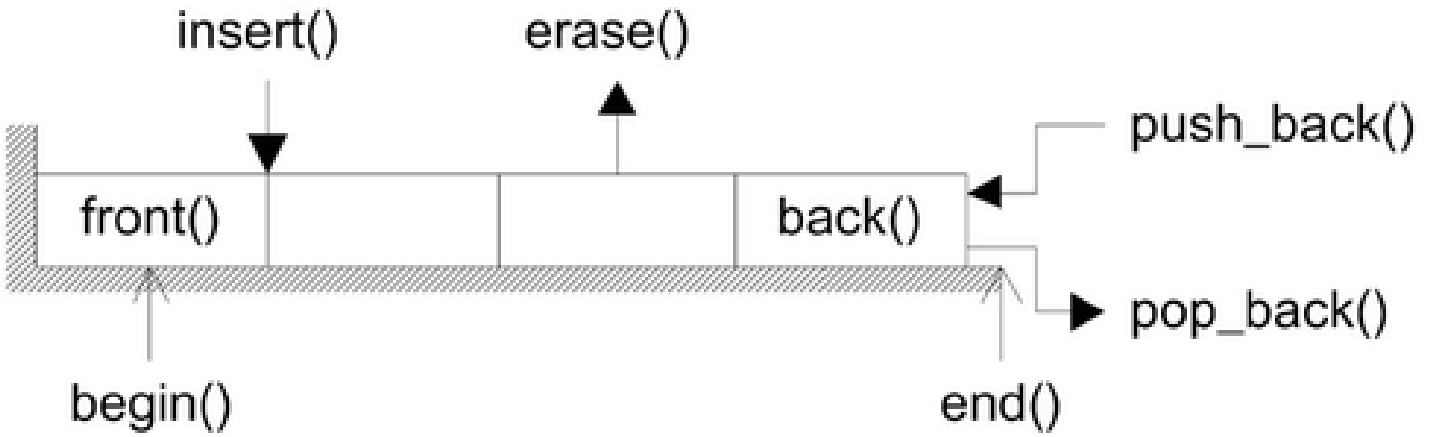
1、序列型容器概览

vector（向量）

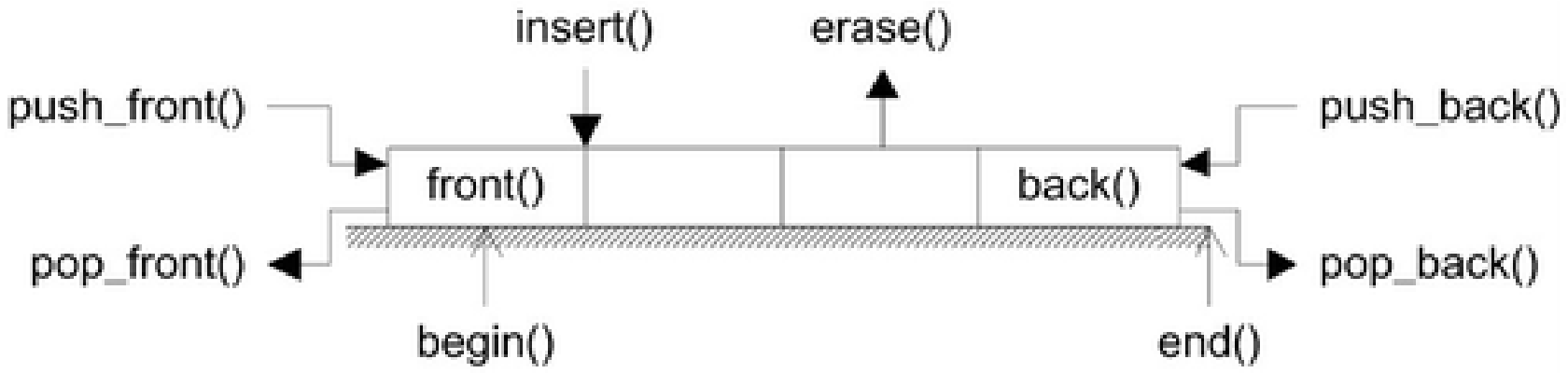
- 定义在头文件 <vector>
- 实际上就是个动态数组。随机存取任何元素都能在常数时间完成。在尾端增删元素具有较佳的性能。



向量（vector）

deque（双端队列）

- 定义于头文件 <deque>
- 也是个动态数组，随机存取任何元素都能在常数时间完成(但性能次于vector)。在两端增删元素具有较佳的性能。

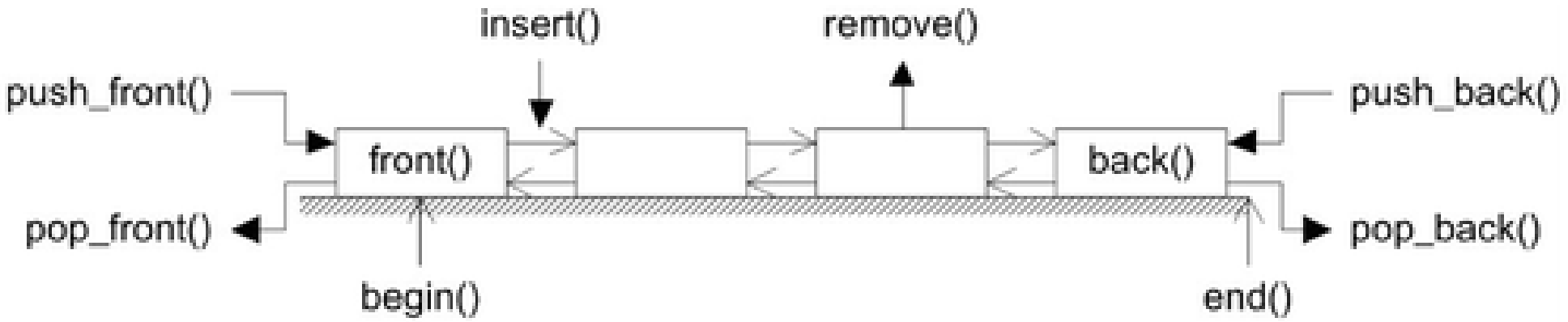


双端队列（deque）

list（双向链表）

- 定义于头文件 <list>
- 任意位置插入和删除元素的效率都很高
- 不支持随机存取
- 每个元素还有指针占用额外空间

在序列容器中，
元素的插入位置
同元素的值无关



列表（list）

序列容器初始化

- 默认构造函数初始化

```
vector<int> vec;
list<string> list1;
deque<float> deq;
```

- 拷贝构造函数初始化

```
vector<int> vec1;
vector<int> vec2(vec1);
list<string> list1;
list<string> list2(list1);
deque<float> deq1;
deque<float> deq2(deq1);
```

- 创建有长度为10的容器

```
vector<string> vec(10);
list<int> list1(10);
deque<string> deq(10);
```

- 创建有10个初值的容器

```
vector<string> vec(10,"hi");
list<int> list1(10, 1);
deque<string> deq(10, "hi");
```

序列容器——添加元素

c.push_back(t)
在容器c的尾部添加值为t的元素。返回void类型

c.push_front(t)
在容器c的前端添加值为t的元素。返回void类型，只适用于list和deque

c.insert(p,t)
在迭代器p所指向的元素前面元素t。返回指向新添加元素的迭代器

c.insert(p,n,t)
在迭代器p所指向的元素前面插入n个值为t的新元素，返回void类型

c.insert(p,b,e)
在迭代器p所指向的元素前面插入迭代器b和e标记的范围内的元素。返回void类型

序列容器——访问元素

c.back()
返回容器c的最后一个元素的引用

c.front()
返回容器c的第一个元素的引用

c[n]
返回下标为n的元素的引用（0<=n<c.size()），只适用于vector和deque容器

c.at[n]
返回下标为n的元素的引用（0<=n<c.size()），只适用于vector和deque容器

序列容器——删除元素

c.pop_back()
删除容器c的最后一个元素

c.pop_front()
删除容器c的第一个元素，只适用于deque和list容器

c.erase(p)
删除迭代器p指向的容器中的元素

c.erase(b,e)
删除迭代器b和e所标记范围内的元素

c.clear()
删除容器中所有的元素

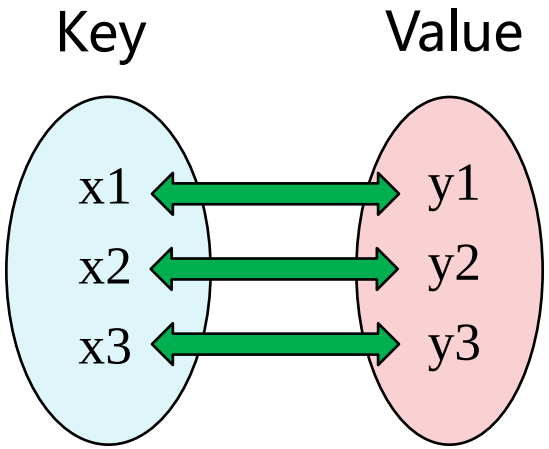
序列容器的选用

- 如果程序要求随机访问元素，则应用vector或者deque容器
- 如果程序必须在容器中间位置插入或删除元素，则应采用list容器
- 如果程序不是在容器的中间位置，而是在容器的首部或尾部插入或删除元素，则应采用deque容器

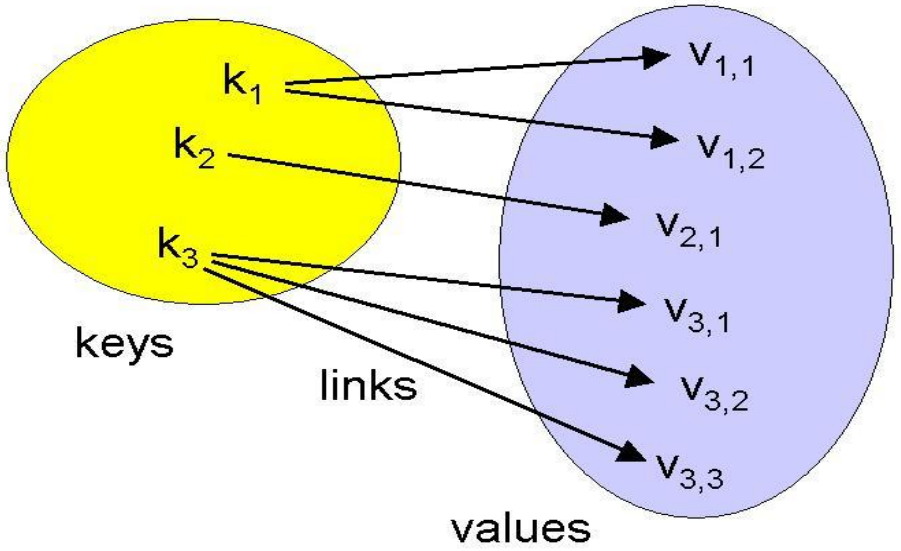
2、关联型容器概览

关联容器的特征

STL提供了4个关联容器，包括：map（映射）、multimap（多重映射）、set（集合）、multiset(多重集合)



map（映射）

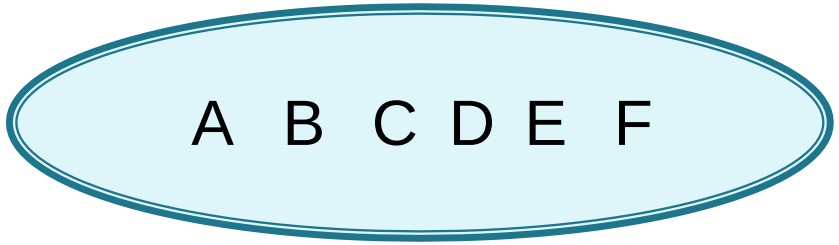


multimap（多重映射）

map、multimap的元素由（key，value）二元组构成，其中键必须是唯一的

关联容器的特征

- set 、multiset 相当于只有键（key），没有对应值（value）的 map 和 mulitimap
- set 支持通过键实现的快速读取，元素唯一



- multiset支持同一个键多次出现的set类型

关联容器与序列容器的差别

- 关联容器是通过键(key)存储和读取元素
- 顺序容器则通过元素在容器中的位置顺序存储和访问元素。

map和set的底层机制都是通过一种称为“红黑树”的数据结构存取数据，这使得它们的数据存取效率相当高

注：“红黑树”是一种常见的数据结构，感兴趣的同学可查看数据结构相关书籍

3、map容器初步

pair类型

- pair 类定义在 <utility> 头文件中。pair 是一个类模板，它将两个值组织在一起，这两个值的类型可不同。可以通过 first 和 second 公共数据成员来访问这两个值
- pair对象常常作为元素被添加到map中
- pair对象的定义：

```
pair<int, string> mypair(5 , "Jack");  //调用构造函数
pair<int, string> otherPair ;   // 直接赋值
otherPair.first = 6;
otherPair.second = “Mike";
```

- 函数模板 make_pair( ) 能从两个变量构造一个 pair

```
pair<int, int > aPair = make_pair( 5, 10 ) ;
```

map创建及添加元素

- map 类定义在 <map> 头文件中
- 创建map对象：

```
map<int, string>  StuInfo;
```

这就定义了一个用int作为键, 相关联string为值的map

- 插入pair对象：

```
pair<int, string> mypair(1, "Tom");
StuInfo.insert(mypair);
StuInfo.insert(pair<int, string>(5, "Jack"));
```

map中使用运算符[ ]

- 用[ ]操作符修改元素的值（键不可修改）

```
StuInfo[1] = "Jim";
```

因为键为 1 的元素存在，因此修改元素

- 用[ ]操作符添加元素

```
StuInfo[2] = "Lily";
```

先查找主键为2的项，没找到，因此添加这个键为 2 的项

- 用[ ]取得元素的值

```
cout<<StuInfo[5] ;    // 输出键 5 对应的值
```

在map中查找元素

- 用find()查找map中是否包含某个关键字

```
int target = 3;
map<int,string>::iterator it;
it = StuInfo.find(target);     //查找关键字target
if( it == StuInfo.end() ){
    cout<<"not existed!"
}else{
    cout<<"find it!"<<endl;
}
```

若查找成功则返回目标项的迭代器，否则返回StuInfo.end() 迭代器

在map中删除元素

- 通过erase()函数按照关键字删除

```
//删掉关键字"1"对应的条目
int r = StuInfo.erase(1);
```

若删除成功，返回 1 ，否则返回 0

- 用clear()清空map

```
StuInfo.clear();
```

再论迭代器

- STL 中的迭代器按功能由弱到强分为5种：
  1. 输入：Input iterators 提供对数据的只读访问。
  1. 输出：Output iterators 提供对数据的只写访问
  2. 正向：Forward iterators 提供读写操作，并能一次一个地向前推进迭代器。
  3. 双向：Bidirectional iterators提供读写操作，并能一次一个地向前和向后移动。
  4. 随机访问：Random access iterators提供读写操作，并能在数据中随机移动。

- 编号大的迭代器拥有编号小的迭代器的所有功能，能当作编号小的迭代器使用。

不同迭代器所能进行的操作

- 所有迭代器： ++p, p++
- 输入迭代器： *p, p = p1, p==p1 , p!=p1
- 输出迭代器： *p, p = p1
- 正向迭代器： 上面全部
- 双向迭代器： 上面全部，--p, p--,
- 随机访问迭代器： 上面全部，以及：
  - p+= i, p -= i,
  - p+ i　　返回指向 p 后面的第i个元素的迭代器
  - p – i　　返回指向 p 前面的第i个元素的迭代器
  - p < p1, p <= p1, p > p1, p>= p1

容器所支持的迭代器类别

容器	迭代器类别
vector	随机
deque	随机
list	双向
set/multiset	双向
map/multimap	双向
stack	不支持迭代器
queue	不支持迭代器

关联容器支持双向迭代器，它支持：

*、++、--、=、==、!=

不支持 <、<=、>=、>

map中迭代器的使用

下面迭代器中”<”使用错误：

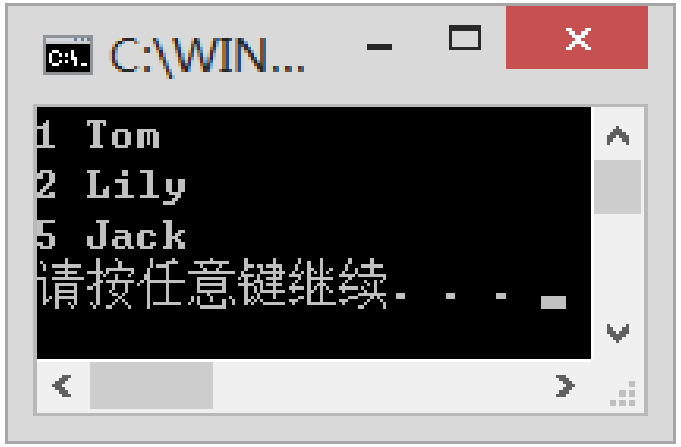
```
map<int,string> m;
map<int,string>::iterator it;
for(it = m.begin();it < m.end(); it++)
{  *****  }
```

下面是map迭代器正确的用法：

```
map<int,string> m;
map<int,string>::iterator it;
for(it = m.begin();it != m.end(); it++)
{  *****  }
```

map中使用迭代器(例)

```
#include <iostream>
#include <string>
#include <utility>
#include <map>
using namespace std;
int main () {
    map<int,string>  StuInfo;
    StuInfo.insert(pair<int, string>(1, "Tom"));
    StuInfo.insert(pair<int, string>(5, "Jack"));
    StuInfo[2]="Lily";
    map<int,string>::iterator it;
    for(it = StuInfo.begin();it != StuInfo.end(); it++)
        cout<<(*it).first<<" "<<(*it).second<<endl;
    return 0;
}
```



4、容器适配器概览

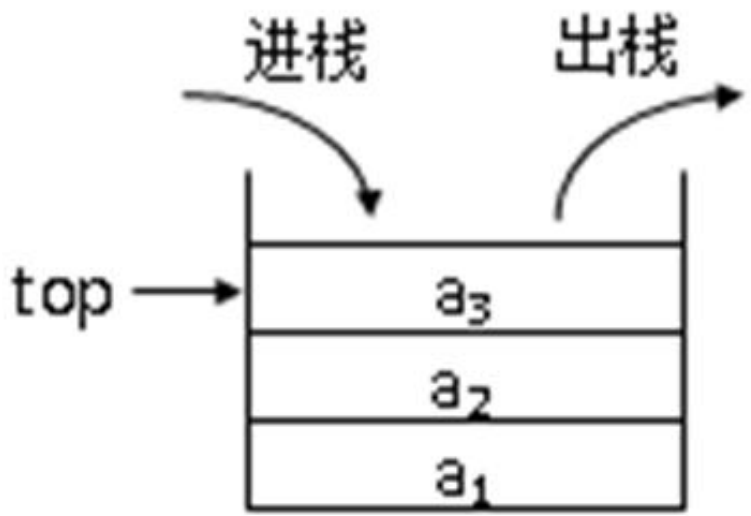
- 容器适配器将其他容器加以包装、改造，变成新的容器。实质上是一种受限容器

- 典型容器适配器：

  stack（栈）

  queue（队列）

stack-堆栈



- 栈是限制在结构的一端进行插入和删除操作
- 允许进行插入和删除操作的一端称为栈顶，另一端称为栈底

stack–堆栈

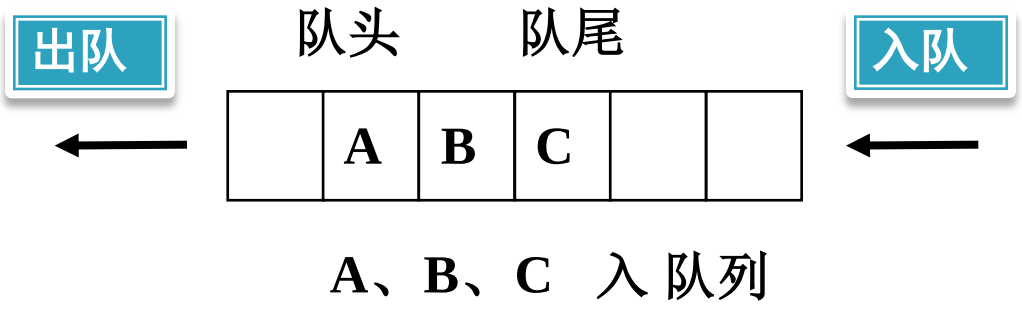
- 编程时加入下列语句：
  #include<stack>

- 栈的常用函数有：

push(elem)	将元素elem入栈
pop()	栈顶元素出栈
top()	求栈顶元素
empty()	判断栈是否空
size()	求栈内元素个数

queue-队列

只能在一端进行插入、在另一端进行删除操作的线性结构



队列示意图

- 加入下列语句：

#include<queue>

- 队列的常用函数有：

push()	入队
pop()	出队
front()	读取队首元素
back()	读取队尾元素
empty()	判断队列是否空
size()	求队列长度

堆栈示例

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
    stack<int> s;          //定义栈 s
    s.push(1);    s.push(2);    s.push(3);    s.push(9); //入栈
    cout<<"栈顶元素："<<s.top()<<endl;    //读栈顶元素
    cout<<"元素数量："<<s.size()<<endl;   //返回元素个数
    cout<<"出栈过程：";
    while(s.empty()!=true)     //栈非空
    {    cout<<s.top()<<" ";   //读栈顶元素
         s.pop();              //出栈，删除栈顶元素
    }
    return 0;
}
```

```
栈顶元素：9
元素数量：4
出栈过程：9 3 2 1
```