TD/TP 12 : piles

Gilles Peskine

10 janvier 2003

1 Interface des piles

Voici le contenu du fichier `pile.h` :

```
#ifndef PILE_H_
#define PILE_H_
struct pile; /* structure incomplète */
typedef struct pile *pile;
pile cree_pile(void); /* crée une pile vide */
void empile(pile p, float x); /* ajoute un élément à la pile */
int est_vide(pile p); /* teste si la pile est vide */
float depile(pile p); /* renvoie et retire la valeur au sommet de la pile */
float lit(pile p); /* renvoie la valeur au sommet de la pile, sans modifier la pile */
void efface(pile p); /* retire tous les éléments de p */
#endif
```

À cette interface, on peut associer deux implémentations : soit on utilise une liste, soit on utilise un tableau.

2 Implémentation d'une pile à l'aide d'une liste

Il y a deux manières classiques de représenter une pile : soit dans une liste, soit dans un tableau.

Lorsqu'on représente une pile par une liste, pousser un élément sur la pile se fait en ajoutant une cellule au début de la liste, et dépiler un élément se fait en retirant la cellule en tête de liste.

Définir un type `pile` et des fonctions `empile`, `cree_pile`, `est_vide`, `depile`, `lit`, `efface`.

3 Implémentation d'une pile à l'aide d'un tableau

Maintenant, on représente une pile par un tableau t et un entier n. Le sommet de la pile est $t[n]$; lorsqu'on empile un élément, on incrémente n, et lorsqu'on dépile, on décrémente n.

Lorsqu'on empile un élément, il faut vérifier qu'il y a de la place dans le tableau. Si ce n'est pas le cas, soit on signale une erreur (débordement de pile), soit on étend la pile (`realloc` du tableau).

Définir un type `pile` et des fonctions `empile`, `cree_pile`, `est_vide`, `depile`, `lit`, `efface`.

4 Calculatrice en notation polonaise inversée (RPN)

En mathématiques, on utilise souvent la *notation infixe* pour les opérateurs, c'est-à-dire qu'on écrit l'opérateur entre ses arguments ($x + 2$). Certaines calculatrices utilisent une autre notation : on écrit d'abord les arguments, et ensuite l'opérateur, par exemple `x 2 +`. C'est la *notation polonaise inversée*. Un avantage de cette notation est qu'on n'a pas besoin de parenthèses : par exemple, $(x + 2) * 3$ s'écrit `x 2 + 3 *`, tandis que $x + (2 * 3)$ s'écrit `x 2 3 * +`.

Le fonctionnement d'une calculatrice RPN est le suivant : lorsque l'utilisateur tape un nombre, celui-ci est empilé ; lorsque l'utilisateur tape un opérateur binaire (`+`, `-`, `*`, `/`), la calulatrice dépile deux nombres, effectue l'opération demandée, et place le résultat sur la pile. L'utilisateur a connaissance du résultat parce que la calculatrice affiche en permanence la valeur au sommet de la pile.

Écrire un programme qui se comporte comme une calculatrice à 4 opérations. Il lit les nombres et les opérateurs sur son entrée standard, et affiche le nombre au sommet de la pile à chaque opération.

Voici un exemple d'utilisation (ce que tape l'utilisateur est souligné) :

```
$ ./calculatrice
3 7 +
10
7 * 5 -
70
65
```

5 Opérateurs supplémentaires

(a) Ajouter à la calculatrice un opérateur unaire `_` (trait de soulignement) qui calcule l'opposé de son argument. `x _` est donc équivalent à `0 x -`.

(b) Ajouter à la calculatrice un opérateur `if=` à quatre arguments : si la pile contient ... $a\,b\,x\,y$ (avec y au sommet), l'opérateur `if=` dépile ces quatre nombre et rempile a si x et y sont égaux, b s'ils sont différents.

De même, ajouter un opérateur `if<` qui revoie a si $x < y$ et b sinon.

6 Opérations sur la pile

Il est quelquefois utile de faire des manipulations de la pile autre que des opérations numériques. Ajouter les opérateurss suivants :

- `dup`, qui duplique la valeur au sommet de la pile (avant : ... x ; après : ... $x\,x$).
- `swap`, qui échange les deux valeurs au sommet de la pile (avant : ... $x\,y$, après : ... $y\,x$).

On peut ajouter encore d'autres opérateurs, notamment pour accéder à des valeurs loin du sommet.

7 Macros

Dans un langage de programmation comme C, lorsqu'on prévoit d'effectuer le même calcul plusieurs fois, on définit une fonction. Nous allons faire un grand pas dans la transformation de notre calculatrice en un vrai ordinateur en ajoutant des *macros*.

Définir une macro, c'est associer à un nom une séquence de mots. Par exemple, si on définit la macro `abs` comme la séquence `"dup dup 0 swap - swap 0 swap if<"`, cela revient à disposer d'un opérateur unaire `abs` qui calcule la valeur absolue de son argument.

On remarque qu'une macro prend « automatiquement » des arguments — elle les trouve sur la pile. Elle place son résultat sur la pile ; rien n'empêche d'ailleurs une macro de renvoyer zéro ou plusieurs résultats.

Modifier la calculatrice pour qu'elle lise des définitions de macros dans un fichier `macros.txt` lorsque le programme démarre. Ce fichier contient une définition de macro par ligne, en donnant le nom, puis une espace, puis le contenu de la macro (des nombres et des opérateurs). Par exemple, voici un fichier `macros.txt` qui définit trois macros `minus`, `if>` et `abs` :

```
minus 0 swap -
if> swap if<
abs dup dup minus swap 0 if>
```