DynamicArray:

cap: Integer
nrElem: Integer
elems: TElem[]

- init(da, cp)
  - **description:** creates a new, empty DynamicArray with initial capacity *cp* (constructor)
  - **pre:** $cp \in N^*$
  - **post:** $da \in \mathcal{DA}$, $da.cap = cp$, $da.nrElem = 0$
  - **throws:** an exception if *cp* is zero or negative
- destroy(da)
  - **description:** destroys a DynamicArray (destructor)
  - **pre:** $da \in \mathcal{DA}$
  - **post:** *da* was destroyed (the memory occupied by the dynamic array was freed)
- size(da)
  - **description:** returns the size (number of elements) of the DynamicArray
  - **pre:** $da \in \mathcal{DA}$
  - **post:** size $\leftarrow$ the size of *da* (the number of elements)
- getElement(da, i)
  - **description:** returns the element from a position from the DynamicArray
  - **pre:** $da \in \mathcal{DA}$, $1 \leq i \leq da.nrElem$
  - **post:** getElement $\leftarrow$ e, $e \in TElem$, $e = da.e_i$ (the element from position *i*)
  - **throws:** an exception if *i* is not a valid position

- setElement(da, i, e)
  - **description:** changes the element from a position to another value
  - **pre:** $da \in \mathcal{DA},\ 1 \leq i \leq da.nrElem,\ e \in TElem$
  - **post:** $da' \in \mathcal{DA}, da'.e_i = e$ (the $i^{th}$ element from $da'$ becomes $e$), setElement $\leftarrow e_{old}$, $e_{old} \in TElem$, $e_{old} \leftarrow da.e_i$ (returns the old value from position $i$)
  - **throws:** an exception if $i$ is not a valid position

- addToEnd(da, e)
  - **description:** adds an element to the end of a DynamicArray. If the array is full, its capacity will be increased
  - **pre:** $da \in \mathcal{DA},\ e \in TElem$
  - **post:** $da' \in \mathcal{DA},\ da'.nrElem = da.nrElem + 1; da'.e_{da'.nrElem} = e\ (da.cap = da.nrElem \Rightarrow da'.cap \leftarrow da.cap * 2)$

- addToPosition(da, i, e)
  - **description:** adds an element to a given position in the DynamicArray. If the array is full, its capacity will be increased
  - **pre:** $da \in \mathcal{DA},\ 1 \leq i \leq da.nrElem + 1,\ e \in TElem$
  - **post:** $da' \in \mathcal{DA},\ da'.nrElem = da.nrElem + 1,\ da'.e_j = da.e_{j-1} \forall j = da'.nrElem, da'.nrElem - 1, ..., i + 1, da'.e_i = e\ , da'.e_j = da.e_j\ \forall j = i - 1, ..., 1\ (da.cap = da.nrElem \Rightarrow da'.cap \leftarrow da.cap * 2)$
  - **throws:** an exception if $i$ is not a valid position (da.nrElem+1 is a valid position when adding a new element)

- deleteFromPosition(da, i)
  - **description:** deletes an element from a given position from the DynamicArray. Returns the deleted element
  - **pre:** $da \in \mathcal{DA},\ 1 \leq i \leq da.nrElem$
  - **post:** deleteFromPosition $\leftarrow$ e, $e \in TElem,\ e = da.e_i,\ da' \in \mathcal{DA},\ da'.nrElem = da.nrElem - 1,\ da'.e_j = da.e_{j+1} \forall i \leq j \leq da'.nrElem, da'.e_j = da.e_j\ \forall 1 \leq j < i$
  - **throws:** an exception if $i$ is not a valid position

- iterator(da, it)
  - **description:** returns an iterator for the DynamicArray
  - **pre:** $da \in \mathcal{DA}$
  - **post:** $it \in \mathcal{I}$, $it$ is an iterator over $da$, the current element from $it$ refers to the first element from $da$, or, if $da$ is empty, $it$ is invalid
- Other possible operations:
  - Delete all elements from the Dynamic Array (make it empty)
  - Verify if the Dynamic Array is empty
  - Delete an element (given as element, not as position)
  - Check if an element appears in the Dynamic Array
  - Remove the element from the end of the Dynamic Array
  - etc.
- Usually, we can discuss the complexity of an operation for an ADT only after we have chosen the representation. Since the ADT Dynamic Array can be represented in a single way, we can discuss the complexity of its operations:
  - size - $\Theta(1)$
  - getElement - $\Theta(1)$
  - setElement - $\Theta(1)$
  - iterator - $\Theta(1)$
  - addToPosition - $O(n)$
  - deleteFromEnd - $\Theta(1)$
  - deleteFromPosition - $O(n)$
  - addToEnd - $\Theta(1)$ *amortized*