* 没有退路可言、不要给自己找接口、以及**顾虑太多**
* 缺乏目的、**目标**、**方法**、效率再勤奋也只是一种时间的牺牲

@[toc]

# deque容器

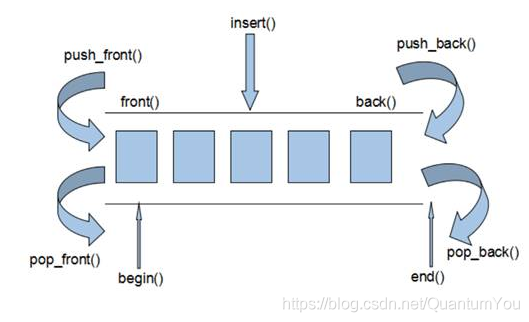
## deque容器基本概念

**功能：**

* 双端数组，可以对头端进行插入删除操作

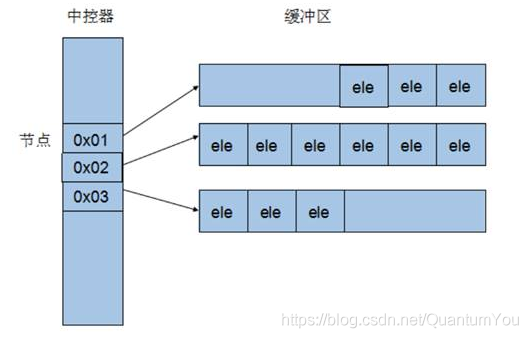
**deque与vector区别：**

* 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内部工作原理**:

* deque内部有个**中控器**，维护每段缓冲区中的内容，缓冲区中存放真实数据
* 中控器维护的是每个缓冲区的地址，使得使用deque时像一片连续的内存空间



* deque容器的迭代器也是支持随机访问的

## deque构造函数

**功能描述：**

* deque容器构造

**函数原型：**

* deque<T> deqT; //默认构造形式
* deque(beg, end); //构造函数将[beg, end)区间中的元素拷贝给本身。
* deque(n, elem); //构造函数将n个elem拷贝给本身。
* deque(const deque &deq); //拷贝构造函数

**示例：**

#include <deque>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
//deque构造  
void test01() {  
  
 deque<int> d1; //无参构造函数  
 for (int i = 0; i < 10; i++)  
 {  
 d1.push\_back(i);  
 }  
 printDeque(d1);  
 deque<int> d2(d1.begin(),d1.end());  
 printDeque(d2);  
  
 deque<int>d3(10,100);  
 printDeque(d3);  
  
 deque<int>d4 = d3;  
 printDeque(d4);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : deque容器和vector容器的构造方式几乎一致，灵活使用即可

注意迭代器的高级使用：  
1、 加入 const deque<int> &d 以及 deque<int>::const\_iterator it = d.begin(); 的作用防止里面参数的修改

void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}

## deque赋值操作

**功能描述：**

* 给deque容器进行赋值

**函数原型：**

* deque& operator=(const deque &deq); //重载等号操作符
* 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。

**示例：**

#include <deque>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
//赋值操作  
void test01()  
{  
 deque<int> d1;  
 for (int i = 0; i < 10; i++)  
 {  
 d1.push\_back(i);  
 }  
 printDeque(d1);  
  
 deque<int>d2;  
 d2 = d1;  
 printDeque(d2);  
  
 deque<int>d3;  
 d3.assign(d1.begin(), d1.end());  
 printDeque(d3);  
  
 deque<int>d4;  
 d4.assign(10, 100);  
 printDeque(d4);  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**：deque赋值操作也与vector相同，需熟练掌握

## deque大小操作

**功能描述：**

* 对deque容器的大小进行操作

**函数原型：**

* deque.empty(); //判断容器是否为空
* deque.size(); //返回容器中元素的个数
* deque.resize(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。
* deque.resize(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。

**注意**; deque 容器没有 capacity 容量的大小的描述

**示例：**

#include <deque>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
  
//大小操作  
void test01()  
{  
 deque<int> d1;  
 for (int i = 0; i < 10; i++)  
 {  
 d1.push\_back(i);  
 }  
 printDeque(d1);  
  
 //判断容器是否为空  
 if (d1.empty()) {  
 cout << "d1为空!" << endl;  
 }  
 else {  
 cout << "d1不为空!" << endl;  
 //统计大小  
 cout << "d1的大小为：" << d1.size() << endl;  
 }  
  
 //重新指定大小  
 d1.resize(15, 1);  
 printDeque(d1);  
  
 d1.resize(5);  
 printDeque(d1);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* deque没有容量的概念
* 判断是否为空 --- empty
* 返回元素个数 --- size
* 重新指定个数 --- resize

## deque 插入和删除

**功能描述：**

* 向deque容器中插入和删除数据

**函数原型：**

两端插入操作：

* push\_back(elem); //在容器尾部添加一个数据
* push\_front(elem); //在容器头部插入一个数据
* pop\_back(); //删除容器最后一个数据
* pop\_front(); //删除容器第一个数据

指定位置操作：

* insert(pos,elem); //在pos位置插入一个elem元素的拷贝，返回新数据的位置。
* insert(pos,n,elem); //在pos位置插入n个elem数据，无返回值。
* insert(pos,beg,end); //在pos位置插入[beg,end)区间的数据，无返回值。
* clear(); //清空容器的所有数据
* erase(beg,end); //删除[beg,end)区间的数据，返回下一个数据的位置。
* erase(pos); //删除pos位置的数据，返回下一个数据的位置。

**示例：**

#include <deque>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
//两端操作  
void test01()  
{  
 deque<int> d;  
 //尾插  
 d.push\_back(10);  
 d.push\_back(20);  
 //头插  
 d.push\_front(100);  
 d.push\_front(200);  
  
 printDeque(d);  
  
 //尾删  
 d.pop\_back();  
 //头删  
 d.pop\_front();  
 printDeque(d);  
}  
  
//插入  
void test02()  
{  
 deque<int> d;  
 d.push\_back(10);  
 d.push\_back(20);  
 d.push\_front(100);  
 d.push\_front(200);  
 printDeque(d);  
  
 d.insert(d.begin(), 1000);  
 printDeque(d);  
  
 d.insert(d.begin(), 2,10000);  
 // 1000 1000 1000 200 100 10 20   
 printDeque(d);  
  
 deque<int>d2;  
 d2.push\_back(1);  
 d2.push\_back(2);  
 d2.push\_back(3);  
  
 d.insert(d.begin(), d2.begin(), d2.end());  
 // 1 2 3 1000 1000 1000 200 100 10 20   
 printDeque(d);  
  
}  
  
//删除  
void test03()  
{  
 deque<int> d;  
 d.push\_back(10);  
 d.push\_back(20);  
 d.push\_front(100);  
 d.push\_front(200);  
 printDeque(d);  
  
 d.erase(d.begin());  
 printDeque(d);  
  
 d.erase(d.begin(), d.end());  
 d.clear();  
 printDeque(d);  
}  
  
int main() {  
  
 //test01();  
  
 //test02();  
  
 test03();  
   
 system("pause");  
  
 return 0;  
}

总结：

* 插入和删除提供的位置是迭代器！(而不是数字下标)
* 尾插 --- push\_back
* 尾删 --- pop\_back
* 头插 --- push\_front
* 头删 --- pop\_front

注意：按照区间插入 算法

P207

## deque 数据存取

**功能描述：**

* 对deque 中的数据的存取操作

**函数原型：**

* at(int idx); //返回索引idx所指的数据
* operator[]; //返回索引idx所指的数据
* 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

**示例：**

#include <deque>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
  
//数据存取  
void test01()  
{  
  
 deque<int> d;  
 d.push\_back(10);  
 d.push\_back(20);  
 d.push\_front(100);  
 d.push\_front(200);  
  
 for (int i = 0; i < d.size(); i++) {  
 cout << d[i] << " ";  
 }  
 cout << endl;  
  
  
 for (int i = 0; i < d.size(); i++) {  
 cout << d.at(i) << " ";  
 }  
 cout << endl;  
  
 cout << "front:" << d.front() << endl;  
  
 cout << "back:" << d.back() << endl;  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 除了用迭代器获取deque容器中元素，[ ]和at也可以
* front返回容器第一个元素
* back返回容器最后一个元素

## deque 排序

**功能描述：**

* 利用算法实现对deque容器进行排序

**算法：**

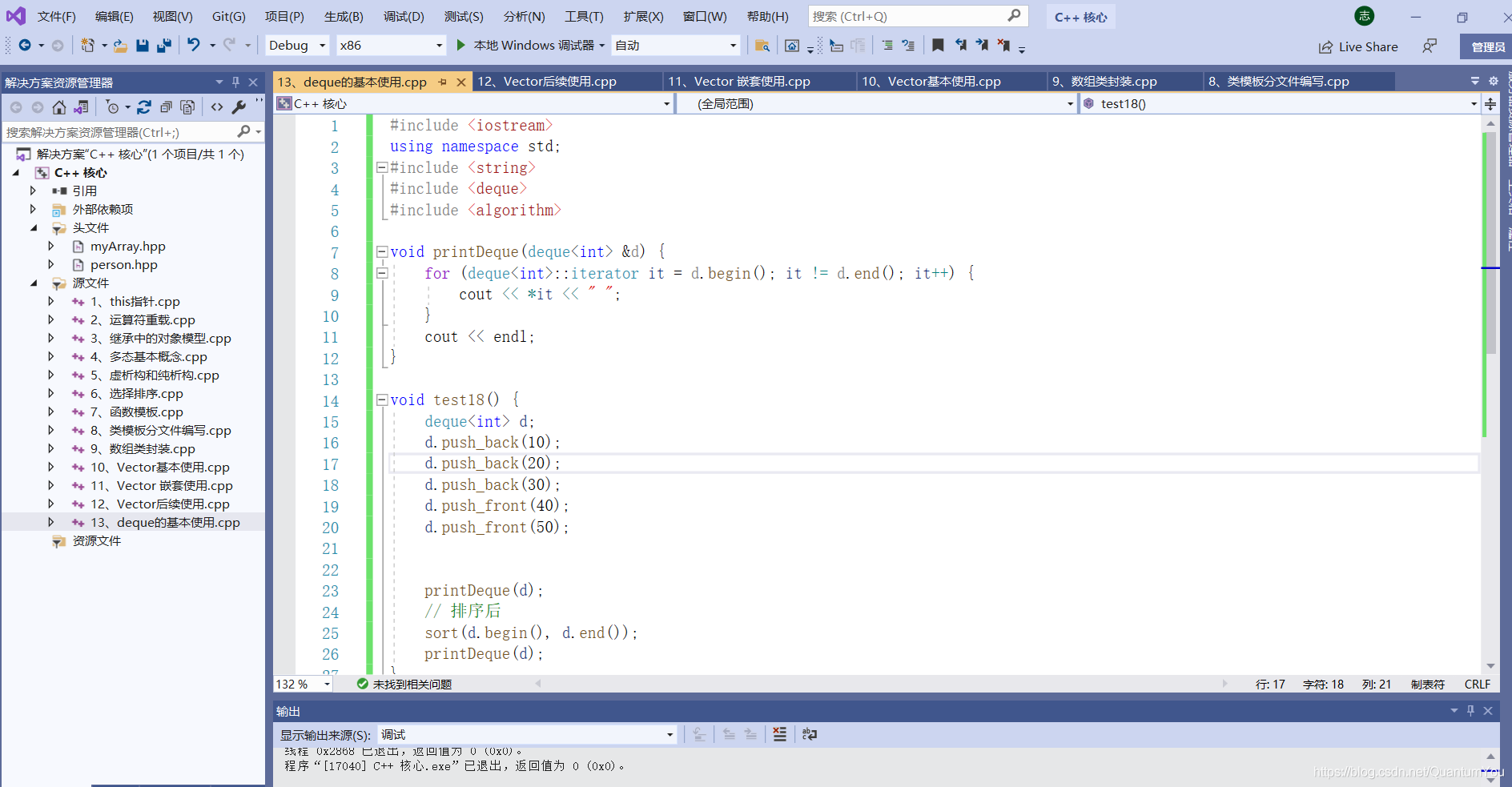
* sort(iterator beg, iterator end) //对beg和end区间内元素进行排序

**示例：**

#include <deque>  
#include <algorithm>  
  
void printDeque(const deque<int>& d)   
{  
 for (deque<int>::const\_iterator it = d.begin(); it != d.end(); it++) {  
 cout << \*it << " ";  
  
 }  
 cout << endl;  
}  
  
void test01()  
{  
  
 deque<int> d;  
 d.push\_back(10);  
 d.push\_back(20);  
 d.push\_front(100);  
 d.push\_front(200);  
  
 printDeque(d);  
 sort(d.begin(), d.end());  
 printDeque(d);  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**：

* sort算法非常实用，使用时包含头文件 algorithm即可
* 默认排序规则 从小到大 升序
* 对于支持 随机访问的迭代器容器、都可以使用 sort 排序对其进行排序



# stack容器

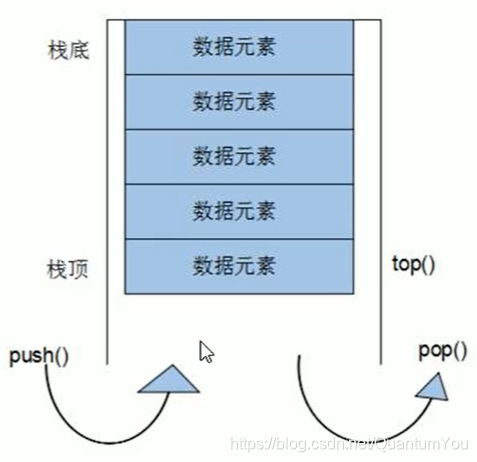
## stack 基本概念

**概念：**stack是一种**先进后出**(First In Last Out,FILO)的数据结构，它只有一个出口

* 栈中只有顶端的元素才可以被外界使用，因此栈不允许有遍历行为，但是可以知道栈中有多少元素
* 栈中进入数据称为 --- **入栈** push
* 栈中弹出数据称为 --- **出栈** pop

## stack 常用接口

功能描述：栈容器常用的对外接口



**构造函数**：

* stack<T> stk; //stack采用模板类实现， stack对象的默认构造形式
* stack(const stack &stk); //拷贝构造函数

赋值操作：

* stack& operator=(const stack &stk); //重载等号操作符

数据存取：

* 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：

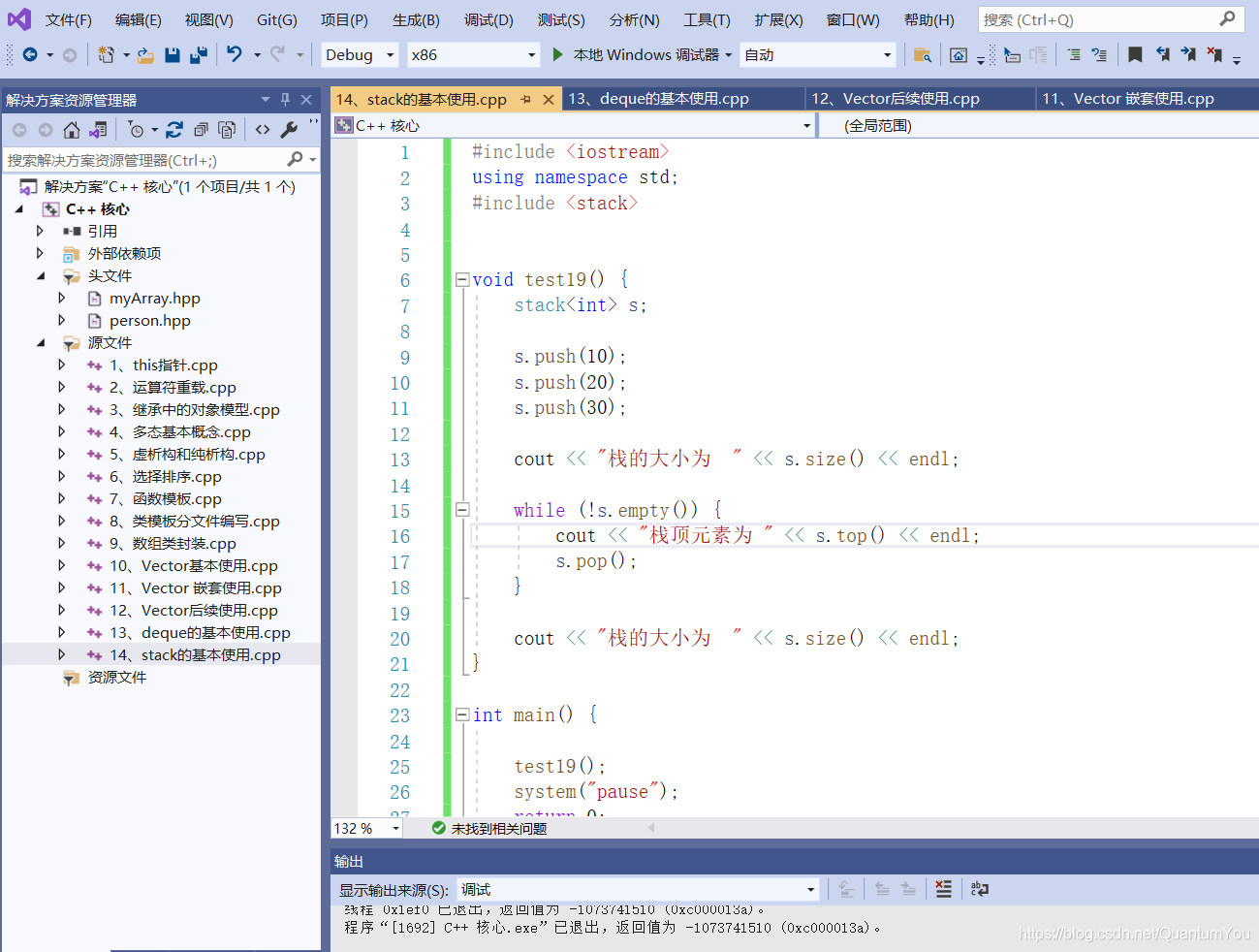
* empty(); //判断堆栈是否为空
* size(); //返回栈的大小

**示例：**

#include <stack>  
  
//栈容器常用接口  
void test01()  
{  
 //创建栈容器 栈容器必须符合先进后出  
 stack<int> s;  
  
 //向栈中添加元素，叫做 压栈 入栈  
 s.push(10);  
 s.push(20);  
 s.push(30);  
  
 while (!s.empty()) {  
 //输出栈顶元素  
 cout << "栈顶元素为： " << s.top() << endl;  
 //弹出栈顶元素  
 s.pop();  
 }  
 cout << "栈的大小为：" << s.size() << endl;  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

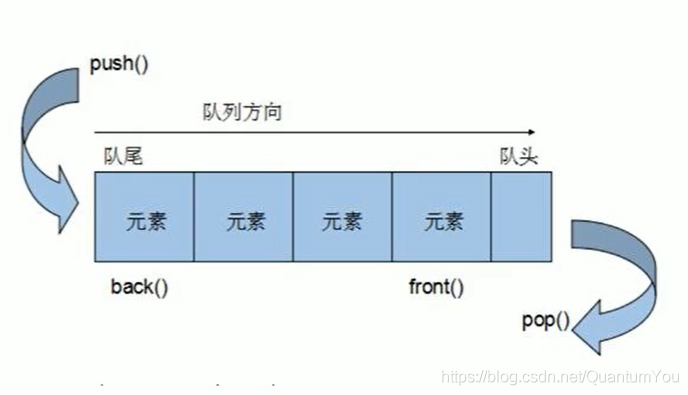
总结：

* 入栈 --- push
* 出栈 --- pop
* 返回栈顶 --- top
* 判断栈是否为空 --- empty
* 返回栈大小 --- size



# queue 容器

## queue 基本概念

**概念：**Queue是一种**先进先出**(First In First Out,FIFO)的数据结构，它有两个出口  


* 队列容器允许从一端新增元素，从另一端移除元素
* 队列中只有队头和队尾才可以被外界使用，因此队列不允许有遍历行为
* 队列中进数据称为 --- **入队** push
* 队列中出数据称为 --- **出队** pop

## queue 常用接口

功能描述：栈容器常用的对外接口

**构造函数**：

* queue<T> que; //queue采用模板类实现，queue对象的默认构造形式
* queue(const queue &que); //拷贝构造函数

赋值操作：

* queue& operator=(const queue &que); //重载等号操作符

数据存取：

* push(elem); //往队尾添加元素
* pop(); //从队头移除第一个元素
* back(); //返回最后一个元素
* front(); //返回第一个元素

大小操作：

* empty(); //判断堆栈是否为空
* size(); //返回栈的大小

**示例：**

#include <queue>  
#include <string>  
class Person  
{  
public:  
 Person(string name, int age)  
 {  
 this->m\_Name = name;  
 this->m\_Age = age;  
 }  
  
 string m\_Name;  
 int m\_Age;  
};  
  
void test01() {  
  
 //创建队列  
 queue<Person> q;  
  
 //准备数据  
 Person p1("唐僧", 30);  
 Person p2("孙悟空", 1000);  
 Person p3("猪八戒", 900);  
 Person p4("沙僧", 800);  
  
 //向队列中添加元素 入队操作  
 q.push(p1);  
 q.push(p2);  
 q.push(p3);  
 q.push(p4);  
  
 //队列不提供迭代器，更不支持随机访问   
 while (!q.empty()) {  
 //输出队头元素  
 cout << "队头元素-- 姓名： " << q.front().m\_Name   
 << " 年龄： "<< q.front().m\_Age << endl;  
   
 cout << "队尾元素-- 姓名： " << q.back().m\_Name   
 << " 年龄： " << q.back().m\_Age << endl;  
   
 cout << endl;  
 //弹出队头元素  
 q.pop();  
 }  
  
 cout << "队列大小为：" << q.size() << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 入队 --- push
* 出队 --- pop
* 返回队头元素 --- front
* 返回队尾元素 --- back
* 判断队是否为空 --- empty
* 返回队列大小 --- size