# Lecture 3

- Iterators & Containers

Lect. PhD. Lupsa Dana
Babes - Bolyai University
Computer Science and Mathematics Faculty
2021 - 2022

Some slides borrowed from: Lect. PhD. Onet-Marian Zsuzsanna

## Previously, in Lecture 2

- Array
- Dynamic array
  - ...
    - Amortized analysis
      - Expansion and contraction
- Matrix

# Previously, in Lecture 2

## DynamicArray: expansion and contraction

Consider DELETE operation:
it is enough to remove the specified item
- It is often desirable to contract the table, to reduce the wasted space
- Table contraction is analogous to table expansion

  allocate a new, smaller table and then copy the items

A natural strategy (?!)

- Expansion: double the table capacity when an item is inserted into a full table
- Contraction: halve the capacity when a deletion would cause the table to become less than half full.

How empty should the array become before resize? Which of the following two strategies do you think is better? Why?

- Wait until the table is only half full (da.nrElem ≈ da.cap/2) and resize it to the half of its capacity
- Wait until the table is only a quarter full (da.nrElem ≈ da.cap/4) and resize it to the half of its capacity

# Containers and iterators

Lecture 1

Container

- is a group of data (collection of elements). There we can add new elements and/or we can remove elements.

Iterator:

abstraction

- is defined over a container
- a mechanism for accessing the elements of the container
  - ➢ is an interface between containers and algorithms

  Iterators are used to offer a common and generic way of moving through all the elements of a container, independently of the representation of the container.

  - Iterators usually benefits from the property that in an object-oriented language, it is possible to have many different implementations for **one interface** (the same).

Remark:

- not all containers have iterators (**e.g.?)**

# Containers and iterators

**See: Lecture03 – ADT Iterator.pdf**

**ADT Iterator**

- interface designed specifically to be used in a loop
  (access to all elements in a container in order to process them)

See the next subalg.:

```
subalgorithm printContainer(c) is:
   //pre: c is a container
   //post: the elements of c were printed
   //we create an iterator using the iterator method of the container
   iterator(c, it)
   while valid(it) execute
      //get the current element from the iterator
      elem ← getCurrent(it)
      print elem
      //go to the next element
      next(it)
   end-while
end-subalgorithm
```

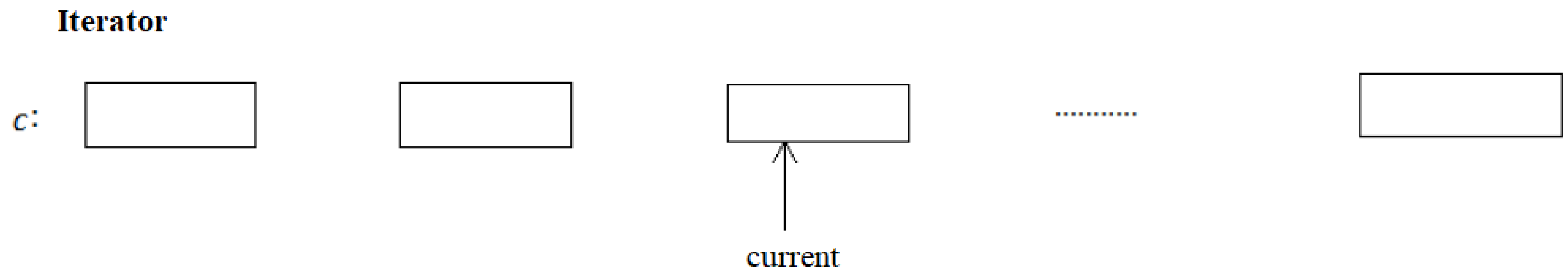
Do we need the init operation? (see ADT Iterator)

It the interface of an iterator is the same, independently of the exact container or its representation, this subalgorithm can be used to print the elements of any container.

# Iterators. Implementation aspects.

An iterator usually contains:

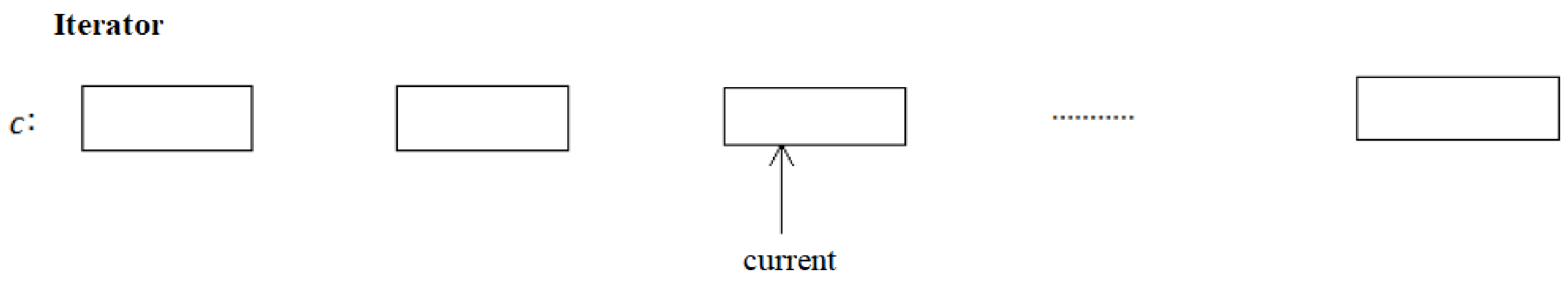
- a reference to the container it iterates over
- a reference to a current element from the container



- How can we represent an iterator for a DymanicArray

    How can we represent that current element from the iterator?

# Iterators. Implementation aspects.



- How can we define an iterator for a Dynamic Array?

  Let us see a short, simplified example

- **C++ example : files in DynArrayExample folder**

Think about:

- What are the time-complexities for iterator operations ?
- How can we print the content of a Dynamic Array?
- What is the complexity of print algorithms?
- What happens when we iterate a container, but the container is modified (add/remove) before the iteration is done?

## Type of iterators

- The iterator presented above describes the simplest iterator:
  unidirectional and read-only

- A unidirectional iterator can be used to iterate through a container in one direction only (usually forward, but we can define a reverse iterator as well).
- A bidirectional iterator can be used to iterate in both directions. Besides the next operation it has an operation called previous and it could also have a last operation (the pair of first).
- A random access iterator can be used to move multiple steps (not just one step forward or one step backward).

- A read-only iterator can be used to iterate through the container, but cannot be used to change it.
- A read-write iterator can be used to add/delete elements to/from the container.

# Type of iterators. Example

***Example : Java util***

| | |
|---|---|
| **Interface Iterator<E>** | boolean hasNext() |
| | E next() |
| | void remove() |

```
// assume a list of Strings in Java:
// ArrayList<String> list

Iterator<String> it = list.iterator();
while(it.hasNext()) {
    System.out.println( it.next() );
}
```

# Iterators. Read-write operations. Example

```
// assume a list of Strings in Java:
// ArrayList<String> list

System.out.println(list);

//Get iterator
Iterator<String> iterator = list.iterator();

//Iterate over all elements
while(iterator.hasNext())
{
    //Get current element
    String value = iterator.next();
    System.out.println( value );

    //Remove element
    if(value.equals("B")) {
        iterator.remove();
    }
}

System.out.println(list);
```

```
[A, B, C, D]
A
B
C
D
[A, C, D]
```

***Example: remove in Java util***

**Interface Iterator<E>**

boolean hasNext()
E next()
void remove()

**Iterator remove()**

- It removes from the underlying collection the last element returned by the iterator.
- This method can be called only once per call to next().
- If the remove() method is not preceded by the next() method, then the exception IllegalStateException is thrown.
- If the underlying collection is modified while the iteration is in progress in any way other than by calling remove() method, iterator will throw an **ConcurrentModificationException**.

# Iterators. Read-write operations.

Problem:
Remove all of the even integers from a list.

Is the next subalgorithm correct?

(Use indexed acces  to elements of a dynamic array.)

```
Subalg. removeEvens(da) {                              //assumes 1-based indexing
    for  idx <- 1, size(list) do
        el <-  getElement(da, idx )
         if (el mod 2 = 0) then
                deleteFromPosition (da, idx);
        endif
    endfor
endSubalg
```

Think about an implementation of remove in DynamicArray example
and about an implementation of removeEvens by using it.