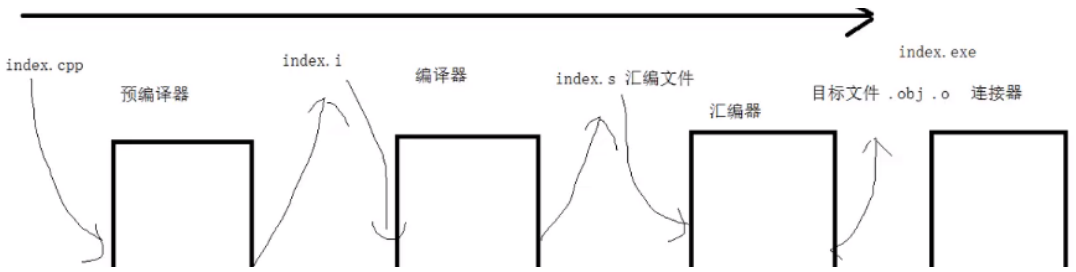



```
#pragma once //防止头文件被重复包含
```

.h 里面做实现，.cpp 里面做申明

友元函数使用模板：

```
template<class T> class Person;
template<class T> void PrintPerson(Person<T>& p);

template<class T>
class Person{
public:

    //普通友元函数
    //template<class T>
    friend void PrintPerson<T>(Person<T>& p);
```

```
//调用拷贝构造 =号操作符
//1. 对象元素必须能够被拷贝
//2. 容器都是值寓意，而非引用寓意 向容器中放入元素，都是放入
//3 如果元素的成员有指针，注意深拷贝和浅拷贝问题
```

static_cast	一般的转换
dynamic_cast	通常在基类和派生类之间转换时使用
const_cast	主要针对 const 的转换
reinterpret_cast	用于进行没有任何关联之间的转换，比如一个字符指针转换为一个整形数

```
//这个函数只能抛出int float char三种类型异常，抛出其他的就报错
void func() throw(int,float,char){
    throw "abc";
}

//不能抛出任何异常
void func02() throw(){
    throw -1;
}

//可以抛出任何类型异常
void func03(){
```

```
//普通类型元素  引用  指针
//普通元素 异常对象catch处理完之后就析构
//引用的话 不用调用拷贝构造，异常对象catch处理完之后就析构
//指针
try{
    func();
}
catch (MyException* e){
    cout << "异常捕获!" << endl;
    delete e;
}
```

4.2.1 标准输入流

标准输入流对象 cin，重点掌握的函数

cin.get() //一次只能读取一个字符

cin.get(一个参数) //读一个字符

cin.get(两个参数) //可以读字符串

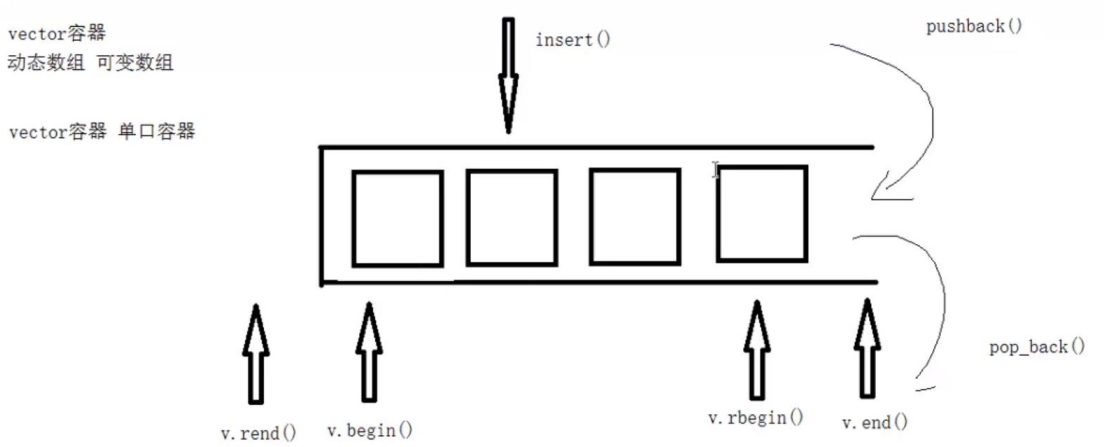
cin.getline()

cin.ignore()

cin.peek()

cin.putback()

序列式容器：元素在容器中的位置由进入的时间和地点决定;
关联式容器：元素在容器中的位置由定义的规则决定



2.3.2.1 vector 构造函数

```
vector<T> v; //采用模板实现类实现，默认构造函数
vector(v.begin(), v.end());//将 v[begin(), end())区间中的元素拷贝给本身。
vector(n, elem);//构造函数将 n 个 elem 拷贝给本身。
vector(const vector &vec);//拷贝构造函数。

//例子 使用第二个构造函数 我们可以...
int arr[] = {2,3,4,1,9};
vector<int> v1(arr, arr + sizeof(arr) / sizeof(int));
```

2.3.2.2 vector 常用赋值操作

```
assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。
assign(n, elem);//将 n 个 elem 拷贝赋值给本身。
vector& operator=(const vector  &vec);//重载等号操作符
swap(vec);// 将 vec 与本身的元素互换。

//第一个赋值函数，可以这么写：
int arr[] = { 0, 1, 2, 3, 4 };
assign(arr, arr + 5);//使用数组初始化 vector
```

2.3.2.3 vector 大小操作

```
size();//返回容器中元素的个数
empty();//判断容器是否为空
resize(int num);//重新指定容器的长度为 num，若容器变长，则以默认值填充新位置。如果容器变短，则末尾超出容器长度的元素被删除。
resize(int num, elem);//重新指定容器的长度为 num，若容器变长，则以 elem 值填充新位置。如果容器变短，则末尾超出容器长>度的元素被删除。
capacity();//容器的容量
reserve(int len);//容器预留 len 个元素长度，预留位置不初始化，元素不可访问。
```

2.3.2.4 vector 数据存取操作

```
at(int idx); //返回索引 idx 所指的数据，如果 idx 越界，抛出 out_of_range 异常。
operator[];//返回索引 idx 所指的数据，越界时，运行直接报错
front();//返回容器中第一个数据元素
back();//返回容器中最后一个数据元素
```

2.3.2.5 vector 插入和删除操作

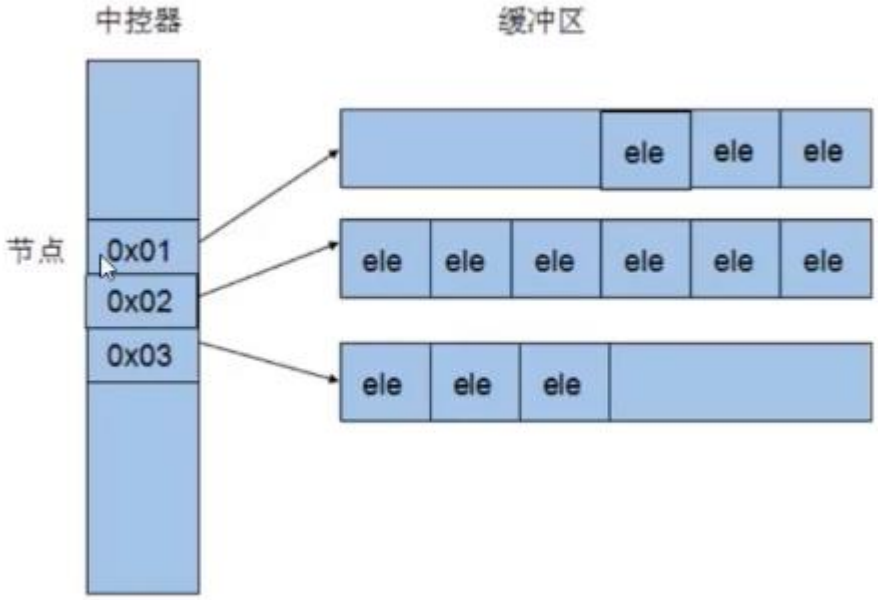
```
insert(const_iterator pos, int count,ele);//迭代器指向位置 pos 插入 count 个元素 ele.
push_back(ele); //尾部插入元素 ele
pop_back();//删除最后一个元素
erase(const_iterator start, const_iterator end);//删除迭代器从 start 到 end 之间的元素
erase(const_iterator pos);//删除迭代器指向的元素
clear();//删除容器中所有元素
```

一般的支持[]访问的都支持随机访问：

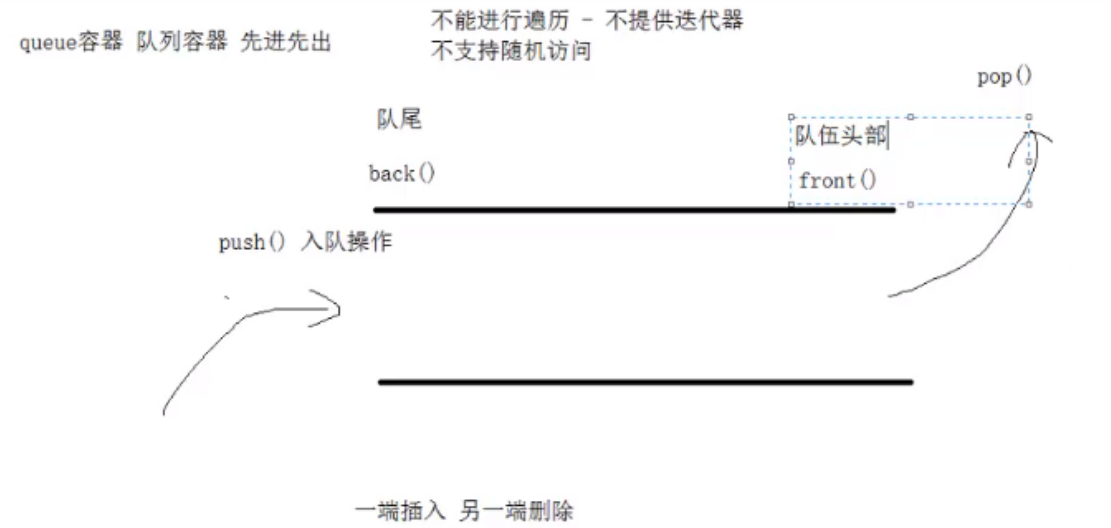
```
v.insert(v.begin() + 2, 100); //vector支持随机访问

//支持数组下标，一般都支持随机访问
//迭代器可以直接+2 +3 -2 -5操作
```

Deque 使用方法类似 vector (deque 排序效率低下，一般都是拷贝到 vector 中排好序再输出到 deque)：



栈不能遍历，不支持随机存取，只能通过 top 从栈顶获取和删除元素



c++	JAVA	目的
array	[ ]	固定大小的数组
vector	ArrayList	可变长度的数组
	Vector	可变长度数组，支持同步操作，效率比ArrayList略低
list	LinkedList	链表，便于增删
forward_list		单链表，注意不提供size()操作
deque	ArrayDeque	双端队列
stack	Stack	栈
queue	Queue	队列
priority_queue	PriorityQueue	支持优先级的队列
set	TreeSet	集合，数据有序，通过二叉搜索树实现
multiset		集合，允许重复元素
unordered_set	HashSet	hash组织的set
unordered_multiset		hash组织的multiset
	LinkedHashSet	按插入有序，支持hash查找
map	TreeMap	key-value映射，按照key有序
multimap		允许重复key的map
unordered_map	HashMap	hash组织的map
unordered_multimap		hash组织的multimap
	LinkedHashMap	按插入有序，支持hash查找
	HashTable	类似HashMap，支持同步操作
bitset	BitSet	位操作

链表是由一系列的节点组成，节点包含两个域，一个数据域，一个指针域。

链表内存是连续的还是非连续的?

1. 非连续，添加删除元素 时间复杂度都是常数项，不需要移动元素，比数组添加删除效率高;
2. 链表只有在需要的时候，才分配内存;
3. 链表需要额外的空间保存结点关系 前驱 后继 关系
4. 链表只要拿到第一个结点就相当于拿到整个链表

2.6.2.1 list 构造函数

```
list<T> lstT;//list 采用采用模板类实现,对象的默认构造形式:
list(beg,end);//构造函数将[beg, end)区间中的元素拷贝给本身。
list(n,elem);//构造函数将 n 个 elem 拷贝给本身。
list(const list &lst);//拷贝构造函数。
```

2.6.2.2 list 数据元素插入和删除操作

```
push_back(elem);//在容器尾部加入一个元素
pop_back();//删除容器中最后一个元素
push_front(elem);//在容器开头插入一个元素
pop_front();//从容器开头移除第一个元素
insert(pos,elem);//在 pos 位置插 elem 元素的拷贝，返回新数据的位置。
insert(pos,n,elem);//在 pos 位置插入 n 个 elem 数据，无返回值。
insert(pos,beg,end);//在 pos 位置插入[beg,end)区间的数据，无返回值。
clear();//移除容器的所有数据
erase(beg,end);//删除[beg,end)区间的数据，返回下一个数据的位置。
erase(pos);//删除 pos 位置的数据，返回下一个数据的位置。
remove(elem);//删除容器中所有与 elem 值匹配的元素。
```

2.6.2.3 list 大小操作

```
size();//返回容器中元素的个数
empty();//判断容器是否为空
resize(num);//重新指定容器的长度为 num,
若容器变长，则以默认值填充新位置。
```

```
如果容器变短，则末尾超出容器长度的元素被删除。
resize(num, elem);//重新指定容器的长度为 num，
若容器变长，则以 elem 值填充新位置。
如果容器变短，则末尾超出容器长度的元素被删除。
```

2.6.2.4 list 赋值操作

```
assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。
assign(n, elem);//将 n 个 elem 拷贝赋值给本身。
list& operator=(const list &lst);//重载等号操作符
swap(lst);//将 lst 与本身的元素互换。
```

2.6.2.5 list 数据的存取

```
front();//返回第一个元素。
back();//返回最后一个元素。
```

2.6.2.5 list 反转排列排序

```
reverse();//反转链表，比如 lst 包含 1,3,5 元素，运行此方法后，lst 就包含 5,3,1 元素。
sort(); //list 排序
```

上面这个 sort()是隶属于 list 容器的成员函数!!

Set 和 MultiSet 自动排序，其迭代器不能修改元素的值（改了就改变了顺序，就乱了）

2.7.2.1 set 构造函数

```
set<T> st;//set 默认构造函数:
mulitset<T> mst; //multiset 默认构造函数:
set(const set &st);//拷贝构造函数
```

2.7.2.2 set 赋值操作

```
set& operator=(const set &st);//重载等号操作符
swap(st);//交换两个集合容器
```

2.7.2.2 set 大小操作

```
size();//返回容器中元素的数目
empty();//判断容器是否为空
```

2.7.2.2 set 插入和删除操作

```
insert(elem);//在容器中插入元素。
clear();//清除所有元素
erase(pos);//删除 pos 迭代器所指的元素，返回下一个元素的迭代器。
erase(beg, end);//删除区间[beg,end)的所有元素 ，返回下一个元素的迭代器。
erase(elem);//删除容器中值为 elem 的元素。
```

2.7.2.3 set 查找操作

```
find(key);//查找键 key 是否存在,若存在，返回该键的元素的迭代器；若不存在，返回 map.end();
lower_bound(keyElem);//返回第一个 key>=keyElem 元素的迭代器。
upper_bound(keyElem);//返回第一个 key>keyElem 元素的迭代器。
equal_range(keyElem);//返回容器中 key 与 keyElem 相等的上下限的两个迭代器。
```

2.8.2 对组

对组(pair)将一对值组合成一个值，这一对值可以具有不同的数据类型，两个值可以分别用 pair 的两个公有函数 first 和 second 访问。

类模板：template <class T1, class T2> struct pair.

如何创建对组?

```
//第一种方法创建一个对组
pair<string, int> pair1(string("name"), 20);
cout << pair1.first << endl; //访问 pair 第一个值
cout << pair1.second << endl;//访问 pair 第二个值
//第二种
```

Set 自定义排序之后，find()查找函数是根据查找（自定义排序的根据---某个元素）来作为是否查找到的根据的：定义一个 Person(age=10,name='s'),将 它放入 set<Person>中，并以 age 为根据排序，这种情况下如果找另一个未放入 set<Person>的 Person(age=10,name="qg")，set 也能找到，但是找到的是 Person(age=10,name='s')

2.10 STL 容器使用时机

	vector	deque	list	set	multiset	map	multimap
典型内存结构	单端数组	双端数组	双向链表	二叉树	二叉树	二叉树	二叉树
可随机存取	是	是	否	否	否	对 key 而言：是	否
元素搜寻速度	慢	慢	非常慢	快	快	对 key 而言：快	对 key 而言：快
元素安插移除	尾端	头尾两端	任何位置	-	-	-	-

- vector 的使用场景：比如软件历史操作记录的存储，我们经常要查看历史记录，比如上一次的记录，上上次的记录，但却不会去删除记录，因为记录是事实的描述。
- deque 的使用场景：比如排队购票系统，对排队者的存储可以采用 deque，支持头端的快速移除，尾端的快速添加。如果采用 vector，则头端移除时，会移动大量的数据，速度慢。

vector 与 deque 的比较：

一：vector.at()比 deque.at()效率高，比如 vector.at(0)是固定的，deque 的开始位置却是不固定的。

二：如果有大量释放操作的话，vector 花的时间更少，这跟二者的内部实现有关。

三：deque 支持头部的快速插入与快速移除，这是 deque 的优点。

- list 的使用场景：比如公交车乘客的存储，随时可能有乘客下车，支持频繁的不确实位置元素的移除插入。
- set 的使用场景：比如对手机游戏的个人得分记录的存储，存储要求从高分到低分的顺序排列。

- map 的使用场景：比如按 ID 号存储十万个用户，想要快速要通过 ID 查找对应的用户。二叉树的查找效率，这时就体现出来了。如果是 vector 容器，最坏的情况下可能要遍历完整个容器才能找到该用户。

3.1 函数对象

3.1.1 函数对象的概念

重载函数调用操作符的类，其对象常称为函数对象（function object），即它们是行为类似函数的对象，也叫仿函数(functor),其实就是重载“()”操作符，使得类对象可以像函数那样调用。

注意:1.函数对象(仿函数)是一个类，不是一个函数。

2.函数对象(仿函数)重载了”()”操作符使得它可以像函数一样调用。

假定某个类有一个重载的 operator()，而且重载的 operator()要求获取一个参数，我们就将这个类称为“一元仿函数”（unary functor）；相反，如果重载的 operator()要求获取两个参数，就将这个类称为“二元仿函数”（binary functor）。

函数对象可以像普通函数一样调用；函数对象可以像普通函数那样接受参数；函数对象超出了函数的概念，函数对象可以保存函数调用的状态

```
//     for_each()返回但三个参数的类型
    myFunc02 m2 = for_each(v.begin(),v.end(),m1);
```

3.1.2 谓词概念

谓词是指普通函数或重载的 operator()返回值是 bool 类型的函数对象(仿函数)。如果 operator 接受一个参数，那么叫做一元谓词,如果接受两个参数，那么叫做二元谓词，谓词可作为一个判断式。

例如：

```
struct myfuncobj01{
      bool operator(int v){}  //接受一个参数，并且返回值为 Bool 即一元谓词
}
bool compare01(int v); //同样是叫做一元谓词

struct myfuncobj02{
      bool operator(int v1, int v2){}  //接受两个参数，返回值为 Bool 即二元谓词
}
bool compare02(int v1, int v2); //同样是叫做二元谓词
```

3.1.4 函数对象适配器

函数对象适配器是完成一些配接工作，这些配接包括绑定(bind)，否定(negate),以及对一般函数或成员函数的修饰，使其成为函数对象，重点掌握函数对象适配器(红色字体):

bind1st ：将参数绑定为函数对象的第一个参数

bind2nd ： 将参数绑定为函数对象的第二个参数

not1 ： 对一元函数对象取反

not2：对二元函数对象取反

ptr_fun ： 将普通函数修饰成函数对象

mem_fun ： 修饰成员函数

mem_fun_ref ： 修饰成员函数

```
find(iterator beg, iterator end, value)
/*
    adjacent_find 算法 查找相邻重复元素
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param  _callback 回调函数或者谓词(返回 bool 类型的函数对象)
    @return 返回相邻元素的第一个位置的迭代器
*/
adjacent_find(iterator beg, iterator end, _callback);
/*
    binary_search 算法 二分查找法
    注意: 在无序序列中不可用
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param value 查找的元素
    @return bool 查找返回 true 否则 false
*/
bool binary_search(iterator beg, iterator end, value);
/*
```

```
find_if(iterator beg, iterator end, _callback);
/*
    count 算法 统计元素出现次数
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param  value 回调函数或者谓词(返回 bool 类型的函数对象)
    @return int 返回元素个数
*/
count(iterator beg, iterator end, value);
/*
    count 算法 统计元素出现次数
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param  callback 回调函数或者谓词(返回 bool 类型的函数对象)
    @return int 返回元素个数
*/
count_if(iterator beg, iterator end, _callback);
```