Algorithmique et structures de données
TP3 – Implémentation statique des piles et files

Lamia BENAMARA

Consigne :
Pour les séances TP, le langage de programmation utilisé sera le langage Java.
Pensez toujours à soigner vos traces d'exécution.
Pensez à testez chaque fonction dans votre programme principal.

Exercice 1 : Fonctionnement pile/file

a) Donner le contenu de la pile pour chaque opération de la suite Q*UES**TI*ON*FAC*IL**E*. Chaque lettre provoque un empilement et chaque astérisque un dépilement.

b) Donner le contenu de la file pour chaque opération de la suite Q*UES**TI*ON*FAC*IL**E*. Chaque lettre provoque un empilement et chaque astérisque un dépilement.

Exercice 2 : Implémentation statique de la pile (LIFO)

1. Définissez une structure **pile** pour représenter une pile d'entiers en utilisant un tableau d'une capacité maximale égale à N (constante).

2. Implémentez toutes les fonctions nécessaires pour manipuler la pile (utilisez les algorithmes vus en cours) :

   - `Initialiser_pile()` : crée et renvoie une pile vide.
   - `est_pile_vide(P)` : renvoie un booléen indiquant si P est vide.
   - `empiler( P,x)` : ajoute x au sommet de la pile P et retourne un booléen (faux si la pile est pleine)
   - `depiler(P)` : supprime et renvoie le sommet de P, si P est non vide
   - `sommet(P)` : renvoie le sommet de P si P est non vide

3. Écrire une fonction `pile_entier` qui crée une pile de capacité n (<N) contenant les entiers de 1 à n, n étant au sommet.

4. Écrire une fonction `afficher_pile(p)` qui affiche le contenu d'une pile d'entiers en commençant par le sommet, la pile devant rester inchangée en fin de fonction.

5. Écrire une fonction `inverser_pile(p)` qui prend en paramètre une pile et qui retourne la pile inverse (la pile p ne doit pas changer)

6. Ecrire un algorithme pour calculer la factorielle en simulant la récursivité à l'aide d'une pile.

Exercice 3 : Implémentation statique de la file (FIFO)

1. Définissez une structure **file** pour représenter une file d'entiers en utilisant un tableau d'une capacité maximale égale à N (constante).

2. Implémentez toutes les fonctions nécessaires pour manipuler la file (utilisez les algorithmes vus en cours) :

   - `Initialiser_file()` : crée et renvoie une file vide.
   - `est_file_vide(F)` : renvoie un booléen indiquant si F est vide et retourne un booléen (faux si la file est pleine)
   - `enfiler( F,x)` : ajoute x en queue de la file F.
   - `defiler(F)` : supprime et renvoie la tête de F, si F est non vide

3. Écrire une fonction `File_entier` qui crée une file de capacité n (<N) contenant les entiers de 1 à n, n étant le dernier élément enfilé.

4. Écrire une fonction `afficher_File(F)` qui affiche le contenu d'une file d'entiers en suivant l'ordre FIFO, la file devant rester inchangée en fin de fonction.

5. Écrire une fonction `inverser_File(F)` qui prend en paramètre une file et qui retourne la file inverse (la file ne doit pas changer)

6. Écrire une fonction `supprimer_element_File(F, e)`, qui prend en paramètre une file, supprime toutes les occurrences de e et remet les éléments restants dans l'ordre initial (vous ne devez pas utiliser d'autres structures de données temporaires)

7. Un guichet s'ouvre : Des clients de la poste font la queue à un guichet, lorsque, par chance, un autre guichet s'ouvre. Pour respecter l'ordre dans la file, le plus juste (même si ce n'est pas le cas en pratique) serait que les deuxième, quatrième, sixième... personnes dans la file partent vers une autre file. Écrire une fonction `separe_file(F)` renvoyant une file contenant les éléments (entiers) déplacés, à la fin, la file F doit contenir uniquement les éléments restants.

Exercice 4 : Implémentation statique du Buffer tournant

Le but de cet exercice est d'implémenter la version statique (via un tableau) d'un buffer tournant. Pour rappel, le principe du buffer tournant est :

- On a un ensemble de données homogènes gérées sur un principe de lecture/écriture concurrente : lecture et écriture peuvent se produire de manière simultanée (mais sont conditionnées par l'autre opération) sur une zone tampon.

- Ce tampon est basé sur le modèle du buffer tournant avec une suite de postes limitée à une capacité maximale *N*
- Une lecture ne peut être faite que si une écriture a déjà été faite, mais une écriture ne peut être faite que si la lecture a été effectuée sous peine de perdre l'information.

Réaliser en java un tampon tournant de caractères, de capacité initiale N(constante) :

1. Définissez d'abord la structure puis les fonctions lire et écrire en utilisant la version à base de tableau (reprenez les algorithmes vus en cours).
2. Concevoir et réaliser une fonction ‘scénario de test' qui exécute une séquence de lectures et d'écritures qui conduit à des débordements en écriture ou des échecs en lecture.
3. Les échecs d'écriture et de lecture doivent afficher des messages d'erreurs.
4. Les traces d'exécution du scénario doivent être soignées et particulièrement lisibles.