1. (30 Punkte) Eine DEQUEUE ist eine Datenstruktur, die eine Folge $X_i, X_{i+1}, \ldots, X_j$ von Stücken speichert, mit i, j ganze Zahlen. Folgende Operationen sind darauf definiert:

   - Q.SIZE() gibt die Länge der Folge Q an
   - Q.EMPTY() gibt an, ob die Folge Q leer ist
   - Q.FIRST() gibt das erste Stück in der Folge Q zurück (also X_i)
   - Q.LAST() gibt das letzte Stück in der Folge Q zurück (also X_j)
   - Q.ENDPUSH(X) hängt das Stück x ans Ende der Folge Q an (sozusagen als X_{j+1})
   - Q.FRONTPUSH(X) hängt das Stück x an den Anfang der Folge Q (sozusagen als X_{i-1})
   - Q.ENDPOP() entfernt das letzte Stück von Q und gibt es zurück
   - Q.FRONTPOP() entfernt das erste Stück von Q und gibt es zurück

   Die beiden Pop-Operationen verursachen einen Fehler, wenn Q leer ist. Man kann also eine Dequeue als eine Art doppelendigen Stack betrachten.

   Geben Sie eine Implementierung dieser Datenstruktur an, die auf Arrays basiert, die jede Operation in konstanter amortisierter Laufzeit realisiert, und die nicht übermäß Speicherplatz verschwendet.

   Erklären Sie, warum Sie konstante amortisierte Laufzeit erreichen.

2. (20 Punkte) In der Vorlesung haben wir eine Array-basierte Implementierung des Datentyps addressierbarer Stack behandelt, die für die Push- und Pop-Operation jeweils $O(1)$ **amortisierte** Laufzeit braucht, und mit $\Theta(m)$ Speicher auskommt, wenn m Elemente im Stack gespeichert sind.

   Geben Sie eine Array-basierte Implementierung dieses Datentyps an, die für die Push- und Pop-Operation jeweils $O(1)$ **worst case** Laufzeit braucht, und mit $\Theta(m)$ Speicher auskommt, wenn m Elemente im Stack gespeichert sind.

   Hinweis: Verwenden Sie zwei Felder.

   Sie dürfen annehmen, dass Allokation und Deallokation eines Feldes nur konstant viel Zeit braucht.