•

Access elements in a container

- **(export) Position**
- **Index**
- **Current position Hidden in the container (cursor)**
- **iterator**

foreach - a *kind of implicit* iteration

- In many modern languages

Python

Java:

```
List<String> someList = new ArrayList<String>()
// add "monkey", "donkey", … to someList
for(String item : someList ){
   System.out.println(item);
}
```

can be used for any class that implements the Iterable interface

UNIX shell: **for *filename* in *.txt; do *cat $filename* ; done**

Iterator

- is defined over a container
- *walks through* the elements of the container

Remarks:

- not all containers have iterators
- is an interface between containers and algorithms

  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)
- **! cursor**

Iterators

- interface designed specifically to be used in a loop (access to all elements in a container in order to process them)
- Subalg. processElem(***c***)

  @initialization *(associate **it** with **c**)*

  while @ there are elements in ***c*** unprocessed

  @ get another element *e*

  @ process *e*

  endwhile

  endprocessElem.

Forward iterator

? ADT

- $\mathcal{D}_{\text{Iterator}}$ = {it| it – iterator over a container }
- Operations

init (ContainerSeq c)	create/destroy (current = first elem.) first , begin	
getCurrent	current	*
moveNext	next	++
isValid	(*not*) isDone, end	

? **Usage** (in a loop)

Forward iterator

Java.util style

next, hasNext

- at the begining the current object iterator is "before" the first element
- hasNext – verify if *current* object has a *next* object
- next **-** get next object and go over the next **!!**

? **Usage** (in a loop)

Iterators: classification

Many types;
not every type of container supports every type of iterator

➔ direction of traversal

- **Forward** — can be *incremented*
- **Backward (/reverse)** — can be de*cremented*
- **Bidirectional** — can be *incremented* and *decremented*
- **Random access**

  can go both forward and backward with a number of positions

  -bi-directional ``long jumps”

More classification:
C++ STL

const : don't allow you to change the values that they point to

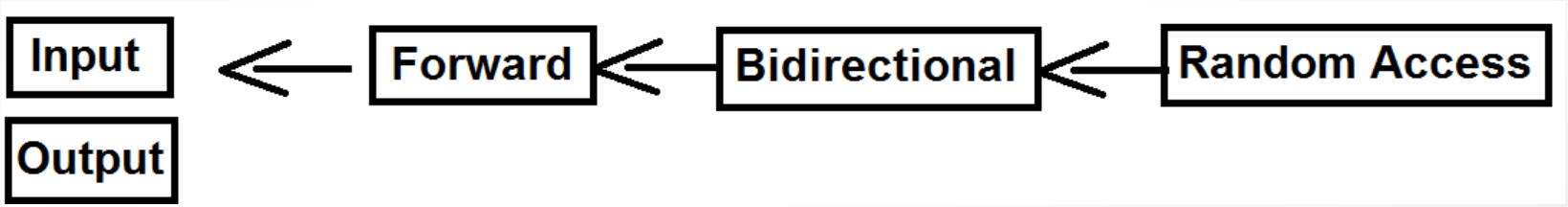
Dereferencing them yields a reference to a constant element (such as const T&)

Constant iterators are iterators that do not fulfill the requirements of an output iterator;

mutable (regular iterators)

More classification: C++ STL

- Input: each value pointed by the iterator is read once then the iterator is incremented.
- Output: each element pointed by the iterator is written a value once then the iterator is incremented



Iterators

? What if a container is modified while iterating through its elements

- Iterator over linked list
- Iterator over vector