一．Vector容器介绍

Vector容器，说白了也是数组，但是vector的空间运用十分灵活，不像 array十分死板。Vector是动态空间分配，随着元素的加入，它的内部机制会自动扩充空间。同时，vector的空间扩充并不是要多少扩充多少，而是预先多分配一些空间，以供之后的需求。

**vector的容量永远大于或等于其大小，一旦容量等于大小，下次再有新增元素，整个**

**vector容器就得再次开辟新空间。**

二、vector的数据结构

Vector的数据结构，是线性连续空间，它以两个迭代器\_Myfirst和\_Mylast分别指向配置得来的连续空间中目前已被使用的范围，并以迭代器\_Myend指向整块连续内存空间的尾端。

Vector迭代器是随机访问的迭代器，支持跳跃式访问

三．Vector的扩充

Vector并不是在原有的空间中添加新的空间，而是寻找一片更大的空间，将原有的空间拷贝过去，并且释放原有的空间。因此，当vector的空间扩展时，原有的迭代器一定会失效。

那么，vector扩充空间有规律么？用代码来看看。

执行完代码后

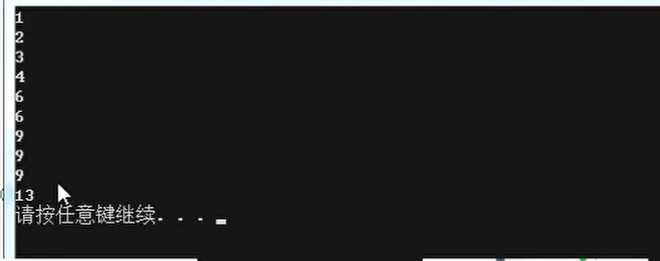
Vector<int> v;

For(int I = 0; I < 10; ++i)

{  
 v.push\_back(i);

Cout << v.capacity() << endl;

}



可以看到，vector的容量扩充并不是说有一定的规律的，不是按照什么倍数增长，看起来似乎没有约定俗成的规律，只是适当地扩充。

四．如何降低vector的容量？

Vector<int> v;

For(int I = 0; I < 10000; ++i)

v.push\_back(i);

v.resize(3);

更改v的容量，从代码中可以看出，初始化v的大小为10000，因此v的容量必定是大于10000的。之后改变v的大小为3，而v的容量依然是不变的。那么这样的vector就会浪费很多空间。如何优化呢？

Vector<int>(v).swap(v)

前面是一个匿名对象，用v来初始化匿名对象x，因此该匿名对象x的空间就是v的空间大小，同时x的容量也为3。

后面用swap的作用是交换v和x的指针，因此，现在v指向的是一个vector，容量为3；而匿名对象x指向的是一个原始的vector，容量大于10000，后续 匿名对象x会自动被销毁。

五．Vector的常用操作

（一）vector构造函数

vector<T> v; //采用模板实现类实现，默认构造函数

vector(v.begin(), v.end());//将v[begin(), end())区间中的元素拷贝给本身。

vector(n, elem);//构造函数将n个elem拷贝给本身。

vector(const vector &vec);//拷贝构造函数。

//例子 使用第二个构造函数 我们可以...

int arr[] = {2,3,4,1,9};

vector<int> v1(arr, arr + sizeof(arr) / sizeof(int));

（二）vector赋值操作

assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身。

assign(n, elem);//将n个elem拷贝赋值给本身。

vector& operator=(const vector &vec);//重载等号操作符

swap(vec);// 将vec与本身的元素互换。

（三）vector大小操作

size();//返回容器中元素的个数

empty();//判断容器是否为空

resize(int num);//重新指定容器的长度为num，若容器变长，则以默认值填充新位置。如果容器变短，则末尾超出容器长度的元素被删除。

resize(int num, elem);//重新指定容器的长度为num，若容器变长，则以elem值填充新位置。如果容器变短，则末尾超出容器长>度的元素被删除。

capacity();//容器的容量

reserve(int len);//容器预留len个元素长度，预留位置不初始化，元素不可访问。

（四）vector数据存取操作

at(int idx); //返回索引idx所指的数据，如果idx越界，抛出out\_of\_range异常。

operator[];//返回索引idx所指的数据，越界时，运行直接报错

front();//返回容器中第一个数据元素

back();//返回容器中最后一个数据元素

（五）vector插入和删除操作

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

push\_back(ele); //尾部插入元素ele

pop\_back();//删除最后一个元素

erase(const\_iterator start, const\_iterator end);//删除迭代器从start到end之间的元素

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

clear();//删除容器中所有元素