Container

A Container

is an object that stores a finite number of other objects

- its *elements*
- *owns them*

methods for accessing its elements

- usually use ***iterator***

  0, 1 or more than one iterator may be active at any one time

In general:

no guarantee that the elements are stored in any definite order

the order might be different upon each iteration

- Empty container : $\varnothing_{\text{Container}}$

ADT …

Container

size the number of elements it contains ADT

area the total number of bytes that it occupies DS

(memory used)

the sum of the elements' areas plus *whatever* overhead

A variable sized container

insert and/or remove elements

A fixed size container constant size throughout the container's lifetime

(*In some fixed-size container types, the size is determined at compile time.*)

- static
- *semi-static*
- dynamic

Other classification

- homogeneous container *elements of same type (vector)*
- heterogeneous container *elements with different types (tuple)*

Operations

Think about properties of data type you model !

For containers, we have to consider:

- create / initialize, destroy *(create empty)*

  (*copy*, *assignment*)
- get

  set: controlled access to all relevant components
- test some properties
- conversion to/from other types

… (other) specific operations

NO: read/write (Why? Argue!)

Set

- no duplicate elements
- no order is guaranteed

In Mathematics: are unchanging

In computer science:

can change over time (grow, shrink)

Small examples:

- {a, b} = {b, a}
- {a} **no**: {a, a}

Bag

- allow duplicate elements
- no order is guaranteed

Small examples:

- {a, b} = {b, a}
- {a}   <> {a, a}

Terminology:		
	bag	(*Smalltalk*)
	multiset	*(C++ STL)*
	collection	*(Java.util)*

(Linear) List

- elements are arranged in a strict linear order

Terminology:	**Sequence containers**	(C++ STL)
	List	(Java.util)

List as ADT: Uniform formal approach

Position – give the position of elements in list

- open to many possible instantiations of Position

ADT List: two parameters

- (i) TE - the type of the constitutive elements
- (ii) Position - the type of elements' positions

specialized ADTs List are obtained by instantiation of the type Position
IndexedList, *SinglyLinkedList*, *DoublyLinkedList.*

What is a Vector?

Stack

container

insertions/extractions are made following a fixed (predefined) strategy:

LIFO: Last In First Out

Remarks:

- **iteration** is ***not specific*** to stack
  stack with iteration → **extension**
  by default: work with stack that only have specific operation

---

True about stack (formal axioms)

- newly created stack is empty;
- after pushing an item to a newly created stack, it becomes nonempty;
- peek returns the most recently pushed item;
- stack remains untouched, after a pair of push and pop commands, executed one after another and with the same element.

Stack

Operations

- init // create , createEmpty, initEmpty
- destroy
- push
- pop
- isEmpty //empty
- peek
- isFull

Queue

- a container
- in which insertions/extractions are made following a fixed (predefined) strategy

  FIFO: First In First Out

Remarks:

- **iteration** is ***not specific*** to stack

  queue with iteration → **extension**

  by default: work with queue that only have specific operation

Queue

Operations

- init // create , createEmpty, initEmpty
- destroy
- enqueue // push, push_back , add, insert
- dequeue // pop, pop_front, extract,remove

- isEmpty //empty

- isFull
- peek

Deque

double ended queue

- insertions/extractions can be made ***from both ends*** **(LIFO + FIFO)**

… operations

Deque – name for operations

Operation	**Ada**	**C++ STL**	**Java.util**
insert at back	append	***push_back***	offerLast
insert at front	appendleft	***push_front***	offerFirst
remove last	pop	***pop_back***	pollLast
remove first	popleft	***pop_front***	pollFirst
examine last		back	***peekLast***
examine first		front	***peekFirst***

Map & Multimap

Elements:

key

value (mapped value)

Other terms:

- associative array, dictionary

Unique associative container

- keys are not unique
- there is no limit on the number of elements with the same key

Multiple associative container

no order is guaranteed

Map

- **operations**
  specific: based on keys

add …, k; v // … *put, reassign*
remove …; k
// → value

search…, k // *get, searchByKey*
// → value
containsKey
// → boolean

----

containsValue
not usual : searchByValue(m,v) // → key;

Map

- **other operations**

create
destroy

isEmpty ,   size

keys   // *key*Set
values   // *valueMultiset*
pairs

getIterator   // iterator