UE Algorithmes et programmation
L1/S2 – TD/TP n^o 7
Pile et file – Année 2015

Vous pouvez récupérer du code ici :
`http://lab-sticc.univ-brest.fr/~rodin/FTP/Enseignements/L1/AlgoEtProg`

Sans indication particulière, les fonctions ne doivent pas être écrites sous forme récursives. La récursivité ne sera utilisée que lorsque cela sera indiqué explicitement dans le sujet.

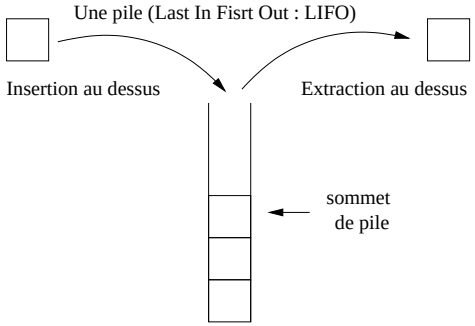
TD

1 TD Exercice 1 : gestion de pile

Rappels sur les piles :
- Les piles sont utilisées en programmation pour gérer des objets qui sont en attente d'un traitement ultérieur, traitement dans l'**ordre inverse de l'ordre d'arrivée**.
- Dans une pile les éléments sont systématiquement : **ajoutés au dessus** et **extraits au dessus**

En cours, non avons défini le type abstrait pile :

- **Type :** pile d'objets de type élément
- **Utilise :** booléen, élément
- **Opérations :**
  - créer : → pile
  - estVide : pile → booléen
  - estPleine : pile → booléen
  - sommet : pile → élément
  - empiler : pile X élément → pile
  - dépiler : pile → pile
- **Préconditions :**
  - sommet(p) ssi estVide(p) = faux
  - empiler(p,e) ssi estPleine(p) = faux
  - dépiler(p) ssi estVide(p) = faux
- **Axiomes :**
  - estVide(créer()) = vrai
  - estVide(empiler(p,e)) = faux
  - sommet(empiler(p,e)) = e
  - dépiler(empiler(p,e)) = p



1. En cours, nous avons défini la structure de pile suivante :

```
#define SIZE 10

struct stpile { int tab[SIZE];
                int taille;    /* taille du tableau tab      */
                int nbElems;   /* nombre de valeurs stockees */
              };

typedef struct stpile pile;
```

   Ecrire les fonctions C suivantes :
   - `pile creerPile(void);`
   - `void afficherPile(pile p);`
   - `int  estVidePile(pile p);`
   - `int  estPleinePile(pile p);`
   - `int  sommetPile(pile p);`
   - `void empilerPile(pile* p, int e);`
   - `void depilerPile(pile* p);`

   Rappel : `creerPile` ne prend pas de paramètre et retourne une `pile` ; `afficherPile` prend une `pile` en paramètre et ne retourne rien (`void`) ; `estVidePile` prend une `pile` en paramètre et retourne un `int` (1 : vrai si la pile est vide, 0 : faux sinon) ; `estPleinePile` prend une `pile` en paramètre et retourne un `int` (1 : vrai si la pile pleine,0 : faux sinon) ; `sommetPile` prend une `pile` en paramètre et retourne un `int` (le sommet de la pile) ; `empilerPile` prend une `pile` (à modfifier) en paramètre, un `int` à empiler et ne retourne rien (`void`) ; `depilerPile` prend une `pile` (à modfifier) en paramètre er ne retourne rien (`void`), `depilerPile` dépile un élément (le sommet).

2. La notation polonaise inversée est une façon d'écrire les expressions arithmétiques sans indiquer de multiples parenthèses. Une expression arithmétique écrite en notation polonaise inversée est une suite de nombres et d'opérateurs (+, -, / ou *) et se lit (et s'évalue) de gauche à droite de la façon suivante :
   - lorsqu'un nombre est lu, il est empilé sur une pile (initialement vide) ;
   - lorsqu'un opérateur est lu, les deux premiers éléments de la pile sont récupérés et dépilés, l'opération est effectuée sur les deux valeurs obtenues, et le résultat est empilé ;
   - le résultat de l'expression est le premier (et normalement le seul) élément de la pile à la fin de la lecture.

   a) Appliquer la méthode exposée pour calculer : `17 10 -` , `3 28 7 / +` et `3 28 + 7 /` Comment s'écrivent ces expressions en notation classique ?

   b) Traduire en notation polonaise inversée les expressions suivantes : `(19 * 6) - 7` et `(4 - 2 * (7 + 6)) + 3 * 5`

2 TD Exercice 2 : gestion de file circulaire

En cours, nous avons vu la notion de file avec une implémentation nécessitant un décalage des éléments lors de l'ajout d'un élément dans la file.
Nous avons également vu qu'il était possible d'implémenter une file à l'aide d'un tableau "circulaire" grâce à de 2 indices sur le tableau (`tete` et `queue`).

```
#define SIZE 10

struct stfile { int tab[SIZE];
                int taille;    /* la taille du tableau tab                  */
                int nbElems;   /* pratique, pas obligatoire, nb val stockees */
                int tete;      /* indice element de tete                    */
                int queue;     /* indice element de queue                   */
              };

typedef struct stfile file;
```

Rappels :
- Lorsque l'on ajoute un élément, on ajoute 1 à `queue` (si `queue` est en dehors des limites du tableau, on le fait passer à 0).
- Lorsque l'on enlève l'élément de tête, on ajoute 1 à `tete` (si `tete` est en dehors des limites du tableau, on le fait passer à 0).

Ré-implémenter toutes les fonctions de gestion de file vues en cours (`creerFile`, `afficherFile`, `estVideFile`, `estPleineFile`, `premierFile`, `enfilerFile`, `defilerFile`).

TP

1 TP Exercice 1 : gestion de pile

1. Programmer les fonctions de gestion de pile de l'exercice 1 du TD :
   - `pile creerPile(void) ;`
   - `void afficherPile(pile p) ;`
   - `int  estVidePile(pile p) ;`
   - `int  estPleinePile(pile p) ;`
   - `int  sommetPile(pile p) ;`
   - `void empilerPile(pile* p, int e) ;`
   - `void depilerPile(pile* p) ;`
2. Afin de tester ces fonctions, utiliser le programme `menuPile.c` (un menu avec la possibilité d'appeler chacune des fonctions) se trouvant :
   `http://lab-sticc.univ-brest.fr/~rodin/FTP/Enseignements/L1/AlgoEtProg`
3. Ecrire un programme `calculette.c` qui lit des commandes en notation polonaise inversée, jusqu'à afficher le résultat. Les commandes (des chaînes de caractères) successives sont lues à l'aide de la fonction `scanf`.
   Ces commandes peuvent être :
   - `"+"`, `"-"`, `"*"`, `"/"` qui fait l'opération correspondante (en entier) sur les deux premiers éléments de la pile de calcul ;
   - `"p"` qui affiche le contenu de la pile ;

   - `"q"` qui permet de sortir de la boucle pour ensuite afficher le résultat (le sommet de la pile) ;
   - un ***nombre***, transcrit de chaîne en entier par la fonction `atoi` puis empilé.
   Exemple d'utilisation de la fonction `atoi` :

```
#include <stdio.h>
#include <stdlib.h>    /* Pour connaitre la fonction atoi */

int main(void)
{
 char str[256]="1234";
 int  val;
 val = atoi(str);
 printf("%d\n",val);   /* Affichage de l'entier 1234 */
 return 0;
}
```

   Le programme devra donc créer une pile vide, lire est exécuter les commandes jusqu'à la commande `"q"`, afficher le sommet de la pile et terminer.

2 TP Exercice 2 : autre application des piles

Ecrire un programme permettant de vérifier qu'un mot est bien parenthésé.
Ainsi, ce programme doit :
- accepter les mots comme `()`, `(()(())` ou `(((()())())`
- rejeter les mots comme `)(`, `()((` ou `(((()))))`

Le programme doit commencer par la lecture d'une chaîne au clavier avec `scanf`. Puis, il doit analyser, un par un, les caractères de la chaîne.

Rappels :

- la fonction `strlen` permet d'obtenir la longueur (le nombre de caractères) d'une chaîne.
- on peut accéder au `i`ème d'une chaine `str` de la façon suivante : `str[i]`.

L'analyse se fait comme décrit ci–après.

Pour chacun des caractères de la chaîne :
- si c'est une parenthèse ouvrante '(', une valeur est empilée ;
- si c'est une parenthèse ')' fermante, une valeur est dépilée.

Le mot est accepté si :
- la pile n'est jamais vide à la lecture d'une fermante ; et si
- la pile est vide lorsque tout le mot a été analysé.