TD 4
Piles, files, tables de hachage

Polytech Marseille - IRM 3ème année
Alexandra Bac

Algorithmique et structures de données
2h TD - 2h TP

1 TD

On s'intéresse ici aux structures de piles et files, implémentées dans une versions "à effet de bord" (c'est-à-dire que les fonctions des types abstraits de données modifieront les piles/files et auront un type retour `void`). Ce TD explore les conséquences de ce choix et aboutit à une implémentation cohérente de cette politique d'accès.

Exercice 1 (Piles et application)**.** Comme indiqué précédemment, on souhaite implémenter une structure de piles dans une perspective "à effet de bord" en s'appuyant pour cela, autant que possible, sur les listes chaînées.

On inclura donc le module de listes du TD3 à partir duquel on définit les types de données suivants :

```
typedef int bool ;
struct zstack {
    int size ;
    List top ;
} ;
typedef struct zstack * Stack ;
```

(i) Commenter ce type. Comment est implémentée une pile vide ? Combien "d'étages" de références comporte une pile ?

(ii) On considère le code suivant :

```
Stack p1, p2 ;
p2 = create_empty_stack() ;
p1 = p2 ;
push(3,p2) ;
```

Représenter la mémoire après l'exécution de ce code.

(iii) Implémenter le type abstrait de piles dans cette version à effet de bord :

```
Stack create_empty_stack() ;
bool is_empty_stack(Stack) ;
void push(Data, Stack) ;
Data pop(Stack) ;
Data top(Stack) ;
```

ainsi que les fonctions de gestion mémoire appropriées.

(iv) Si on avait simplement implémenté les piles par le type suivant et les prototypes :

```
typedef List Stack ;
void create_empty_stack(Stack *) ;
bool is_empty_stack(Stack *) ;
void push(Data, Stack *) ;
Data pop(Stack *) ;
Data top(Stack *) ;
```

Représenter la mémoire après l'exécution du code 2 ; quel est le problème ?

Exercice 2 (Files)**.** De même, on souhaite maintenant implémenter une structure de files dans cette même philosophie "à effet de bord".

(i) Adapter le type de données `Stack` pour définir une structure de files `Queue`.

(ii) Implémenter les fonctions du TAD des files :

```
Queue create_empty_queue() ;
bool is_empty_queue(Queue) ;
void push_back(Data, Queue) ;
Data pop_front(Queue) ;
Data top(Queue) ;
```

ainsi que les fonctions de gestion mémoire appropriées.

Les files seront utilisées et mises en application au chapitre suivant.

2 TP

Implémenter en C :
- un module `stack`.
- un module `queue`.

On utilisera la compilation séparée.

3 Challenge

Exercice challenge. On veut écrire une programme permettant d'évaluer une expression arithmétique en notation postfixée. Dans cette notation, les arguments sont donnés avant l'opération. Par exemple :
- `1 4 + 5 * 3 2 * +` correpond à $((1+4)*5)+(3*2)$ et doit donc être évalué à 31
- `1 4 5 + * 3 2 * +` correpond à $((1*(4+5)))+(3*2)$ et doit donc être évalué à 15

Les chaînes seront stockées dans un tableau de caractères et on considèrera les opérations +, -, *, /. Pour les évaluer, on utilisera une pile :
- en rencontrant des entiers, ces derniers doivent être empilés
- en rencontrant une opération, elle doit être appliquée aux éléments du sommet de la pile

Votre programme déterminera si l'opération est correcte (on peut arrive à la fin de l'évaluation ET à la fin la pile contient seulement un entier) et renverra la valeur évaluée.

Indications C. A partir du tableau de caractères, il faut lire un entier quand cela est possible (plusieurs caractères ...), soit lire une opération, et sauter les espaces. Pour cela, on utilisera la fonction `strtol`. On commencera par lancer sur les données :

```
n = strtol(data, &fin, 10);
```

où `char *fin` reçoit l'adresse du premier caractère non conforme à un entier. Cette fonction, dans tous les cas, permet de passer les espaces. Pour tester si un entier a été rencontré ou non, on utilisera :

```
if ((n!=0) || (errno != EINVAL)) // On a bien trouvé un entier
```

Vous trouverez dans `materiel_challenge.zip` des modules de listes et piles vous permettant de partir sur cette base.