1 tp6_listes

1.1 Objectifs du travail pratique

1. Création d'une librairie en Rust.
2. Implémentation d'une pile statique et d'une pile dynamique.
3. Instanciation et utilisation de cette.
4. Écriture de programmes de tests.
5. Indentation, commentaires et modularisation du code.
6. Utilisation des traits.
7. Gestion des erreurs.

1.2 Enoncé

Écrire une librairie Rust permettant la gestion de piles statiques et dynamiques. Cette librairie sera utilisée pour effectuer le tri d'un tableau à l'aide de deux piles. Le tri fonctionne de la manière suivante.

- Le tableau est parcouru élément par élément.
- Une première pile (celle de gauche dans l'exemple de la figure 1 contient toujours des éléments triés en ordre croissant en partant du sommet. Une seconde pile (celle de droite dans l'exemple de la figure 1 contient toujours des éléments en ordre décroissant en partant du sommet.
- A chaque étape, on dépile les éléments d'une des piles pour les empiler dans l'autre, jusqu'à ce que l'élément traité soit empilable dans la pile de gauche en respectant l'ordre dans chacune des deux piles.
- En fin de tri, tous les éléments se trouvent dans la première pile; on termine donc le tri en vidant cette pile dans le tableau de départ.

1.3 Cahier des charges (première partie)

- Écrire (sur du papier) les spécifications (structures de données, signatures de fonctions, …) d'une librairie permettant la gestion d'une pile statique et dynamique d'entiers.
- Implémenter deux trait `Empiler` dépiler `Dépiler`, permettant d'empiler et dépiler les éléments d'un type.
- Implémenter ces traits pour vos piles statiques et dynamiques.
- Écrire un programme qui utilise la librairie de gestion de pile pour implémenter le tri décrit ci-dessus. Créer une procédure pour le tri.
- Le lancement du programme avec une liste quelconque de nombres entiers sur la ligne de commande doit afficher la liste triée avec un nombre par ligne.
- Vous devez écrire des programmes de test pour valider les fonctions implémentées pour vos piles et pour vos tris. Dans ce but, les macros `assert!()` et `assert_eq!()` pourraient s'avérer très utiles.

Rien d'autre ne devra être affiché

1.4 Exemple

```
> cargo run 18 2 34 21 7 4
2
4
7
18
21
34
```

1.5 Cahier des charges (deuxième partie)

Une fois cette première partie terminée, vous devez rendre votre code complètement générique. Dans une premier temps de rendre vos implémentation de piles génériques (fonctionant pour n'importe quel type de données)

```
struct Pile<T> {
    // Votre implémentation générique
}
```

Puis, votre tri à deux piles devra être également être rendu générique: il faudra prendre une pile d'un type générique et être capable de la trier (attention, il faudra rajouter une contrainte sur les traits implémentés (un *trait bound*) par votre type générique dans ce cas).

1.6 Exemple

```
> cargo run 18.5 2.2 34.1 33.8 7.1 4.89
2.2
4.89
7.1
18.5
33.8
34.1
```

1.7 Règles du jeu

L'exemple décrit ci-dessous est illustré à la figure 1. Soit le tableau à trier suivant :

```
+====+====+====+====+====+
| 17 | 34 | 20 | 40 | 25 |
+====+====+====+====+====+
```

Le tableau est parcouru de gauche à droite.

- 17 est placé dans la pile de gauche.
- 34 est plus grand que 17: 17 est dépilé de la pile de gauche pour être empilé dans la pile de droite; 34 est placé dans la pile de gauche.

- 20 est plus petit que 34 et plus grand que 17: il est empilé dans la pile de gauche.
- 40 est comparé avec 20 et 34: ceux-ci sont dépilés de la pile de gauche pour être empilés dans la pile de droite; 40 est empilé à gauche.
- 25 est plus petit que 34: celui-ci est dépilé de droite pour être empilé à gauche; 25 est empilé à gauche.
- Il n'y a plus d'élément à traiter: la pile de droite est vidée, chacun de ces éléments étant empilé dans la pile de gauche; cette pile contient au final tous les éléments en ordre croissant en partant du sommet.
- Il ne reste plus qu'à vider la première pile dans le tableau.

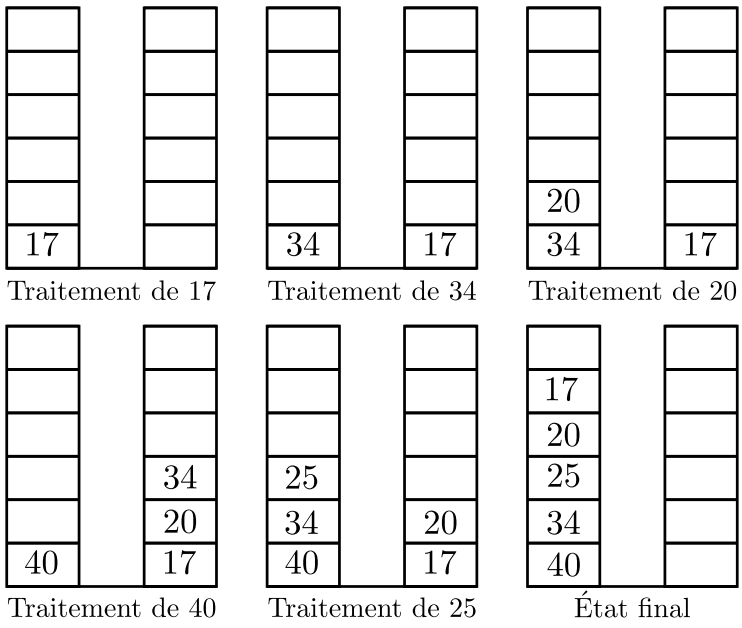

Figure 1: Utilisation des piles pour le tri.

1.8 Indications

1.8.1 Lecture à la ligne de commande

Afin de lire la ligne de commande il faut importer le module `std::env`:

```
let args: Vec<String> = env::args().collect();
```

Le résultat étant stocké dans un vecteur de strings, il faut ensuite transformer chaque élément du vecteur et le convertir en entier (on a déjà fait quelque chose de similaire dans les TPs précédents).

1.8.2 Considérations générales

Avant de commencer à programmer, réfléchissez bien aux méthodes que vous voulez exposer à l'utilisateur pour chacune des parties. Par exemple, quelles sont les méthodes que devront implémenter vos piles statiques et dynamiques? Quelles actions peuvent-elles effectuer toutes les deux?