thème 4 — Références et 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 4 traite des références et pointeurs en C, des opérateurs de référence `&`, et `*` de déréférencement. (L'arithmétique des pointeurs n'est pas abordée ici, elle fera l'objet de futurs thèmes).

Mettre 42 — *exercice de TD*

Soit une variable de type `int` que l'on désire affecter à la valeur 42.

1. Pourquoi est-ce qu'une fonction de prototype

   ```
   void mettre42_bogus(int n);
   ```

   ne peut être utilisée pour modifier la valeur de la variable.

2. Donner le prototype d'une fonction `mettre42()` qui permet de modifier la valeur d'une variable entière désignée par un paramètre.

3. Soient les déclarations suivantes :

   ```
   int n;
   int t[12];
   ```

   Donner les appels de la fonction `mettre42()` permettant d'affecter la valeur 42 à `n` d'une part, à l'élément `t[3]` du tableau d'autre part.

Incrémenter un entier — *exercice de TD*

1. Réalisez une fonction `inc()` qui prend une référence sur un entier et incrémente l'entier.

2. Utilisez cette fonction pour incrémenter une variable entière, un élément d'un tableau d'entiers, un champs de type entier d'une structure.

Échanger deux variables — *exercice de TP*

1. Réalisez une fonction `swap_int()` pour échanger les valeurs de deux variables entières.

2. Réalisez un programme principal qui

   - lit deux valeurs entières à l'aide d'appels à la fonction `scanf()` de la bibliothèque C standard, dans deux variables entières,
   - échange les valeurs de ces variables,
   - affiche les nouvelles valeurs des variables avec `printf()`

   On consultera éventuellement la page de manuel de la fonction `scanf()` : `man scanf`.

Division — *exercice de TP*

Une fonction C ne permet de renvoyer qu'une unique valeur via le `return`.
Une fonction peut néanmoins accepter des paramètres qui soient des références et ainsi "renvoyer" plusieurs valeurs résultats.

1. Réalisez une fonction `division()` qui prend en argument un diviseur et un dividende (entiers), qui calcule le quotient et le reste.

   Cette fonction `division()` recevra également des références sur des entiers qui recueilleront le quotient et le reste.

2. Réalisez un programme principal qui appelle cette fonction avec les valeurs de votre choix, et testez ses résultats avec `putdec()`.

Chercher dans un tableau — *exercice de TD*

- Réalisez une fonction `chercher()` qui accepte en paramètre un tableau d'entiers `tab`, et une valeur `v`, et qui renvoie une référence vers le premier élement du tableau égal à `v`.
  Quelle valeur peut renvoyer la fonction si la valeur n'est pas présente dans le tableau ?

Arguments de `main()` – *exercice de TD et TP*

Le prototype de la fonction `main()` d'un programme C est le suivant :

```
int main (int argc, char *argv[]);
```

Les paramètres `argc` et `argv` permettent de récupérer les arguments de la ligne de commande.

- `argc` correspond au nombre d'arguments +1
- `argv[0]` est une chaîne de caractères correspondant au nom de la commande. Autrement dit, c'est une référence sur le 1er caractère de cette chaîne de caractères.
- de manière plus générale, `char *argv[]` est un tableau de références vers des caractères. Autrement dit, un tableau de chaînes de caractères.

  Chacune de ces chaînes de caractères correspond à un argument de la ligne de commande.

1. Représentez par un dessin la valeur du paramètre `argv` de `main()` lors de l'exécution des commandes suivantes d'un programme C compilé en une commande `mecho` :

   ```
   % mecho "Hello world !"
   % mecho Hello world !
   ```

2. Expliquez, compilez et exécutez le programme suivant :

   ```c
   /* Ma commande echo */

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

   int
   main (int argc, char *argv[])
   {
       int i;

       for(i = 1; i < argc ; i++) {
           printf("%s ", argv[i]);
       }
       putchar('\n');

       exit(EXIT_SUCCESS);
   }
   ```

Sauter les espaces — *exercice de TP*

1. Réalisez une fonction `skip_spaces()` qui accepte une chaîne de caractères `s`, et qui renvoie une référence sur le premier élement du tableau qui n'est pas une espace (si la chaîne est entièrement constituée d'espaces, on renverra un pointeur sur le caractère zéro terminal).

Soit le programme principal suivant utilisant cette fonction `skip_spaces()` compilé en un exécutable `strip_spaces_tst` :

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

```c
int
main(int argc, char *argv[])
{
    char * strip;
    int i;
    assert(argc == 2);

    printf("argv  : %s\n", argv[1]);
    strip = skip_spaces(argv[1]);
    printf("strip : %s\n", strip);

    for (i=0 ; strip[i]; i++)
        strip[i] = toupper(strip[i]);

    printf("strip : %s\n", strip);
    printf("argv  : %s\n", argv[1]);

    exit(EXIT_SUCCESS);
}
```

2. Quel sont les résultats attendus des exécutions suivantes du programme :

```
% ./strip_spaces_tst "FOO BAR"
% ./strip_spaces_tst "       FOO BAR"
% ./strip_spaces_tst "Foo Bar"
% ./strip_spaces_tst "       Foo Bar"
```

Expliquez ces résultats.

Pointeur sur un pointeur — *exercice de TD*

Un pointeur est aussi une variable. De la même manière que nous pouvons manipuler une référence sur une variable entière, nous pouvons manipuler une référence sur un pointeur.

Considérez le code suivant :

```c
int n, m;
int * foo;
foo = &n;
```

1. En utilisant la règle de typage des pointeurs (un pointeur qui référence quelque chose de type `TOTO` a pour type `TOTO*`), déduire le type d'un pointeur `bar` pouvant contenir une référence vers `foo`.
2. Comment déclarer un tel pointeur, et y affecter une référence sur `foo` ?
3. Une fois l'affectation précédente réalisée, comment faire pointer `foo` sur `m` en utilisant le pointeur `bar` et l'opérateur de déréférencement ?

Échanger deux pointeurs — *exercice de TP*

1. Quel est le prototype d'une fonction qui réalise l'échange de valeurs de deux pointeurs de type `int *` ?
2. Réaliser une fonction `swap_ptr()` qui échange les valeurs de deux pointeurs de type `int*`.
3. Vous pourrez utiliser le code suivant pour tester votre fonction :

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

int
main_swap_ptr() {

    int a, b;
```

```c
    int *p = &a;
    int *q = &b;

    swap_ptr(&p, &q);

    if ((p == &b) && (q == &a)) {
        printf("OK ;)\n");
        exit(EXIT_SUCCESS);
    } else {
        printf("KO ;(\n");
        exit(EXIT_FAILURE);
    }
}
```

Recherche dichotomique — *exercice de TD et TP*

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

   ```c
   float * search_dicho(float v, float *tab, int size);
   ```

2. Proposez une fonction `main()` pour tester cette fonction de recherche dichotomique.