目录

[小知识 1](#_Toc97648467)

[STL 2](#_Toc97648468)

[自定义数据类型 2](#_Toc97648469)

[二维数组，即vector 的嵌套 3](#_Toc97648470)

[string p223-222(差異點) 3](#_Toc97648471)

[vector 4](#_Toc97648472)

[#include<stack> 5](#_Toc97648473)

[#include<list> 5](#_Toc97648474)

[#deque 5](#_Toc97648475)

[*并且queue只能从队首删除元素,但是两端都能访问* 8](#_Toc97648476)

[priority\_queue 优先队列 8](#_Toc97648477)

[stak 栈 8](#_Toc97648478)

[map map<int, string> mapStudent; 9](#_Toc97648479)

[unordered\_map<string(key), int>mp 10](#_Toc97648480)

# 小知识

//局部数组未初始化 默认数值会是最小

//局部变量则不行，会提示使用了未初始化的局部变量

//创建动态数组 使用 vector<bool>arrAy(10, false);左先说多长，再说其内数值大小

0x8000 0000=-2147483648

0x7fff ffff = **2147483647**

int \*arr\_f = new int[10];//new堆区空间

按“F6”，这将使用设置中的默认参数编译和运行文件。或按“F7”，这将使用您为程序指定的参数。

//读入缓冲区

freopen()函数的原型为：

FILE \*freopen(char \*filename, char \*type, FILE \*stream);

【参数】filename为要重定向到的文件名；type为文件打开方式，请参考fopen()函数；stream为被重定向的文件流（一般是 stdin、stdout 或 stderr）

freopen("1.txt", "r", stdin);//读入

freopen("1.txt", "w", stdin);//输入输出

freopen("../../数据读取.txt","r",stdin); //输入重定向

freopen("../../数据输出.txt","w",stdout); //输出重定向

fclose(stdin);//关闭重定向输入

fclose(stdout);//关闭重定向输出

bool is = true;//->1 is = false; //->0

int n = s.length();// 数组长度

s.substr(0, 1);//返回截取数组某一段

#include <math.h>//pow(x,y);

递归 和 递推

# STL

STL六大组件

容器,算法,迭代器,仿函数,适配器(配接器),空间配置器

解决逻辑或数学上的问题叫做算法;

算法分为质变算法和非质变算法。

质变算法:指运算过程中会更改区间内的元素内容。如:拷贝，替换，删除等。

非质变算法:是指运算过程中不会更改区间内元素的内容。如查找，计数，遍历等。

1.容器:各种数据结构，如vector,list,deque,set,map等，用来存放数据

2.算法:各种常用的算法，如sort,find,copy,for\_each等

3.迭代器:扮演了容器与算法之间的胶合剂

4.仿函数:行为类似函数,可作为算法的莫种策略。

5.适配器:一种用来修饰容器或者仿函数或迭代器接口的东西

6.空间适配器:负责空间的配置与管理。

容器vector

算法 for\_each 遍历

迭代器 vector<int>::iterator 变量；

变量可以赋值----v.begin()起始 v.end()结束

**遍历方式1：**

for (vector<int>::iterator it = v.begin(); it!= v.end(); \*it++) {

cout << \*it << endl;

}

**遍历方式2:**

void myPrint(int val){

cout << val << endl;

}

for\_each(v.begin(), v.end(), myPrint);//for\_each(首，尾，函数名)(使用了回调技术)

}

**！！！代码::** **vector容器存放内置的数据类型**

# 自定义数据类型

**Person 是类名**

vector<Person>v; 或 vector<Person\*>v是创建名字为v动态数组v

for (vector<Person>::iterator it = v.begin(); it != v.end(); it++)

**it** 可以看成一个指针 指向的是Person的地址 向Person\*

(\*it).m\_Name it->m\_Name //访问类中属性方式

for (vector<Person\*>::iterator it = v.begin(); it != v.end(); it++)

**it** 同样可以看成一个地址 是指向Person\*的地址

--Person\* 本来就是一个地址，故it是指向地址的地址

(\*it)->m\_Name (\*\*it).m\_Name //访问类中属性方式

**！！！代码::存放自定义数据类型**

# 二维数组，即vector 的嵌套

vector<vector<int>>v;//创建一个嵌套的动态数组

vector<int>v1;//创建一个一维动态数组

v1.push\_back(1);//一维数组赋值

v.push\_back(v1);//将一维数组整个填充到二维数组

遍历方式如下----- 迭代器 出来的变量为<这其中的东西>

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

cout << v[1][2]; //可以如此访问 //vit ( vector<int> )是这个二维数组的列

for (vector<int>::iterator vit = (\*it).begin(); vit != (\*it).end(); vit++) {

cout << \*vit << " ";

}

**！！！代码::容器嵌套容器**

# string p223-222(差異點)

deque 容器

双端数组，可以头端进行插入和删除操作

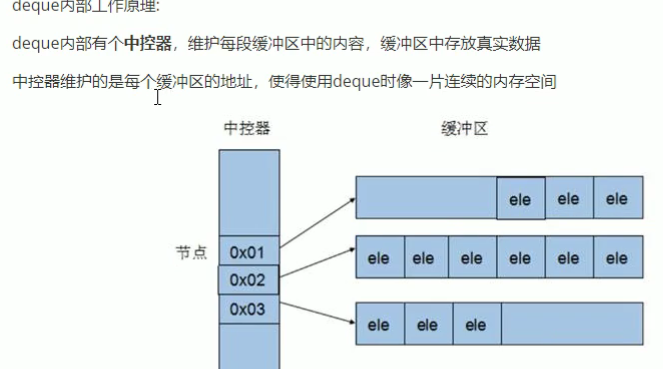
deque与vector区别:

●vector对于头部的插入删除效率低，数据量越大，效率越低

●deque相对而言，对头部的插入删除速度回比vector快

●vector访问元素时的速度会比deque快,这和两者内部实现有关

deque 的工作原理



deque容器支持迭代器的随机访问

String

string s1; //默认构造

string s2(str);//ba c\_string 转换成string

string s4(10,'a');

str4.assign("sa");

str5.assign("hello C++",5);// 前5个拼接到该字符串

str6.assign(str2);

str7.assign(5, 'G');

char s = 'H';

str1 +=s;//s可以是字符，string，

str5.append("?45fffds6", 4);//前4个拼接到该字符串

str2.append(str4, 0, 3); // str4=Holle,+ //0开始往后截取三个字符

str5.append(str2);//str5连接拼接至str2

str1.find("fg",3); //有返回下标，反之返回负一,后面一个参数是查找起始位置

str1.rfind("de"); //find 与 find区别 //find 从左往右查找 ， rfind从右往左查找

str1.replace(1, 3, "1111");//从下标1开始，往后3个替换

= 返回 0 //函数原型

> 返回 1 int compare(const string &s) const; //与字符串s比较

< 放回 -1 int compare(const char \*s) const; //与字符串s比较

//修改单个字符

char& operator[](int 0); //通过[]方式取字符

char& at(int n); //通过at方法获取字符

string& insert(int pos, const char\* s); //插入字符串

string& insert(int pos, const string& str); //插入字符串

string& insert(int pos, int n, char C); //在指定位置插入n个字符c

string& erase(int pos, int n = npos);， //删除从Pos开始的n个字符

string substr(int pos = 0, int n = npos) const; //返回由pos开始的n个字符组成的字符串

str2.substr(0, str2.find("@"))



# vector

vector<vector<int>> vec(row, vector<int> (col,1)); //二维初始化

push\_ back(ele); //尾部插入元素ele

pop\_ back(); //删除最后一个元素

back(); //返回容器中最后-个数据元素

front(); //返回容器中第一个数据元素

v.begin();v.end();//返回首位地址

clear(); //删除容器中所有元素

erase(const\_ iterator pos); //删除迭代器指向的元素

insert(const\_ iterator pos, int count,ele); //迭代器指向位置pos插入count个元素ele

empty();//判断容器是否为空

assign(n，elem);//将n个elem拷贝赋值给本身. //会清空数组原先数据

vec[i].resize(col)//调整长度为col

# #include<stack>

push(elem); //向栈顶添加元素

pop(); //从栈顶移除第一个元素

top(); //返回栈顶元素

!!!大小操作:

empty(); //判断堆栈是否为空

size(); //返回栈的大小.

# #include<list>

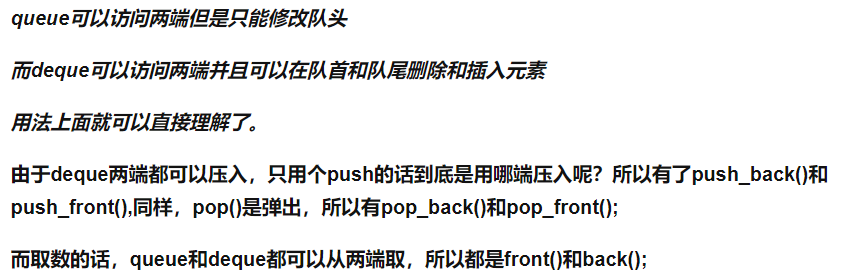
push\_ back(elem); //在容器尾部加入一个元素

pop\_ back(); //删除容器中最后一个元素

push\_front(elem); //在容器开头插入-个元素

pop\_ front(); //从容器开头移除第一个元素

# #deque



**std::deque<int> d;**

**std::deque<int> d(10);**

**std::deque<int> d(10, 5)**

**std::deque<int> d1(5);**

**std::deque<int> d2(d1);**

**//拷贝普通数组，创建deque容器**

**int a[] = { 1,2,3,4,5 };**

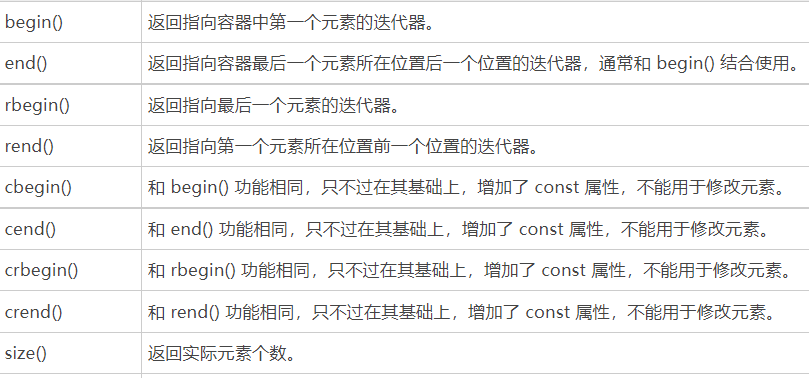
**std::deque<int>d(a, a + 5);**

**//适用于所有类型的容器**

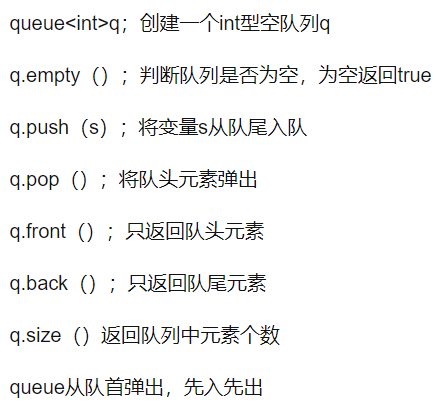
**std::array<int, 5>arr{11,12,13,14,15};**

**std::deque<int>d(arr.begin()+2, arr.end());//拷贝arr容器中的{13,14,15}**





# 并且queue只能从队首删除元素,但是两端都能访问

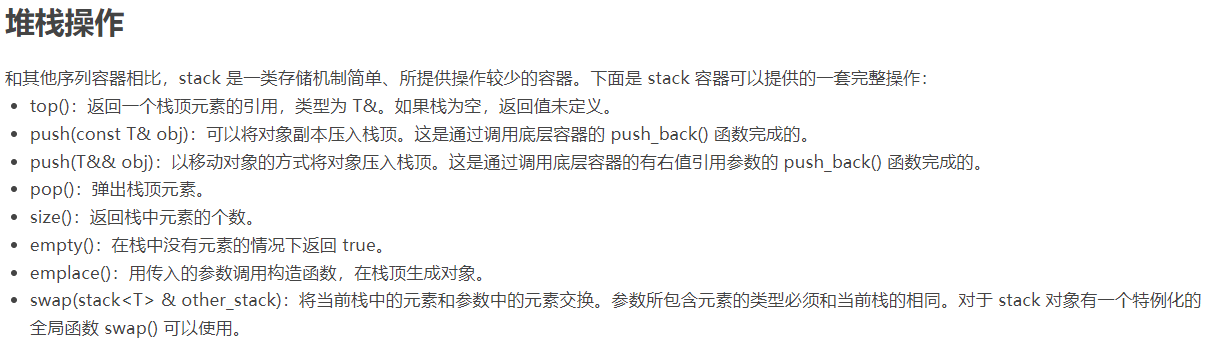


# priority\_queue 优先队列

priority\_queue<int, vector<int>, less<int> > p;

priority\_queue<int, vector<int>, greater<int> > q;

# stak 栈



# map map<int, string> mapStudent;



**map是模板，一个map变量key和value两个值，你在这里是想用类似map<int,int> m\_map的变量来表示背包里的东西，**

**m\_map->first可以取得key值，**

**m\_map->second可以取得value值；**

**map自动按照key值按升序排列，key的值不能修改，可以修改value的值。**

**begin()**          返回指向map头部的迭代器

**clear(）**          删除所有元素

**count()**          返回指定元素出现的次数

**empty()**          如果map为空则返回true

**end()**          返回指向map末尾的迭代器

**equal\_range()**          返回特殊条目的迭代器对

**erase()**          删除一个元素

**find()**          查找一个元素

**get\_allocator()**         返回map的配置器

**insert()**          插入元素

**key\_comp()**          返回比较元素key的函数

**lower\_bound()**         返回键值>=给定元素的第一个位置

**max\_size()**          返回可以容纳的最大元素个数

**rbegin()**          返回一个指向map尾部的逆向迭代器

**rend()**          返回一个指向map头部的逆向迭代器

**size()**          返回map中元素的个数

**swap()**          交换两个map

**upper\_bound()**          返回键值>给定元素的第一个位置

**value\_comp()**          返回比较元素value的函数

6   迭代数据   
  for   (auot my\_Itr=my\_Map.begin();   my\_Itr!=my\_Map.end();   ++my\_Itr)   {}

# unordered\_map<string(key), int>mp