Travaux dirigés #4 – Files et piles

Sauf mention contraire, les codes sont à donner en `Pseudo-code`.

Exercice 1

Oh there's stacks to do and there's stacks to see.

L'objectif de cet exercice est de manipuler la structure de données `Pile`. Les seules fonctions autorisées sont décrites ci-après. En particulier, il n'est pas permis d'utiliser directement des listes ou des tableaux.

Le *type abstrait* Pile

Les opérations suivantes sont autorisées, mais leur implémentation n'est pas connue. La notation pointée indique l'existence d'une classe : ce choix est fait pour insister sur la Programmation Orientée Objet mais ce n'est pas une obligation.

- `Pile()` : retourne une pile vide
- `P.est_vide()` : retourne `Vrai` si et seulement si `P` est vide
- `P.empiler(x)` : empile `x` sur la pile `P`
- `P.depiler()` : dépile (supprime **et** retourne) le sommet de `P`
- `P.sommet()` : retourne le sommet de `P`.

Par souci de simplicité, une pile est représentée par la notation $\underline{2} \cdot 5 \cdot 8 \cdot 1$ où l'élément souligné représente le sommet de la pile. La pile vide est représentée par $\emptyset$.

1. Donner la trace du programme suivant en supposant que la pile `P` vaut : $\underline{1} \cdot 2 \cdot 3 \cdot 4$

```
1 P' ← Pile()
2 tant que non(P.est_vide()) faire
3 |   x ← P.depiler()
4 |   P'.empiler(x)
```

2. Définir une fonction `inverser_pile(P)` retournant l'inverse de la pile `P`. **Au retour de la fonction, la pile d'origine pourra être vide**. Par exemple : `inverser_pile(P)` pour $\texttt{P} = \underline{1} \cdot 4 \cdot 2 \cdot 3 \cdot 5$ retourne la pile $\underline{5} \cdot 3 \cdot 2 \cdot 4 \cdot 1$.
3. Définir une fonction `appartient_pile(P,e)` qui retourne `Vrai` si l'élément `e` est présent dans la pile `P`, et `Faux` sinon. **La pile `P` doit rester inchangée au retour de la fonction.**
4. Définir une fonction `copier_pile(P)` retournant une copie de la pile `P` passée en paramètre. **La pile `P` doit rester inchangée au retour de la fonction.**

Exercice 2

I jump the queue 'cause I'm smarter than you.

L'objectif de cet exercice est de manipuler la structure de données `File`. Les seules fonctions autorisées sont décrites ci-après. En particulier, il n'est pas permis d'utiliser directement des listes ou des tableaux.

Le *type abstrait* File

Les opérations suivantes sont autorisées, mais leur implémentation n'est pas connue. La notation pointée indique l'existence d'une classe : ce choix est fait pour insister sur la Programmation Orientée Objet mais ce n'est pas une obligation.

- `File()` : retourne une file vide
- `F.est_vide()` : retourne `Vrai` si et seulement si `F` est vide
- `F.enfiler(x)` : enfile `x` dans la file `F`
- `F.defiler(x)` : défile (supprime **et** retourne) l'élément en tête de `F`
- `F.tete()` : retourne l'élément en tête de la file `F`.

Par souci de simplicité, une file est représentée par la notation $2 \cdot 5 \cdot 8 \cdot \underline{1}$ où l'élément souligné représente le premier élément ajouté à la file. La file vide est représentée par $\emptyset$.

1. Donner la trace du programme suivant en supposant que la file `F` vaut : $4 \cdot 3 \cdot 2 \cdot \underline{1}$

```
1 F' ← File()
2 tant que non(F.est_vide()) faire
3 |   x ← F.defiler()
4 |   F'.enfiler(x)
```

2. Définir une fonction `copie_file(F)` prenant en paramètre une file `F` et retournant une copie de `F`. **La file passée en paramètre est inchangée au retour de la fonction.**
3. Définir une fonction `pile_vers_file(P)` prenant en paramètre une pile `P` et retournant une file contenant les mêmes valeurs. L'ordre de construction de la file sera celui d'extraction de la pile.
4. Définir une fonction `file_vers_pile(F)` prenant en paramètre une file `F` et retournant une pile contenant les mêmes valeurs. L'ordre de construction de la pile sera celui d'extraction de la file.