知识点1【vectorAPI】

1、vector赋值与交换语句

案例1：赋值交换

2、vector大小操作

案例：1

案例2：resize 作用的容器的大小 不会更改容器的容量

3、使用resize swap收缩容器的容量

4、reserve预留空间大小

5、数据的存取

6、vector容器的插入和删除

知识点2【deque容器 双端动态数组】

1、deque容器和vector容器最大的差异：

知识点3【deque容器的API】

1、deque容器的构造 和赋值

2、deque容器的大小操作、双端插入删除操作、元素访问操作

3、容器的插入删除

4、案例：有5名选手：选手ABCDE，10个评委分别对每一名选手打分，去除最高分，去除评委中最低分，取平均分，最后输出每位选手的分数

5、srand设置随机种子 rand 产生随机数

知识点4【stack容器 栈容器】

知识点5【队列容器 queue】

知识点6【list容器】链表容器

知识点7【链表的API】

知识点1【vectorAPI】

1、vector赋值与交换语句

案例1：赋值交换

```
/*
3.2.4.2 vector常用赋值操作
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。
*/
void printVectorInt(vector<int> &v)
{
 for(vector<int>::iterator it=v.begin(); it!=v.end();it++)
 {
 cout<<*it<<" ";
 }
 cout<<endl;
}
void test01()
{
 vector<int> v1(5,10);
 vector<int> v2;

 //vector& operator=(const vector &vec);//重载等号操作符
 v2 = v1;
 printVectorInt(v2);

 //assign(n, elem);//将n个elem拷贝赋值给本身
 vector<int> v3;
 v3.assign(5,100);
 printVectorInt(v3);

 //assign(beg, end);//将[beg, end)区间中的数据拷贝赋值给本身
 vector<int> v4;
 v4.assign(v3.begin(), v3.end());
 printVectorInt(v4);
```

```
34
35  //swap(vec);// 将vec与本身的元素互换。
36  vector<int> v5(5,20);
37  vector<int> v6(10,40);
38  printVectorInt(v5);
39  printVectorInt(v6);
40  v5.swap(v6);
41  printVectorInt(v5);
42  printVectorInt(v6);
43 }
```

运行结果：

```
10 10 10 10 10
100 100 100 100 100
100 100 100 100 100
20 20 20 20 20
40 40 40 40 40 40 40 40 40 40
40 40 40 40 40 40 40 40 40 40
20 20 20 20 20
```

2、vector大小操作

案例：1

```
1  /*
2  3.2.4.3 vector大小操作
3  size();//返回容器中元素的个数
4  empty();//判断容器是否为空
5  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。
6  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。
7  capacity();//容器的容量
8  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
9  */
10  void test02()
11  {
```

```
 vector<int> v;
 v.push_back(10);
 v.push_back(20);
 v.push_back(30);
 v.push_back(40);

 if(v.empty())
 {
 cout<<"v容器为空"<<endl;
 }
 else
 {
 cout<<"容器非空"<<endl;
 cout<<"size = "<<v.size()<<endl;
 cout<<"capacity = "<<v.capacity()<<endl;
 //容量 >= size
 }

 printVectorInt(v);//10 20 30 40
 //resize(int num);//重新指定容器的长度为num
 //多出的部分 自动补0
 v.resize(8);
 printVectorInt(v);//10 20 30 40 0 0 0 0

 //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
 v.resize(10,5);
 printVectorInt(v);//10 20 30 40 0 0 0 0 5 5

 v.resize(2);
 printVectorInt(v);//10 20
}
```

案例2：resize 作用的容器的大小 不会更改容器的容量

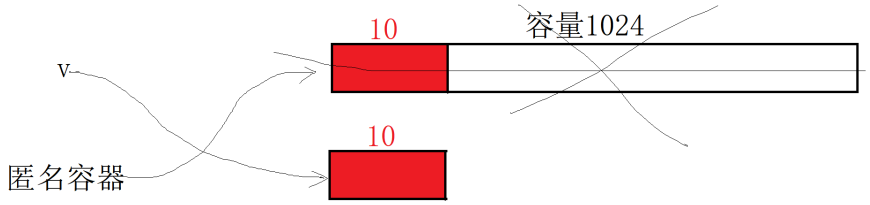
```
void test03()
{
 vector<int> v;
 v.push_back(10);
 v.push_back(20);
 v.push_back(30);
 v.push_back(40);
```

```
8   v.push_back(50);
9   v.push_back(60);
10
11   cout<<"size = "<<v.size()<<endl;
12   cout<<"capactiy = "<<v.capacity()<<endl;
13   printVectorInt(v);
14
15   cout<<"----------------"<<endl;
16   v.resize(2);
17   cout<<"size = "<<v.size()<<endl;
18   cout<<"capactiy = "<<v.capacity()<<endl;
19   printVectorInt(v);
20  }
```

运行结果：

```
size = 6
capactiy = 8
10 20 30 40 50 60
----------------

size = 2
capactiy = 8
10 20
```

3、使用resize swap收缩容器的容量



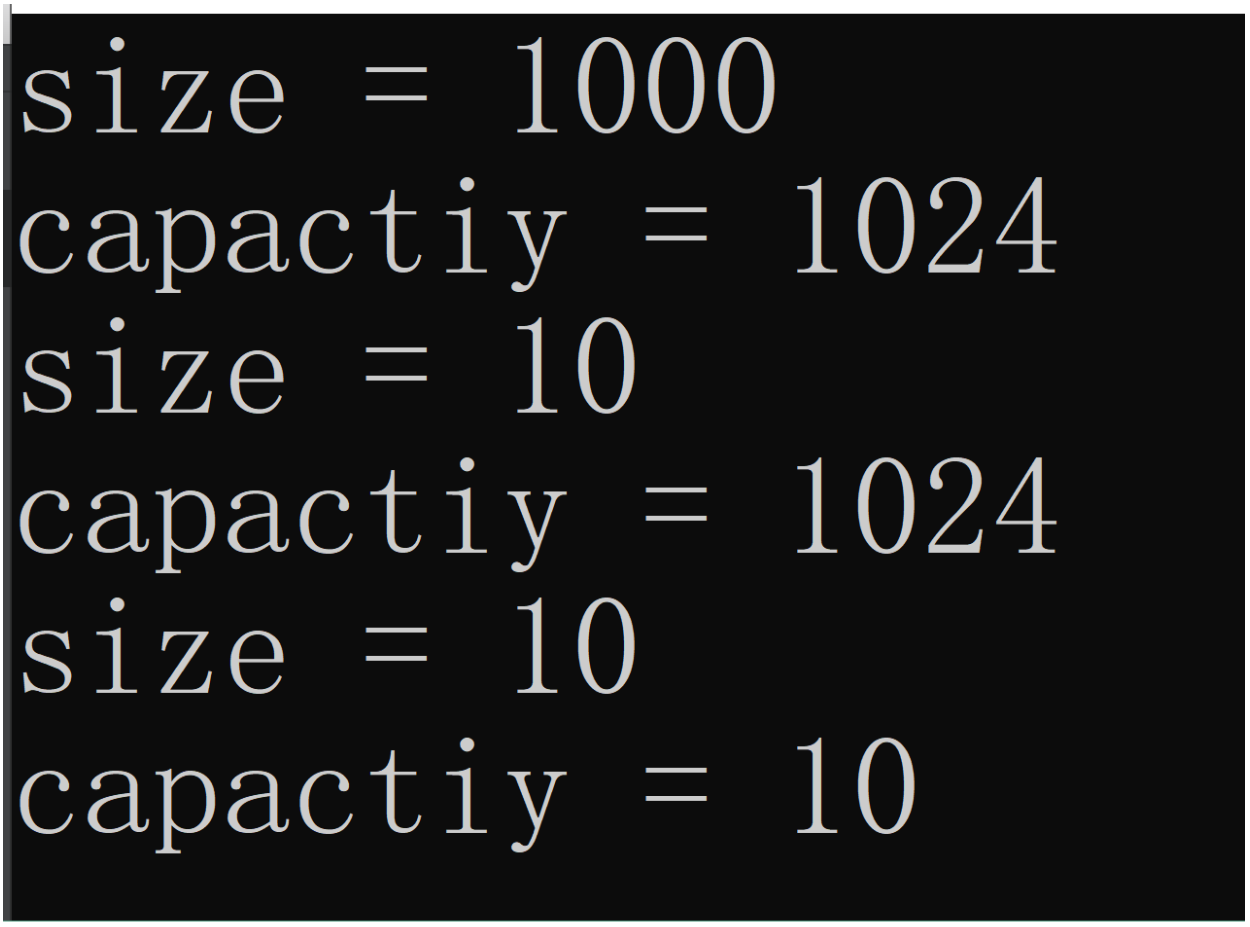
v.resize(10);

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

将v的实际元素 赋值给 匿名容器

```
void test04()
{
 vector<int> v;
 for(int i=0;i<1000;i++)
 {
 v.push_back(i);
 }
 cout<<"size = "<<v.size()<<endl;//1000
 cout<<"capactiy = "<<v.capacity()<<endl;//1024

 //使用reszie将空间 置成10个元素（可以吗?）
 v.resize(10);//不能修改容量 只能修改size
 cout<<"size = "<<v.size()<<endl;//10
 cout<<"capactiy = "<<v.capacity()<<endl;//1024

 //使用swap收缩容器的容量
 vector<int>(v).swap(v);

 cout<<"size = "<<v.size()<<endl;//10
 cout<<"capactiy = "<<v.capacity()<<endl;//10
}
```

运行结果：



4、reserve预留空间大小

```
//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
void test05()
{
 vector<int> v;

 //一次性 给够空间 叫空间预留
 v.reserve(1000);//预留空间 1000个元素

 int *p = NULL;
  int count = 0;
  for(int i=0;i<1000;i++)
  {
  v.push_back(i);
  if(p != &v[0])
  {
  count++;
  p = &v[0];
  }
  }
  cout<<"重新另寻空间次数:"<<count<<endl;
```

```
}
```

运行结果：

如果没有：v.reserve(1000); 结果为11

如果有:v.reserve(1000); 结果为1

5、数据的存取

```
/*
3.2.4.4 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
*/
void test06()
{
  vector<int> v;
  v.push_back(10);
  v.push_back(20);
  v.push_back(30);
  v.push_back(40);

  printVectorInt(v);//10 20 30 40
  cout<<v[2]<<endl;//30
  cout<<v.at(2)<<endl;//30
  //[] 越界 不抛出异常
  //at 越界 抛出异常

  cout<<"front = "<<v.front()<<endl;//10
  cout<<"back = "<<v.back()<<endl;//40
}
```

6、vector容器的插入和删除

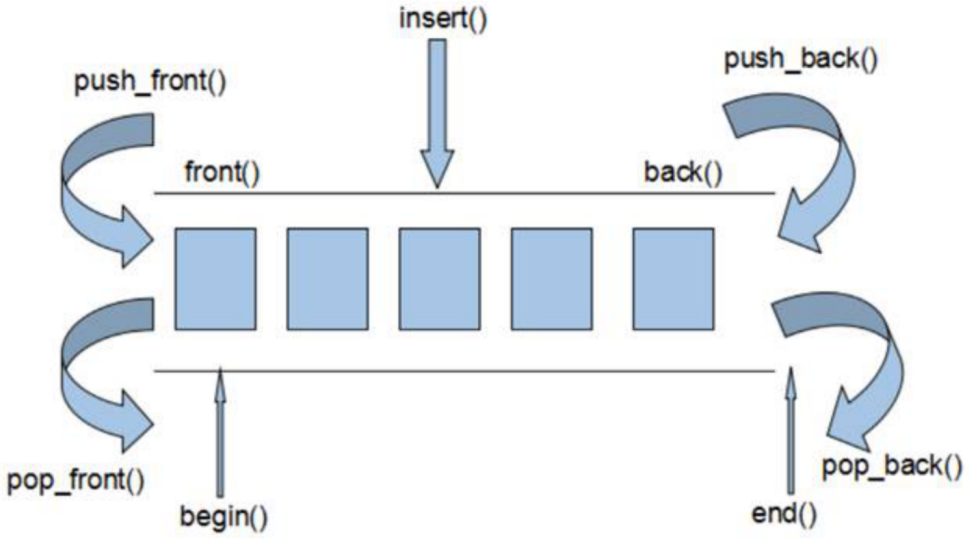
```
/*
3.2.4.5 vector插入和删除操作
insert(const_iterator pos, int count,ele);//迭代器指向位置pos插入count个元素ele.
push_back(ele); //尾部插入元素ele
pop_back();//删除最后一个元素
```

```
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
*/
 void test07()
 {
  vector<int> v;
  v.push_back(10);
  v.push_back(20);
  v.push_back(30);
  v.push_back(40);
  printVectorInt(v);//10 20 30 40

  //insert(const_iterator pos, int count,ele);
  //迭代器指向位置pos插入count个元素ele.
  v.insert(v.begin()+2,3,100);
  printVectorInt(v);//10 20 100 100 100 30 40

  //尾部删除：pop_back();//删除最后一个元素
  v.pop_back();//将40删除了
  printVectorInt(v);//10 20 100 100 100 30

  //erase(const_iterator start, const_iterator end);
  //删除迭代器从start到end之间的元素
  v.erase(v.begin()+2, v.end()-1);
  printVectorInt(v);//10 20 30

  //erase(const_iterator pos);//删除迭代器指向的元素
  v.erase(v.begin()+1);//删除20的位置
  printVectorInt(v);//10 30

  cout<<"size = "<<v.size()<<", capacity = "<<v.capacity()<<endl;

  //clear();//删除容器中所有元素
  v.clear();
  printVectorInt(v);//啥也没有
  cout<<"size = "<<v.size()<<", capacity = "<<v.capacity()<<endl;
 }
```

运行结果：

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
10 20 30 40
10 20 100 100 100 30 40
10 20 100 100 100 30
10 20 30
10 30
size = 2, capacity = 8

size = 0, capacity = 8
```

知识点2【deque容器 双端动态数组】

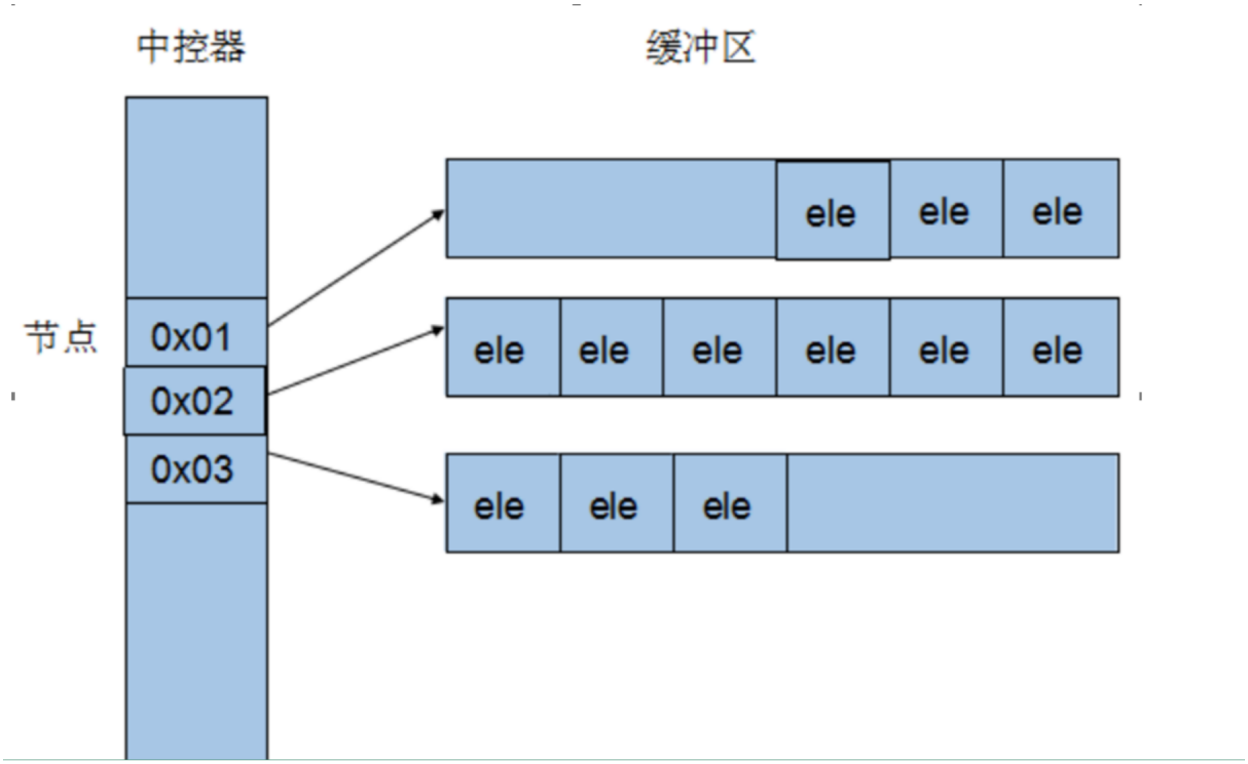


1、deque容器和vector容器最大的差异：

a、deque插入和删除 常数项时间（不会因为数据量的大小 改变操作所耗时）

b、deque容器没有容量的概念 以分段连续空间组成。

c、Deque是由一段一段的定量的连续空间构成。



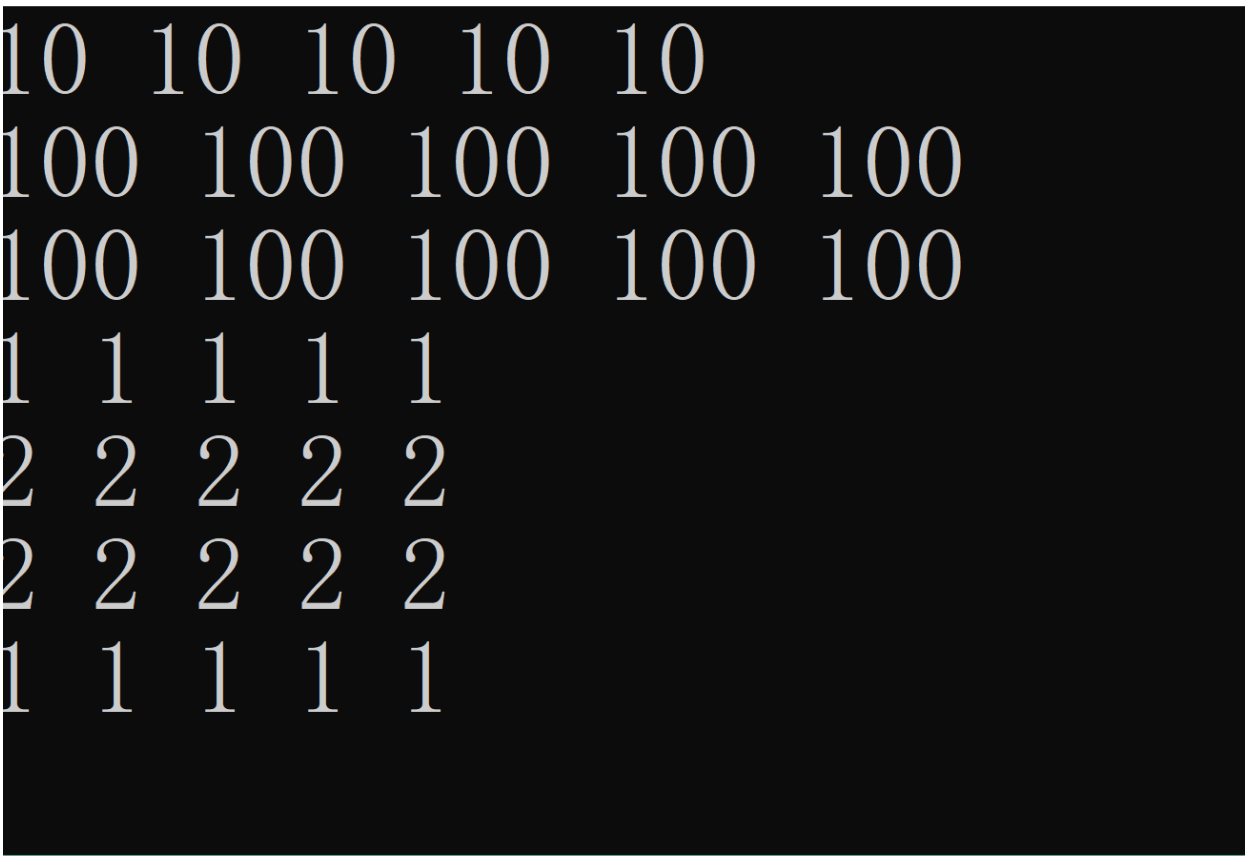
知识点3【deque容器的API】

1、deque容器的构造 和赋值

```
/*
3.3.3.1 deque构造函数
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。
3.3.3.2 deque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。
deque& operator=(const deque &deq); //重载等号操作符
swap(deq);// 将deq与本身的元素互换
*/
void printDequeInt(deque<int> &d)
{
 for(deque<int>::iterator it=d.begin();it!=d.end();it++)
 {
 cout<<*it<<" ";
 }
 cout<<endl;
}
void test01()
```

```
{
 deque<int> d(5,10);
 printDequeInt(d);//10 10 10 10 10

 //assign(n, elem);//将n个elem拷贝赋值给本身。
 deque<int> d1;
 d1.assign(5,100);
 printDequeInt(d1);//100 100 100 100 100

 //deque& operator=(const deque &deq); //重载等号操作符
 deque<int> d2;
 d2 = d1;
 printDequeInt(d2);//100 100 100 100 100

 //swap(deq);// 将deq与本身的元素互换
 deque<int> d3(5,1);
 deque<int> d4(5,2);
 printDequeInt(d3);//1 1 1 1 1
 printDequeInt(d4);//2 2 2 2 2
 d3.swap(d4);
 printDequeInt(d3);//2 2 2 2 2
 printDequeInt(d4);//1 1 1 1 1
}
```

运行结果：

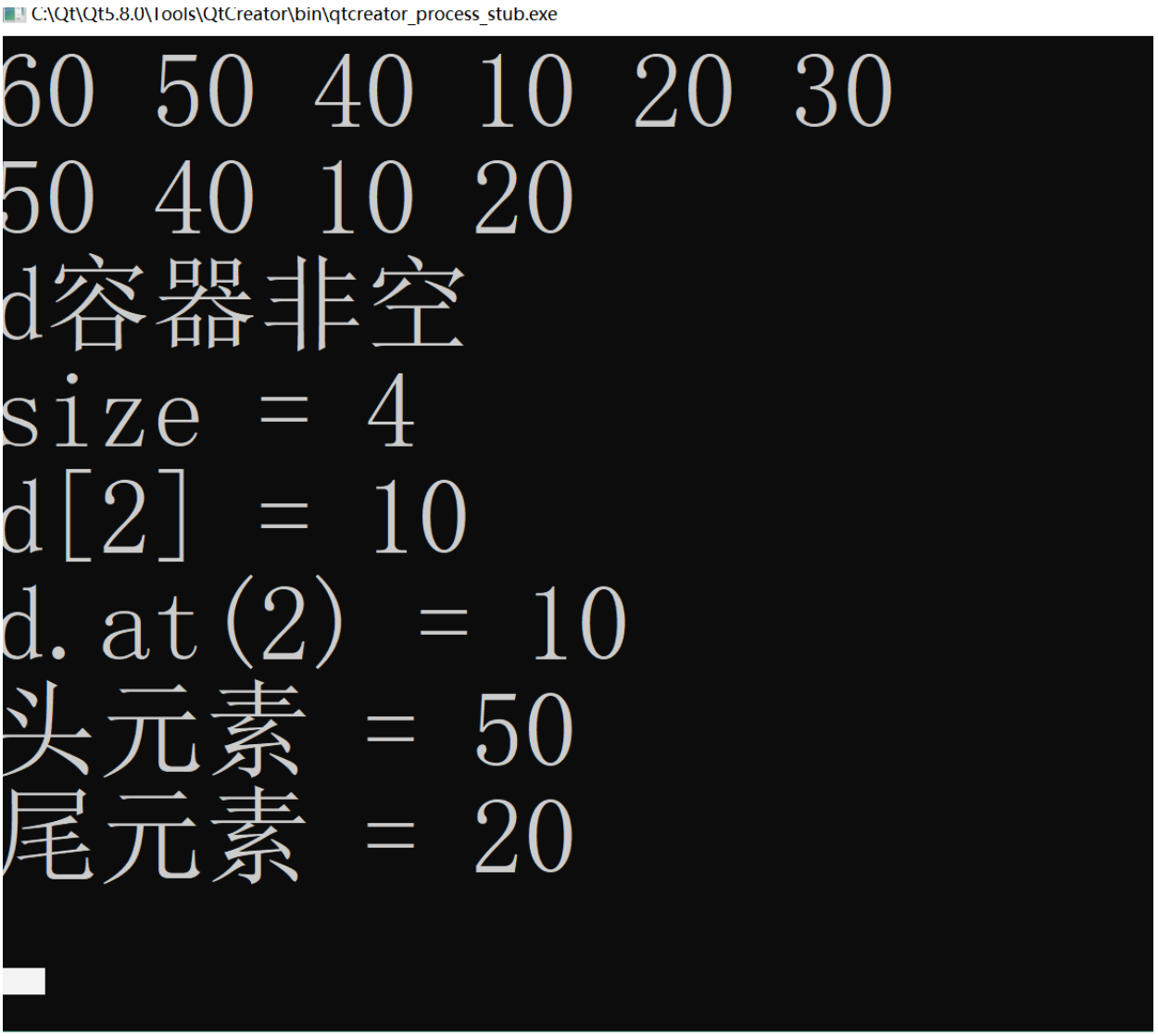


2、deque容器的大小操作、双端插入删除操作、元素访问操作

```
/*
3.3.3.3 deque大小操作
deque.size();//返回容器中元素的个数
deque.empty();//判断容器是否为空
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。
3.3.3.4 deque双端插入和删除操作
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
3.3.3.5 deque数据存取
at(idx);//返回索引idx所指的数据，如果idx越界，抛出out_of_range。
operator[];//返回索引idx所指的数据，如果idx越界，不抛出异常，直接出错。
front();//返回第一个数据。
back();//返回最后一个数据
*/
void test02()
{
```

```
20  deque<int> d;
21  //尾部插入
22  d.push_back(10);
23  d.push_back(20);
24  d.push_back(30);//10 20 30
25
26  //头部插入
27  d.push_front(40);
28  d.push_front(50);
29  d.push_front(60);
30  printDequeInt(d);//60 50 40 10 20 30
31
32  //头部删除
33  d.pop_front();//50 40 10 20 30
34  //尾部删除
35  d.pop_back();//50 40 10 20
36  printDequeInt(d);//50 40 10 20
37
38  if(d.empty())
39  {
40  cout<<"d容器为空"<<endl;
41  }
42  else
43  {
44  cout<<"d容器非空"<<endl;
45  cout<<"size = "<<d.size()<<endl;//4
46  }
47
48  //[]访问第二个元素
49  cout<<"d[2] = "<<d[2]<<endl;//10
50  cout<<"d.at(2) = "<<d.at(2)<<endl;//10
51  cout<<"头元素 = "<<d.front()<<endl;//50
52  cout<<"尾元素 = "<<d.back()<<endl;//20
53 }
```

运行结果：

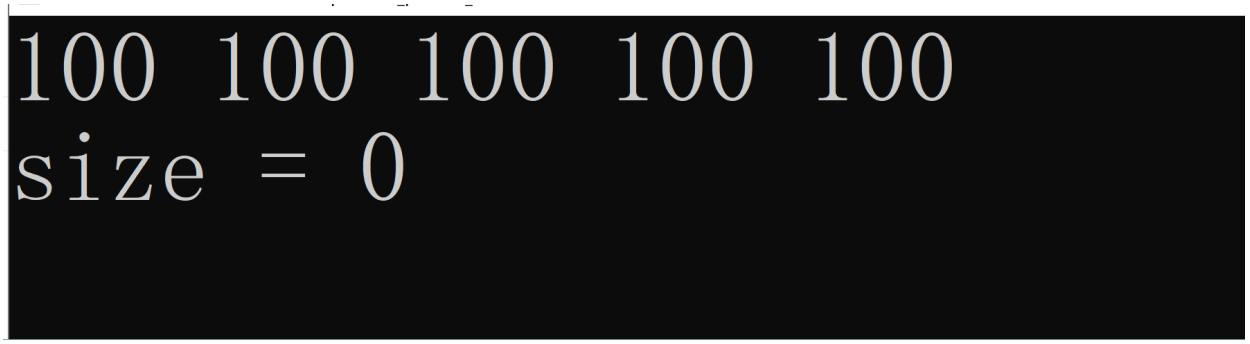


3、容器的插入删除

```
/*
3.3.3.6 deque插入操作
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。
3.3.3.7 deque删除操作
clear();//移除容器的所有数据
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
*/

void test03()
{
 deque<int> d;
 d.insert(d.begin(),5, 100);
 printDequeInt(d);//100 100 100 100 100
```

```
17
18   d.clear();
19   cout<<"size = "<<d.size()<<endl;//0
20  }
```

运行结果：



4、案例：有5名选手：选手ABCDE，10个评委分别对每一名选手打分，去除最高分，去除评委中最低分，取平均分，最后输出每位选手的分数

步骤分析：

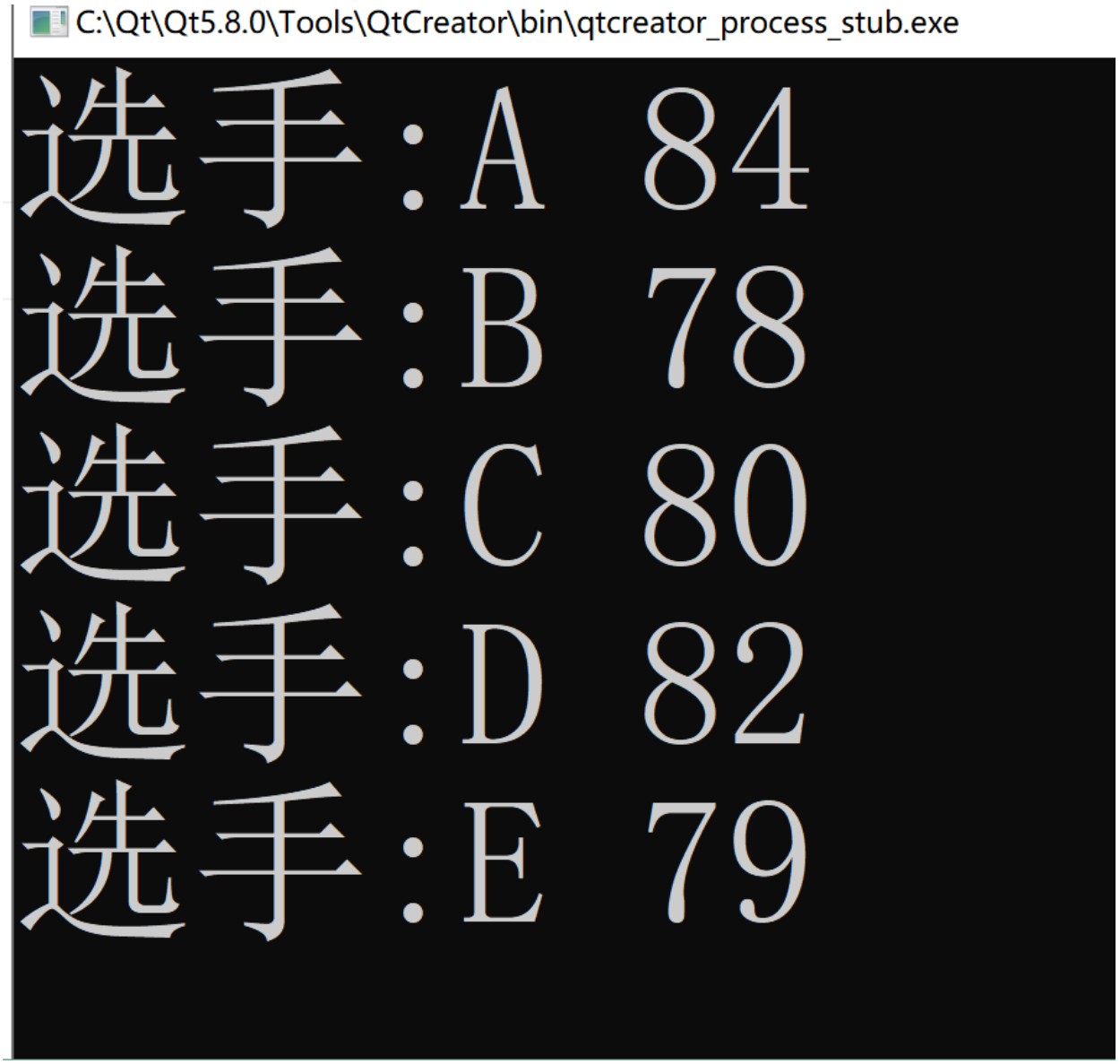
```
1. 创建五名选手，放到vector中
2. 遍历vector容器，取出来每一个选手，执行for循环，可以把10个评分打分存到deque容器中
3. sort算法对deque容器中分数排序，pop_back pop_front去除最高和最低分
4. deque容器遍历一遍，累加分数，累加分数/d.size()
5. person.score = 平均分
```

```
#include <iostream>
#include<string>
#include<vector>
#include<deque>
#include<stdlib.h>
#include<time.h>
#include<algorithm>
using namespace std;

//选手类
class Person
{
public:
 string name;
 int score;
 Person(string name,int score)
```

```
 {
 this->name = name;
 this->score =score;
 }
};
void createPerson(vector<Person> &v)
{
 //5名选手是ABCDE
 string nameTmp="ABCDE";
 for(int i=0;i<5;i++)
 {
 string name="选手:";
 name += nameTmp[i];

 //将选手的姓名 分数0 放入vector容器中
 v.push_back(Person(name,0));
 }
}
void printVectorPerson(vector<Person> &v)
{
 for(vector<Person>::iterator it=v.begin();it!=v.end();it++)
 {
 //*it == Person
 cout<<(*it).name<<" "<<(*it).score<<endl;
 }
}
void playGame(vector<Person> &v)
{
 //设置随机种子
 srand(time(NULL));

 //容器v中的每个人 逐一比赛
 for(vector<Person>::iterator it=v.begin();it!=v.end();it++)
 {
 //*it == Person
 //每位选手 都要被10个评委打分 放入deque容器中
 deque<int> d;
 for(int i=0;i<10;i++)//10个评委
 {
 int score = rand()%41+60;//60~100
```

```
 d.push_back(score);
 }

 //对deque容器（评委的10个分数）排序
 sort(d.begin(), d.end());//去掉一个最低分 最高分

 //去掉一个最低分
 d.pop_front();
 //去掉一个最高分
 d.pop_back();

 //得到每个选手的总分数
 int sum = accumulate(d.begin(),d.end(), 0);

 //获取平均分 赋值 选手的score
 (*it).score = sum/d.size();
 }
}
int main(int argc, char *argv[])
{
 //1、定义一个vector容器存放5名选手
 vector<Person> v;
 createPerson(v);

 //2、5名选手 逐一比赛
 playGame(v);

 //3、将5名选手的成绩打印出来
 printVectorPerson(v);
 return 0;
}

```

运行结果：

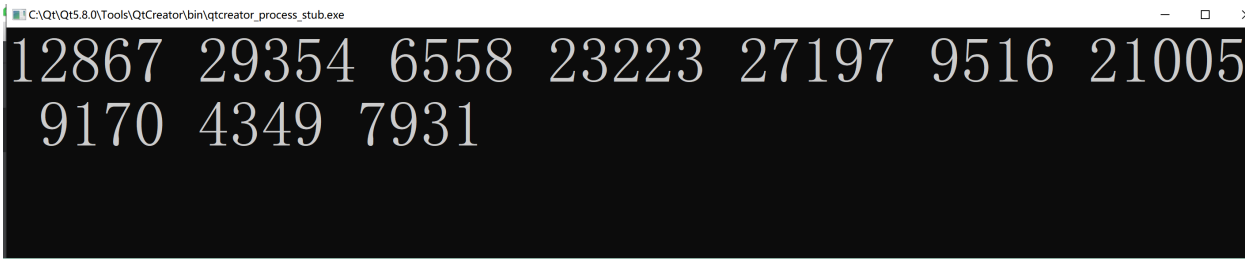


5、srand设置随机种子 rand 产生随机数

```
#include <iostream>
#include<time.h>
using namespace std;

int main(int argc, char *argv[])
{
 //设置随机数种子time(NULL)获取当前时间
 srand(time(NULL));

 for(int i=0;i<10; i++)
 {
 //rand()函数的返回值就是随机数
 int num = rand();
 cout<<num<<" ";
```

```
15    }
16    cout<<endl;
17    return 0;
18  }
```

运行结果：

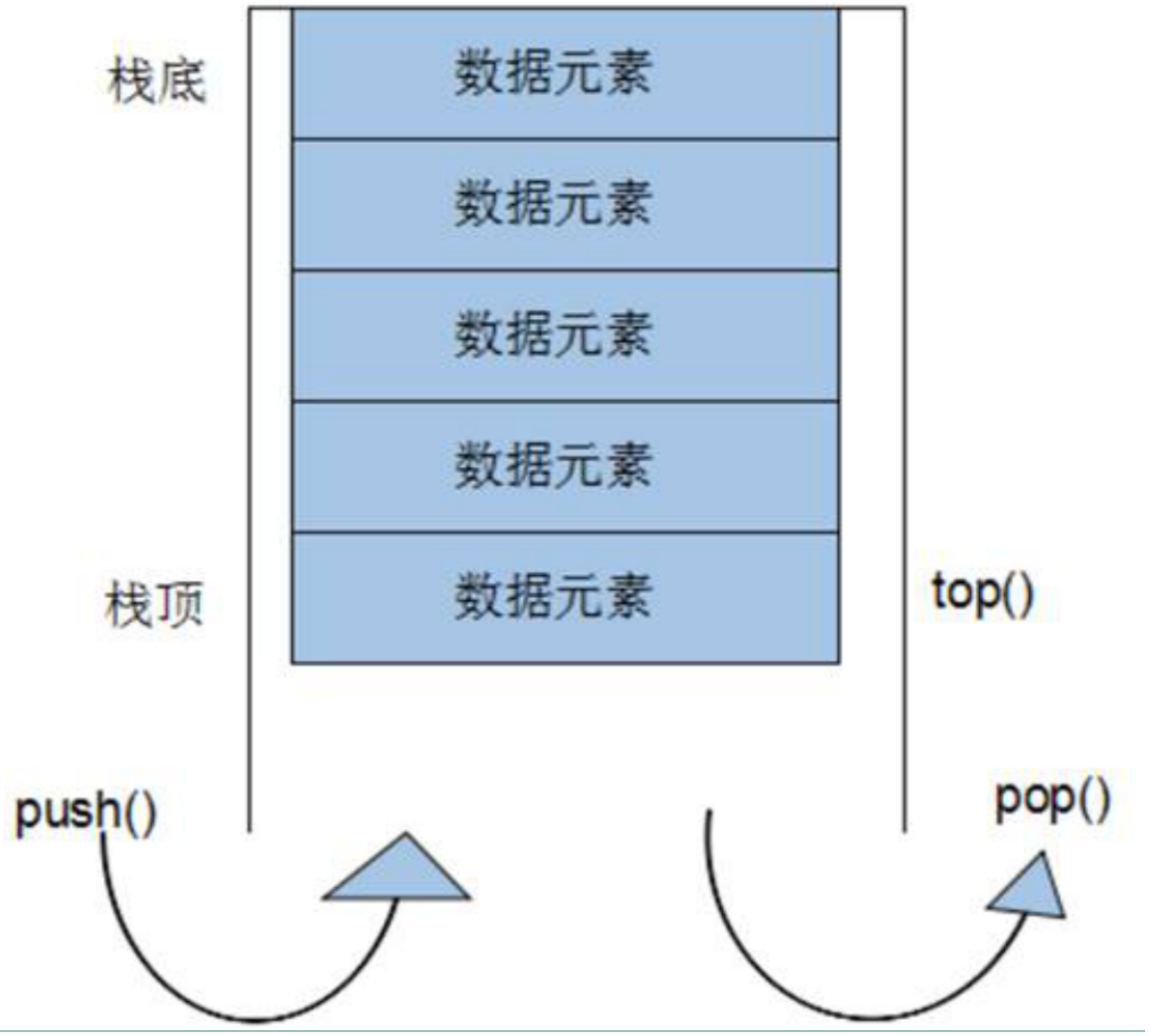


知识点4【stack容器 栈容器】

先进后出的数据结构。

push入栈、pop出栈、top永远指向栈顶元素

stack没有迭代器



```cpp
/*
3.4.3.1 stack构造函数
stack<T> stkT;//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
3.4.3.2 stack赋值操作
stack& operator=(const stack &stk);//重载等号操作符
3.4.3.3 stack数据存取操作
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
 3.4.3.4 stack大小操作
 empty();//判断堆栈是否为空
 size();//返回堆栈的大小
 */
 void test01()
 {
  stack<int> s;
  //入栈
  s.push(10);
  s.push(20);
  s.push(30);
  s.push(40);

  if(s.empty())
  {
  cout<<"栈容器为空"<<endl;
  }
  else
  {
  cout<<"栈容器非空"<<endl;
  cout<<"size = "<<s.size()<<endl;
  }

  while(!s.empty())//非空 返回false
  {
  cout<<s.top()<<endl;
  //出栈
  s.pop();
  }
```

```
40  }
```

运行结果：

```
栈容器非空
size = 4
40
30
20
10
```

知识点5【队列容器 queue】

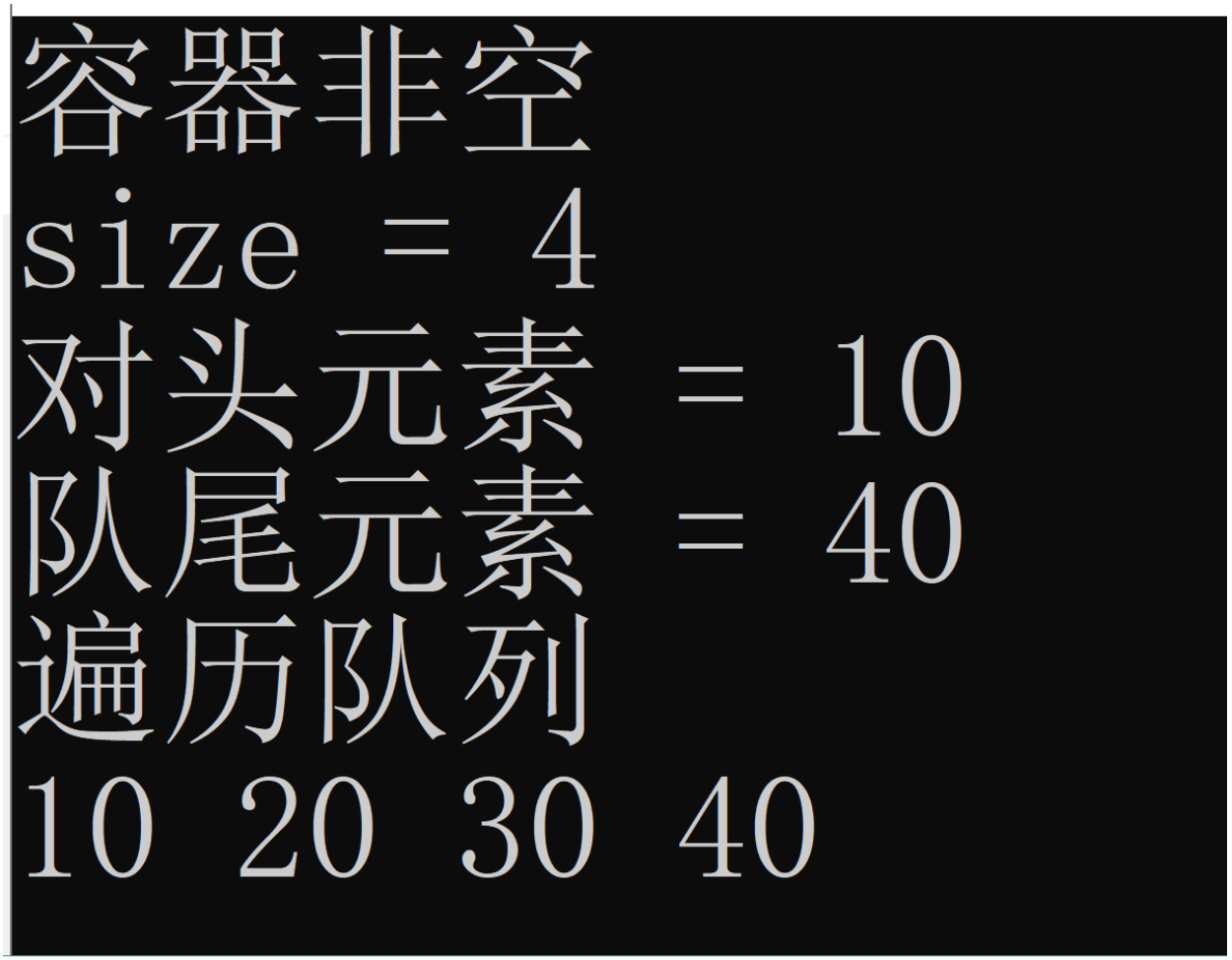
队列容器：先进先出 队尾插入数据 对头删除数据

queue容器：没有迭代器 不具备遍历功能 只能通过front、back访问

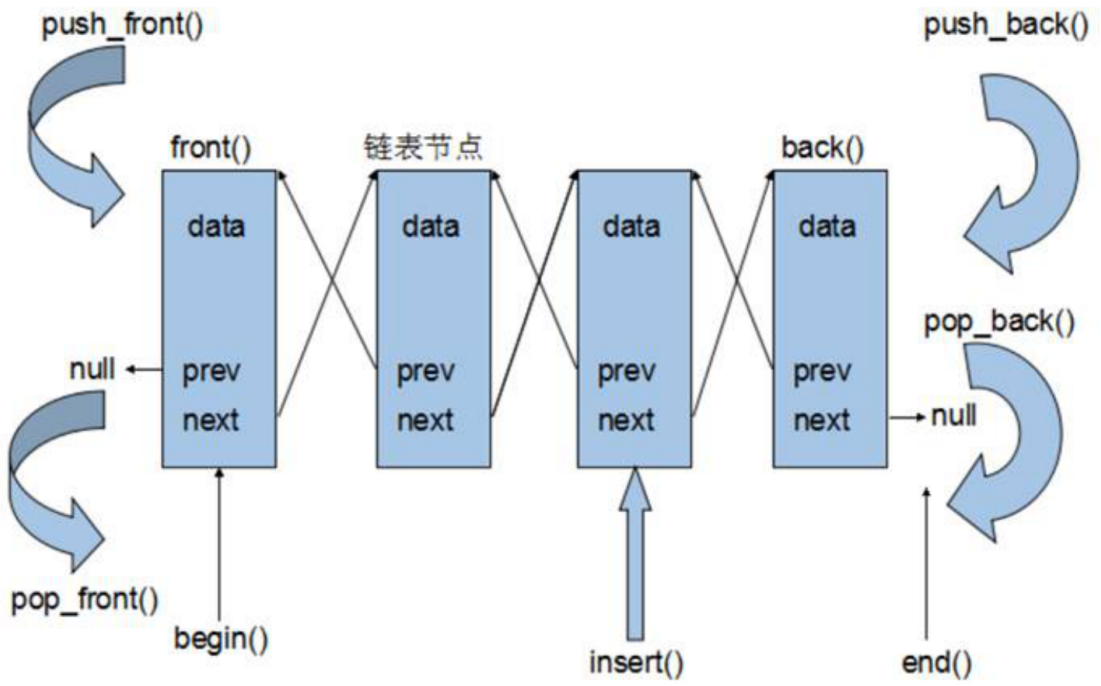
```
/*
3.5.3.1 queue构造函数
queue<T> queT;//queue采用模板类实现，queue对象的默认构造形式：
queue(const queue &que);//拷贝构造函数
3.5.3.2 queue存取、插入和删除操作
push(elem);//往队尾添加元素
pop();//从队头移除第一个元素
back();//返回最后一个元素
front();//返回第一个元素
 3.5.3.3 queue赋值操作
 queue& operator=(const queue &que);//重载等号操作符
 3.5.3.4 queue大小操作
 empty();//判断队列是否为空
 size();//返回队列的大小
 */
 #include<queue>
 void test02()
 {
  queue<int> q;
  q.push(10);
  q.push(20);
  q.push(30);
  q.push(40);

  if(q.empty())
  {
  cout<<"容器为空"<<endl;
  }
  else
  {
  cout<<"容器非空"<<endl;
  cout<<"size = "<<q.size()<<endl;
  cout<<"对头元素 = "<<q.front()<<endl;//10
  cout<<"队尾元素 = "<<q.back()<<endl;//40
  }

  cout<<"遍历队列"<<endl;
  while(!q.empty())
  {
```

```
40    cout<<q.front()<<" ";
41    q.pop();//出队
42    }
43  }
```

运行结果：



知识点6【list容器】链表容器



list容器（双向链表）的迭代器 必须具备前移、后移 提供的是双向迭代器

vector 单端动态数组（随机访问迭代器）

deque:双端动态数组（随机访问迭代器）

stack：栈容器（没有迭代器）

queue：队列容器（没有迭代器）

list：链表容器（双向迭代器）

知识点7【链表的API】

1、list的常规操作

```
#include <iostream>

using namespace std;
/*
3.6.4.1 list构造函数
list<T> lstT;//list采用采用模板类实现,对象的默认构造形式:
list(beg,end);//构造函数将[beg, end)区间中的元素拷贝给本身。
list(n,elem);//构造函数将n个elem拷贝给本身。
list(const list &lst);//拷贝构造函数。
 3.6.4.2 list数据元素插入和删除操作
 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);//在容器开头插入一个元素
 pop_front();//从容器开头移除第一个元素
```

```
insert(pos,elem);//在pos位置插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();//移除容器的所有数据
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。
remove(elem);//删除容器中所有与elem值匹配的元素。
3.6.4.3 list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(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(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。
3.6.4.4 list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。
list& operator=(const list &lst);//重载等号操作符
swap(lst);//将lst与本身的元素互换。
3.6.4.5 list数据的存取
front();//返回第一个元素。
back();//返回最后一个元素。
3.6.4.6 list反转排序
reverse();//反转链表，比如lst包含1,3,5元素，运行此方法后，lst就包含5,3,1元素。
sort(); //list排序
*/
#include<list>
#include<algorithm>
void printListInt(list<int> &L)
{
 for(list<int>::iterator it=L.begin();it!=L.end();it++)
 {
 cout<<(*it)<<" ";
 }
 cout<<endl;
}
void test01()
```

```
{
 list<int> L;
 L.push_back(10);
 L.push_back(20);
 L.push_back(30);
 L.push_back(40);

 printListInt(L);//10 20 30 40
 //迭代器+n 只有随机访问迭代器支持
 //而list容器的迭代器是双向迭代器 不支持+n
 //L.insert(L.begin()+2, 3, 100);//err
 list<int>::iterator it = L.begin();
 //++ 随机访问迭代器 以及 双向迭代器 都支持
 it++;
 it++;
 L.insert(it,3,100);
 printListInt(L);//10 20 100 100 100 30 40

 //remove(elem);//删除容器中所有与elem值匹配的元素。
 L.remove(100);//删除所有100
 printListInt(L);//10 20 30 40

 //链表反转
 L.reverse();
 printListInt(L);//40 30 20 10

 //sort是系统提供的算法 仅支持 随机访问迭代器（不支持list）
 //list容器不能使用系统算法 list会自己提供算法
 //sort(L.begin(),L.end());
 L.sort();
 printListInt(L);//10 20 30 40
}
int main(int argc, char *argv[])
{
 test01();
 return 0;
}

```

运行结果：

```
10 20 30 40
10 20 100 100 100 30 40
10 20 30 40
40 30 20 10
10 20 30 40
```