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 àn-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 $10, 20, 30, \ldots, 80$ en sommet de pile
6. Dépiler et afficher tous les éléments de la pile

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

```c
#define TAILLE_MAX 10000

/* structure permettant de manipuler des séquences contenant n entiers, */
/* n entre 1 et 10000 */
/* les valeurs des éléments de la séquence sont stockés dans le */
/* champ tab, des indices 0 àn-1 */
typedef struct {
  int n;
  int tab[TAILLE_MAX];
} SequenceEntiers;

/* Opération de construction */

/* Créer une séquence de taille n avec des valeurs nulles */
void creer_sequence(int n, SequenceEntiers *s);

/* Opérations d'accè s */

/* Taille de la séquence */
int nb_elements(SequenceEntiers *s);

/* Retourner le i-è me élément de la séquence s, 1 <= i <= n */
int get_element(SequenceEntiers *s, int i);

/* Opérations de modification */

/* Modifier la taille de la séquence */
/* n >= 0 est la nouvelle taille de la séquence */
/* NB : si la séquence est agrandie, la valeur des éléments ajoutés
   n'est pas définie */
void modifier_taille(SequenceEntiers *s, int n);

/* Changer la valeur du i-è me élément de s avec la valeur v, 1 <= i <= n */
/* La fonction n'est pas définie si i>nb_elements(s) */
void put_element(SequenceEntiers *s, int i, int v);
```

```c
#include "type_sequence.h"

void creer_sequence(int n, SequenceEntiers *s) {
  int i;
  s->n = n;
  for (i = 0; i < n; i++) {
    s->tab[i] = 0;
  }
}

int nb_elements(SequenceEntiers *s) { return s->n; }

int get_element(SequenceEntiers *s, int i) { return s->tab[i - 1]; }

void modifier_taille(SequenceEntiers *s, int n) { s->n = n; }

void put_element(SequenceEntiers *s, int i, int v) { s->tab[i - 1] = v; }
```