thème 5 — Arithmétique des pointeurs

Université de Lille
Licence d'informatique, 2e année
Module de MPC — Maîtrise de la Programmation en C
Équipe pédagogique de MPC, d'après un document CC BY-SA de Gilles Grimaud Philippe Marquet, 2018-2020.
mars 2021

Une version PDF pour impression est accessible sur nextcloud.univ-lille.fr

Ce document est le support d'exercices de TD et TP.

Le thème 5 traite de l'arithmétique des pointeurs, de l'équivalence tableaux/pointeurs, de la possible généricité des pointeurs `void *`, de la représentation des tableaux à plusieurs dimensions, et enfin des pointeurs de fonctions.

→ Adresses et arithmétique de pointeurs

Adresses mémoires des types construits — *exercice de TD*

Soit le programme `tab_duo_trio.c` suivant

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

struct duo_s {
    int x;
    int y;
};

union trio_u {
    int n;
    char c;
    float x;
};

int
main()
{
    int tab[10];
    struct duo_s duo;
    union trio_u trio;

    printf("sizeof(int) : %lu\n", sizeof(int));
    printf("tab :\t%p\n", tab);
    printf("&duo :\t%p\n", &duo);
    printf("&trio :\t%p\n", &trio);

    exit(EXIT_SUCCESS);
}
```

dont une exécution produit :

```
% ./tab_duo_trio
sizeof(int) : 4
tab :   0x7ffeebe6a6d0
&duo :  0x7ffeebe6a6c8
&trio : 0x7ffeebe6a6c0
```

1. Quelle est l'adresse du champ `duo.x` ? `duo.y` ?
2. Quelle est l'adresse du champ `trio.n` ? `trio.c` ? `trio.x` ?

3. Quelles sont les valeurs des expressions suivantes
   - `&trio.n + 1`
   - `&trio.c + 1`
4. Quelle est l'adresse de l'élément `tab[5]` du tableau ?
   De manière générale, quelle est l'adresse de l'élément `tab[i]` ?
5. Soit la déclaration et initialisation :

   ```
   int *ptr;
   ptr = tab;
   ```

   En utilisant uniquement la variable `ptr`, et l'arithmétique des pointeurs, proposez une boucle qui va écrire la valeur `0` dans toutes les cases du tableau.

Pointeurs et chaînes de caractères — *exercice de TD*

1. Écrire une fonction `char *strend(char *str)` qui renvoie l'adresse du zéro terminal de la chaîne `str`.
2. (Pourquoi le prototype de cette fonction ne peut-il être `char *strend(const char *str)` ?)
3. En utilisant la fonction `strend()`, proposez une fonction qui renvoie le nombre de caractères d'une chaîne donnée - le caractère `'\0'` final non compris :

   ```
   int mstrlen(const char *str);
   ```

   Pour information. Cette fonction `mstrlen()` est similaire à la fonction `strlen()` fournie par la bibliothèque standard `string.h`.
4. La bibliothèque `string.h` fournie également les fonctions suivantes :

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

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

   Proposez des fonctions `mstrcpy()` et `mstrcat()`, réécritures de ces fonctions.

Autour des variables d'environnement – *exercice de TP*

La variable globale

```
extern char **environ;
```

est un pointeur sur le premier élément d'un tableau.
Chacun des éléments de ce tableau est une chaîne de caractères, donc un pointeur `char *`.
Une valeur `(char *) 0` indique la fin du tableau.

Chacune des chaînes est de la forme `"VAR=valeur"`, `VAR` correspondant à une des variables d'environnement.
Cette variable `environ` est par exemple exploitée par la commande Unix `printenv` qui affiche l'ensemble des variables d'environnement et leur valeur :

```
% printenv | head
SHELL=/bin/bash
LANGUAGE=en_US:
PWD=/home/12/duchmol
LOGNAME=duchmol
HOME=/home/12/duchmol
LANG=en_US.UTF-8
TERM=xterm-256color
USER=duchmol
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SSH_TTY=/dev/pts/0
```

1. Proposez une fonction `nvar()` qui exploite cette variable `environ` et renvoie le nombre de variables d'environnement définies.

   Proposez un programme qui fait appel à cette fonction et affiche ce nombre de variables.

   On pourra comparer le résultat avec celui fourni par la commande

   ```
   % printenv | wc -l
   ```

2. Proposez une commande `mprintenv` qui reproduit le comportement de `printenv`.

3. La commande `printenv` admet des paramètres qu'elle considère comme des noms de variables d'environnement donc elle affiche la valeur :

   ```
   % printenv USER SHELL
   duchmol
   /bin/bash
   ```

   Proposez une nouvelle version de votre commande `mprintenv` pour inclure cette fonctionnalité.

Nouvelle recherche dichotomique — *exercice de TP*

1. Proposez une fonction récursive qui recherche par dichotomie une valeur `v` dans un tableau trié `tab` :

   ```
   float * search_interval(float v, const float *tab, const float *end);
   ```

   L'argument `end` pointe sur le dernier élément du tableau.

2. On pourra reprendre le programme principal utilisé pour la fonction `search_dicho()` du TP précédent, *thème 4* pour valider cette nouvelle fonction de recherche.

→ Pointeurs génériques `void *`

Mise à zéro — *exercice de TD*

1. Proposez une fonction "générique" qui prend l'adresse d'un entier de type non spécifié, ainsi que sa taille, et met cet entier à zéro :

   ```
   void memzero(void *addr, unsigned int size);
   ```

   Exemples d'utilisations de la fonction :

   ```
   int a;
   short int b;
   memzero(&a, sizeof(int));            /* équivalent à a = 0 */
   memzero(&b, sizeof(short int));      /* équivalent à b = 0 */
   ```

2. Cette fonction peut-elle être utilisée pour mettre à zéro l'ensemble des valeurs d'un tableau, par exemple :

   ```
   #define MAX    128
   int t[MAX];
   ```

Pour information. La bibliothèque `string.h` propose une fonction `memset()` qui est généralement utilisée pour mettre à zéro une zone mémoire donnée.

Comparaison générique — *exercice de TD*

L'objet de l'exercice est de proposer une fonction `mmemcmp()` qui compare deux zones mémoire octet par octet.

1. Donnez un prototype possible pour cette fonction.

2. Donnez une définition de cette fonction.
   La fonction renverra une valeur nulle si et seulement si les deux zones sont égales.

Pour information. La bibliothèque `string.h` propose une fonction `memcmp()` qui réalise une telle comparaison.

Copie générique — *exercice de TP*

1. Proposez une fonction qui copie un objet d'adresse `from`, de type non spécifié, et de taille `size`, à l'adresse `to` :

   ```
   void mmemcpy(void *to, const void *from, unsigned int size);
   ```

2. Testez votre proposition, par exemple avec le code suivant

   ```
   #include <stdlib.h>             /* pour random() */
   #include <string.h>             /* pour memcmp() */
   #include <assert.h>             /* pour assert() */

   #define SIZE    1021

   void
   test_mmemcpy()
   {
       char     tc_orig[SIZE], tc_dest[SIZE];
       long int ti_orig[SIZE], ti_dest[SIZE];
       int i;

       /* initialisation */
       for(i=0 ; i<SIZE ; i++) {
           tc_orig[i] = random() % 256;
           tc_dest[i] = random() % 256;
           ti_orig[i] = random();
           ti_dest[i] = random();
       }

       /* copie */
       mmemcpy(tc_dest, tc_orig, SIZE);
       mmemcpy(ti_dest, ti_orig, SIZE * sizeof(long int));

       /* vérification */
       assert(memcmp(tc_orig, tc_dest, SIZE) == 0);
       assert(memcmp(ti_orig, ti_dest, SIZE * sizeof(long int)) == 0);
   }
   ```

Pour information. La bibliothèque `string.h` propose une fonction `memcpy()` qui réalise une telle copie.

Échange générique — *exercice de TP*

1. Proposez une fonction `memswap()` qui réalise un échange générique entre deux variables de même taille.
   Dans un premier temps, on pourra supposer que les deux objets ne se recouvrent pas en mémoire.
2. Testez votre proposition avec une fonction semblable à celle proposée pour l'exercice précédent.

→ Tableaux à plusieurs dimensions

Arrangement mémoire des éléments d'un tableau à plusieurs dimensions — *exercice de TD*

Soit la déclaration d'un tableau

```
int b[3][5];
```

En considérant que l'allocation du tableau se fait linéairement en mémoire (les 3 "tranches" de `b` sont allouées à des adresses contiguës), donnez l'état du tableau `b` après l'exécution du code C suivant :

```
int b[3][5];
int *a = *b, i;

for (i=0 ; i<15 ; *a++ = i++)
```

```
    ;
**b = 15;           **(b+1) = 16;        *(b[0]+1) = 17;
*(*b+8) = 18;       *(b[1]+2) = 19;      *(*(b+1)+5) = 20;
*(b[2]+3) = 21;     *(*(b+2)+2) = 22;
```

→ Pointeurs de fonction

à venir...