## Day01

##### 1、代码规范

1 统一平台，便于兼容；

Windows 建议win7 x64

Windows下各开发工具大致如下：

Visual studio 建议2013

OpenCV 建议2.4.11

Halcon 建议2012

PCL 建议1.8.0

QT 建议5.5

Python视情况选择2.7或3.5，另python开发推荐使用pycharm软件，可视化与编辑等功能较全。

Ubuntu 建议Ubuntu 14.04 LTS

Ubuntu下各开发工具大致如下：

OpenCV 建议2.4.11

PCL 建议1.8.0

QT 建议5.5

Python 建议2.7，python开发推荐使用sublime Text

2 选择合适的代码管理平台，明细化访问、读写、维护等各种权限，推荐使用git平台。

3 代码的规范性

建议从每一行代码的开发与编写过程中，就尽力规范化，有代码规范工具StyleCorp，但不推荐使用。

3.1 命名规范

3.1.1 变量前缀：以变量实际名称为Var为例：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变量类型 | 变量名 | 变量类型 | 变量名 | 变量类型 | 变量名 |
| int | nVar/iVar | float | fVar | double | dVar |
| bool | bVar | byte | btVar | string | strVar |
| char | cVar | unsigned char | ucVar | unsigned int | unVar |

各类型指针前缀如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 指针类型 | 前缀 | 指针类型 | 前缀 | 指针类型 | 前缀 |
| 普通指针 | ptr | shared\_ptr | shr\_ptr | unique\_ptr | unq\_ptr |
| weak\_ptr | weak\_ptr |  |  |  |  |

关于如vector、list、map的STL泛型容器一类变量，各容器具体前缀如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 容器类型 | 前缀 | 容器类型 | 前缀 | 容器类型 | 前缀 |
| vector | vec | list | lst | set | set |
| queue | que | map | map | stack | stk |

对于全局变量与局部变量的区分，全局变量加g\_前缀，成员变量加m\_前缀予以表示。

**结构体名称前加大写字母S前缀**，**类名称前加大写字母C前缀**，**共用体名称前加大写字母U前缀。**

3.1.2 变量起名规则

1 严禁局部变量与全局变量重名；

2 变量定义应尽可能接近第一次使用地方，并尽早销毁，减少内存消耗；

3 变量定义需初始化，尤其是指针类型变量，以避免出现未知风险，如内存泄漏、野指针，浮动变量等；

4 变量名体现出变量实际凸显的功能或用途，命名相对意义的变量，用明确的反义词对；

5 变量的实际意义可能包含多个单词，单词之间的区分可以使用Unix风格，即单词之间以”\_”分隔，也可以使用Windows风格，即每个单词的第一个字母予以大写，但整体风格应保持一致

6尽量用const替换掉#define

3.2 函数规范

1 函数定义时参数顺序应保持为，输入参数在前，输出参数在后，输入参数一般为值和常数引用，输出参数或输入/输出参数一般为非常数指针；

2 函数尽量定义为bool类型，并明确是否成功执行，给出正确与否的返回值与日志信息，对于需要对输入数据进行处理获得结果的函数，尽量将需要返回的数据通过引用方式传递；

3 类一定要自己完成类中成员的初始化，无论是否在构造函数内，以避免不可预知的问题；

4 可以多定义相关成员函数，但尽量避免对操作符进行重载；

5 函数以实现具体功能为主，如果没有特殊的需求，尽量控制在40行以内，如果过长不影响程序的运行的情况，可以将长函数予以分隔；

6 尽量避免定义参数缺省函数，以避免错误理解及运行结果；

7 尽量避免使用RTTI，即dynamic\_cast一类，推荐使用virtual方法；

8 在函数与函数之间、逻辑段落之间、变量申明和函数之间多加以空格予以区分；

9 在==、=、+、‐、\*、/、>、<、>=、<=、:这些运算符两端加空格；标点符号如for循环内的”;”前面不加空格，后面加一个空格；if、for、while与括号之间加一个空格；多重括号运算用空格标明层次关系；->、.、++、--前后不加空格；数组名与[]之间及函数名和括号之间不加空格；尽量用.at()运算符替代[]运算符；

10 函数内每行代码不要超过80列，在逻辑分割点予以折行，并且折行内容尽量对齐；

11 逻辑判断时尽量将使用如if (2 == x)的结构，并且如果判断是否为零，尽量使用！运算符；

3.3 代码注释

1 代码注释用英文编写，尽量避免中文及其他语言；

2 关键代码及部分过程需要进行注释，代码注释应在20%以上；

3 注释和程序一同更新；

4 减少缩写，避免歧义；

5 对于协议标准或对paper方法的实现编程应加入标准文献的索引；

6 逻辑段落的注释放在段落前一行，语句的注释放在语句之后，注释与被注释的代码之间无空行；

7 头文件开头需要注释段落，头文件中函数声明前需要注释段落，全局变量需要详细注释，特殊功能的局部变量需要详细注释；

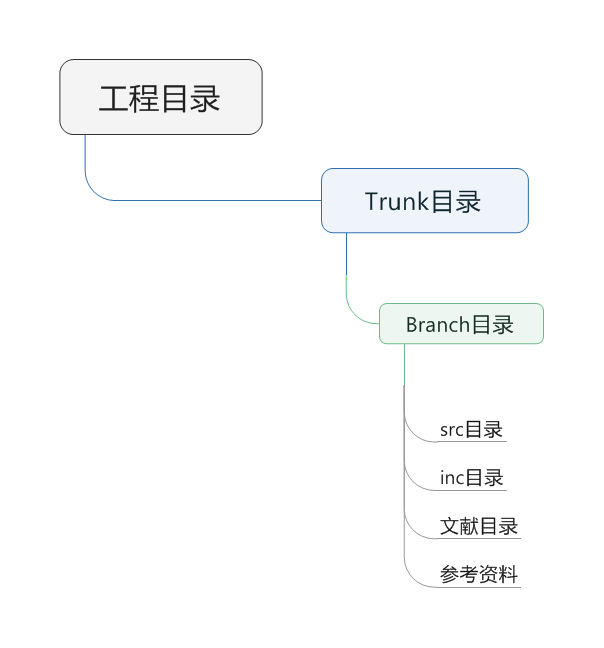
8 头文件开头注释段落参考模板如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (C), 20xx‐20xx, xxx Group

File name: xxx

Description: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Function List:

func1: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

func2: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

History:

Author: xxx Version: xxx Date: xxx Email: xxx

Description: Create

Author: xxx Version: xxx Date: xxx Email: xxx

Description: Fix xxx bugs

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

图1.工程结构

9 函数开头注释段落参考模板如下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: xxxx

Description :xxxxxxxxxxxxxxxxxx

Para: xxxxxx [输入输出类型]

Return: xxxx

Notes: xxxx

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

3.4 文件规范

1 头文件包含依次为：类对应头文件、C系统头文件、C++系统头文件、其他库头文件、本项目内头文件，且各部分头文件之间通过空行区分；

2 文件编译声明尽量使用前向声明，减少文件依赖；

3 尽量使用#ifndef #define结构，避免使用#pragma once方法；

4 工程结构

工程结构建议为四层树形结构，即工程目录、Trunk目录、Branch目录与具体文档目录。其中，Trunk目录主要为大方向上的分类并列目录，Branch目录为其中一个方向上具体一个技术点的目录，第四级目录为具体到各文件实现的目录，该目录中，需要存放的除了代码文件外，还应该有具体的文献索引或文件，技术手册或链接，开发参考的URL链接或具体文档资料。

##### 2、课程内容

**三个项目**：**①** 东方水利 **②** 三维重建 **③** Slam-AGV导航小车

**Everyday**:作业+两篇英文文献

**导论**：指针、引用、默认参数、结构体<结构体传参、结构体构造函数>、C++基本函数、类的构建

2.1 头文件

#ifndef LESSON\_H\_ //编译文件，全部大写，最后在加一个下划线防止重名

#define LESSON\_H\_

#include <iostream> //系统头文件<用尖括号>

#include <string>

#include <sstream>

#include <fstream>

#include <opencv2/opencv.hpp> //外部依赖头文件<用尖括号>

#include <boost/thread/thread.hpp>

#include”lesson2.h” //自定义头文件”用双引号”

//三种类型的头文件分三段

namespace IPSG //自定义空间名，命名空间1代码封装让别人知道哪个团队的2防重名

{

**typedef** struct Student

{ //成员变量加前缀m\_

std::string m\_strName; //姓名

std::string m\_strStudentNO; //学号

float m\_fCHNScore; //中文成绩

float m\_fMathScore; //数学成绩

float m\_fENGScore; //英文成绩

//结构体构造函数，有的编译器要求结构体必须赋初值（跟类不同）

Student(std::string strName = "123", std::string strStudentNO = "456", float fCHNScore = 12.0,

float fMathScore = 23.5, float fENGScore = 56.33) **:** m\_strName(strName), m\_strStudentNO(strStudentNO),

m\_fCHNScore(fCHNScore), m\_fMathScore(fMathScore), m\_fENGScore(fENGScore)**{};//大括号勿忘**

}SStudent; //结构体名前加大写S

class CStudent ////类名前加大写C

{

public:

CStudent() :m\_bUpdate(true), m\_fNature(85.65),

m\_strAccount("admin"), m\_strPassword("admin"){};

~CStudent();

bool Login();

private:

bool Index();

public:

bool m\_bUpdate;

private:

SStudent m\_Student;

float m\_fNature;

std::string m\_strAccount;

std::string m\_strPassword;

};

}

#endif // !LESSON\_H\_

2.2 CPP文件

#include "lesson1.h"

bool IPSG::Lesson() //空间名+类名+函数名==>作用域大小排列

{

std::fstream file;

file.open("file.txt",std::ios::out); //可用于打印日志log.txt

file << "stream" << std::endl;

SStudent Student("12345","67890",54.0,65.3,100.5);

std::cout << Student.m\_strName << std::endl;

Student.PrintStudentInfo();

return true;

}

2.3 main函数

Int main(int argc , char \*argv[], char \*\*env) main.exe -v main.exe -p

if(argv[1] == "-v") printf("hello");

if(argv[1] == "-p") printf("world");

有类似选择的功能。

C++定义接近使用的地方，Tab定义和注释时对齐。

2.4 指针

Int\* b = nullptr; //不用Int\* b = null; null定义为0，而0 的位置一般在系统的启动位置，比较危险。

2.5 引用

取别名，**在函数参数做传出参数的用法bool lesson （int nPara, int& nPara2）**

**函数中，return false 一般放在最前面<意会>**

**Return false**

**Return ture…**

2.6 默认参数

在头文件里面函数的声明处给定默认参数，注意二意性！

2.7 头文件注释，函数注释

VASSITX插件：VASSTIX-->Insert VA Snippet…-->Edit VA Snippets…

文件注释

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Copyright (C), 2018-2021, Southwest University of Science and Technology.

//Filename : $FILE\_BASE$.$FILE\_EXT$

//Author : IPSG

//Date : $DATE$ $HOUR$:$MINUTE$

//Version : 1.0

//Description:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

函数注释

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*!

2 \*@brief

3 \*@author %USERNAME%

4 \*@date $YEAR$年$MONTH$月$DAY$日

5 \*@param[out]

6 \*@param[in] $MethodArg$

7 \*@return $SymbolType$

8 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2.8杂项

sstrream数据流系统 fstream文件流系统 文件读入可以按符号 空格等。

stringstream类型 一般用于刷buffer 类型转换。

算术性使用频率很高用const而不是define，#define用于开关性质的地方。

Vector<BYTE> vuc。

需要一个输出 就定义一个引用 或者指针。

解码时为保护数据拷贝一份。

##### 3.资料博客查阅

3.1 c++中vector的用法详解

vector(向量): C++中的一种数据结构,确切的说是一个类.它相当于一个动态的数组,当程序员无法知道自己需要的数组的规模多大时,用其来解决问题可以达到最大节约空间的目的.

用法:

1.文件包含:

首先在程序开头处加上#include<vector>以包含所需要的类文件vector

还有一定要加上using namespace std;

2.变量声明:

2.1 例:声明一个int向量以替代一维的数组:vector <int> a;(等于声明了一个int数组a[],大小没有指定,可以动态的向里面添加删除)。

2.2 例:用vector代替二维数组.其实只要声明一个一维数组向量即可,而一个数组的名字其实代表的是它的首地址,所以只要声明一个地址的向量即可,即:vector <int \*> a.同理想用向量代替三维数组也是一样,vector <int\*\*>a;再往上面依此类推.

3.具体的用法以及函数调用:

3.1 如何得到向量中的元素?其用法和数组一样:

例如:

vector <int \*> a

int b = 5;

a.push\_back(b);//该函数下面有详解

cout<<a[0]; //输出结果为5

1.push\_back 在数组的最后添加一个数据

2.pop\_back 去掉数组的最后一个数据

3.at 得到编号位置的数据

4.begin 得到数组头的指针

5.end 得到数组的最后一个单元+1的指针

6．front 得到数组头的引用

7.back 得到数组的最后一个单元的引用

8.max\_size 得到vector最大可以是多大

9.capacity 当前vector分配的大小

10.size 当前使用数据的大小

11.resize 改变当前使用数据的大小，如果它比当前使用的大，者填充默认值

12.reserve 改变当前vecotr所分配空间的大小

13.erase 删除指针指向的数据项

14.clear 清空当前的vector

15.rbegin 将vector反转后的开始指针返回(其实就是原来的end-1)

16.rend 将vector反转构的结束指针返回(其实就是原来的begin-1)

17.empty 判断vector是否为空

18.swap 与另一个vector交换数据

3.2 详细的函数实现功能：其中vector<int> c.

c.clear() 移除容器中所有数据。

c.empty() 判断容器是否为空。

c.erase(pos) 删除pos位置的数据

c.erase(beg,end) 删除[beg,end)区间的数据

c.front() 传回第一个数据。

c.insert(pos,elem) 在pos位置插入一个elem拷贝

c.pop\_back() 删除最后一个数据。

c.push\_back(elem) 在尾部加入一个数据。

c.resize(num) 重新设置该容器的大小

c.size() 回容器中实际数据的个数。

c.begin() 返回指向容器第一个元素的迭代器

c.end() 返回指向容器最后一个元素的迭代器

4.内存管理与效率

1》使用reserve()函数提前设定容量大小，避免多次容量扩充操作导致效率低下。

关于STL容器，最令人称赞的特性之一就是是只要不超过它们的最大大小，它们就可以自动增长到足以容纳你放进去的数据。（要知道这个最大值，只要调用名叫max\_size的成员函数。）对于vector和string，如果需要更多空间，就以类似realloc的思想来增长大小。vector容器支持随机访问，因此为了提高效率，它内部使用动态数组的方式实现的。在通过 reserve() 来申请特定大小的时候总是按指数边界来增大其内部缓冲区。当进行insert或push\_back等增加元素的操作时，如果此时动态数组的内存不够用，就要动态的重新分配当前大小的1.5~2倍的新内存区，再把原数组的内容复制过去。所以，在一般情况下，其访问速度同一般数组，只有在重新分配发生时，其性能才会下降。正如上面的代码告诉你的那样。而进行pop\_back操作时，capacity并不会因为vector容器里的元素减少而有所下降，还会维持操作之前的大小。对于vector容器来说，如果有大量的数据需要进行push\_back，应当使用reserve()函数提前设定其容量大小，否则会出现许多次容量扩充操作，导致效率低下。

reserve成员函数允许你最小化必须进行的重新分配的次数，因而可以避免真分配的开销和迭代器/指针/引用失效。但在我解释reserve为什么可以那么做之前，让我简要介绍有时候令人困惑的四个相关成员函数。在标准容器中，只有vector和string提供了所有这些函数。

(1) size()告诉你容器中有多少元素。它没有告诉你容器为它容纳的元素分配了多少内存。

(2) capacity()告诉你容器在它已经分配的内存中可以容纳多少元素。那是容器在那块内存中总共可以容纳多少元素，而不是还可以容纳多少元素。如果你想知道一个vector或string中有多少没有被占用的内存，你必须从capacity()中减去size()。如果size和capacity返回同样的值，容器中就没有剩余空间了，而下一次插入（通过insert或push\_back等）会引发上面的重新分配步骤。

(3) resize(Container::size\_type n)强制把容器改为容纳n个元素。调用resize之后，size将会返回n。如果n小于当前大小，容器尾部的元素会被销毁。如果n大于当前大小，新默认构造的元素会添加到容器尾部。如果n大于当前容量，在元素加入之前会发生重新分配。

(4) reserve(Container::size\_type n)强制容器把它的容量改为至少n，提供的n不小于当前大小。这一般强迫进行一次重新分配，因为容量需要增加。（如果n小于当前容量，vector忽略它，这个调用什么都不做，string可能把它的容量减少为size()和n中大的数，但string的大小没有改变。在我的经验中，使用reserve来从一个string中修整多余容量一般不如使用“交换技巧”，那是条款17的主题。）

这个简介表示了只要有元素需要插入而且容器的容量不足时就会发生重新分配（包括它们维护的原始内存分配和回收，对象的拷贝和析构和迭代器、指针和引用的失效）。所以，避免重新分配的关键是使用reserve尽快把容器的容量设置为足够大，最好在容器被构造之后立刻进行。

例如，假定你想建立一个容纳1-1000值的vector<int>。没有使用reserve，你可以像这样来做：

vector<int> v;

for (int i = 1; i <= 1000; ++i) v.push\_back(i);

在大多数STL实现中，这段代码在循环过程中将会导致2到10次重新分配。（10这个数没什么奇怪的。记住vector在重新分配发生时一般把容量翻倍，而1000约等于210。）

把代码改为使用reserve，我们得到这个：

vector<int> v;

v.reserve(1000);

for (int i = 1; i <= 1000; ++i) v.push\_back(i);

这在循环中不会发生重新分配。

在大小和容量之间的关系让我们可以预言什么时候插入将引起vector或string执行重新分配，而且，可以预言什么时候插入会使指向容器中的迭代器、指针和引用失效。例如，给出这段代码，

string s;

...

if (s.size() < s.capacity()) {

s.push\_back('x');

}

push\_back的调用不会使指向这个string中的迭代器、指针或引用失效，因为string的容量保证大于它的大小。如果不是执行push\_back，代码在string的任意位置进行一个insert，我们仍然可以保证在插入期间没有发生重新分配，但是，与伴随string插入时迭代器失效的一般规则一致，所有从插入位置到string结尾的迭代器/指针/引用将失效。

回到本条款的主旨，通常有两情况使用reserve来避免不必要的重新分配。第一个可用的情况是当你确切或者大约知道有多少元素将最后出现在容器中。那样的话，就像上面的vector代码，你只是提前reserve适当数量的空间。第二种情况是保留你可能需要的最大的空间，然后，一旦你添加完全部数据，修整掉任何多余的容量。

2》使用“交换技巧”来修整vector过剩空间/内存

有一种方法来把它从曾经最大的容量减少到它现在需要的容量。这样减少容量的方法常常被称为“收缩到合适（shrink to fit）”。该方法只需一条语句：vector<int>(ivec).swap(ivec);

表达式vector<int>(ivec)建立一个临时vector，它是ivec的一份拷贝：vector的拷贝构造函数做了这个工作。但是，vector的拷贝构造函数只分配拷贝的元素需要的内存，所以这个临时vector没有多余的容量。然后我们让临时vector和ivec交换数据，这时我们完成了，ivec只有临时变量的修整过的容量，而这个临时变量则持有了曾经在ivec中的没用到的过剩容量。在这里（这个语句结尾），临时vector被销毁，因此释放了以前ivec使用的内存，收缩到合适。

3》用swap方法强行释放STL Vector所占内存

template < class T> void ClearVector( vector<T>& v )

{

vector<T>vtTemp;

vtTemp.swap( v );

}

如

vector<int> v ;

nums.push\_back(1);

nums.push\_back(3);

nums.push\_back(2);

nums.push\_back(4);

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

/\* 或者v.swap(vector<int>()); \*/

/\*或者{ std::vector<int> tmp = v; v.swap(tmp); }; //加大括号{ }是让tmp退出{ }时自动析构\*/

5.Vector 内存管理成员函数的行为测试

C++ STL的vector使用非常广泛，但是对其内存的管理模型一直有多种猜测，下面用实例代码测试来了解其内存管理方式，测试代码如下：

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> iVec;

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //1个元素， 容器容量为1

iVec.push\_back(1);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //2个元素， 容器容量为2

iVec.push\_back(2);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //3个元素， 容器容量为4

iVec.push\_back(3);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //4个元素， 容器容量为4

iVec.push\_back(4);

iVec.push\_back(5);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //5个元素， 容器容量为8

iVec.push\_back(6);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //6个元素， 容器容量为8

iVec.push\_back(7);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //7个元素， 容器容量为8

iVec.push\_back(8);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //8个元素， 容器容量为8

iVec.push\_back(9);

cout << "容器 大小为: " << iVec.size() << endl;

cout << "容器 容量为: " << iVec.capacity() << endl; //9个元素， 容器容量为16

/\* vs2005/8 容量增长不是翻倍的，如

9个元素 容量9

10个元素 容量13 \*/

/\* 测试effective stl中的特殊的交换 swap() \*/

cout << "当前vector 的大小为: " << iVec.size() << endl;

cout << "当前vector 的容量为: " << iVec.capacity() << endl;

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

cout << "临时的vector<int>对象 的大小为: " << (vector<int>(iVec)).size() << endl;

cout << "临时的vector<int>对象 的容量为: " << (vector<int>(iVec)).capacity() << endl;

cout << "交换后，当前vector 的大小为: " << iVec.size() << endl;

cout << "交换后，当前vector 的容量为: " << iVec.capacity() << endl;

return 0;

}

6.vector的其他成员函数

c.assign(beg,end)：将[beg; end)区间中的数据赋值给c。

c.assign(n,elem)：将n个elem的拷贝赋值给c。

c.at(idx)：传回索引idx所指的数据，如果idx越界，抛出out\_of\_range。

c.back()：传回最后一个数据，不检查这个数据是否存在。

c.front()：传回地一个数据。

get\_allocator：使用构造函数返回一个拷贝。

c.rbegin()：传回一个逆向队列的第一个数据。

c.rend()：传回一个逆向队列的最后一个数据的下一个位置。

c.~ vector <Elem>()：销毁所有数据，释放内存。

7.备注:在用vector的过程中的一些问题,特此列出讨论:

1)

vector <int > a;

int b = 5;

a.push\_back(b);

此时若对b另外赋值时不会影响a[0]的值

2)

vector <int\*> a;

int \*b;

b= new int[4];

b[0]=0;

b[1]=1;

b[2]=2;

a.push\_back(b);

delete b; //释放b的地址空间

for(int i=0 ; i <3 ; i++)

{

cout<<a[0][i]<<endl;

}

此时输出的值并不是一开始b数组初始化的值,而是一些无法预计的值.

分析:根据1) 2)的结果,可以想到,在1)中, 往a向量中压入的是b的值,即a[0]=b,此时a[0]和b是存储在两个不同的地址中的.因此改变b的值不会影响a[0];而在2)中,因为是把一个地址(指针)压入向量a,即a[0]=b,因此释放了b的地址也就释放了a[0]的地址,因此a[0]数组中存放的数值也就不得而知了.