abstraction "essense"

1. save implementation efforts (template: type abstraction)
2. "easy" change of data structure

$\boxed{\text{extendable}}$ array ← C++ vector

* functionality abstraction

fast / storage-saving

(dense array & sparse array

indexed (random) access : $\boxed{\text{vector}}$

at(i)	.	getatIndex(i)
set(i)	.	put to Index(i)
insert	.	
erase	.	

* exendable array

if array A "overflow"

grow the array

1. allocate new array B O(1)
2. copy contents from A to B O(N)
3. remove A and assign B to A O(1)

consider M "pushes" to array

a. size(B) = size(A) + 1

N		size(A)
1		1
	!, allocate, copy(1)	
2		2
	!, allocate, copy(2)	
3		3
	!, allocate, copy(3)	
4		4
⋮		⋮
M	!, allocate, copy(M-1)	

$$\underbrace{\begin{matrix} \Downarrow & \Downarrow \\ M-1 & 1+2+3+\cdots+(M-1) \\ \text{allocate} & \text{copy} \end{matrix}}\Big) \quad \Theta(M^2)$$

on average $\frac{O(M^2)}{M} = O(M)$

b. size(B) = size(A) * 2 C++ vector

N		size(A)
1		1
	!, allocate, copy(1)	
2		2
	!, allocate, copy(2),	4
3		
4		4
5	!, allocate, copy(4),	8
6		8
7		8
8		8
9	!, allocate, copy(8),	16

$\sim 2^K$

2^K M   K allocate $O(\log M)$ $1+2+4+8+\cdots+2^{K-1}$ copy $O(M)$

on average $\frac{O(M)}{M} = O(1)$ ☆

indexed
random

vector

dense array : fast

sparse array : space-saving

extendable array : dynamic

list

iterative

sequential

postional

doubly linked list

singly linked list

dense array
(consecutive)

"address"

remove (p)
insert (p, e) slow

get (p)

begin() : head arr

isEnd () : == NULL "arr+len"

next () : ++

	dense array	linked list
begin	&arr[0]	head
end	&arr[N]	NULL
next	p++	p->next

```
p.next() ===> p++

for(iterator<vector<int> > p = c.begin(); p != c.end(); p = p.next()){
   sum += elem(p);
}

elem(p) ===> (*p)
```