C/C++学习记录

STL(Standard Template Library)标准模板库

STL六大组件

容器、算法、迭代器、仿函数、适配器、空间配置器

容器：各种数据结构，如vector、list、deque、set、map等

算法：常用算法，如sort、find、copy、for\_each等

迭代器：容器与算法之间的胶合剂

仿函数：行为类似函数，作为算法的某种策略

适配器：一种用来修饰容器或仿函数或迭代器接口的东西

空间配置器：用来配置与管理空间

容器、算法、迭代器

容器：

将应用最广泛的一些数据结构实现出来

常用的数据结构有：数组、链表、树、栈、队列、集合、映射表等。

容器可以分为序列式容器和关联式容器

C++\_study\_container.cpp

算法：（Algorithms）

有限的步骤解决逻辑或者数学问题，根据运算过程中是否改变区间内的元素内容分为质变算法和非质变算法。

迭代器：

算法要通过迭代器访问容器中的元素。类似于指针的操作

迭代器分为：输入迭代器、输出迭代器、向前迭代器、双向迭代器和随机访问迭代器。

容器

string容器

string和char\*的区别：

char\*是一个指针

string是一个类，类中封装了char\*，管理这个字符串，是一个char\*型的容器。string中封装了很多方法如：copy、delete、find、replace、insert等。string管理char\*所分配的内存，无须担心越界和取值越界等问题，由类的内部负责。

string构造函数：

string(); 空字符串

string(const char\* s); 用字符串s初始化

string(const string& str); 用一个string对象初始化另一个string对象

string(int n, char c); 用n个字符c初始化

string赋值函数

1. 直接等号赋值
2. 使用assign方法赋值

string& assign(const char\* s, int n); // 把s的前n个字符赋给字符串

string& assign(const string & s);

string& assign(int n, char c); // 用n个c赋值给字符串

string拼接

1. 直接+=拼接
2. 使用append方法拼接

string& append(const char\* s, int n); // 把s的前n个字符拼接到末尾

string& append(const string & s, int pos, int n); // 字符串s中从pos开始的n个字符拼接到末尾

string查找和替换

没有查找内容时返回-1，否则返回索引

int find(const string & str, int pos); //查找str第一次出现的位置，从pos开始

int find(const char\* s, int pos, int n); //查找s中前n个字符第一次出现的位置，从pos开始

// 当使用rfind时表示反序查找

string& replace(int pos, int n, const string & str); // 替换从pos开始的n个字符为str

string比较

字符串按照ASCII码进行对比

相等返回0，大于返回1，小于返回-1

int compare(const string& s);

int compare(const char\* s);

一般用来判断字符串是否相等。

string字符存取

int str\_size = str.size();

获取str长度

char& operator[(int n)];

char& at(int n);

string插入和删除

string& insert(int pos, const char\* s);

string& insert(int pos, const string& str);

string& insert(int pos, int n, char c);

// 在pos插入字符串，或n个c字符

string& insert(int pos, int n);

// 删除从pos开始的n个字符

string截取

string substr(int pos, int n);

// 返回从pos开始的n个字符组成的字符串

vector容器

vector与数组非常相似，也称为单端数组，不同在于数组为静态空间，而vector可以动态扩展，当当前空间不够时，开辟一块更大的空间将原空间中的数据拷贝到新空间中。

常用迭代器有

vector<int>::iterator v\_0 = v.begin();

vector<int>::iterator v\_end = v.end();

vector<int>::iterator v\_r0 = v.rbegin();

vector<int>::iterator v\_rend = v.rbegin();

构造函数

vector<int> v1;

// 默认构造函数

vector<int> v2(v1.begin(), v1.end());

// 区间构造，v2 = v1[v1.begin(), v1.end())，当括号中只有v1时表示拷贝构造

vector<int> v3(10, 100);

// 10个100组成的vector

赋值操作

1. 直接等号赋值
2. 使用assign方法赋值

v2 = v1;

v2.assign(v1.begin(), v1.end());

v2.assign(10, 100)

容量和大小

empty();

// 判断是否为空

capacity();

// 返回容器容量

size();

// 返回元素中容器个数

resize(int num, elem);

//重新指定容器长度，当长度变大是用elem填充，不需要填充时没有elem，当长度变小删除超出num的元素，resize仅改变元素数量，不改变容器容量

reserve(int len);

//容器预留len个元素长度的空间，预留位置不初始化，元素不可访问，避免多次扩展空间

插入和删除

push\_back(ele);

//尾部插入ele

pop\_back();

//删除最后一个元素

insert(const\_iterator pos, ele)

//在迭代器pos位置插入ele

insert(const\_iterator pos, int count, ele);

//在迭代器pos位置插入count个ele

erase(const\_iterator pos);

//删除迭代器pos指向元素

erase(const\_iterator start, const\_iterator end);

//删除迭代器[start, end)区间元素

clear();

//清空容器

数据存取

at(int index);

//返回索引index指向元素

operator[int index];

//返回索引index指向元素

front();

//返回第一个元素

back();

//返回最后一个元素

swap(vector v);

// 容器中的元素与容器v中的元素互换，使用swap可以缩小大容量少元素容器的大小。

vector<int>(v).swap(v);

//首先创建一个匿名对象拷贝v中的数据，在将v与匿名容器对象的元素互换，从而实现v中元素数量与容器容量匹配，从而减小内存消耗

deque容器

双端数组，可以对头端进行插入和删除操作，但是访问元素比vector慢。deque中有个中控器，维护每段缓冲区中的内容，缓冲区存放数据，中控器维护每个缓冲区的地址，使得使用deque时像一片连续空间。

迭代器

支持随机访问。

begin();

end();

构造函数

deque<int> d;

deque<int> d2(d.begin(), d.end());

//拷贝d[begin, end)的元素构造d2

deque<int> d2(int n, elem);

//构造n个elem的容器

deque<int> d2(const deque & d);

//拷贝d构造d2

赋值操作

d2 = d;

d2.assign(d.begin(), d.end());

//将d[begin, end)赋值给d2

d2.assign(int n, elem);

//将n个elem赋值给d2

**大小操作**

empty();

// 判断是否为空

size();

// 返回容器中元素个数

resize(int num, elem);

//重新指定容器长度，当长度变大是用elem填充，不需要填充时没有elem，当长度变小删除超出num的元素

插入和删除

两端操作：

d.push\_back(elem);

//尾部插入elem

d.push\_front(elem);

//头部插入elem

d.pop\_back();

//结尾删除

d.pop\_front();

//头部删除

指定位置操作：

d.insert(index, elem);

//在index插入elem

d.insert(index, n, elem);

//在index插入n个elem

d.insert(index, d2.begin(), d2.end());

//在index插入d2[begin, end)

d.clear();

//清空d

d.erase(d.begin(), d.end());

//删除d[begin, end)

d.erase(index);

**//删除d[index]**

数据存取

**at(int index);**

**//返回索引index指向元素**

**operator[int index];**

**//返回索引index指向元素**

**front();**

**//返回第一个元素**

**back();**

**//返回最后一个元素**

排序

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

stack容器(栈)

先进后出的数据结构，只有一个口用来进出数据。

构造函数：

stack<int> s;

stack<int> s1(s);

赋值：

s1 = s;

数据存取：

s.push(1);

//将1压入栈

s.pop();

//移除栈顶元素

s.top();

// 返回栈顶元素

大小操作

s.empty();

//判断栈是否为空

s.size();

//返回栈的大小

queue容器

先进先出的数据结构，队尾只能入数据，队头只能出数据，只能访问到队头(front)和队尾(back)的数据。

构造函数

queue<int> q;

queue<int> q1(q);

赋值

q1 = q

数据存取

q.push(1);

//向队尾添加元素

q.pop();

//删除队头元素

q.back();

//返回队尾元素

q.front();

//返回队头元素

大小操作

q.empty();

q.size();

list容器(链表)

将数据进行链式存储，在物理存储单元上是非连续的，数据元素的逻辑顺序通过链表中的指针链接实现的。优点：可以在任意位置快速插入或删除元素，动态存储分配，不会有内存浪费；缺点：遍历速度没有数组快，占用空间大

STL中的list是一个双向循环的链表，既有指向下一个元素的指针next也有指向前一个元素的指针prev，普通链表第一个元素的prev和最后一个元素的next指向Null，循环链表分别指向对方。

因为list不是存储在连续的内存空间，因此迭代器只能前移后移，不能跳跃访问。

构造函数

list<int> l;

list<int> l1(l.begin(), l.end());

list<int> l(10, 100);

//构造10个100的list

list<int> l1(l);

赋值

l = l1;

l.assign(l1.begin(), l1.end());

//将l1[begin, end)的元素赋值给l

l.assign(10, 100);

//将10个100赋值给l

l.swap(l1);

//l和l1的元素互换

大小操作

l.size();

l.empty();

l.resize(10);

//重新指定list长度为10，若容器变长以默认值0填充新位置，变短则删除超出容器长度的元素

l.resize(10, 8);

//重新指定list长度为10，若容器变长以设定值8填充新位置，变短则删除超出容器长度的元素

插入和删除

//两端操作

l.push\_back(1);

//尾部插入1

l.push\_front(1);

//头部插入1

l.pop\_back();

//尾部删除

l.pop\_front();

//头部删除

//指定位置操作

l.insert(l.begin(), 10);

//在begin处插入10

l.insert(l.begin(), 10, 100);

//在begin处插入10个100

l.insert(l.begin(), l1.begin(), l1.end());

//在begin处插入l1[begin，end)

l.erase(l.begin(), l.end());

//删除l[begin, end)元素

l.erase(l.begin());

//删除l[begin]元素

l.remove(10);

//删除所有等于10的元素

l.clear();

数据存取

l.front();

//返回第一个元素

l.back();

//返回最后一个元素

反转和排序

l.reverse();

//链表反序

l.sort();

//链表排序

bool ComparePersonAge(const Person p1, const Person p2)

{

return p1.m\_Age < p2.m\_Age;

}

l.sort(ComparePersonAge);

//按照年龄升序排列，当p1.m\_Age > p2.m\_Age为降序排列

set容器

所有元素在插入时会自动排序，属于关联式容器，底层使用二叉树实现，不内存在重复元素。

构造

set<Person> s1;

set<Person> s2(s1);

赋值

s2 = s1;

插入和删除

s1.insert(p1);

s1.clear();

s1.erase(s1.begin());

s1.erase(s1.begin(), s1.end());

s1.erase(p1);

// 删除s1中的p1

大小和交换

s1.size();

s1.empty();

s1.swap(s2);

查找和统计

s1.find(p1);

//查找，返回该元素的迭代器，否则返回.end()

s1.count(p1);

//统计p1的元素个数，但是set中没有重复的值，所有只会是0/1

set和multiset

multiset可以插入相同元素

pair对组

pair<int, Person> p(1, p1);

pair<int, Person> p = make\_pair(1, p1);

p.first();

//第一个数据

p.second();

//第二个数据

set容器排序

set默认由小到大进行排序，利用仿函数可以改变排序规则

class MyCompare

{

public:

bool operator() (int v1, int v2) const;

};

bool MyCompare::operator()(int v1, int v2)const

{

return v1 > v2;

}

set<int, MyCompare> s;

map、multimap容器

map中的所有元素都是pair，pair中第一个元素为key，第二个为value，所有元素都会根据value自动排序。map和multimap都是关联式容器，底层由二叉树实现。

map不允许重复的key，multimap允许重复的key。

构造和赋值

map<int, Person> m;

map<int, Person> m2(m);

m2 = m;

大小和交换

m.size();

m.empty();

m.swap(m2);

插入和删除

m.insert(pair<int, Person>(1, Person("aaa", 40, 177)));

m.clear();

m.erase(m.begin());

m.erase(m.begin(), m.end());

m.erase(1);

//按照key删除元素

查找和统计

m.find(1);

//查找key为1的元素，返回元素迭代器，不存在该元素则返回.end()

m.count(1);

//统计key为1的元素个数，因为key值不重复，只有0/1

排序

默认为value由小到大排列，可以通过仿函数改变排序方式

class MyCompare

{

public:

bool operator() (const Person& p1, const Person& p2)const;

};

bool MyCompare::operator() (const Person& p1, const Person& p2)const

{

return p1.m\_Age > p2.m\_Age;

}

map<int, Person, MyCompare> m;

函数对象

函数对象基本概念

重载函数调用操作符的类，其对象称为函数对象，函数对象使用重载的()时，行为类似函数调用，也叫仿函数。其本质为类而非函数。

函数对象可以像普通函数一样调用，可以有参数、返回值；

函数对象可以有自己的状态；

可以像参数一样传递。

谓词

返回bool类型的仿函数。

一元谓词

接收一个参数

二元谓词

接收两个参数

内建函数对象

#include<functional>

STL内建了一些函数对象，可以分为算数仿函数、关系仿函数和逻辑仿函数。这些仿函数是类，需要实例化后使用。

算数仿函数

negate<int> n;

n(50);

// 将整形变量取反

plus<int> p;

p(10, 29);

// 两个数相加

minus<int> m;

//减法

multiplies<int> mult;

//乘法

divides<int> div;

//除法

modulus<int> mod;

//取模

关系仿函数

返回布尔类型

equal\_to<int> eq;

//等于

not\_equal\_to<int> neq;

//不等于

greater<int> gre;

//大于

greater\_equal<int> gre\_eq;

//大于等于

less<int> les;

//小于

less\_equal<int> les\_eq;

//小于等于

常见用法：降序排列

vector<int> v;

v.push\_back(10);

v.push\_back(5);

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

PrintVector(v);

sort(v.begin(), v.end(), greater<int>());

PrintVector(v);

逻辑运算

返回布尔类型

logical\_and<bool> log\_a;

//逻辑与

logical\_or<bool> log\_o;

//逻辑或

logical\_not<bool> log\_n;

//逻辑非

常用算法

算法主要由<algorithm> / <functional> / <numeric>组成，其中<algorithm>是所有STL头文件中最大的，涉及比较、交换、查找、遍历、复制、修改等操作；<functional>定义了一些模板类，用来声明函数对象；<numeric>体积很小，只有几个在序列上面进行简单数学运算的模板函数。

常用遍历算法

常用遍历算法有for\_each、transform

for\_each

for\_each(v.begin(), v.end(), PrintVector);

//遍历v，并将每一个\*i送入PrintVector函数

transform

transform(v.begin(), v.end(), v1.begin(), PrintVector);

// 将v[begin，end)的内容搬运到v1，并将每一个\*i传入PrintVector函数

常用查找算法

常用查找算法有find、find\_if、adjacent\_find、binary\_search、count和count\_if

find

find(v.begin(), v.end(), val);

//在v[begin, end)查找val是否存在，若存在返回对应迭代器，否则返回v.end()

对于自定义数据类型，需要给自定义数据类型重载==号

find\_if条件查找

find(v.begin(), v.end(), func);

//将v[begin, end)中的迭代器依次传入func，如果func返回True则返回对应迭代器，若到了v.end()依然没有符合func的元素返回v.end()

adjacent\_find查找相邻重复元素

adjacent\_find(v.begin(), v.end());

// 查找v[begin, end)中相邻的重复元素，返回第一组相邻重复元素的第一个元素的迭代器。

binary\_search二分查找

查找指定元素是否存在，存在返回true否则返回false，只能用于有序序列。

binary\_search(v.begin(), v.end(), val);

// 在v[begin, end)中以二分查找方法查找val，若val存在返回true否则返回false

count统计

count(v.begin(), v.end(), val);

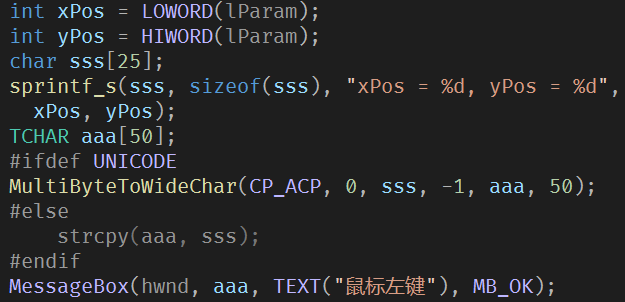
// 统计v[begin, end)中val的个数，返回int类型

count\_if条件统计

count(v.begin(), v.end(), func);

// 统计v[begin, end)中能让func返回True的元素个数，返回int类型

char类型转换为TCHAR



char\*和CString的相互转化

char\* p = “ccc”;

CString str = CString(p);

CStringA tmp；

tmp = str;

p = tmp.GetBuffer();