② better than ①, implemented in stl::vector w/ "reserve" functionality
(caveats ?)

* functionality abstraction

doubly & singly linked list & array

list : iterative (positional) access

insert(p, e) insert at position
elem(p) element at position
begin() starting position

p != end() ⇐ isEnd(p) is p the end?
nextOf(p) go to next element of p
erase(p)

iterator (p): abstraction for { index in sparse array; index / pointer in dense array; pointer in linked list }

	sparse array	dense array	list
begin	0	&arr[0]	head
end	n+1	&arr[n+1]	NULL
nextOf(p)	p+1	p++	p→next
elem(p)	at(p)	(*p)	p→value

- iterator<container_type>, "safe" pointer in some sense
- overload "++" to do nextOf, override "*" to do { --? +5? }
- int sum = 0;
  for (iterator<list<int>> p = c.begin(); p != c.end(); p++) {
  can now "freely" change this one
  sum += (*p);
  }

STL list : doubly linked list

* sequence : vector + list + i ⇔ p