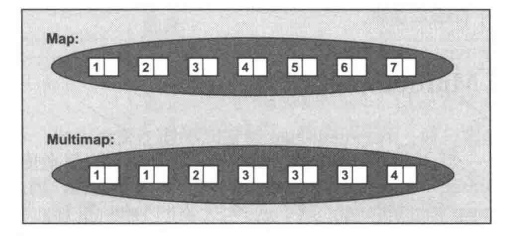
map和multimap将key/value pair当作元素进行管理，根据key的排序准则自动为元素排序。



map和multimap的排序准则与set相同。

对于map和multimap而言，key和value都必须是copyable或movable。根据指定的排序准则key必须是comparable。

对于multimap，等价的key的顺序是随机且固定的。

map和multimap内部的实现结构通常是红黑树，与set和multiset相同。

map和multimap不能直接修改key值，只能移除某个key后重新添加新的key/value。

如果需要替换key：

例：程序stl\_test49

namespace MyLib {

// 替换map容器的key

template<typename Cont>

inline bool ReplaceKey(Cont& c,

const typename Cont::*key\_type*& old\_key,

const typename Cont::*key\_type*& new\_key)

{

typename Cont::*iterator* pos;

pos = c.*find*(old\_key);

if (pos != c.*end*())

{

c.*insert*(typename Cont::*value\_type*(new\_key, pos->*second*));

c.*erase*(pos);

return true;

}

else

{

return false;

}

}

}

int *main*(int argc, char\* argv[])

{

*map*<*string*, int> c{ { "old\_key", 8 }, {"hello", 9} };

*cout* << "original: " << *endl*;

for (auto& elem : c)

{

*cout* << elem.*first* << " : " << elem.*second* << *endl*;

}

MyLib::ReplaceKey(c, "old\_key", "new\_key");

*cout* << *endl*;

*cout* << "replace key" << *endl*;

for (auto& elem : c)

{

*cout* << elem.*first* << " : " << elem.*second* << *endl*;

}

return 0;

}

输出为：

original:

hello : 9

old\_key : 8

replace key:

hello : 9

new\_key : 8

map容器支持[]，而multimap不支持。

map的[]使用需要谨慎：

map<string, float> coll1

coll1[“oto”] = 9;

上面的这个[]访问，如果key值oto存在，则修改其value为9。但如果oto不存在，则会插入一个key为oto，值为9的新元素，而这到底是不是想要的结果却不得而知。

map和multimap上使用算法和lambda

例：程序stl\_test49

*map*<*string*, double> coll{ { "tim", 9.9 }, {"struppi", 11.77} };

*for\_each*(coll.*begin*(), coll.*end*(),

[](*pair*<const *string*, double>& elem) {

elem.*second* \*= elem.*second*;

});

*for\_each*(coll.*cbegin*(), coll.*cend*(),

[](const *map*<*string*, double>::*value\_type*& elem) {

*cout* << elem.*first* << " : " << elem.*second* << *endl*;

});

输出为：

struppi : 138.533

tim : 98.01

map当作管理数组：

例：程序stl\_test49

typedef *map*<*string*, float> StringFloatMap;

StringFloatMap stocks;

stocks["BASF"] = 369.5f;

stocks["VW"] = 413.5f;

stocks["Daimler"] = 819.0f;

stocks["BMW"] = 834.0f;

stocks["Siemens"] = 842.2f;

StringFloatMap::*iterator* pos;

*cout* << *left*; // 输出左对齐

for (pos = stocks.*begin*(); pos != stocks.*end*(); ++pos)

{

*cout* << "stock: " << *setw*(12) << pos->*first*

<< "price: " << pos->*second* << *endl*;

}

*cout* << *endl*;

for (pos = stocks.*begin*(); pos != stocks.*end*(); ++pos)

{

pos->*second* \*= 2;

}

for (pos = stocks.*begin*(); pos != stocks.*end*(); ++pos)

{

*cout* << "stock: " << *setw*(12) << pos->*first*

<< "price: " << pos->*second* << *endl*;

}

*cout* << *endl*;

// 替换key

stocks["Volkswagen"] = stocks["VW"];

stocks.*erase*("VW");

for (pos = stocks.*begin*(); pos != stocks.*end*(); ++pos)

{

*cout* << "stock: " << *setw*(12) << pos->*first*

<< "price: " << pos->*second* << *endl*;

}

输出为：

stock: BASF price: 369.5

stock: BMW price: 834

stock: Daimler price: 819

stock: Siemens price: 842.2

stock: VW price: 413.5

stock: BASF price: 739

stock: BMW price: 1668

stock: Daimler price: 1638

stock: Siemens price: 1684.4

stock: VW price: 827

stock: BASF price: 739

stock: BMW price: 1668

stock: Daimler price: 1638

stock: Siemens price: 1684.4

stock: Volkswagen price: 827

将multimap当作字典（Dictionary）

例：程序stl\_test49

*multimap*<*string*, *string*> dictionary;

dictionary.*insert*({ { "day", "Tag" }, { "strange", "fremd" },

{ "car", "Auto" }, { "smart", "elegant" }, { "trait", "Merkmal" }, {"strange", "seltsam"},

{ "smart", "raffiniert" }, { "smart", "klug" }, {"clever", "raffiniert"} });

*cout*.*setf*(*ios*::*left*, *ios*::*adjustfield*);

for (const auto& elem : dictionary)

{

*cout* << ' ' << *setw*(10) << elem.*first*

<< elem.*second* << *endl*;

}

*cout* << *endl*;

*cout* << "smart: " << *endl*;

// 所有key为smart的元素的value

for (auto p = dictionary.*lower\_bound*("smart");

p != dictionary.*upper\_bound*("smart"); ++p)

{

*cout* << " " << p->*second* << *endl*;

}

输出为：

car Auto

clever raffiniert

day Tag

smart elegant

smart raffiniert

smart klug

strange fremd

strange seltsam

trait Merkmal

smart:

elegant

raffiniert

klug

map综合应用，运行期指定排序规则

例：程序stl\_test50

// 运行期定义排序准则

class RuntimeStringCompare

{

public:

enum compare\_mode { normal, nocase };

public:

RuntimeStringCompare() : mode\_(normal)

{

}

RuntimeStringCompare(compare\_mode mode) : mode\_(mode)

{}

// 函数对象，需要重载()运算符

bool operator() (const *string*& s1, const *string*& s2)

{

if (mode\_ == normal)

{

return s1 < s2;

}

else

{

// 忽略大小写，按字典序比较

return *lexicographical\_compare*(s1.*begin*(), s1.*end*(),

s2.*begin*(), s2.*end*(),

nocase\_compare);

}

}

private:

static bool nocase\_compare(char c1, char c2)

{

return *toupper*(c1) < *toupper*(c2);

}

private:

const compare\_mode mode\_;

};

typedef *map*<*string*, *string*, RuntimeStringCompare> StringMap;

void FillAndPrint(StringMap& coll);

int *main*(int argc, char\* argv[])

{

StringMap coll1;

FillAndPrint(coll1);

*cout* << "ingore case: " << *endl*;

RuntimeStringCompare ignorecase(RuntimeStringCompare::nocase);

// 在构造函数参数中定义排序准则

// 忽略大小写后，coll2将会少一个元素

StringMap coll2(ignorecase);

FillAndPrint(coll2);

return 0;

}

void FillAndPrint(StringMap& coll)

{

coll["Deutschland"] = "Germany";

coll["deutsch"] = "German";

coll["Haken"] = "snag";

coll["arbeiten"] = "work";

coll["Hund"] = "dog";

coll["gehen"] = "go";

coll["Unternehmen"] = "enterprise";

coll["unternehmen"] = "undertake";

coll["gehen"] = "walk";

coll["Bestatter"] = "undertaker";

*cout*.*setf*(*ios*::*left*, *ios*::*adjustfield*);

for (const auto& elem : coll)

{

*cout* << *setw*(15) << elem.*first* <<

" " << elem.*second* << *endl*;

}

*cout* << *endl*;

}

输出：

Bestatter undertaker

Deutschland Germany

Haken snag

Hund dog

Unternehmen enterprise

arbeiten work

deutsch German

gehen walk

unternehmen undertake

ingore case:

arbeiten work

Bestatter undertaker

deutsch German

Deutschland Germany

gehen walk

Haken snag

Hund dog

Unternehmen undertake

在忽略大小写后，map容器coll2少了一个key为unternehmen的元素。