STL容器使用

# stack

c++ stl栈stack的头文件为:

#include <stack>

c++ stl栈stack的成员函数介绍

操作 比较和分配堆栈

empty() 堆栈为空则返回真

pop() 移除栈顶元素

push() 在栈顶增加元素

size() 返回栈中元素数目

top() 返回栈顶元素

# queue

queue模板类的定义在<queue>头文件中。

与stack模板类很相似，queue模板类也需要两个模板参数，一个是元素类型，一个容器类型，元素类型是必要的，容器类型是可选的，默认为deque类型。

定义queue对象的示例代码如下：

queue<int> q1;

queue<double> q2;

queue的基本操作有：

入队，如例：q.push(x); 将x接到队列的末端。

出队，如例：q.pop(); 弹出队列的第一个元素，注意，并不会返回被弹出元素的值。

访问队首元素，如例：q.front()，即最早被压入队列的元素。

访问队尾元素，如例：q.back()，即最后被压入队列的元素。

判断队列空，如例：q.empty()，当队列空时，返回true。

访问队列中的元素个数，如例：q.size()

（push\_back, pop\_front）

# vector

头文件

#include <vector>

声明及初始化

vector<int> vec; //声明一个int型向量

vector<int> vec(5); //声明一个初始大小为5的int向量

vector<int> vec(10, 1); //声明一个初始大小为10且值都是1的向量

vector<int> vec(tmp); //声明并用tmp向量初始化vec向量

容量

vec.size(); //大小

vec.capacity(); //真实大小

vec.max\_size(); //最大容量

vec.resize(); //更改大小

vec.empty(); //判空

修改

vec.push\_back(); //末尾添加元素

vec.insert(); //任意位置插入元素

vec.pop\_back(); //末尾删除元素

vec.erase(); //任意位置删除元素

vec.swap(); //交换两个元素

vec.clear(); //清空元素

5.元素的访问

vec[1]; //下标访问,并不会检查是否越界

vec.at(1); //at方法访问,以上两者的区别就是at会检查是否越界

vec.front(); //访问第一个元素

vec.back(); //访问最后一个元素

int\* p = vec.data(); //返回一个指针,可行的原因在于vector在内存中就是一个连续存储的数组，所以可以返回一个指针指向这个数组。这是是C++11的特性。

6.算法

遍历

vector<int>::iterator it;

for (it = vec.begin(); it != vec.end(); it++)

cout << \*it << endl;

//或者

for (size\_t i = 0; i < vec.size(); i++) {

cout << vec.at(i) << endl;

}

排序

sort(vec.begin(), vec.end()); //采用的是从小到大的排序

//如果想从大到小排序，可以采用下面的反转函数，也可以采用下面方法:

bool Comp(const int& a, const int& b) {

return a > b;

}

sort(vec.begin(), vec.end(), Comp);

翻转

#include <algorithm>

reverse(vec.begin(), vec.end());

# list

双向链表。只支持双向顺序访问。在list中任何位置进行插入、删除操作速度都很快。

适用于少量读写，大量插入，删除的情况。

#include <list>

声明及初始化

list<int> l;

容量

l.empty(); //判空

元素的访问

l.front(); // 取队头元素

l.back(); // 取队尾元素

修改

l.push\_back(); // 尾后压入元素

l.push\_front(); // 队头压入元素

l.pop\_back(); // 尾后弹出一个元素

l.pop\_front(); // 队头弹出一个元素

l.insert(l.begin(), 88); // 某个位置插入元素(性能好)

l.remove(2); // 删除某个元素(和所给值相同的都删除)

l.erase(--l.end()); // 删除某个位置的元素(性能好)

算法

遍历

list<int>::iterator it;

for(it = l.begin(); it!=l.end(); it++)

{

cout<<\*it<<endl;

}

排序

l.sort(); //注意与vector的排序使用方法不一样

翻转

l.reverse(); // 倒置所有元素

forward\_list

单向链表。只支持单向顺序访问。在链表的任何位置进行插入、删除操作都很快。

用法总结

与list用法基本一致，用的也不多，先稍微总结下，没有认真总结。

头文件

声明及初始化

forward\_list<int> fl = {1, 2, 3, 4, 5, 6, 7, 8, 9};

容量

元素的访问

auto prev = fl.before\_begin(); // 表示fl的"首前元素"

auto curr = fl.begin(); // 表示fl的第一个元素

修改

fl.push\_front(0); // 压入元素，该容器没有push\_back方法

算法

遍历

排序

fl.sort();

翻转

# deque

双端队列。支持快速随机访问。在头尾位置插入、删除速度很快。

deque折中了vector和list， 如果你需要随机存取又关心数据的插入和删除，那么可以选择deque。

#include <deque>

声明及初始化

deque<int> d1;

容量

dl.empty() //判空

元素的访问

d1.front(); // 取队头元素

d1.back(); // 取队尾元素

修改

d1.push\_back(); // 尾后压入元素

d1.push\_front(4); // 队头压入元素

d1.pop\_back(); // 尾后弹出一个元素

d1.pop\_front(); // 队头弹出一个元素

算法

遍历

for(it = dl.begin(); it!=dl.end(); it++) //迭代输出

{

cout<<\*it<<endl;

}

排序

sort(dl.begin(), dl.end());

翻转

reverse(dl.begin(), dl.end());

# string

与vector相似的容器，但专门用于保存字符。随机访问快，在尾部插入删除快。

string用于和字符串操作有关的一些情况，也是实际开发中应用最多的。

用法总结

头文件

#include <string>

声明及初始化

string str1 = "Hello Ace"; // string的几种构造方法

string str2("Hello World");

string str3(str1, 6); // 从str1下标6开始构造， str3 -> Ace

容量

元素的访问

修改

string str4 = str2.substr(0, 5); // 求子串： str4 -> Hello

string str5 = str2.substr(6); // 求子串： str5 -> World

string str6 = str2.substr(6, 11); // 求子串： str6 -> World

string str8 = str2.replace(6, 5, "Game"); // 替换：str8 -> Hello Game 从位置6开始，删除5个字符，并替换成"Game"

string str9 = str2.append(", Hello Beauty");// 追加字符串： str9 -> Hello World, Hello Beauty

auto pos1 = str1.find("Ace"); // 查找字符串:pos1 -> 6 ,返回第一次出现字符串的位置，如果没找着，则返回npos

int res = str1.compare("Hello, Ace"); // 比较字符串： res -> -1, 根据str1是等于、大于还是小于参数指定的字符串， 返回0、整数或者负数

关联容器

关联容器支持高效的关键字查询和访问。标准库一共定义了8个关联容器，最主要的类型是map和set。8个容器中，每个容器：

是一个map或者是一个set。map保存关键字-值对；set只保存关键字。

要求关键字唯一或者不要求。

保持关键字有序或者不保证有序。

以下是八个常用的容器类：

set / unordered\_set

map / unordered\_map

multiset / unordered\_multiset

multimap / unordered\_multimap

从名字中就可以看出是否唯一、是否有序，允许重复关键字的容器名字都包含“multi”，不是有序容器的集合都包含“unordered”。

所以set 和map都是有序且无重复元素的。

# set

头文件

#include <set>

声明及初始化

set<int> s;

容量

empty() 　　　 //判断set容器是否为空

size() 　　　　//返回当前set容器中的元素个数

max\_size() 　 //返回set容器可能包含的元素最大个数

元素的访问

begin() 　　 //返回set容器的第一个元素

end() 　　　　 //返回set容器的最后一个元素

修改

erase(iterator) //删除定位器iterator指向的值

erase(first,second) //删除定位器first和second之间的值

erase(key\_value) //删除键值key\_value的值

insert(key\_value) //将key\_value插入到set中

insert(first,second) //将定位器first到second之间的元素插入到set中

6.算法

遍历

set<int>::iterator it; //声明迭代器

for(it = s.begin(); it!=s.end(); it++) //迭代输出

{

cout<<\*it<<endl;

}

排序

set本身就是有序的。

翻转

reverse(s.begin(), s.end()); //翻转

# map

头文件

#include <map>

声明及初始化

map<string,int> my\_Map;

容量

my\_Map.size() //返回元素数目

my\_Map.empty() //判断是否为空

my\_Map.clear() //清空所有元素

元素的访问

int i = my\_Map["a"];

MY\_MAP::iterator my\_Itr;

my\_Itr.find("b");

int j = my\_Itr->second; //first返回键，second返回值

5.修改

//插入数据

my\_Map["a"]； //只插入键，对应的值默认为0

my\_Map["a"] = 1; //同时插入键和值

my\_Map.insert(map<string,int>::value\_type("b",2));

my\_Map.insert(pair<string,int>("c",3));

my\_Map.insert(make\_pair<string,int>("d",4));

//删除数据

my\_Map.erase(my\_Itr);

MY\_MAP::iterator my\_Itr;

my\_Map.erase("c");

6.算法

遍历

map<string,int>::iterator it; //声明迭代器

for(it = p.begin(); it!=p.end(); it++) //迭代输出

{

cout<<it->first<<it->second<<endl;

}

排序

map本身就是有序的

代码举例

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v; // 定义一个vector容器

v.push\_back(1); // 向容器中添加3个元素

v.push\_back(2);

v.push\_back(3);

// 遍历向量的元素

vector<int>::iterator b = v.begin(); // 指向容器的第一个元素

vector<int>::iterator e = v.end(); // 指向容器尾元素的下一个位置

// C++11新标准的写法, auto关键字为类型推断，由编译器自动完成

// auto b = v.begin();

// auto e = v.end();

for (vector<int>::iterator iter = b; iter != e; ++iter)

{

cout << \*iter << endl;

}

return 0;

}

要点

迭代器最常用到的就是begin和end成员。其中begin成员负责返回指向第一个元素。end成员则负责返回指向容器的“尾元素的下一个位置。要特别注意end成员不是指向尾元素，而是指向尾元素的下一个位置！

表达式语句 ++iter。 建议使用前置++而非后置++。 在迭代器中，前置++的效率高于后置++。

初始化语句：vector<int>::iterator iter = b; 如果你的环境支撑C++11标准，那么强烈建议你写成auto iter = b;，即使用类型自动推断关键字auto。使用auto使程序更为简洁，也不会出错，由编译器自动推断。

迭代器的运算符

\*iter： 返回迭代器iter所指元素的引用

iter->mem: 解引用iter并获取该元素的名为mem的成员，等价于(\*item).mem

++iter: 另iter指向容器的下一个元素

--iter: 另iter指向元素的前一个元素

iter1 == iter2：判断两个迭代器是否相等

iter1 != iter2: 判断两个迭代器是否不相等

算法

算法模式

大多数的算法具有如下4种形式之一：

alg(beg, end, other args);

alg(beg, end, dest, other args);

alg(beg, end, beg2, other args);

alg(beg, end, beg2, end2, other args);

其中alg是算法的名字,beg和end表示算法所操作的输入范围。dest表示指定目的位置,beg2和end2表示接受第二个范围。

常用算法

# find()

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

int val = 5;

int arr[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

vector<int> vec = { 11, 22, 33, 44, 55, 66, 77, 88, 99 };

// 查找元素的范围是第2个元素到第8个元素，支持内置数组

// 如果找到想要的元素，则返回结果指向它

auto result = find(arr + 1, arr + 7, val);

cout << \*result << endl; // 输出结果为 5,如果没找到返回7，想一下为什么

int val2 = 100;

// 没有找到这个值，返回vec.cend()

auto res = find(vec.begin(), vec.end(), val2);

if (res == vec.cend())

cout << "没找到元素!" << endl;

else

cout << \*res << endl;

return 0;

}

# sort()

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

using namespace std;

struct CoureSocre

{

string name; // 姓名

int math; // 数学成绩

int chinese; // 语文成绩

int total; // 总成绩

CoureSocre(string \_name, int \_math, int \_chinese)

{

name = \_name;

math = \_math;

chinese = \_chinese;

total = math + chinese;

}

};

bool myCmp(CoureSocre c1, CoureSocre c2)

{

// 如果总成绩相同

if (c1.total == c2.total)

{

return c1.math >= c2.math;

}

return c1.total > c2.total;

}

int main()

{

// 初始化5个学生的程序

CoureSocre c1("Ace", 90, 95);

CoureSocre c2("Shawna", 99, 100);

CoureSocre c3("Kelly", 100, 99);

CoureSocre c4("Jordan", 88, 90);

CoureSocre c5("Kobe", 90, 88);

// 加入容器

vector<CoureSocre> vecScoreList = { c1, c2, c3, c4, c5 };

// 调用sort算法进行排序

sort(vecScoreList.begin(), vecScoreList.end(), myCmp);

cout << "学生的成绩排名为:" << endl;

for each (CoureSocre c in vecScoreList) // 使用for each 算法进行遍历

{

cout << "姓名:" << c.name << "\t总成绩:" << c.total << "\t数学:" << c.math << "\t语文:" << c.chinese << endl;

}

return 0;

}