# vector

#include <vector>

using namespace std

vector<typename> name;

vector<vector<int> > name; // >>之间要加空格

vector数组

vector<typename> arrayname[SIZE];

一维长度固定为SIZE

通过下标访问: vi[0] ~ vi[vi.size() - 1]

迭代器:

vector<int>::iterator it = vi.begin();

vi[i]和\*(vi.begin()+i)等价, 也和\*(it+i)等价

不支持it<vi.end(), 只能写it!=vi.edn()

只有vector和string才支持vi.begin()+3这种迭代器加上整数的写法

push\_back(x)在vector最后添加一个元素x

pop\_back()删除vector的尾元素

size()返回unsigned类型的元素个数

clear()清除所有元素, 复杂度O(N)

insert(it, x)在vector的迭代器it处插入元素x, O(N)

erase(it)删除迭代器为it处的元素

erase(first, last)删除[first, last)内的所有元素

# set

set是一个内部自动有序且不含重复元素的容器.

#include <set>

using namespace std

set<typename> name;

set迭代器不支持\*(it+i)的方式访问, 通过以下方式遍历集合:

for(set<int>::iterator it = st.begin(); it != st.end(); it++){

printf("%d", \*it);

}

set内的元素自动递增排序, 自动去除重复元素.

insert(x), 将x插入set中,自动递增排序和去重, O(logN)

find(x), 返回元素x在set中的迭代器, 若不存在返回st.end(), O(logN)

earse(it), 删除迭代器it处的元素, O(1)

erase(x), 删除值为x的元素, O(logN)

erase(first, last), 删除[first, last)的元素, O(last - first)

size()获取set内元素的个数

clear() 清除set内的所有元素

# string

#include <string>

using namespace std;

可以通过下标和迭代器访问

length() 返回string的字符数

size()返回string的长度.

insert(pos, string), 在pos位置插入字符串string

insert(it, it2, it3), 在原字符串it的位置插入待插入字符串的[it2, it3)

erase(it)删除迭代器it处的元素

erase(first, last), 删除[first, last)的元素

erase(pos, len)删除位置pos开始的len个字符

clear() 清空string, O(1)

substr(pos, len)返回从pos号位开始, 长度位len的字串

string::nops, 是unsigned int类型的最大值, 作为find()函数失配时的返回值. 值为-1或者4295967295

str.find(str2)当str2是str1的子串时返回str2在str中第一次出现的位置,否则反后string::nops. O(nm)

replace(pos, len, str2)把str从pos号位开始, 长度为len的子串替换为str2.

replace(it1, it2, str2) 把str的迭代器[it, it2)范围的字串替换为str2.

# map

#include <map>

using namespace std;

map<typename1, typename2> mp;

如果时字符串到整数的映射, 必须使用string而不能使用char数组

map<string, int> mp;

map的键和值可以时STL容器, 例如可以将一个set容器映射到一个字符串

map<set<int>, string> mp;

通过下表访问: mp['c'] = 20;

通过迭代器访问

map<typename1, typename2>::iterator it;

it->first是当前映射的键, it->second是当前映射的值

find(key) 返回键为key的映射的迭代器, O(logN), 查找失败返回mp.end()

erase(it)删除迭代器为it的元素

erase(key)删除键为key的元素, O(logN)

erase(first, last), 删除[first, last)迭代器范围内的元素

size()获得map中映射的对数

clear()清空map中所有元素, O(N)

# queue

#include <queue>

using namespace std;

queue<typename> name;

push(x)将元素x进行入队

front()获得队首元素

back()获得队尾元素

pop()队首元素出队

empty()检测queue是否为空

使用front()和pop()函数前必须使用empty()判空

size()返回queue内元素的个数

# priority\_queue

# stack

#include <stack>

using namespce std;

stack<typename> name;

push(x)将x入栈

top()获得栈顶元素

pop()弹出栈顶元素

empty()检测栈是否为空

size()返回stack内元素的个数

# pair

#include <utility>

using namespce std;

pair<typename1, typename2> name;

pair<string, int> p;

pair<string, int> p("haha", 5);

p = pair<string, int>("haha", 5);

p = make\_pair("haha", 5);

pair只有两个元素, 分别是p.first, p.second

比较大小是先以first的大小作为标准,只有first相等时才去比较second的大小.

map<string ,int> mp;

mp.insert(make\_pair("heihei", 5));

mp.insert(pair<string, int>("haha", 10));

algorithm头文件常用函数

using namespce std;

max(x, y);

min(x, y);

abs(x)返回x的绝对值,x必须是整数

浮点绝对值使用math头文件下的fabs

swap(x, y) 交换x和y的值

reverse(it1, it2)将数组指针在[it1, it2)之间的元素或者容器的迭代器在[it1, it2)的元素进行反转.

next\_permutation(a, a + i)给出一个序列在全排列的下一个序列

fill(a, a + 5, 233)把a[0]~a[4]赋值为233

sort()

lower\_bound(first, last, val)用来寻找在数组或容器的[first, last)范围内第一个值**大于等于**val的元素的位置,如果是数组, 返回该位置的指针, 如果是容器, 返回该位置的迭代器.

upper\_bound(first, last, val)用来寻找在数组或容器的[first, last)范围内第一个值**大于**val的元素的位置,如果是数组, 返回该位置的指针, 如果是容器, 返回该位置的迭代器.

如果数组或容器中没有需要寻找的元素, 则lower\_bound()和upper\_bound()均返回可以插入该元素的位置的指针或迭代器.

C语言实现版本见算法笔记的二分查找这一小节

定义并初始化二维数组的方法:

m行n列

vector<vector<bool>> Pacificmark(m,vector<bool>(n,false));

# C++的lambda表达式

语法形式:

[函数对象参数] (操作符重载函数参数) mutable 或 exception 声明 -> 返回值类型 {函数体}

(1) [函数对象参数]

不能省略.

空 没有任何函数对象参数.

= 函数体内可以使用lambda所在范围内所有可见的局部变量, 按值传递.

& 函数体内可以使用lambda所在范围内所有可见的局部变量, 按值引用递.

this 函数体内可以使用lambda所在类中的成员变量.

a 将a按值传递, 但默认情况下参数a有const属性, 要修改传递进来的a, 可加mutable修饰符.

&a a按引用传递.

=, &a, &b 除a和b按引用传递外, 其它参数都是按值传递.

&, a, b 除a和b按值传递外, 其它参数都是按引用传递.

(2) (操作符重载函数参数)

可以省略

(3) mutable 或 exception 声明

可以省略. 加上 mutable 修饰符后, 可以修改传递进来的拷贝(注意是能修改拷贝,而不是值本身). exception 声明用于指定函数抛出的异常, 如抛出整数类型的异常, 可以使用 throw(int).

(4) -> 返回值类型

当返回值为void时可以省略.

(5)

变量可能被另一个在嵌套声明区域中声明的变量隐藏.

using声明

using编译指令

使用构造函数

(1) Stock food = Stock("foodname", 250, 1.25);

(2) Stock garment("Furry Mason", 50, 2.5);

(3) Stock \*pstock = new Stock("Electroshock", 18, 19.20);

(4) Stock food; // 将使用默认构造函数Stock()

如果提供了非默认的构造函数但是没有提供默认构造函数, 则以下声明出错:

Stock food;

原因是禁止创建未初始化的对象.

使用析构函数

析构函数的名称: 在类名前加上~, 如~Stock(). 析构函数没有参数.

析构函数的调用由编译器决定, 通常不在代码中显示调用析构函数:

(1) 如果是静态存储类对象, 析构函数在程序结束时被自动调用.

(2) 如果是自动存储类对象, 析构函数在程序执行完代码块时自动被调用.

(3) 如果对象是通过new创建的, 则当调用delete时, 析构函数自动被调用.

const int \* p; // p is a pointer to int const.

int const \* p; // p is a pointer to const int.

int \* const p; // p is a const pointer to int.

const int \* const p; // p is a const pointer to int const.

int const \* const p; // p is a const pointer to const int.

不能把const对象, 字面值或者需要类型转换的对象传递给普通的引用形参.

如果要对字符串进行遍历str[i], i最好定义为string::size\_type i.

使用数组作为函数的形参以下三种定义等价:

void print(const int\*);

void print(const int[]);

void print(const int[10]); // 10并不起作用

管理数组实参的规范方法是传递指向数组首元素和尾后元素的指针.

void print(const int \*beg, const \*end){}

数组引用形参:

void print(int (&arr)[10]) {} // 注意括号不能省

int \*matrix[10] // 十个指针构成的数组

int (\*matrix)[10] // 指向含有10整数的指针

C++11提供两种方法编写处理不同数量实参的函数:

(1) 所有实参类型相同, 可以传递一个名为initializer\_list的标准库类型.

(2) 实参的类型不同, 可以编写可变参数模板.

**initializer\_list形参**

initializer\_list<T> lst;

initializer\_list<T> lst{a, b, c, ...};

----------------------------------

lst2(lst) 拷贝或赋值一个initializer\_lis对象不会拷贝列表中的元素

lst2 = lst 拷贝后, 原始列表和副本共享元素

----------------------------------

lst.size() 列表中元素的数量

lst.begin() lst中首元素的指针

lst.end() lst中尾元素的下一个指针

initializer\_list对象中的元素值永远是常量, 无法改变.

如果想向initializer\_list中传递一个值的序列, 则必须把序列放在一对花括号内:

error\_msg({"xyz", var1, var2});

省略符形参

省略符形参是为了便于C++程序访问某些特殊的C代码而设置的, 这些代码使用了名为varargs的 C标准库功能.

两种形式:

void foo(parm\_list, ...);

void foo(...);

引用返回左值

调用一个返回引用的函数得到左值, 其它返回类型得到右值. 可以像使用其它左值那样来使用返回引用的函数的调用. 我们可以尾返回类型是非常量引用的函数的结果赋值.

char &getVal(string &str, sting::size\_type ix) {return str[ix];}

可以这样用

string s("a value");

getVal(s, 0) = 'A';

列表初始化返回值

C++11新标准规定, 函数可以返回花括号包围的值的列表.

vector<string> process() {

return {"str1", "str2", "str3"};

}

类型别名

一种方法是typedef, 一种方法是使用别名声明, 关键字为using.

typedef int arrT[10]; // arrT是一个类型别名, 它表示的类型是含有10个整数的数组.

using arrT = int[10]; // 同上

arrT\* func(int i); // func返回一个指向含有10个整数的数组的指针

声明一个返回数组指针的函数

数组的大小必须跟在函数名字之后, 返回数组指针的函数形式如下

type (\*func(parm\_list)) [dimension]

使用尾置返回类型

C++11规定, 任何函数的定义都能使用尾置返回, 这种形式对于返回类型比较复杂的函数最有效, 比如返回类型是数组的指针或数组的引用.

auto func(int i) -> int(\*) [10];

func返回一个指针, 该指针指向含有10个整数的数组.

decltype

如果我们知道函数返回的指针指向哪个数组, 就可以使用decltype关键字声明返回类型.

int odd[] = {1, 3, 5, 7, 9};

int even[] = {0, 2, 4, 6, 8};

decltype(odd) \*arrPtr(int i) {return (i % 2) ? &odd : &even;}

arrPtr使用关键字decltype表示它的返回类型是个指针并且该指针所指向的对象与odd类型一致.

decltype并不负责把数组类型转换为对应的指针, 所以decltype的结果是个数组, 要想表示arrPtr返回指针还必须在函数声明时加一个\*符号.

一个指向整型数组的指针 int (\*p)[10];

执行p++时, p会加上10个int空间的字节数.

引用为对象起了另外一个名字, 引用类型易怒用另外一种类型.

引用无法更换绑定的对象, 必须赋初始值.

引用并被对象, 不能定义引用的引用.

以引用作为初始值, 实际上是以与引用绑定的对象作为初始值.

变量的定义包括一个基本数据类型和一组声明符.

\*和&就是类型修饰符, 是声明符的一部分.

引用不是对象, 不能定义指向引用的指针.

定义一个对指针的引用: int \*p; int \*&r = p;

引用类型必须与其所引用的对象的类型一致, 有两种例外:

(1) 在初始化常量引用时, 允许用任意表达式作为初始值, 只要该表达式的结果能够转换成引用的类型即可.

(2)