TD/TP 12 — corrigés partiels

1 Interface des piles

Le fichier `pile.h` contient l'interface d'une bibliothèque manipulant des piles. Dans le source de chaque morceau de programme utilisant cette bibliothèque, on met la ligne

```
#include <pile.h>
```

Nous allons écrire deux implémentations des piles : avec un tableau (`pile_tableau.c`), et avec une liste (`pile_liste.c`). Pour lier un programme utilisant la bibliotèque des piles, on choisit l'une ou l'autre de ces implémentations et on met `pile_tableau.o` ou `pile_liste.o` dans la liste des fichiers objets à compiler.

2 Implémentation d'une pile par une liste

Voir le fichier `pile_liste.c`.

L'interface impose que le type `pile` soit un pointeur sur `struct pile`. De plus, les fonctions qui agissent sur des piles doivent les modifier *en place*, c'est-à-dire qu'elles peuvent modifier la `struct pile` mais pas le pointeur lui-même. On ne peut donc pas représenter la pile vide par un pointeur nul, on utilise à la place un pointeur vers une structure qui contient un pointeur nul.

Pour empiler un nombre, on crée une nouvelle cellule, on y place le nombre, on fait pointer la nouvelle cellule sur l'ancien sommet de la pile, et on fait de la nouvelle cellule le nouveau sommet de la pile. Pour dépiler un nombre, on fait essentiellement l'inverse : après avoir vérifié que la pile n'est pas vide, on stocke la valeur au sommet et l'adresse de la cellule suivante dans des variables locales, on efface la cellule devenue inutile, on met en place l'adresse du nouveau sommet et on renvoie la valeur anciennement au sommet.

3 Implémentation d'une pile par un tableau

Voir le fichier `pile_tableau.c`.

La structure `struct pile` comporte trois champs : un pointeur `p` vers un tableaux de flottants, un entier `taille` qui est la taille allouée pour `p` (`taille` sert à repérer les débordements), et un entier `n` qui indique la position du sommet de la pile. Le fond de la pile est `p[0]`, et le sommet est `p[n-1]`. Pour une pile vide, on a `n` $= 0$.

Lorsqu'on empile un nombre, si la pile est pleine (`n` = `taille`), on augmente la taille du tableau. Lorsqu'on dépile, on diminue la taille du tableau ; mais on ne le fait que si `(p->n < p->taille - TAILLE_INITIALE)`, histoire que si la pile passe son temps à osciller autour d'un multiple `TAILLE_INITIALE`, on ne fasse pas un `realloc` à chaque fois.