14};
    int tb[] = {27, 25, 23, 22, 24};
    int tc[5] = {37, 39};
    int td[5];

    int i;

    printf("ta : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", ta[i]);

    printf("\ntb : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", tb[i]);

    printf("\ntc : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", tc[i]);

    printf("\ntd : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", td[i]);

    printf("\nte : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", te[i]);

    printf("\ntf : ");
    for (i=0 ; i<5 ; i++)
        printf("%d, ", tf[i]);

    putchar('\n');

    exit (EXIT_SUCCESS);
}
```

(• ***Éléments de solution 1*** On note
- la déclaration des tableaux ;
- leur possible initialisation ; possiblement incomplète.
- la boucle d'itération sur les éléments d'un tableau, formule idiomatique du C.

Une exécution peut donner quelque chose comme

```
ta : 17, 15, 13, 12, 14,
tb : 27, 25, 23, 22, 24,
tc : 37, 39, 0, 0, 0,
td : 134513028, -1073745500, 1073835688, 1, 1075245144,
te : 0, 0, 0, 0, 0,
tf : 63, 64, 0, 0, 0,
```

- les variables *automatiques* ne sont pas initialisées par défaut ;
- les variables *statiques* sont initialisées à zéro par défaut. Une initialisation explicite n'est néanmoins, pas mal venue.

•)

Exercice 2 (Copie de tableaux)
Soient les deux tableaux définis par

```
#define SIZE          12
int tsrc[SIZE], tdest[SIZE];
```

Question 2.1 Donnez le code C permettant d'affecter à `tdest` les éléments de `tsrc`.
(• ***Éléments de solution 2.1***

- On remarque la déclaration des deux tableaux en une seule fois ; il faut répéter les `[]` et la taille.
- On ne peut affecter directement les tableaux, il faut itérer sur les éléments.

```
int i;
for (i=0 ; i<SIZE ; i++)
    tdest[i] = tsrc[i];
```

•)

Question 2.2 Donnez le code C permettant d'affecter à `tdest` les seuls éléments strictement positifs de `tsrc`. On complétera la fin du tableau par des valeurs nulles.
(• ***Éléments de solution 2.2*** Une solution bête en deux itérations, initialisation à zéro puis copie :

```
int i, j;
for (i=0 ; i<SIZE ; i++)
    tdest[i] = 0;
for (i=0, j=0 ; i<SIZE ; i++)
    if (tsrc[i] > 0)
        tdest[j++] = tsrc[i];
```

Une solution en une itération

```
int i, j;
for (i=0, j=0 ; i<SIZE ; i++)
    if (tsrc[i] != 0)
        tdest[j++] = tsrc[i];
for ( ; j<SIZE ; j++)
    tdest[j] = 0;
```

on remarque la double initialisation dans le premier `for` et l'instruction vide d'initialisation dans le second.
•)

Exercice 3 (Compteur de chiffres et de blancs)
On veut, au sein d'une fonction, compter les occurrences de chaque chiffre (`'0'` à `'9'`) et des blancs (`' '` ou `\t`) d'un texte lu sur `stdin`. Le résultat sera stocké dans un tableau de 10 entiers pour les chiffres et dans une variable entière pour les blancs.
(• ***Éléments de solution 3***

- On ne peut initialiser un tableau automatique avec des zéros sans se répéter.
- pourquoi pas un `switch` ?

```
void occurrences()
{
    int chiffre[10] = {0,0,0,0,0,0,0,0,0,0};
```

```
    int blancs = 0;
    int c;
    while ((c=getchar()) != EOF)
        switch (c) {
            case '0': case '1': case '2': case '3':
            case '4': case '5': case '6': case '7':
            case '8': case '9':
                chiffre[c-'0']++;
                break;
            case ' ': case '\t':
                blancs++;
                break;
    }
    ...
}
```

•)

Exercice 4 (Valeur maximale)

Question 4.1 Donnez la déclaration d'une fonction `max_val()` qui retourne la plus grande valeur, en valeur absolue, d'un tableau de valeurs `double` passé en paramètre.

(• ***Éléments de solution 4.1***

- Comment passer le tableau en paramètre ? Il faut pouvoir connaître son nombre d'éléments.
- Ce nombre d'éléments peut être connu et fixe pour tous les appels de la fonction (c'est par exemple une macro CPP). On utilise alors cette valeur dans le corps de la fonction.
- Le prototype de la fonction est alors l'un des suivants (d'autres sont possibles, mais on ne parlera pas encore de pointeurs...)

  ```
  #define SIZE     12
  double max_val(double t[SIZE]);
  double max_val(double t[]);
  double max_val(double t[2]);
  ```

  La valeur indiquant la taille du paramètre ne sera jamais utilisée par le compilateur. On peut mettre n'importe quelle valeur... mieux vaut donc ne pas en mettre !
- La taille du tableau peut varier d'un appel à l'autre. Il nous faut alors un moyen de connaître cette taille. On utilise couramment un paramètre supplémentaire.
- Chacun des prototypes suivants est valide :

  ```
  double max_val(double t[SIZE], int size);
  double max_val(double t[], int size);
  ```

  mais on ne peut pas écrire

  ```
  double max_val(double t[size], int size);
  ```

  ni même

  ```
  double max_val(int size, double t[size]);
  ```

•)

Question 4.2 Donnez la définition de la fonction `max_val()`.

(• ***Éléments de solution 4.2*** Pas de difficultés particulières, on notera l'emploi de `fabs()` définie dans `math.h` qui requière l'option de compilation `-lm` lors de la phase d'édition de liens.

```
double
max_val(double t[], int size)
{
    double max = 0;
    int i;

    for (i=0 ; i < size ; i++)
        if (max < fabs(t[i]))
            max = fabs(t[i]);
```

```
    return max;
}
```

•)

Question 4.3 Soient les déclarations

```
#define SIZE                200
double m, t[SIZE];
```

donnez le code C qui range dans `m` la plus grande valeur absolue des éléments de `t`.

(• ***Éléments de solution 4.3*** Il s'agit uniquement de voir comment passer le tableau `t` en paramètre.

```
m = max_val(t, SIZE);
```

•)

Exercice 5 (Recherche par dichotomie)

On écrit une fonction qui recherche l'indice d'un nombre `x` dans un tableau `v` de `n` flottants triés par ordre croissant :

```
int search(float x, float v[], int n);
```

(• ***Éléments de solution 5***

– Une version itérative est la suivante. Facile une fois indiqué que l'on utilise les indices `debut`, `fin`, et `milieu`.

```
int
search(float x, float v[], int n)
{
    int debut = 0,
        fin = n-1,
        milieu;

    while (debut <= fin) {
        milieu = (debut + fin)/2;
        if (v[milieu] == x)
            return milieu;
        else if (v[milieu] > x)
            /* recherche dans la moitie gauche */
            fin = milieu-1;
        else
            /* recherche dans la moitie droite */
            debut = milieu+1;
    }

    /* pas trouvé */
    return -1;
}
}
```

– on peut préférer une version récursive, qui pose le problème de l'appel sur la seconde moitié du tableau... Mais bon ils savent ce qu'est une adresse, ils l'ont vu en archi...

```
int
search(float x, float v[], int n)
{
    if (n==1)
        if (v[0] == x)
            return 0;  /* tableau indice a partir de 0 */
        else
            return -1;  /* valeur absente */
    else if (x < v[n/2])
        return search(x, v, n/2);
```

```
    else
        return n/2 + search(x, &v[n/2], n-n/2); /* attention impair */
}
```

•)

Exercice 6 (Crible d'Érathostène)
Le crible d'Érathostène permet d'identifier les nombres premiers inférieurs à un entier donné. Il est basé sur la construction d'une liste contenant initialement tous les entiers. On supprime successivement les entiers multiples du plus petit élément de la liste identifié comme premier.

Proposez une fonction qui affiche les nombres premiers inférieurs à une valeur `N` donnée.

(• ***Éléments de solution 6*** On mémorise la liste dans un tableau, `prime[i]` contient l'état de l'entier `i` (`i` pour simplifier, on peut décaler...) parmi :

- premier
- candidat
- non-premier

•)

Exercice 7 (Inversion d'une chaîne de caractères)
En langage C, les chaînes de caractères sont rangées sous forme de tableaux de caractères, terminés par le caractère `'\0'`.

On veut inverser une chaîne de caractères, sans déclarer un second tableau. Pour ce faire, on pourra avoir besoin de la fonction `strlen(const char s[])` déclarée dans `string.h` qui retourne la longueur d'une chaîne (non compris le caractère `'\0'`).

```
void str_reverse(char str[]);
```

(• ***Éléments de solution 7***

- La fonction `strlen()` est définie dans la bibliothèque `string`. On peut la réécrire.

```
int
mystrlen(char s[])
{
    int cpt=0;
    while (s[cpt] != '\0')
        cpt++;
    return cpt;
}

void
str_reverse (char str[])
{
    char c ;
    int i, lg = mystrlen (str);

    for (i=0 ; i<lg/2 ; i++) {
        c = str[i] ;
       str[i] = str[lg-i-1];
       str[lg-i-1] = c;
    }
}
```

- Si on admet introduire les pointeurs, on peut aussi imaginer une version a l'aide de 2 pointeurs : `montant` et `descendant` :

```
int
mystrlen(char *s)
{
    int cpt=0;
    while (*s++ != '\0')
        cpt++;
    return cpt;
```

```
    }

    void
    str_reverse (char *str)
    {
        char *montant = chaine ;
        char *descendant = *chaine + strlen (chaine) -1;
        while (descendant > montant)
            ...
    }
```

on note que `*s++` est lu `*(s++)` et que

```
    {
      char *montant = chaine;
      ...
```

est lu

```
    {
      char *montant;
      montant = chaine;
```

•)

Exercice 8 (Conversion décimal/caractères)
La fonction `itoa(int n, char s[])` retourne dans `s` la chaîne de caractères représentant l'entier signé passé en paramètre. Pour l'écrire, on se méfiera du signe, et du fait que l'on ne veut pas de zéros inutiles. Pour cela, on pourra réutiliser la fonction `str_reverse()`.
(• ***Éléments de solution 8*** Attention `bogus_itoa(0)` retourne `""`!

```
void bogus_itoa(int n, char s[])
{
    char sgn;
    int ind=0;
    sgn = n<0 ? '-' : '+';
    n = n<0 ? -n : n;
    while(n!=0) {
        s[ind++] = n%10 + '0';
        n /= 10;
    }
    if (sgn == '-')
        s[ind++] = sgn;
    s[ind] = '\0';
    str_reverse(s);
 }
```

•)

Exercice 9 (Chaînes de caractères)
Réécrire des fonctions de la bibliothèque de manipulation des chaînes de caractères. Ces fonctions ne gèrent pas l'allocation mémoire, le programmeur doit s'assurer de la validité avant l'appel à ces fonctions.

```
/* recopie le contenu de src dans dest
   retourne dest */
char *strcpy(char *dest, const char *src);

/* recopie src a la fin de dest (concat)
   retourne dest */
char *strcat(char *dest, const char *src);
```

```
/* compare les contenus de s2 et s1;
   renvoie la difference entre les deux premiers caracteres differents */
int strcmp(const char *s1, const char *s2);
```

(• ***Éléments de solution 9***

- Les prototypes sont ici donnés, comme dans le manuel en ligne, avec des pointeurs. On explique (affirme ?) que c'est une autre forme syntaxique de

  ```
  char *strcpy(char dest[], const char src[]);
  ```

  une fonction ne peut pas retourner un tableau (de caractères), mais peut retourner un pointeur (sur un caractère)...
- On note aussi le `const`.
- Quelques étapes de réécriture pour en arriver à la fameuse version du strcpy(). Conseiller à un étudiant de s'arrêter là ou il est encore à l'aise.

```
char *
mystrcpy(char *dest, const char *src)
{
    while (*src != '\0') {
        *dest = *src;
        src++, dest++;
    }
    *dest = '\0';
}
char *
mystrcpy(char *dest, const char *src)
{
    while (*src != '\0')
        *dest++ = *src++;
    *dest = '\0';
}
char *
mystrcpy(char *dest, const char *src)
{
    while ((*dest++ = *src++) != '\0')
        ;
}
char *
mystrcpy(char *dest, const char *src)
{
    while (*dest++ = *src++)
        ;
}
char *
mystrcat (char *dest, const char *src)
{
    while (*dest!='\0')
        dest++;
    while ((*dest++ = *src++) != '\0')
        ;
}

int
mystrcmp (char *s1, char *s2)
{
    while ((*s1==*s2) && (*s1!='\0'))
        s1++;s2++;
    return *s2 - *s1;
}
```

•)