DLLA

- Obviously, we can define a doubly linked list as well without pointers, using arrays.
- For the DLLA we will see another way of representing a linked list on arrays:
  - The main idea is the same, we will use array indexes as links between elements
  - We are using the same information, but we are going to structure it differently
  - However, we can make it look more similar to linked lists with dynamic allocation

DLLA - Node

- Linked Lists with dynamic allocation are made of nodes. We can define a structure to represent a node, even if we are working with arrays.
- A node (for a doubly linked list) contains the information and links towards the previous and the next nodes:

DLLANode:
  info: TElem
  next: Integer
  prev: Integer

DLLA

- Having defined the *DLLANode* structure, we only need one array, which will contain *DLLANodes*.
- Since it is a doubly linked list, we keep both the head and the tail of the list.

DLLA:
nodes: DLLANode[]
cap: Integer
head: Integer
tail: Integer
firstEmpty: Integer
size: Integer *//it is not mandatory, but useful*

DLLA - Allocate and free

- To make the representation and implementation even more similar to a dynamically allocated linked list, we can define the *allocate* and *free* functions as well.

```
function allocate(dlla) is:
//pre: dlla is a DLLA
//post: a new element will be allocated and its position returned
    newElem ← dlla.firstEmpty
    if newElem ≠ -1 then
        dlla.firstEmpty ← dlla.nodes[dlla.firstEmpty].next
        if dlla.firstEmpty ≠ -1 then
            dlla.nodes[dlla.firstEmpty].prev ← -1
        end-if
        dlla.nodes[newElem].next ← -1
        dlla.nodes[newElem].prev ← -1
    end-if
    allocate ← newElem
end-function
```

DLLA - Allocate and free

subalgorithm free (dlla, poz) **is:**
//pre: dlla is a DLLA, poz is an integer number
//post: the position poz was freed
    dlla.nodes[poz].next $\leftarrow$ dlla.firstEmpty
    dlla.nodes[poz].prev $\leftarrow$ -1
    if dlla.firstEmpty $\neq$ -1 **then**
        dlla.nodes[dlla.firstEmpty].prev $\leftarrow$ poz
    end-if
    dlla.firstEmpty $\leftarrow$ poz
end-subalgorithm

DLLA - InsertPosition

subalgorithm insertPosition(dlla, elem, poz) **is:**
//pre: dlla is a DLLA, elem is a TElem, poz is an integer number
//post: the element elem is inserted in dlla at position poz
    if poz $<$ 1 **OR** poz $>$ dlla.size + 1 **execute**
        @throw exception
    end-if
    newElem $\leftarrow$ alocate(dlla)
    if newElem = -1 **then**
        @resize
        newElem $\leftarrow$ alocate(dlla)
    end-if
    dlla.nodes[newElem].info $\leftarrow$ elem
    if poz = 1 **then**
        if dlla.head = -1 **then**
            dlla.head $\leftarrow$ newElem
            dlla.tail $\leftarrow$ newElem
        else
//continued on the next slide...

DLLA - InsertPosition

```
        dlla.nodes[newElem].next ← dlla.head
        dlla.nodes[dlla.head].prev ← newElem
        dlla.head ← newElem
    end-if
else
    nodC ← dlla.head
    pozC ← 1
    while nodC ≠ -1 and pozC < poz - 1 execute
        nodC ← dlla.nodes[nodC].next
        pozC ← pozC + 1
    end-while
    if nodC ≠ -1 then //it should never be -1, the position is correct
        nodNext ← dlla.nodes[nodC].next
        dlla.nodes[newElem].next ← nodNext
        dlla.nodes[newElem].prev ← nodC
        dlla.nodes[nodC].next ← newElem
//continued on the next slide...
```

DLLA - InsertPosition

```
        if nodNext = -1 then
            dlla.tail ← newElem
        else
            dlla.nodes[nodNext].prev ← newElem
        end-if
    end-if
end-if
end-subalgorithm
```

- Complexity: $O(n)$

DLLA - Iterator

- The iterator for a DLLA contains as *current element* the index of the current node from the array.

DLLAIterator:
list: DLLA
currentElement: Integer

DLLAIterator - init

subalgorithm init(it, dlla) **is:**
//pre: dlla is a DLLA
//post: it is a DLLAIterator for dlla
  it.list ← dlla
  it.currentElement ← dlla.head
end-subalgorithm

- For a (dynamic) array, currentElement is set to 1 when an iterator is created. For a DLLA we need to set it to the head of the list (which might be position 1, but it might be a different position as well).
- Complexity:

DLLAIterator - init

subalgorithm init(it, dlla) **is:**
//pre: dlla is a DLLA
//post: it is a DLLAIterator for dlla
  it.list $\leftarrow$ dlla
  it.currentElement $\leftarrow$ dlla.head
end-subalgorithm

- For a (dynamic) array, currentElement is set to 1 when an iterator is created. For a DLLA we need to set it to the head of the list (which might be position 1, but it might be a different position as well).
- Complexity: $\Theta(1)$

DLLAIterator - getCurrent

```
subalgorithm getCurrent(it) is:
//pre: it is a DLLAIterator, it is valid
//post: e is a TElem, e is the current element from it
//throws exception if the iterator is not valid
    if it.currentElement = -1 then
        @throw exception
    end-if
    getCurrent ← it.list.nodes[it.currentElement].info
end-subalgorithm
```

- Complexity:

DLLAIterator - getCurrent

```
subalgorithm getCurrent(it) is:
//pre: it is a DLLAIterator, it is valid
//post: e is a TElem, e is the current element from it
//throws exception if the iterator is not valid
    if it.currentElement = -1 then
        @throw exception
    end-if
    getCurrent ← it.list.nodes[it.currentElement].info
end-subalgorithm
```

- Complexity: $\Theta(1)$

DLLAIterator - next

```
subalgoritm next (it) is:
//pre: it is a DLLAIterator, it is valid
//post: the current elements from it is moved to the next element
//throws exception if the iterator is not valid
    if it.currentElement = -1 then
        @throw exception
    end-if
    it.currentElement ← it.list.nodes[it.currentElement].next
end-subalgorithm
```

- In case of a (dynamic) array, going to the next element means incrementing the *currentElement* by one. For a DLLA we need to follow the links.
- Complexity:

DLLAIterator - next

```
subalgoritm next (it) is:
//pre: it is a DLLAIterator, it is valid
//post: the current elements from it is moved to the next element
//throws exception if the iterator is not valid
    if it.currentElement = -1 then
        @throw exception
    end-if
    it.currentElement ← it.list.nodes[it.currentElement].next
end-subalgorithm
```

- In case of a (dynamic) array, going to the next element means incrementing the *currentElement* by one. For a DLLA we need to follow the links.
- Complexity: $\Theta(1)$

DLLAIterator - valid

```
function valid (it) is:
//pre: it is a DLLAIterator
//post: valid return true is the current element is valid, false otherwise
    if it.currentElement = -1 then
        valid ← False
    else
        valid ← True
    end-if
end-function
```

- Complexity:

DLLAIterator - valid

```
function valid (it) is:
//pre: it is a DLLAIterator
//post: valid return true is the current element is valid, false otherwise
    if it.currentElement = -1 then
        valid ← False
    else
        valid ← True
    end-if
end-function
```

- Complexity: $\Theta(1)$