•

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

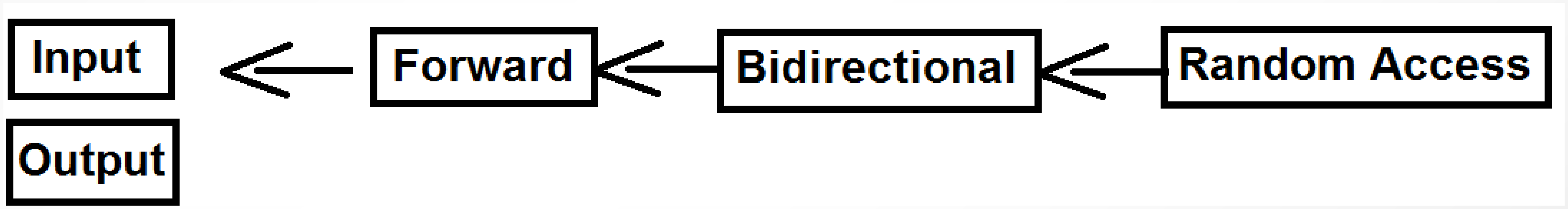
Dereferencing them yields a reference to a constant element (such as const T&)

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

```
//...
typedef int TElement;
class Vector
{
   int cap;                    //          int _capacity;
   int n;                      //          int _size;
   TElement* els;        //          TElement* _elements

public:
      //...
friend class IteratorIdx1;
friend class IteratorIdx2;
friend class IteratorAddr;

private:
      // ...
};
// FORWARD Iterator
class Iterator{
public:
   virtual bool isValid()=0;
   virtual void moveNext()=0;
   virtual TElement getCurrent()=0;
};

class IteratorIdx1:public Iterator
{
   int idx;
   Vector* v;

public:
   explicit IteratorIdx1(Vector& x)
    {
      idx=0;
      v=&x;
      }
   bool isValid()
   {
      return (idx<(*v).n);

   void moveNext()
   {
      idx++;
   }

   TElement getCurrent()
   {
      return (*v).els[idx];
   }
};
```

```
class IteratorIdx2:public Iterator
{
   int idx;
   Vector& v;
public:
   explicit IteratorIdx2(Vector& x):idx(0),v(x)
   {

   }

   bool isValid()
   {
      return (idx<v.n);
   }

   void moveNext()
   {
      idx++;
   }

   TElement getCurrent()
   {
      return v.els[idx];
   }
};

class IteratorAddr:public Iterator
{
   TElement* current;
   TElement* afterLast;
public:
   explicit IteratorAddr(Vector& x)
   {
     current=x.els;
     afterLast=x.els+x.n;
   }
   bool isValid()
   {
      return (current<afterLast);
   }

   void moveNext()
   {
      current++;
   }
```

```
  TElement getCurrent()
   {
      return *current;
   }
};

//...

void testIterat()
{
  Vector v;
  TElement el;
  for(int i=0;i<5;i++)
  {
     el=i+10 ;
     v.addLast(el);
  };

  Iterator* it;
  it = new IteratorIdx1(v);  //Idx1, Idx2, Addr
  while ((*it).isValid())
  {
            el = (*it).getCurrent();
            cout << el << " ";
            (*it).moveNext();
  };
}
//...
```
===

Check this out:
http://docs.oracle.com/javase/7/docs/api/
http://www.cplusplus.com/reference/stl/