Idea

The Queue ADT follows the "First-In, First-Out" Principle, which means the element that was first added is also first removed again. It has similar methods to the Stack, with a different removing-function.

Definition

We define the ADT as the following 5-Tuple:

$$\mathcal{D} = (N, P, Fs, Ts, Ax),$$

where the components are defined as follows:

1. $N :=$ Queue
2. $P := \{\text{Element}\}$
3. $Fs := \{\text{queue, enqueue, dequeue, peek, length}\}$
4. Ts is the set containing the following type specifications:
   (a) queue : Queue
   (b) length : Queue $\rightarrow \mathbb{N}_0$
   (c) enqueue : Queue $\times$ Element $\rightarrow$ Queue
   (d) dequeue : Queue $\rightarrow$ Queue $\cup \{\Omega\}$
   (e) peek : Queue $\rightarrow$ Element $\cup \{\Omega\}$
5. Ax is the set containing the following axioms.
   $\forall Q \in$ Queue : $x, y \in$ Element :
   (a) queue().peek() $= \Omega$
   (b) queue().length() $= 0$
   (c) Q.enqueue(x).length() $= Q$.length() + 1
   (d) Q.length() $> 0 \rightarrow Q$.dequeue().length() $= Q$.length() - 1
   (e) Q.length() $> 0 \rightarrow Q$.enqueue(x).dequeue() $= Q$.dequeue().enqueue(x)
   (f) queue().enqueue(x).dequeue() $=$ queue()
   (g) queue().dequeue() $= \Omega$
   (h) Q.length() $> 0 \rightarrow Q$.enqueue(x).peek() $= Q$.peek()
   (i) Q.length() $= 0 \rightarrow Q$.enqueue(x).peek() $= x$

Theorems

In this section, the definition from above is used to prove useful theorems about the Queue ADT.

Corollary 1. *Q.length() is equals to the amount of function calls of "enqueue" minus the amount of function calls of "dequeue"*

Theorem 1. *Dequeuing an element from a Queue Q removes the element obtained through Q.peek().*

Proof. Let $x = Q.\text{peek}()$ and $x \in \text{Element}$. Note, that this means, that $Q.\text{length}() > 0$, since x would not be an Element otherwise.
We will prove the theorem via induction over n, where $n = Q.\text{length}()$ now:

Base Case: $n = 1$

$$\begin{aligned} Q_1 &= queue().enqueue(x) && \mid \text{Corollary 1} \\ Q_1.peek() &= queue().enqueue(x).peek() \\ Q_1.peek() &= x \end{aligned}$$

Induction Step: $n \to n+1$

$$\begin{aligned} Q_{n+1} &= Q_n.enqueue(y) \\ Q_{n+1} &= queue().enqueue(x_1)....enqueue(x_n).enqueue(y) \\ &\mid \text{Corollary 1} \\ Q_{n+1}.peek() &= queue().enqueue(x_1)....enqueue(x_n).enqueue(y).peek() \\ Q_{n+1}.peek() &= queue().enqueue(x_1)....enqueue(x_n).peek() \\ &\mid \text{Axiom (h)} \\ Q_{n+1}.peek() &= queue().enqueue(x_1).peek() \\ &\mid \text{Axiom (h) applied n-1 times} \\ Q_{n+1}.peek() &= x_1 \\ &\mid \text{Axiom (i)} \\ Q_{n+1}.peek() &= queue().enqueue(x_1).peek() \\ Q_{n+1}.peek() &= Q_1.peek() \end{aligned}$$

□

Theorem 2. *Queues follow the "First-In, First-Out"-Principle. That means, that the 1st,2nd,...,nth element to enqueued is also going to be the 1st,2nd,...,nth element to be dequeued.*

Proof. Let Q_n be a Queue, such that $Q.\text{length}() = n$. It follows then from Corollary 1, that

$$Q_n = queue().enqueue(x_1)....enqueue(x_n)$$

holds. Further, we know from Axiom (e), that the following holds for all $n > 0$:

$$\begin{aligned} Q_n.dequeue() &= queue().enqueue(x_1).enqueue(x_2)....enqueue(x_n).dequeue() \\ &= queue().enqueue(x_1).enqueue(x_2)....dequeue().enqueue(x_n) \\ &= queue().enqueue(x_1).enqueue(x_2).dequeue()....enqueue(x_n) \\ &= queue().enqueue(x_1).dequeue().enqueue(x_2)....enqueue(x_n) \end{aligned}$$

From Axiom (f), that the above is equivalent with the following:

$$Q_n.dequeue() = queue().enqueue(x_2)....enqueue(x_n)$$

It can easily be proved (per induction for example), that by repeating the same steps n times this holds for all elements aside from x_1 as well, yet that proof is left out here for brevity. $\square$

Implementation

TBD