C++ 标准模板库
C++ Standard Template Libarary

ACM/ICPC 培训

瞿绍军

主要内容

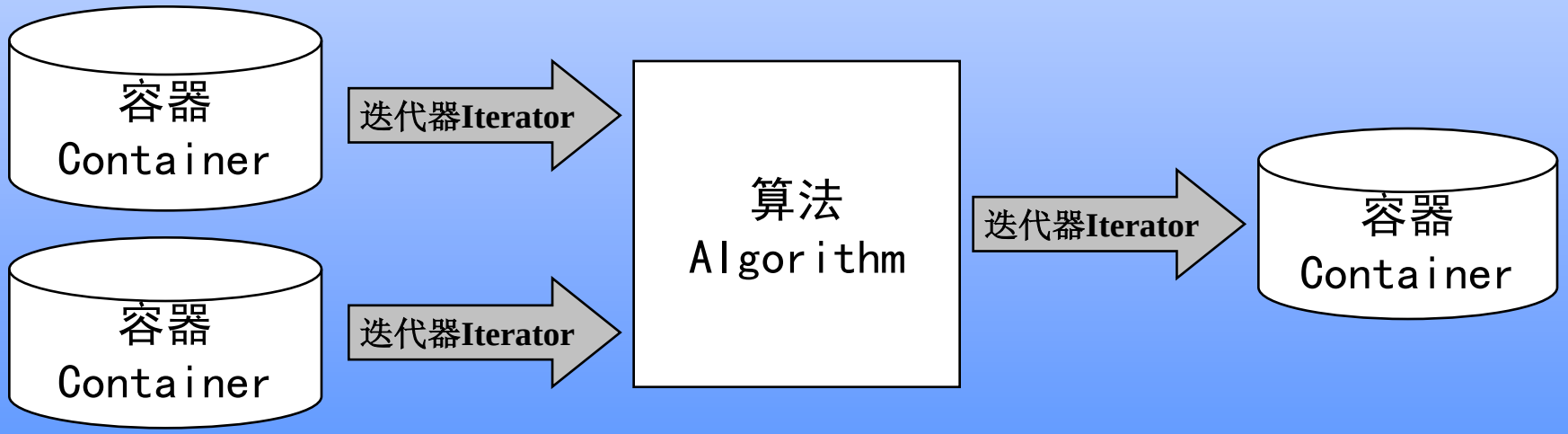
- **STL概述：组件、容器、迭代器（iterator）、算法**
- **STL容器：vector、list、stack、queue**
- **STL算法：搜寻、排序、拷贝、数值运算**

STL概述

- STL是C++标准程序库的核心，深刻影响了标准程序库的整体结构
- STL是泛型（generic）程序库，利用先进、高效的算法来管理数据
- STL由一些可适应不同需求的集合类（collection class），以及在这些数据集合上操作的算法（algorithm）构成
- STL内的所有组件都由模板（template）构成，其元素可以是任意类型

STL概述

- **STL组件**
  - 容器（Container） — 管理某类对象的集合
  - 迭代器（Iterator） — 在对象集合上进行遍历
  - 算法（Algorithm） — 处理集合内的元素



STL组件之间的协作

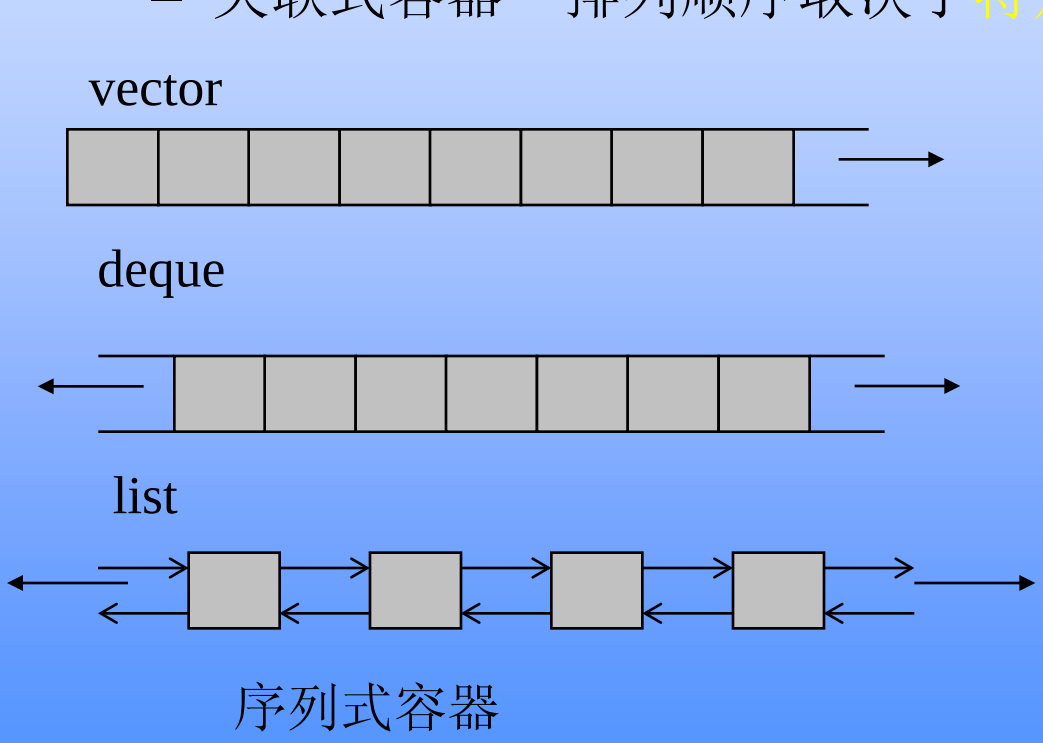
STL概述

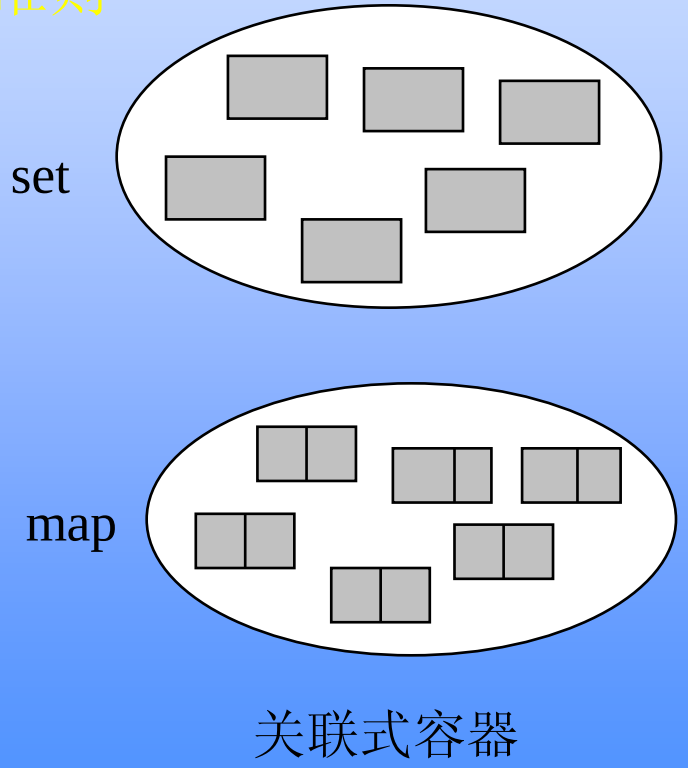
- **STL容器元素的条件**
  - 必须能够通过拷贝构造函数进行复制
  - 必须可以通过赋值运算符完成赋值操作
  - 必须可以通过析构函数完称销毁动作
  - 序列式容器元素的默认构造函数必须可用
  - 某些动作必须定义operator ==，例如搜寻操作
  - 关联式容器必须定义出排序准则，默认情况是重载operator <

对于基本数据类型（int, long, char, double, ...）而言，以上条件总是满足

STL概述

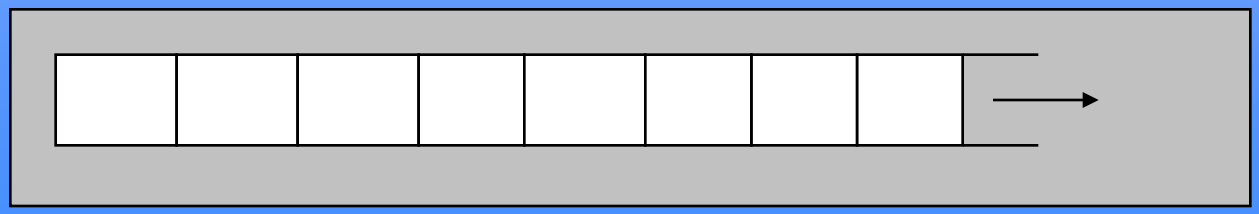
- **STL容器类别**
  - 序列式容器－排列次序取决于插入时机和位置
  - 关联式容器－排列顺序取决于特定准则





顺序容器：向量容器

- **vector**
  - 使用动态数组来存储和管理它的对象.
  - vector模拟动态数组
  - vector的元素可以是任意类型T，但必须具备赋值和拷贝能力（具有public拷贝构造函数和重载的赋值操作符）
  - 必须包含的头文件**#include <vector>**
  - vector支持随机存取
  - vector的大小（size）和容量（capacity）通常是不同的，size返回实际元素个数，capacity返回vector能容纳的元素最大数量。如果插入元素时，元素个数超过capacity，需要重新配置内部存储器。



vector

- **vector**
  - 构造、拷贝和析构

操作	效果
vector<T> c	产生空的vector
vector<T> c1(c2)	产生同类型的c1，并将复制c2的所有元素
vector<T> c(n)	利用类型T的默认构造函数和拷贝构造函数生成一个大小为n的vector
vector<T> c(n,e)	产生一个大小为n的vector，每个元素都是e
vector<T> c(beg,end)	产生一个vector，以区间[beg,end]为元素初值
~vector<T>()	销毁所有元素并释放内存。

vector

- **vector**
  - 非变动操作

操作	效果
c.size()	返回元素个数
c.empty()	判断容器是否为空
c.max_size()	返回元素最大可能数量（固定值）
c.capacity()	返回重新分配空间前可容纳的最大元素数量
c.reserve(n)	扩大容量为n
c1==c2	判断c1是否等于c2
c1!=c2	判断c1是否不等于c2
c1<c2	判断c1是否小于c2
c1>c2	判断c1是否大于c2
c1<=c2	判断c1是否大于等于c2
c1>=c2	判断c1是否小于等于c2

vector

- **vector**
  - 赋值操作

操作	效果
c1 = c2	将c2的全部元素赋值给c1
c.assign(n,e)	将元素e的n个拷贝赋值给c
c.assign(beg,end)	将区间[beg;end]的元素赋值给c
c1.swap(c2)	将c1和c2元素互换
swap(c1,c2)	同上，全局函数

```
std::list<T> l;
std::vector<T> v;
…
v.assign(l.begin(),l.end());
```

所有的赋值操作都有可能调用元素类型的默认构造函数，拷贝构造函数，赋值操作符和析构函数

vector

- **vector**
  - 元素存取

操作	效果
at(idx)	返回索引idx所标识的元素，对idx会进行越界检查
operator [](idx)	返回索引idx所标识的元素，对idx不进行越界检查
front()	返回第一个元素，不检查第一个元素是否存在
back()	返回最后一个元素，不检查最后一个元素是否存在

```
std::vector<T> v;//empty

v[5]= t;                        //runtime error
std::cout << v.front();         //runtime error
```

vector

- 为vector容器声明迭代器

  作用：遍历容器中的元素

  vector<int>::iterator intVecIter;

vector

- **vector**
  - 迭代器相关函数

操作	效果
begin()	返回一个迭代器，指向第一个元素
end()	返回一个迭代器，指向最后一个元素之后
rbegin()	返回一个逆向迭代器，指向逆向遍历的第一个元素
rend()	返回一个逆向迭代器，指向逆向遍历的最后一个元素

迭代器持续有效，除非发生以下两种情况：
（1）或插入元素
（2）容量变化而引起内存重新分配

vector

- **vector**
  - 安插（insert）元素

操作	效果
c.insert(pos,e)	在pos位置插入元素e的副本，并返回新元素位置
c.insert(pos,n,e)	在pos位置插入n个元素e的副本
c.insert(pos,beg,end)	在pos位置插入区间[beg;end]内所有元素的副本
c.push_back(e)	在尾部添加一个元素e的副本

vector

- **vector**
  - 移除（remove）元素

操作	效果
c.pop_back()	移除最后一个元素但不返回最后一个元素
c.erase(pos)	删除pos位置的元素，返回下一个元素的位置
c.erase(beg,end)	删除区间[beg;end]内所有元素，返回下一个元素的位置
c.clear()	移除所有元素，清空容器
c.resize(num)	将元素数量改为num（增加的元素用defalut构造函数产生，多余的元素被删除）
c.resize(num,e)	将元素数量改为num（增加的元素是e的副本）

vector

- **vector**
  - vector应用实例：vector

STL容器通用的成员函数

- **STL容器的通用能力**
  - 所有容器中存放的都是值而非引用，即容器进行安插操作时内部实施的是拷贝操作。因此容器的每个元素必须能够被拷贝。如果希望存放的不是副本，容器元素只能是指针。
  - 所有元素都形成一个次序（order），可以按相同的次序一次或多次遍历每个元素
  - 各项操作并非绝对安全，调用者必须确保传给操作函数的参数符合需求，否则会导致未定义的行为

STL容器通用的成员函数

- **STL容器的共通操作**
  - 初始化（initialization）
    - 产生一个空容器

```
std::list<int> l;
```

    - 以另一个容器元素为初值完成初始化

```
std::list<int> l;
…
std::vector<float> c(l.begin(),l.end());
```

    - 以数组元素为初值完成初始化

```
int array[]={2,4,6,1345};
…
std::set<int> c(array,array+sizeof(array)/sizeof(array[0]));
```

STL容器通用的成员函数

- **STL容器的共通操作**
  - 与大小相关的操作（size operator）
    - size()—返回当前容器的元素数量
    - empty()—判断容器是否为空
    - max_size()—返回容器能容纳的最大元素数量
  - 比较（comparison）
    - ==,!=,<,<=,>,>=
    - 比较操作两端的容器必须属于同一类型
    - 如果两个容器内的所有元素按序相等，那么这两个容器相等
    - 采用字典式顺序判断某个容器是否小于另一个容器
  - 赋值（assignment）和交换（swap）
    - swap用于提高赋值操作效率

STL容器通用的成员函数

- **容器的共通操作**
  - 与迭代器（iterator）相关的操作
    - begin()—返回一个迭代器，指向第一个元素
    - end()—返回一个迭代器，指向最后一个元素之后
    - rbegin()—返回一个逆向迭代器，指向逆向遍历的第一个元素
    - rend()—返回一个逆向迭代器，指向逆向遍历的最后一个元素之后
  - 元素操作
    - insert(pos,e)—将元素e的拷贝安插于迭代器pos所指的位置
    - erase(beg,end)—移除[beg，end]区间内的所有元素
    - clear()—移除所有元素

copy算法

- 输出容器中的 元素
- 将元素从一个地方复制到另一个地方
- 函数模版原型(#include<algorithm>)

template <class inputIterator,class outputIterator>

outputItr copy(inputIterator first1, inputIterator last,outputIterator first2);

first1: 复制元素的 开始位置

last:复制元素的 结束位置

first2:指定将元素复制到哪里

copy算法

```
int intArray[]={5,6,8,3,23,35,67,34,55};
vector<int> vecList(9);
copy(intArray,intArray+9,vecList.begin());
```

ostream迭代器和copy函数

输出容器元素方法1:for

```
vector<string>::iterator it;
for(it=sentence.begin();it!=sentence.end();++it)
    cout<<*it<<" ";
cout<<endl;
```

ostream迭代器和copy函数

输出容器元素方法2:copy和ostream

```
copy (sentence.begin(), sentence.end(),
        ostream_iterator<string>(cout," "));
  cout << endl;
```

或

```
ostream_iterator<string>  screen(cout," ");
copy (sentence.begin(), sentence.end(),screen);
```

迭代器

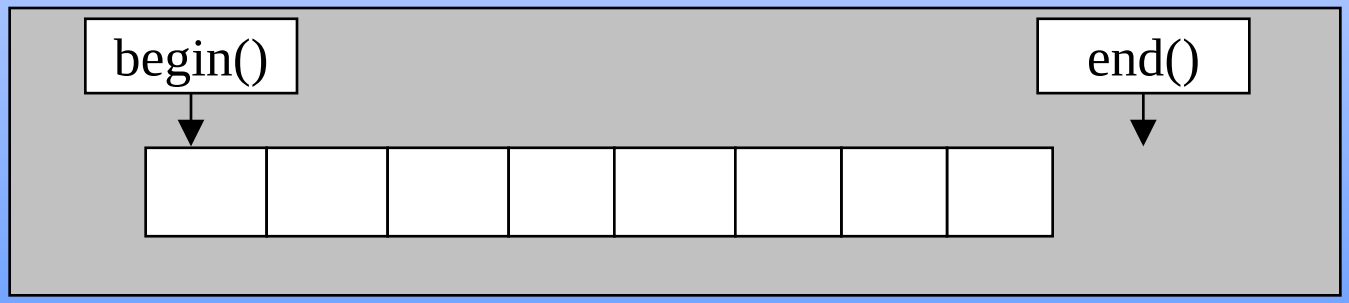
- **迭代器（iterator）（示例:iterator）**
  - 可遍历STL容器内全部或部分元素的对象
  - 指出容器中的一个特定位置
  - 迭代器的基本操作

操作	效果
*	返回当前位置上的元素值。如果该元素有成员，可以通过迭代器以operator ->取用
++	将迭代器前进至下一元素
==和!=	判断两个迭代器是否指向同一位置
=	为迭代器赋值（将所指元素的位置赋值过去）

迭代器

- **迭代器（iterator）**
  - 所有容器都提供获得迭代器的函数

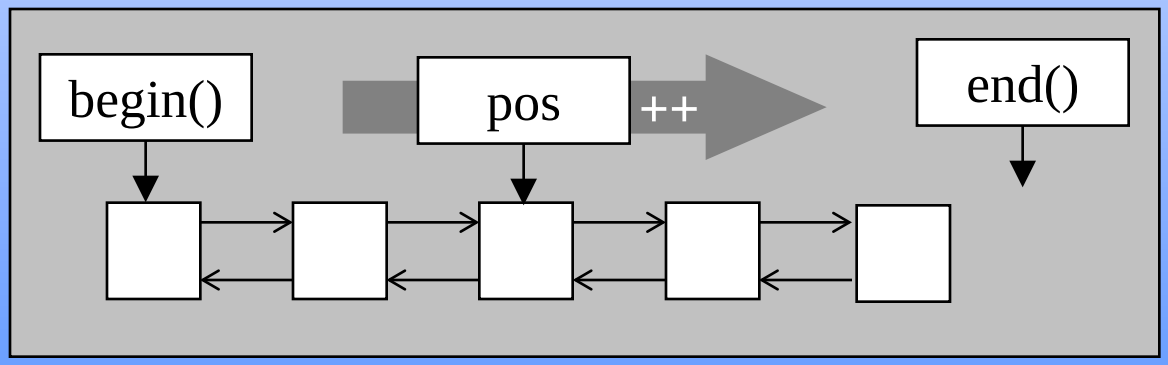
操作	效果
begin()	返回一个迭代器，指向第一个元素
end()	返回一个迭代器，指向最后一个元素之后



半开区间[beg, end)的好处：
1.为遍历元素时循环的结束时机提供了简单的判断依据（只要未到达end()，循环就可以继续）
2.不必对空区间采取特殊处理（空区间的begin()就等于end()）

迭代器

- **迭代器（iterator）**
  - 所有容器都提供两种迭代器
    - container::iterator以“读/写”模式遍历元素
    - container::const_iterator以“只读”模式遍历元素
  - 迭代器示例：iterator



迭代器

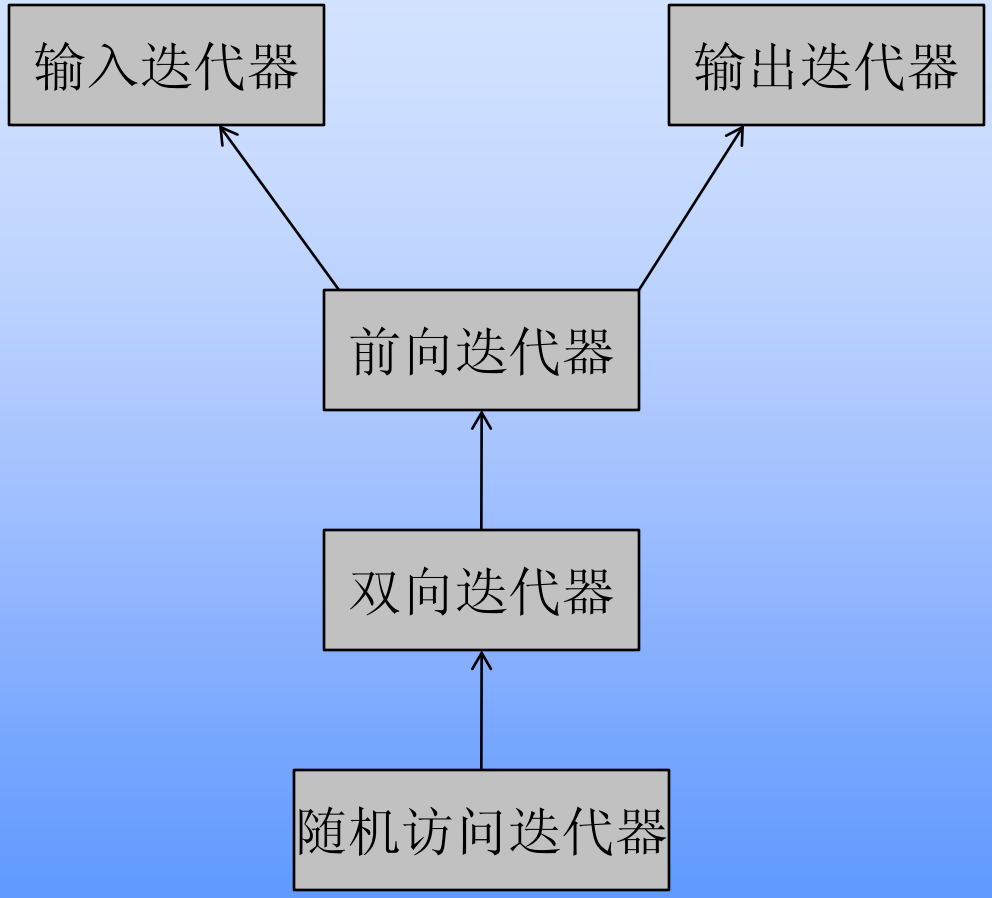
- **迭代器（iterator）**
  - 迭代器分类
    - 输入迭代器
      - 可逐个读取元素
      - 用于从输入流中读取数据
    - 输出迭代器
      - 可逐个元素进行写操作
      - 用于向一个输出流中写入数据
    - 前向迭代器
      - 综合了输入迭代器和输出迭代器大部分功能

迭代器

- 双向迭代器
  - 可以双向行进,以递增运算前进或以递减运算后退。
  - vector、deque 、list、set和map提供双向迭代器
- 随机存取迭代器
  - 除了具备双向迭代器的所有属性，还具备随机访问能力。
  - 可以对迭代器增加或减少一个偏移量、处理迭代器之间的距离或者使用<和>之类的关系运算符比较两个迭代器。
  - vector、deque和string提供随机存取迭代器

```
Vector<int> v;
for(pos=v.begin();pos<v.end();++pos{
        …
 }
```

迭代器



流迭代器

- istream迭代器

  istream_iterator<Type> isIdentifier(istream&);

- ostream迭代器

  ostream_iterator<Type> osIdentifier(ostream&);

  或

  ostream_iterator<Type> osIdentifier(ostream&,char * deLimit);

迭代器

```
#include <vector>
#include <list>
#include <deque>
#include <algorithm>
#include <iostream>
using namespace std;

int main()
{
   const int arraySize = 7;
   int ia[arraySize] = { 0,1,2,3,4,5,6 };

   vector<int> ivect(ia, ia+arraySize);
   list<int> ilist(ia, ia+arraySize);
   deque<int> ideque(ia, ia+arraySize);
```

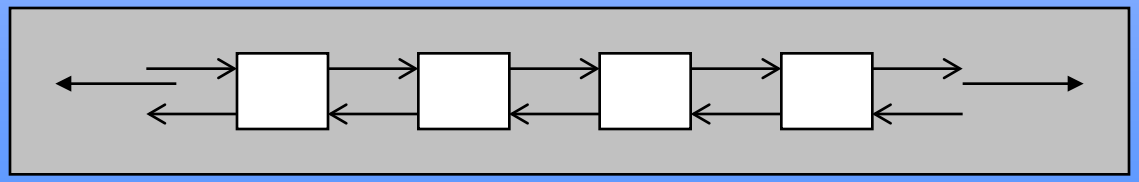
```
    vector<int>::iterator it1 = find(ivect.begin(), ivect.end(), 4);
    if (it1 == ivect.end())
        cout << "4 not found." << endl;
    else
        cout << "4 found. " << *it1 << endl;
    // 执行结果: 4 found. 4

    list<int>::iterator it2 = find(ilist.begin(), ilist.end(), 6);
    if (it2 == ilist.end())
        cout << "6 not found." << endl;
    else
        cout << "6 found. " << *it2 << endl;
    // 执行结果: 6 found. 6

    deque<int>::iterator it3 = find(ideque.begin(), ideque.end(), 8);
    if (it3 == ideque.end())
        cout << "8 not found." << endl;
    else
        cout << "8 found. " << *it3 << endl;
    // 执行结果: 8 not found
}
```

顺序容器：list

- **list**
  - 使用双向链表管理元素
  - list的元素可以是任意类型T，但必须具备赋值和拷贝能力
  - 必须包含的头文件#include <list>
  - list不支持随机存取，因此不提供下标操作符
  - 在任何位置上执行元素的安插和移除都非常快。
  - 安插和删除不会导致指向其他元素的指针、引用、iterator失效。



顺序容器：list

- **list**
  - 构造、拷贝和析构

操作	效果
list<T> c·	产生空的list
list<T> c1(c2)	产生同类型的c1，并将复制c2的所有元素
list<T> c(n)	利用类型T的默认构造函数和拷贝构造函数生成一个大小为n的list
list<T> c(n,e)	产生一个大小为n的list，每个元素都是e
list<T> c(beg,end)	产生一个list，以区间[beg,end]为元素初值
~list<T>()	销毁所有元素并释放内存。

顺序容器：list

- **list**
  - 非变动性操作

操作	效果
c.size()	返回元素个数
c.empty()	判断容器是否为空
c.max_size()	返回元素最大可能数量
c1==c2	判断c1是否等于c2
c1!=c2	判断c1是否不等于c2
c1<c2	判断c1是否小于c2
c1>c2	判断c1是否大于c2
c1<=c2	判断c1是否大于等于c2
c1>=c2	判断c1是否小于等于c2

顺序容器：list

- **list**
  - 赋值

操作	效果
c1 = c2	将c2的全部元素赋值给c1
c.assign(n,e)	将e的n个拷贝赋值给c
c.assign(beg,end)	将区间[beg;end]的元素赋值给c
c1.swap(c2)	将c1和c2的元素互换
swap(c1,c2)	同上，全局函数

顺序容器：list

- **list**
  - 元素存取

操作	效果
front()	返回第一个元素，不检查第一个元素是否存在
back()	返回最后一个元素，不检查最后一个元素是否存在

```
std::list<T> l;//empty

std::cout << l.front();          //runtime error
if(!l.empty()){
        std::cout<<l.back();          //ok
}
```

顺序容器：list

- **list**
  - 迭代器相关函数

操作	效果
begin()	返回一个双向迭代器，指向第一个元素
end()	返回一个双向迭代器，指向最后一个元素之后
rbegin()	返回一个逆向迭代器，指向逆向遍历的第一个元素
rend()	返回一个逆向迭代器，指向逆向遍历的最后一个元素

顺序容器：list

- **list**
  - 安插（insert）元素

操作	效果
c.insert(pos,e)	在pos位置插入e的副本，并返回新元素位置
c.insert(pos,n,e)	在pos位置插入n个e的副本
c.insert(pos,beg,end)	在pos位置插入区间[beg;end]内所有元素的副本
c.push_back(e)	在尾部添加一个e的副本
c.push_front(e)	在头部添加一个e的副本

顺序容器：list

- **list**
  - 移除（remove）元素

操作	效果
c.pop_back()	移除最后一个元素但不返回
c.pop_front()	移除第一个元素但不返回
c.erase(pos)	删除pos位置的元素，返回下一个元素的位置
c.remove(val)	移除所有值为val的元素
c.remove_if(op)	移除所有“op(val)==true”的元素
c.erase(beg,end)	删除区间[beg;end]内所有元素，返回下一个元素的位置
c.clear()	移除所有元素，清空容器
c.resize(num)	将元素数量改为num（多出的元素用defalut构造函数产生）
c.resize(num,e)	将元素数量改为num（多出的元素是e的副本）

顺序容器：list

- **list**
  - 特殊变动性操作

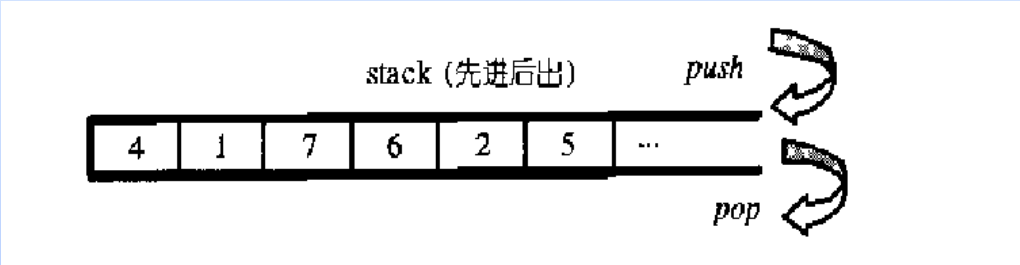
操作	效果
c.unique	移除重复元素，只留下一个
c.unique(op)	移除使op()结果为true的重复元素
c1.splice(pos,c2)	将c2内的所有元素转移到c1的迭代器pos之前
c1.splice(pos,c2,c2pos)	将c2内c2pos所指元素转移到c1内的pos之前
c1.splice(pos,c2,c2beg,c2end)	将c2内[c2beg;c2end]区间内所有元素转移到c2的pos之前
c.sort()	以operator <为准则对所有元素排序
c.sort(op)	以op为准则对所有元素排序
c1.merge(c2)	假设c1和c2都已排序，将c2全部元素转移到c1并保证合并后list仍为已排序
c.reverse()	将所有元素反序

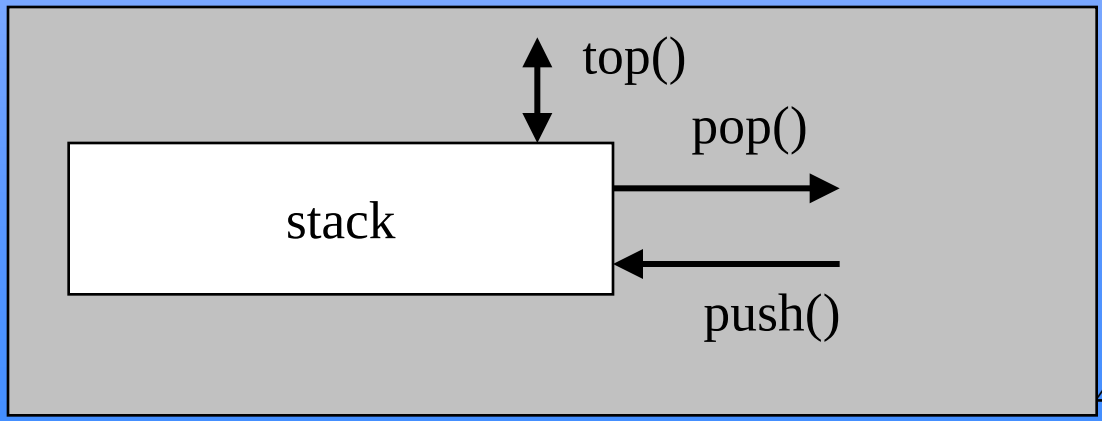
顺序容器：list

- **list**
  - list应用实例：list

STL容器：堆栈

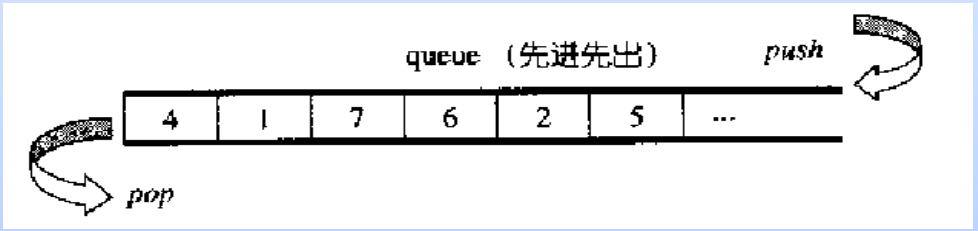
- **stack**
  - 后进先出（LIFO）
  - #include <stack>
  - 核心接口
    - empty() -- 返回bool型，表示栈内是否为空 (s.empty())
    - size() -- 返回栈内元素个数 (s.size())
    - push(value)—将元素压栈
    - top()—返回栈顶元素，但不移除
    - pop()—从栈中移除栈顶元素，但不返回
    - 不提供迭代器
  - 实例：stack

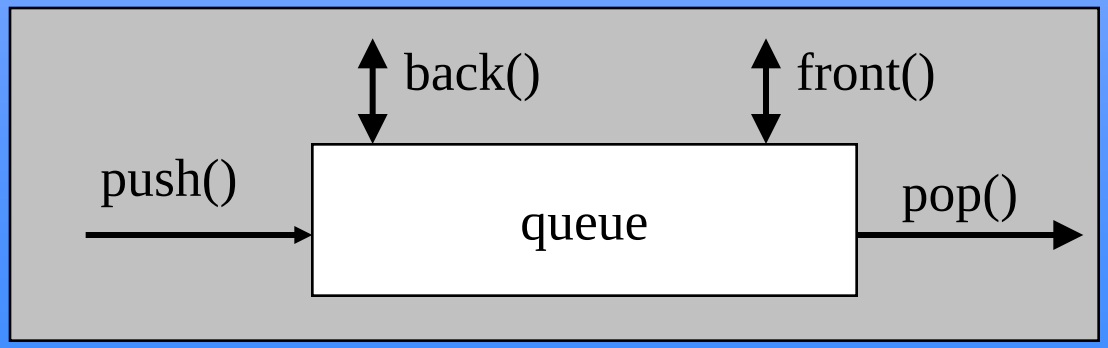




STL容器：队列

- **queue**
  - 先进先出（FIFO）
  - #include <queue>
  - 核心接口
    - empty() -- 返回bool型，表示queue是否为空 (q.empty())
    - size() -- 返回queue内元素个数 (q.size())
    - push(value)—将元素置入队列
    - front()—返回队列头部元素，但不移除
    - back()—返回队列尾部元素，但不移除
    - pop()—从队列中移除元素，但不返回
  - 实例：queue

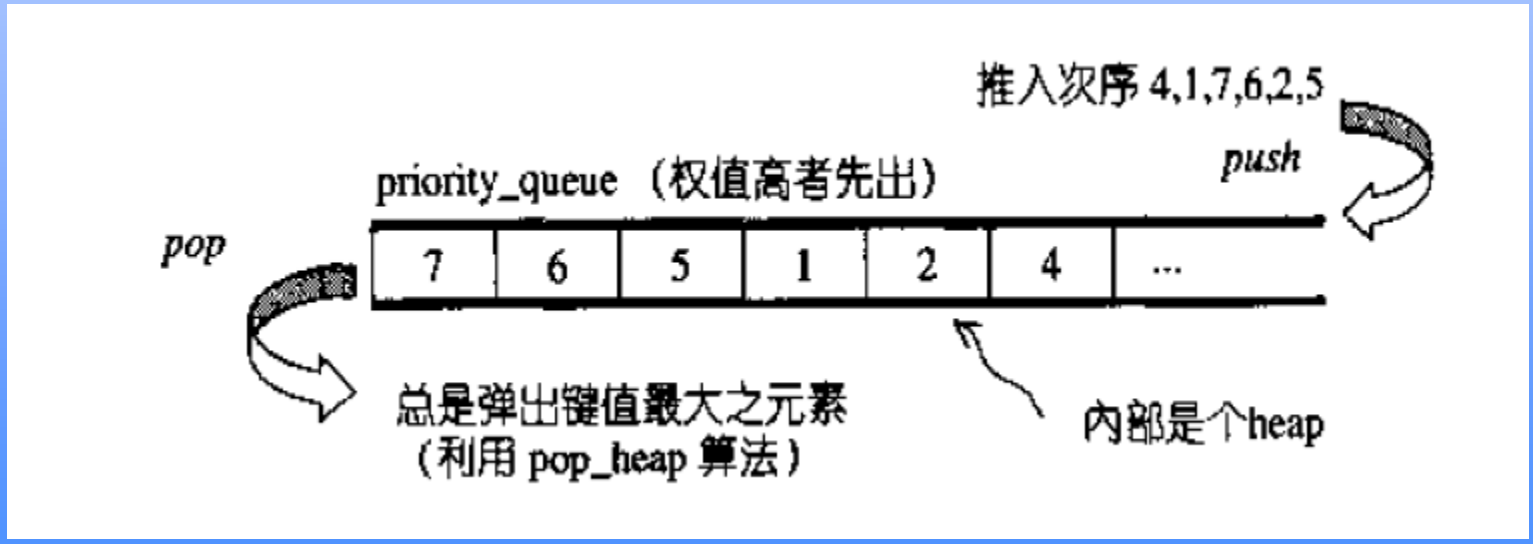




STL容器：优先队列

- 优先队列（**priority_queue**）

一个拥有权值观念的queue，自动依照元素的权值排列，权值最高排在前面。缺省情况下，priority_queue是利用一个max_heap完成的



STL容器：优先队列

- 优先队列（**priority_queue**）

头文件: #include <queue>
定义：priority_queue <data_type> priority_queue_name;
　　如： priority_queue <int> q;//默认是大顶堆
操作：
　　q.push(elem) 将元素elem置入优先队列
　　q.top() 返回优先队列的下一个元素
　　q.pop() 移除一个元素
　　q.size() 返回队列中元素的个数
　　q.empty() 返回优先队列是否为空

双端队列

- **Deque**
  - Deque也是一个动态的Array，不过两端都可以添加元素，不过向中间添加元素时耗时比较多，因为要移动元素。用法语vector类似。
  - #include<deque>

双端队列

- **Member functions**
- **(constructor) Construct deque container (public member function)**
- **(destructor) Deque destructor (public member function)**
- **operator= Copy container content (public member function)**

双端队列

- **Iterators:**
  begin Return iterator to beginning (public member function)
- **end Return iterator to end (public member function)**
- **rbegin Return reverse iterator to reverse beginning (public member function)**
- **rend Return reverse iterator to reverse end (public member function)**

双端队列

- **Capacity:**
  size Return size (public member function)
- **max_size Return maximum size (public member function)**
- **resize Change size (public member functions)**
- **empty Test whether container is empty (public member function)**

双端队列

- **Element access:**
- **operator[] Access element (public member function)**
- **at Access element (public member function)**
- **front Access first element (public member function)**
- **back Access last element (public member function)**

双端队列

- **Modifiers:**
  assign Assign container content (public member function)
- **push_back Add element at the end (public member function)**
- **push_front Insert element at beginning (public member function)**
- **pop_back Delete last element (public member function)**
- **pop_front Delete first element (public member function)**
- **insert Insert elements (public member function)**
- **erase Erase elements (public member function)**
- **swap Swap content (public member function)**
- **clear Clear content (public member function)**

pair

- 利用pair类可以将两个值合并成一个单元，作为一个整体来处理。

  #include<utility>

- 构造函数
- pair<Type1,Type2>pElement;
- pair<Type1,Type2>pElement(expr1,expr2);
- pair模板类对象有两个成员：first和second，分别表示首元素和尾元素。
- pair类重载了关系运算符（=,<,<=,>,>=,!=）

make_pair

- 除了直接定义一个pair对象外，如果需要即时生成一个pair对象，也可以调用在<utility>中定义的一个模板函数：make_pair。make_pair需要两个参数，分别为元素对的首元素和尾元素。

```
template<class T1,class T2>
pair<T1,T2>make_pair(const T1 &x,const T2&y)
{
    return(pair<T1,T2>(x,y));
}
```

pair

- **pair: 保存两个元素的一个struct，位于std**

```
namespace std{
    template <class T1, class T2>;
  struct pair{
        T1 first;
        T2 second;
        … //other functions
  };
}
pair<T1, T2> make_pair(T1, T2);
```