Algorithmique et Structures de données 1

LIV 2022-2023

Travaux Dirigés 8

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.

Exercice 1. *Enfiler des piles*

Qu'affiche ce code :

```
pileI_t p; // pile d'int
fileI_t f; // file d'int
initialiserPI(&p);
initialiserFI(&f);

emPilerI(&p,1);
enFilerI(&f,2);
emPilerI(&p,3);
emPilerI(&p,4);
enFilerI(&f,5);
printf("%d\n",deFilerI(&f));
printf("%d\n",dePilerI(&p));
emPilerI(&p,6);
emPilerI(&p,7);
printf("%d\n",dePilerI(&p));
enFilerI(&f,8);
printf("%d\n",deFilerI(&f));
printf("%d\n",dePilerI(&p));
enFilerI(&f,dePilerI(&p));
printf("%d\n",dePilerI(&p));
printf("%d\n",deFilerI(&p));
printf("%d\n",deFilerI(&f));

detruirePI(&p);
detruireFI(&f);
```

Exercice 2. *Parcours profondeur sans appels récursif*

Écrire une fonction `void parcouPrefPile(noeud_t* racine)` qui effectue un parcours en profondeur préfixe d'un arbre SANS aucun appel récursif mais en utilisant une pile.

La structure d'un noeud de l'arbre est donnée par

```
typedef struct s_noeud_t
{
   int v; // étiquette du noeud (v pour valeur)
   struct s_noeud_t* g; // pointeur vers la racine du sous-arbre  gauche
   struct s_noeud_t* d; // pointeur vers la racine du sous-arbre droit
} noeud_t;
```

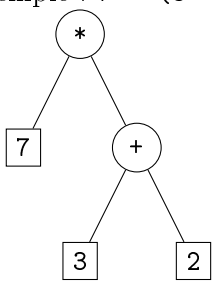
On suppose que l'on peut utiliser une pile de type `pileA_t` et les opérations

- `void initialiserPA(pileA_t* p)`
- `void detruirePA(pileA_t* p)`
- `void emPilerA(pileA_t* p,noeud_t* n)`
- `noeud_t* dePilerA(pileA_t* p)`
- `int estVidePA(pileA_t* p)// renvoie 1 si *p est une pile vide`

Notation Polonaise inversée

Une opération arithmétique peut être vue comme un arbres étiqueté :
— où les **feuilles** sont étiquetées par des nombres,
— où les **noeuds internes** sont étiquetés par des symboles d'opérations.

Exemple : `7 * (3 + 2)`



Voici deux conventions d'écritures d'opérations :
— l'écriture **infixée** (la plus utilisée) : `7 * (3 + 2)` qui nécessite l'utilisation de parenthèses.
— l'écriture **postfixée** (moins répandue) : `7 3 2 + *` qui n'a pas besoin de parenthèses. Il s'agit d'un parcours postfixe (ou parcours suffixe) de l'arbre. (Cette écriture est aussi connue comme <<la notation polonaise inversée>>)

Exercice 3. *Un interpréteur*

On souhaite construire un interpréteur qui calcule le résultat d'une opération écrite en postfixée.
On suppose (pour simplifier le problème) que :
— L'opération contient uniquement des nombres à un seul chiffre et les opérations `+` et `*`.
— L'expression de l'opération est contenue dans un tableau de `char` terminée par le caractère nul.
— Chaque symbole de l'opération est représenté par son caractère ASCII. Exemple :

'7'	'3'	'2'	'+'	'*'	'\0'		

— On peut utiliser une pile d'entiers (de type `pileInt_t`) et les opérations `emPiler`, `dePiler`, `estVide`, `initialiserP`, `detruireP`.

Compléter le corps de la fonction suivante. (Aide : utiliser une pile d'entiers).

```
int calcul_infixe(char* tab)
// renvoie le résultat de l'opération contenue dans tab
{
   // à compléter
}
```