[LC120]

# Pointeurs vers des fonctions

On peut déclarer en C des variables pointant vers des fonctions. Ceci permet de passer ces fonctions comme arguments d'autres fonctions. L'exemple qui suit est inspiré de Kernighan & Ritchie [K&R - § 5.12]. Il s'agit d'une fonction permettant d'effectuer un tri assez élémentaire.

```
int tabli[]={1,12,23,.... };
#define NBI sizeof(tabli)/sizeof(int)
int *tablipt[NBI];
char *tabchp={"chaine","essai",... };

int nbchp= sizeof(tabchp)/sizeof(char *);

void sort(void *v, int n, int (*cmp)(  ),
                           void (*xch)(void * ,void *))
{
int gap,i,j;
for (gap=n/2 ; gap>0 ; gap /= 2)
    for (i=gap ; i<n ; i++)
        for (j=i-gap ; j>=0 ; j-=gap)
            {
            if  ((*cmp)(&v[j],&v[j+gap])>0)
                (*xch)(&v[j],&v[j+gap]);
            }
}
.............
extern int numcmp(int x,int y);    // comparaison d'entiers

#include<string.h >
extern void swapn(int * x,int * y),    // échange d'entiers
       swapchpt(char  * * x,char * * y);  // echange de chaînes


main()
{
for (i=0 ; i<NBI ; i++)    // init tableau de pointeurs (tablipt)
    tablipt[i]=&tabli[i];
......
......
if (numeric)
    sort ((void *)tablipt, NBI, numcmp, swapn);
else /* non numérique → chaînes de caractères */
    sort ((void *)tabchp, nbchp, strcmp, swapchpt);
......
......
}
```

```
// voici les définitions (externes) des fonctions de comparaison et échange
int numcmp(int * x,int * y)     // comparaison d'entiers
    { return *x-*y; }
void swapn(int * * x,int * * y)     // échange d'entiers
    { int * tmp; tmp= *x; *x=*y; *y=tmp; }
void swapchpt(char  * * x,char * * y)  // echange de chaînes
    { char * tmp; tmp= *x; *x=*y; *y=tmp;}
```

en fait, échange de ptr.

---

Un autre exemple similaire est celui de deux fonctions qui sont dans la bibliothèque standard : Il s'agit des fonctions **qsort** et **bsearch** dont voici les déclarations :

```
void *   bsearch   /* recherche binaire dans un tableau déjà trié */
(void *  clef, void * tabl, size_t nb, size_t taille, int (* cmp)(const void * p1, const void * p2));
void     qsort     /* tri suivant algorithme quicksort */
          (void * tabl, size_t nb, size_t taille, int (* cmp)( const void * p1, const void * p2));
```

• pointeur sur élt si trouvé ◊ NULL */

↑ clé — ↑ tableau — ↑ nb élt — ↑ taille élt — ↑ fonct. de comparaison

interdit de modifier les objets pointés.

Voici à ce sujet un extrait de la FAQ[1] que l'on peut trouver sur les "news" concernant le langage C (§ 13.8) :

« I'm trying to sort an array of strings with qsort(), using strcmp() as the comparison function, but it's not working.»

Answer : By "array of strings" you probably mean "array of pointers to char." The arguments to qsort's comparison function are pointers to the objects being sorted, in this case, pointers to pointers to char. strcmp(), however, accepts simple pointers to char. Therefore, strcmp() can't be used directly. Write an intermediate comparison function like this:

```
/* compare strings via pointers */
int pstrcmp(const void *p1, const void *p2)
{
    return strcmp(*(char * const *)p1, *(char * const *)p2);
}
```

The comparison function's arguments are expressed as "generic pointers," const void *. They are converted back to what they "really are" (pointers to pointers to char) and dereferenced, yielding char *'s which can be passed to strcmp().

Il ressort de ce qui précède, que :

- les arguments de la fonction de comparaison de **qsort** sont des pointeurs vers les objets à trier ;
- si on souhaite trier des chaînes de caractères (= tableaux de même longueur), on peut utiliser **strcmp** directement ;
- mais si on souhaite utiliser des chaînes de caractères (= pointeurs), alors il faut utiliser une fonction ayant pour arguments des pointeurs vers des pointeurs (non modifiables) vers des caractères c'est-à-dire des char * const *. Il faut donc utiliser alors une fonction telle que la fonction donnée en exemple : **pstrcmp** qui appellera elle-même **strcmp** en lui passant les (char *) pointés par p1 et p2, après avoir adapté le type de ces deux pointeurs p1 et p2. En fait, on peut écrire l'exemple précédent sous la forme suivante, ce qui revient à faire l'adaptation de type lors de l'appel de cette fonction, plutôt que dans la fonction elle-même :

http://iutsox03/prof/Blanc/Cfaq

[1] (= Frequently Asked Questions = Foire Aux Questions).
On peut trouver la FAQ en anglais à l'adresse suivante : http://www.eskimo.com/~scs/C-faq/top.html
et une version française (pas les mêmes questions) à l'adresse suivante : http://www.isty-info.uvsq.fr/~rumeau/fclc/
ou encore sur l'INTRANET de l'IUT où je l'ai mise : http://193.49.118.104/prof/Blanc/faqC/

```
int pstrcmp(char *      * p1, char *      * p2)
 {
 return strcmp(* p1, * p2);
 }
```

## Tableaux de fonctions

Une autre utilisation des pointeurs vers des fonctions sont les "tableaux de fonctions" qui sont en fait des tableaux de pointeurs vers des fonctions. Les éléments d'un tel tableau contiennent donc des adresses de fonctions.

Exemple :

```
    /* tableau des opérateurs */
#define NBFONC 5
char op[NBFONC]="+-*/%";   /* Attention :                    */
    /* les fonctions */
int plus (int x, int y)    {return (    );}
int moins (int x, int y)   {return (x-y);}
int prod (int x, int y)    {return (x*y);}
int quot (int x, int y)    {if (y) return (x/y); else...}
int reste (int x, int y)   {if (y) return (x%y); else...}

/* déclaration du tableau des fonctions (elle est obligatoirement
   générique si les fonctions ne sont pas toutes de même type) */
int (*f[NBFONC])()={ plus , moins, prod , quot , reste };


int main()       /* programme principal */
{
int m,n,i; char c;
while (1)
    {
    scanf("%d %c %d =",&m,&c,&n);  /* saisie */
    i=ind(op,c,NBFONC);
    if (i==-1) printf ("erreur\n");
    else printf ("%d\n",(*f[i])(m,n));
    }
}

int ind( char * ch, char c, int n) /* recherche de l'index */
{
int i;
for (i=0 ; i<n && ch[i]!=c ;   )   i++;
return (i<n?i:-1);
}
```