Module04-02
C++ 标准库：标准容器

- 数据结构简介
- → 标准容器
- 常用算法简介
- 标准算法与函数对象
- 迭代器
- 字符串
- I/O 流
- 数值

标准库 - 标准容器

- 标准容器 (STL Containers)
  - 标准容器综述
  - 序列容器 (Sequence Containers)
    - vector、 deque、 list
  - 容器适配器 (Container Adapters)
    - stack、 queue、 priority_queue
  - 关联容器 (Assosiative Containers)
    - map、 multimap、 set、 multiset
  - 特殊容器
    - string(安排在字符串课程中)、 array、 valarray （安排在数值课程中)、 bitset

- 操作综述
  - 容器内部成员类型

value_type	元素的类型
allocator_type	分配器的类型
size_type	下标、元素计数等的类型
difference_type	迭代器之差的类型
iterator	迭代器
const_iterator	const 迭代器
reverse_iterator	反向迭代器
const_reverse_iterator	const 反向迭代器
reference	元素类型的引用
const_reference	元素类型的 const 引用
key_type	关联容器的 Key 的类型
mapped_type	关联容器的元素值的类型
key_compare	关联容器的 Key 比较准则的类型

- 操作综述（续 1 ）
  - 预定义的迭代器

begin()	指向第一个元素
end()	指向最后一个元素之后的位置
rbegin()	指向反向顺序的第一个元素
rend()	指向反向顺序最后一个元素之后的位置

  - 直接访问元素

front()	访问第一个元素
back()	访问最后一个元素
[]	下标操作，不检查数组越界
at()	下标操作，检查数组越界（ deque 和 vector ）

- 操作综述（续 2 ）
  - 栈和队列操作

push_back()	在末尾追加
pop_back()	删除最后一个元素
push_front()	在最前端加入 (deque 和 list)
pop_front()	删除最前端一个元素 (deque 和 list)

  - 插入和删除 (所有容器， p 为迭代器，指定位置)

insert(p,v)	在 p 前插入 v
insert(p,n,v)	在 p 前插入 n 个 v
insert(p,first,last)	在 p 前插入 first(含) 到 last(不含) 之间的元素
erase(p)	删除 p 位置的元素
erase(first,last)	删除 first(含) 到 last(不含) 之间的元素
clear()	删除所有元素

- 操作综述（续 3 ）
  - 容器 size 和其它

size()	元素的个数
empty()	查看容器是否为空
max_size()	容器最多能容纳元素的个数
capacity()	容器当前容量（仅适用于 vector）
reserve()	为扩充而预留空间（仅适用于 vector）
resize()	改变容器的大小（vector、 list、 deque）
swap()	交换 2 个容器的元素
get_allocator()	取得容器分配器的副本
==	判断 2 个容器的元素是否完全相等
!=	
<	

- 操作综述（续 4 ）
  - 容器的构造和复制

container()	构造空容器
container(n)	n 个默认值的元素的容器（关联容器无）
container(n,v)	n 个 v 的拷贝（关联容器无）
container(first,last)	用 first（含）、last（不含）的元素创建一个新容器
container(other)	复制构造
~container()	
operator=(other)	赋值操作符
assign(n,v)	赋值 n 个 v 的拷贝
assign(first,last)	使用 first（含）、last（不含）的元素赋值

标准容器 - 标准容器综述

- 操作综述（续 5 ）
  - 针对关联容器的 Key 的操作

operator[](k)	访问 key 为 k 的元素（仅对 map 有效）
find(k)	查找 key 为 k 的元素（一个或多个）
lower_bound(k)	查找 key 为 k 的元素中的第一个
upper_bound(k)	查找 key 为 k 的元素中的最后一个
equal_range(k)	查找 key 为 k 的元素的 lower_bound 和 upper_bound
key_cmp()	key 比较器的副本
value_cmp()	值比较器的副本

- 容器操作的性能评估

容器	下标操作	删除、插入	前端操作	后端操作	迭代器
vector	O(1)	O(n)+		O(1)+	Ran
list		O(1)		O(1)	Bi
deque	O(1)	O(n)	O(1)	O(1)+	Ran
stack				O(1)+	
queue			O(1)	O(1)+	
priority_queue			O(log(n))	O(log(n))	
map	O(log(n))	O(log(n))+			Bi
multimap		O(log(n))+			Bi
set		O(log(n))+			Bi
multiset		O(log(n))+			Bi
string	O(1)	O(n)+	O(n)+	O(1)+	Ran
array	O(1)				Ran
valarray	O(1)				Ran
bitset	O(1)				

- 容器操作的性能评估（续）
  - 关于记法
    - O(1)+：对于 vector 和 string 等基于 array（数组）的序列容器，任何添加元素的动作均有可能出现重新分配空间、拷贝原数组对象到新空间的情况，所以会有额外的开销
    - O(log(n))+：对于基于 RB-Tree 的关联容器，添加、删除元素后为维护 RB-Tree 的性质，需做 2 ~ 3 次旋转，总体的时间复杂度接近 O(log(n))

标准容器 - 标准容器综述

- 对放入容器的元素的要求
  - 对象的构造和拷贝
    - 容器的 insert 、 assign 等操作实际上是对一个对象的副本进行操作
    - 一个类型的对象能被放入容器、且支持容器的各项操作，需符合下列条件：
      - (最好) 有默认构造函数
      - 必须有复制构造函数
      - 必须有 T& operator=(const T& other) 形式的赋值操作
  - 比较操作
    - 基于 RB-Tree 的关联容器是有序的，为保证对容器中的元素或 key 进行排序，容器中的对象或 key 必须保证 < 可用，或提供合适的 cmp 比较器
    - 对于部分操作如 find ，需对比较容器中的元素是否相等，容器中的对象必须保证 == 可用，或结合 !cmp(x,y) && !cmp(y,x)

标准容器 - vector

- 关于 vector
  - vector 的迭代器为随机迭代器
  - 添加元素的操作：
    - 从尾部追加
    - 在任意位置插入，造成插入位置以后的元素依次向后拷贝
    - 如果上述 2 种操作导致元素的个数超出 vector 的当前容量，则会：
      - 重新分配更大的新的空间，将原来的所有元素复制到新分配的空中
      - 导致原来所有的迭代器失效
  - 删除、添加元素的效率：
    - 在尾端以外操作，通常代价较大，所以在需要频繁的在容器中间添加、删除元素的场合，应该考虑选用其它容器
  - 元素的访问：很高效，同数组

标准容器 - vector

- vector 相关操作 (DEMO)
  - 创建 vector 容器
  - 尾部追加、删除元素
  - 添加、删除元素
  - 访问元素、 vector 的随机迭代器

标准容器 - list

- 关于 list
  - STL list 通常实现为 double linked list
  - 除下标操作、 capacity() 、 reserve() 操作外， list 提供了 vector 的其它所有操作
  - list 的元素删除、添加很高效，为 O(1) ，没有如 vector 的迭代器失效的问题
  - list 元素的访问较低效，为 O(n)
  - list 的迭代器是双向迭代器：
    - 不能如 vector 的迭代器样提供类似 v.begin() + 2 之类的操作
    - 迭代器的比较操作用 != ，而不是 < 等

标准容器 - list

- list 操作 (DEMO)
  - 拼接、排序和合并 (splice, sort, merge)
  - 前端添加、删除元素
  - 其它操作
    - remove 、 remove_if
    - unique
    - reverse

标准容器 - deque

- 关于 deque
  - 类似于 vector， 可视为头尾双端都可以高效添加、删除元素的 vector

标准容器 - stack

- 关于 stack
  - 栈，先进后出 (First in last out, FILO)
  - stack 只提供 top()、 pop()、 push() 操作
  - 可基于 vector、 deque、 list 三种序列容器中的一种实现 stack，只保留三种 stack 操作
  - stack 是一种序列容器的适配器（不需单独实现一种容器）
  - 访问元素的唯一方法： top()，访问栈顶的元素。没有迭代器、下标操作等
- stack 操作 (DEMO)
  - 构造
  - top()、 pop()、 push()

标准容器 - queue

- 关于 queue
  - 队列，先进先出 (First in first out, FIFO)
  - queue 只提供 front() 、 back() 、 pop() 、 push() 操作
  - 可基于 deque 、 list 两种序列容器中的一种实现 queue ， 只保留四种队列操作
  - queue 也是一种序列容器的适配器（不需单独实现一种容器）
  - 访问元素的唯一方法： front() 、 back() ，访问队列头尾的元素。没有迭代器、下标操作等
- stack 操作 (DEMO)
  - 构造
  - front() 、 back() 、 pop() 、 push()

标准容器 - priority_queue

- 关于 priority_queue
  - 接口类似 stack
  - 保证每次出列的元素是队列中优先级最高（元素的值最大或最小）的元素
  - 内部数据结构通常是 heap
- priority_queue 操作 (DEMO)
  - 构造
  - top()、 push()、 pop()

标准容器 - map

- 关于 map
  - map 中的元素类型是： pair<const Key, Value>
  - 每个 key 只关联唯一一个值，也即不会有重复 key
  - 容器中的 key 是经过排序的、并且在删除、添加元素后需维护其有序性
- map 操作 (DEMO)
  - 构造
  - 元素的访问：迭代器、下标
  - find 、 lower_bound 、 upper_bound 、 equal_range
  - 添加、删除

标准容器 - multimap

- 关于 multimap
  - 类似于 map

标准容器 - set 和 multiset

- set 和 multiset
  - 类似于 map 和 multimap

标准容器 - bitset

- 关于 bitset
  - bitset 是用来容纳二进制位表示的容器，提供了一组位操作的集合
- bitset 操作
  - 构造
  - 位操作
  - 其它操作
    - to_ulong()
    - count()、 size()
    - test()、 any()、 none()

标准容器 - 更多容器

- 更多容器：
  - C++ boost Library 提供了如
    - Array（内建数组的封装）
    - Unordered Associative Container：
      - unordered_map、 unordered_set (hashmap、 hashset)
      - unordered_multimap、 unordered_multiset

    等使用容器类型。

标准容器 - Bjarne's Advices

- Bjarne's Advices
  - 如果需要容器，首先考虑 vector
  - 了解你经常使用的每个操作的代价
  - 可以依据不同的准则去排序和搜索
  - 不要使用 C 风格的字符串作为 key，除非你提供了适当的比较准则
  - 在添加和删除元素时，最好是使用序列末端的操作
  - 当需要频繁的在容器的头部和中间插入、删除元素时，请使用 list
  - 当你主要通过 key 来访问元素时，请用 map 或 multimap
  - 尽量使用最小的操作集合，以取得最大的灵活性
  - 如果要保持元素的顺序，使用 map 而不是 hashmap

标准容器 - Bjarne's Advices

- Bjarne's Advices（续）
  - 如果查找速度很重要，用 hashmap 而不是 map
  - 如果无法对元素定义小于操作时，选 hashmap 而不是 map
  - 当需检查某个 key 是否在关联容器里的时候，用 find（而不是通过下标操作 [ ] ）