TD1 – Pile et File (corrigé)

ING1 – Algorithmique II

Année 2011–2012

1 **Inversion d'une pile**

Écrire un algorithme qui inverse une pile. On se servira ici d'une file intermédiaire. □

```
procedure inverserPile(p: Pile de Élément (E/S))
  f: File de Élément
  créerFile(f)
  tant que non (estVide(p)) faire
    enfiler(f,sommet(p))
    dépiler(p)
  fin tant que
  tant que non (estVide(f)) faire
    empiler(p,tête(f))
    défiler(f)
  fin tant que
fin procedure
```

2 **Suppression de la première occurence dans une pile**

Écrire un algorithme qui supprime *la première* occurence d'un élément donné dans une pile. Les éléments restants doivent respecter l'ordre initial. □

```
procedure supprimerUneOccurence(p: Pile de Élément (E/S); e: Élément)
  ptmp: Pile de Élément
  trouvé: booléen
  trouvé←FAUX
  créerPile(ptmp)
  tant que non (estVide(p)) et non (trouvé) faire
    si sommet(p)=e alors
      trouvé←VRAI
    sinon
      empiler(ptmp,sommet(p))
```

```
      fin si
      dépiler(p)
   fin tant que
   tant que non (estVide(ptmp)) faire
      empiler(p,sommet(ptmp))
      dépiler(ptmp)
   fin tant que
fin procedure
```

(3) **Suppression de toutes les occurences dans une pile**

Écrire un algorithme qui supprime *toutes* les occurences d'un élément donné dans une pile. Les éléments restants doivent respecter l'ordre initial. □

```
procedure supprimerUneOccurence(p: Pile de Élément (E/S); e: Élément)
   ptmp: Pile de Élément
   créerPile(ptmp)
   tant que non (estVide(p)) faire
      si sommet(p)≠ e alors
         empiler(ptmp,sommet(p))
      fin si
      dépiler(p)
   fin tant que
   tant que non (estVide(ptmp)) faire
      empiler(p,sommet(ptmp))
      dépiler(ptmp)
   fin tant que
fin procedure
```

(4) **Évaluation d'une expression postfixée**

Écrire un algorithme qui évalue une expression postfixée. On rappelle que dans une expression postfixée, on écrit l'opérateur *après* les opérandes.

Exemples :

$(5+3)\times 2-7 \quad \Rightarrow \quad 5\ 3+2\times 7-$

$9-(5+3) \quad \Rightarrow \quad 9\ 5\ 3+-$

□

```
fonction evaluerEPF(epf: Tableau de Élément): réel
```

```
dernier, courant: entier
operateur: chaîne
terme: Élément
operande1, operande2, operande3: réel
div0: booléen
pile: Pile de Élément
dernier←longueur(epf)
courant←0
créerPile(pile)
div0←FAUX
tant que (courant ⩽ dernier) et non (div0) faire
    terme←epf[courant]
    si estRéel(terme) alors
        empiler(pile,terme)
    sinon
        operateur←terme
        operande2←sommet(pile)
        dépiler(pile)
        operande1←sommet(pile)
        dépiler(pile)
        choix operateur parmi
            "+": operande3←operande1 + operande2
            "−": operande3←operande1 − operande2
            "×": operande3←operande1 × operande2
            "÷":
            si operande2 = 0 alors
                div0←VRAI
            sinon
                operande3←operande1 ÷ operande2
            fin si
        fin choix
        si non (div0) alors
            empiler(pile,operande3)
        fin si
    fin si
```

```
        courant←courant + 1
    fin tant que
    si div0 alors
        erreur "Division par 0"
    fin si
    retourner sommet(pile)
fin fonction
```

(5) **Tri d'une pile**

On désire trier une pile. On utilisera deux piles supplémentaires temporaires afin de pouvoir modifier la pile passée en entrée.

Exemple :

```
E : (27, 82, 37, 0) S : () A : ()
E : (82, 37, 0) S : (27) A : ()
E : (82, 37, 0) S : () A : (27)
E : (37, 0) S : (82) A : (27)
E : (37, 0) S : (27, 82) A : ()
E : (37, 0) S : (82) A : (27)
E : (0) S : (37, 82) A : (27)
E : (0) S : (27, 37, 82) A : ()
E : () S : (0, 27, 37, 82) A : ()
```

□

On utilise une pile de sortie qui stocke les éléments dans l'ordre croissant et une pile auxilière qui les stocke dans l'ordre inverse. On opère comme suit :

- Lorsqu'un élément en sommet de la pile d'entrée est inférieur ou égal à l'élément au sommet de la pile de sortie (ou que celle-ci est vide), l'élément est dépilé de l'entrée et empilé sur la sortie.
- S'il est au contraire plus grand, alors :
  - la pile de sortie est dépilée sur la pile auxilière tant que le sommet de la pile de sortie est inférieur au sommet de la pile d'entrée et que la pile de sortie est non vide
  - le sommet de la pile d'entrée est placé dans la pile de sortie
  - la pile auxilière est vidée dans la pile de sortie

```
procedure trierPile( entree: Pile de Élément (E/S))
    sortie, inter: Pile de Élément
```

```
e: Élément
encore: booléen
créerPile(inter)
créerPile(sortie)
tant que non (estVide(entree)) faire
    e←sommet(entree)
    si estVide(sortie) alors
        empiler(sortie,e)
        dépiler(entree)
    sinon si e ⩽ sommet(sortie) alors
        empiler(sortie,e)
        dépiler(entree)
    sinon
        encore←VRAI
        tant que non (estVide(sortie)) et encore faire
            si sommet(sortie) < e alors
                empiler(inter,sommet(sortie))
                dépiler(sortie)
            sinon
                encore←FAUX
            fin si
        fin tant que
        empiler(sortie,e)
        dépiler(entree)
        tant que non (estVide(inter)) faire
            empiler(sortie,sommet(inter))
            dépiler(inter)
        fin tant que
    fin si
fin tant que
tant que non (estVide(sortie)) faire
    empiler(inter,sommet(sortie))
    dépiler(sortie)
fin tant que
tant que non (estVide(inter)) faire
```

```
        empiler(entree,sommet(inter))
        dépiler(inter)
    fin tant que
fin procedure
```