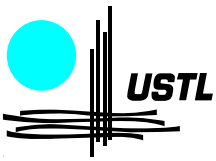


Licence d'informatique
Module de Pratique du C

Travaux dirigés

Manipulation de références et pointeurs

Philippe MARQUET

Novembre 2004

Les exercices proposés illustrent l'utilisation de références et de pointeurs. La représentation des tableaux à plusieurs dimensions est détaillée. On introduit les différentes classes d'allocation mémoire, dont l'allocation dynamique de mémoire. On traite aussi de la définition de types et de la manipulation de structures autoréférentes.

Exercice 1 (Passage de paramètres)
Écrivez une fonction qui échange les valeurs de deux variables entières.

Exercice 2 (Copie de tableau/chaîne de caractères, allocation dynamique de mémoire)
Question 2.1 Donnez le code C de la fonction, non normalisée ANSI, de la bibliothèque `string`

```
char *strdup(const char *str);
```

qui retourne une copie fraîchement allouée de la chaîne de caractère `str`.

On fera appel à la fonction de bibliothèque

```
#include <stdlib.h>
void *malloc(size_t size);
```

qui renvoie une référence sur une nouvelle zone mémoire de longueur de `size` octets. On pourra aussi utiliser les autres fonctions de la bibliothèque `string`.

Que retourne cette fonction `strdup()` en cas de problème ?

Question 2.2 Écrivez une fonction qui renvoie une copie « fraîche » d'un tableau de `n` entiers passé en paramètre.

Exercice 3 (Noms temporaires, allocations automatique, statique ou dynamique)
Donnez la définition d'une fonction qui retourne à chaque invocation une chaîne de caractères différente, par exemple en vue de l'utiliser comme un nom de fichier temporaire.

Exercice 4 (Tableaux à plusieurs dimensions)
Soit la déclaration d'un tableau

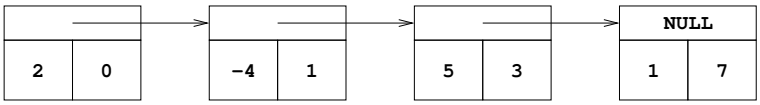
```
int b[3][5];
```

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;
```

Exercice 5 (Manipulation de polynômes)
Un polynôme est une liste de monômes. Un monôme est caractérisé par un coefficient (considéré entier) et un degré. On ne rangera en mémoire que les monômes de coefficient différent de zéro.

Par exemple le polynôme $x^7 + 5x^3 - 4x + 2$ constitué de 4 monômes sera représenté par :



Question 5.1 Donnez des définitions des types de données `monome_t` et `polynome_t`.

Question 5.2 Écrivez une fonction qui incrémente le degré de tous les monômes d'un polynôme.

Question 5.3 Écrivez une fonction qui évalue un polynôme pour une valeur passée en paramètre.

Question 5.4 Écrivez une fonction qui imprime sur la sortie standard la représentation d'un polynôme.

Question 5.5 Écrivez une fonction qui libère la mémoire occupée par un polynôme.

Question 5.6 Écrivez une fonction qui lit sur l'entrée standard un polynôme donné sous la forme d'une suite de coefficients des monômes de degrés décroissants. Le premier entier entré représente le coefficient du monôme de degré le plus élevé ; le nombre de coefficients entrés est égal au degré du polynôme plus 1. Par exemple, le polynôme de l'exemple est entré par la suite :

```
1 0 0 0 5 0 -4 2
```

Question 5.7 Écrivez une fonction qui additionne un monôme à un polynôme. Attention, on ne stocke jamais les monômes dont le coefficient est nul.

Question 5.8 Soyez rassurés, on ne vous demande pas de donner la définition d'une fonction qui additionne (ou multiplie !) deux polynômes. Donnez simplement les prototypes de telles fonctions.

Exercice 6 (Gestion de piles)
Il s'agit de donner une implantation des fonctions suivantes de manipulation d'une pile d'entiers

```
int is_empty(pile_t p);
int push(pile_t *pp, int val);
int pop(pile_t *pp, int *val);
```

Chacune de ces fonctions retourne une valeur non nulle en cas d'erreur.

La spécification précise que la taille de la pile n'est pas bornée, aussi utilisera-t-on une allocation dynamique de mémoire.

Question 6.1
- Donnez la définition d'un type `pile_t`.
- Précisez la représentation de la pile vide.
- Expliquez les prototypes donnés des fonctions.

Question 6.2 Donnez les définitions des fonctions `is_empty()`, `push()`, et `pop()`.

Question 6.3 Identifiez les fonctions supplémentaires que requière l'utilisation de valeurs de type `pile_t`. Donnez le fichier d'entête d'une bibliothèque de manipulation de piles.

Question 6.4 Soit la fonction `next_token()` suivante

```
/* next_token() retourne un token et positionne val en cas de token INT_TK */
enum token_e {INT_TK,           /* un entier */
              PRINT_TK,         /* affichage */
              ADD_TK,           /* addition  */
              MUL_TK,           /* produit */
              EOF_TK,           /* EOF */
              NONE_TK};         /* erreur */
static int val;
enum token_e next_token(void);
```

Écrivez un programme pour une simple calculette postfixée munie des opérations d'addition et de multiplication.

Exercice 7 (Tableaux grandissants)
Il s'agit de gérer des tableaux, définis comme une structure de données avec des accès indexés à partir de zéro aux données, dont la taille augmente avec les accès aux éléments grandissants.

L'interface de cette structure de données est, par exemple pour des grands tableaux d'entiers :

```
int ga_set(struct ga_s *ga, unsigned int index, int val);
int ga_get(struct ga_s *ga, unsigned int index, int *val);
struct ga_s ga_init(void);
int ga_destroy(struct ga_s *ga);
```

Question 7.1 Donnez la définition de l'implantation d'une telle structure par un tableau alloué dynamiquement dont on conserve la taille.

Quels sont les avantages et inconvénients d'une telle implantation ?

Question 7.2 Les inconvénients de la première implantation peuvent être supprimés en multipliant les zones mémoires nécessaires au fur et à mesure de l'agrandissement de la structure. Il s'agit alors de définir une structure de données pour regrouper ces zones mémoires. Donnez la définition d'une nouvelle implantation basée sur ce principe.

Exercice 8 (Allocation via un pool)
Plutôt que d'allouer dynamiquement au fur et à mesure de nombreuses petites structures, on peut gérer, au niveau applicatif, un pool de structures. Les requêtes d'allocation sont alors satisfaites en retournant des structures de ce pool, les désallocations venant à nouveau nourrir le pool, une allocation dynamique effective n'ayant lieu que lorsque le pool est vide.

Proposez une interface et son implantation pour la gestion d'un pool de valeurs d'un type `struct_t` connu.