

## TD4 — Types abstraits

### 1. Vers l’abstraction

Considérons une première version du paquetage **type\_sequence** définissant un type `SequenceEntiers`. Dans cette version l’utilisateur a un accès direct aux champs de la structure `SequenceEntiers` via l’opérateur `..`.

**Exercice 1.** Écrire un programme qui réalise les actions suivantes :

1. initialiser une séquence d’entiers `S` de taille 100 avec la valeur 0
2. lire à la console 20 entiers et les ranger dans la séquence à partir de l’indice 0
3. échanger le premier et le dernier entier de la séquence
4. changer la taille de la séquence

```
1
2 #define TAILLE_MAX 10000
3
4 /* structure permettant de manipuler des séquences contenant n entiers, */
5 /* n entre 1 et 10000 */
6 /* les valeurs des éléments de la séquence sont stockés dans le */
7 /* champ tab, des indices 0 à n-1 */
8 typedef struct {
9     int n;
10    int tab[TAILLE_MAX];
11 } SequenceEntiers;
```

**Exercice 2.** Considérons une deuxième version du paquetage **type\_sequence** qui fournit à l’utilisateur des fonctions et procédures lui permettant d’utiliser le type `SequenceEntiers` sans accéder aux champs de la structure. Cette deuxième version est donnée en annexe.

Écrire un programme qui réalise les mêmes actions que précédemment.

NB : nous avons vu qu’il n’est pas nécessaire de connaître la réalisation des procédures et fonctions de l’objet pour utiliser celui-ci.

### 2. Une pile d’entiers

**Exercice 3.** Quelles sont les opérations nécessaires à la manipulation d’une pile d’entiers ?

Décrire un paquetage **type\_pile** définissant un type `PileEntiers` et les opérations associés.

Écrire un programme de test du paquetage **type\_pile** constitué des opérations :

1. Créer une pile `p`
2. Mettre 4 en sommet de pile
3. Mettre 9 en sommet de pile
4. Dépiler le dernier élément, l’afficher
5. Mettre les éléments 10, 20, 30, ..., 80 en sommet de pile
6. Dépiler et afficher tous les éléments de la pile

## A. Annexe : fichiers `type_sequence.h` et `type_sequence.c`

```
1 #define TAILLE_MAX 10000
2
3 /* structure permettant de manipuler des séquences contenant n entiers, */
4 /* n entre 1 et 10000 */
5 /* les valeurs des éléments de la séquence sont stockés dans le */
6 /* champ tab, des indices 0 à n-1 */
7 typedef struct {
8     int n;
9     int tab[TAILLE_MAX];
10 } SequenceEntiers;
11
12 /* Opération de construction */
13
14 /* Créer une séquence de taille n avec des valeurs nulles */
15 void creer_sequence(int n, SequenceEntiers *s);
16
17 /* Opérations d'accès */
18
19 /* Taille de la séquence */
20 int nb_elements(SequenceEntiers *s);
21
22 /* Retourner le i-ème élément de la séquence s, 1 <= i <= n */
23 int get_element(SequenceEntiers *s, int i);
24
25 /* Opérations de modification */
26
27 /* Modifier la taille de la séquence */
28 /* n >= 0 est la nouvelle taille de la séquence */
29 /* NB : si la séquence est agrandie, la valeur des éléments ajoutés
30    n'est pas définie */
31 void modifier_taille(SequenceEntiers *s, int n);
32
33 /* Changer la valeur du i-ème élément de s avec la valeur v, 1 <= i <= n */
34 /* La fonction n'est pas définie si i > nb_elements(s) */
35 void put_element(SequenceEntiers *s, int i, int v);
```

```
1 #include "type_sequence.h"
2
3 void creer_sequence(int n, SequenceEntiers *s) {
4     int i;
5     s->n = n;
6     for (i = 0; i < n; i++) {
7         s->tab[i] = 0;
8     }
9 }
10
11 int nb_elements(SequenceEntiers *s) { return s->n; }
12
13 int get_element(SequenceEntiers *s, int i) { return s->tab[i - 1]; }
14
15 void modifier_taille(SequenceEntiers *s, int n) { s->n = n; }
16
17 void put_element(SequenceEntiers *s, int i, int v) { s->tab[i - 1] = v; }
```