

知识点1【STL的概述】（了解）

知识点2【string容器】（了解）

- 1、构造函数以及赋值
 - 2、string存取字符操作
 - 3、string拼接操作
 - 4、string查找和替换
 - 5、string比较操作
 - 6、提取string子串
 - 7、string插入和删除操作
 - 8、string和c-style字符串转换
-

知识点3【vector容器】（重要）

- 1、vector的概述
 - 2、vector API
 - 3、巧用swap收缩空间
 - 4、vector容器 嵌套 容器
 - 5、使用算法 对 vector容器排序
 - 6、vector存放自定义数据类型
-

知识点4【deque容器】（了解）

- 1、deque概述
 - 2、deque的API
 - 3、deque容器的案例
-

知识点5【stack栈容器】（了解）

知识点6【queue队列容器】（了解）

知识点7【list链表容器】（了解）

知识点8【set容器】（了解）

1、set容器概述

2、更改set容器的排序规则（定义set容器时 修改）

3、如果set容器存放自定义数据 必须更改排序规则

4、set的API

5、查找元素的上下限

案例2：

知识点9【multiset容器】（了解）

知识点10【pair对组】（了解）

知识点11【map容器】（重要）

案例1：键值的map容器设计

知识点12【multimap案例】

知识点1【STL的概述】（了解）

STL(Standard Template Library,标准模板库)

STL的6大组件：容器、算法、迭代器、适配器、仿函数、空间配置

容器：存放数据

算法：操作数据

迭代器：算法 通过迭代器 操作 容器

适配器：为算法 提供更多的接口

仿函数：为算法 提供策略

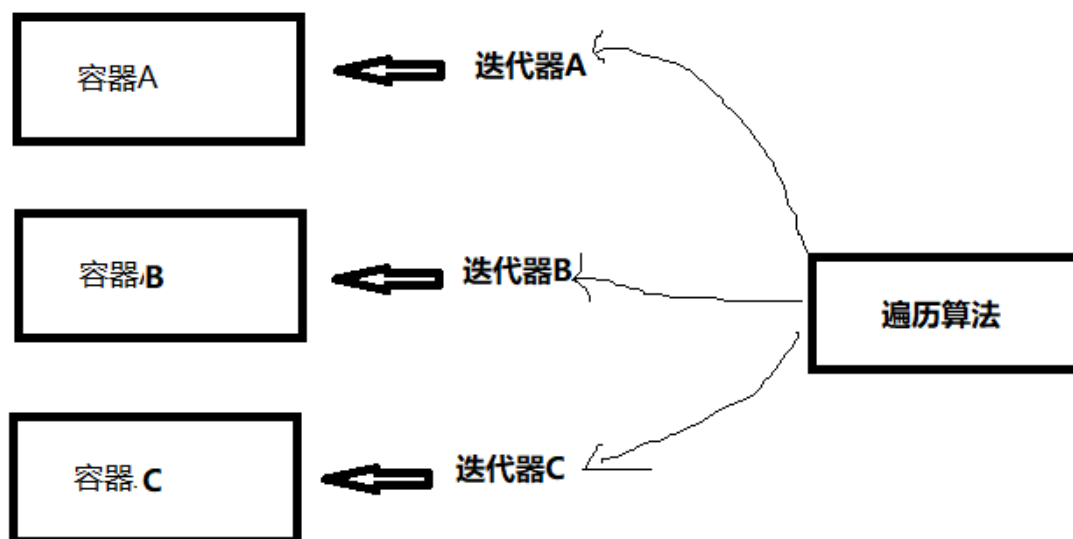
空间配置：为算法、容器提供动态空间

算法分类：质变算法、非质变算法

质变算法：会更改容器的值（拷贝，替换，删除等等）

非质变算法：是指运算过程中不会更改区间内的元素内容，例如查找、计数、遍历、寻找极值等等

迭代器：算法和容器的桥梁



容器 和 迭代器 一一对应的。

知识点2【string容器】（了解）

1、构造函数以及赋值

```
1 3.1.2.1 string 构造函数
2 string();//创建一个空的字符串 例如: string str;
3 string(const string& str);//使用一个string对象初始化另一个string对象
4 string(const char* s);//使用字符串s初始化
5 string(int n, char c);//使用n个字符c初始化 v
6 3.1.2.2 string基本赋值操作
7 string& operator=(const char* s);//char*类型字符串 赋值给当前的字符串
8 string& operator=(const string &s);//把字符串s赋给当前的字符串
9 string& operator=(char c);//字符赋值给当前的字符串
10 string& assign(const char *s);//把字符串s赋给当前的字符串
11 string& assign(const char *s, int n);//把字符串s的前n个字符赋给当前的字符串
12 string& assign(const string &s);//把字符串s赋给当前字符串
13 string& assign(int n, char c);//用n个字符c赋给当前字符串
14 string& assign(const string &s, int start, int n);//将s从start开始n个字符
    赋值给字符串
```

```
1 void test01()
2 {
3     string str1("hello world");
```

```

4  cout<<str1<<endl;
5  string str2(5, 'A');
6  cout<<str2<<endl;
7  string str3 = str2;
8  cout<<str3<<endl;
9
10 string str4;
11 str4 = "hello world";
12 cout<<str4<<endl;
13 str4 = 'W';
14 cout<<str4<<endl;
15
16 str4.assign("hello world", 5);
17 cout<<str4<<endl;
18 str4.assign(str1, 2, 3);
19 cout<<str4<<endl;
20 }

```

```

1
hello world
AAAAA
AAAAA
hello world
W
hello
llo

```

2、string存取字符操作

```

1 char& operator[](int n); //通过[]方式取字符
2 char& at(int n); //通过at方法获取字符

```

```

1 void test02()
2 {
3     string str1="hello world";
4     cout<<str1[1]<<" "<<str1.at(1)<<endl;
5     str1[1]='E';
6     str1.at(6)='H';
7     cout<<str1<<endl;

```

```

8
9 //[] 越界不会抛出异常 at越界会抛出异常
10 try
11 {
12 //str1[1000]='A';
13 str1.at(1000)='A';
14 }
15 catch(exception &e)
16 {
17 cout<<"捕获到异常:"<<e.what()<<endl;
18 }
19 }

```

```

e e
hEllo World
捕获到异常:basic_string::at: __n (which is 1000) >= this->size() (which is 11)

```

3、string拼接操作

```

1 3.1.2.4
2 string& operator+=(const string& str);//重载+=操作符
3 string& operator+=(const char* str);//重载+=操作符
4 string& operator+=(const char c);//重载+=操作符
5 string& append(const char *s);//把字符串s连接到当前字符串结尾
6 string& append(const char *s, int n);//把字符串s的前n个字符连接到当前字符串
  结尾
7 string& append(const string &s);//同operator+=()
8 string& append(const string &s, int pos, int n);//把字符串s中从pos开始的n个
  字符连接到当前字符串结尾
9 string& append(int n, char c);//在当前字符串结尾添加n个字符c

```

```

1 void test03()
2 {
3 string str1="hello";
4 str1 += "world";
5 cout<<str1<<endl;
6
7 string str2="hehe";
8 str1 += str2;
9 cout<<str1<<endl;
10
11 string str3="hello";
12 string str4="world";

```

```

13  cout<<str3+str4<<endl;
14
15  string str5="hello";
16  string str6="world";
17  str5.append(str6, 2, 3);
18  cout<<str5<<endl;
19  str5.append("world", 3);
20  cout<<str5<<endl;
21  }

```

```

helloworld
helloworldhehe
helloworld
hellorlrd
hellorlrdwor

```

4、string查找和替换

```

1  int find(const string& str, int pos = 0) const; //查找str第一次出现位置,从pos开始查找
2  int find(const char* s, int pos = 0) const; //查找s第一次出现位置,从pos开始查找
3  int find(const char* s, int pos, int n) const; //从pos位置查找s的前n个字符第一次位置
4  int find(const char c, int pos = 0) const; //查找字符c第一次出现位置
5  int rfind(const string& str, int pos = npos) const; //查找str最后一次位置,从pos开始查找
6  int rfind(const char* s, int pos = npos) const; //查找s最后一次出现位置,从pos开始查找
7  int rfind(const char* s, int pos, int n) const; //从pos查找s的前n个字符最后一次位置
8  int rfind(const char c, int pos = 0) const; //查找字符c最后一次出现位置
9  string& replace(int pos, int n, const string& str); //替换从pos开始n个字符为字符串str
10 string& replace(int pos, int n, const char* s); //替换从pos开始的n个字符为字符串s

```

```

void test04()
{
    string str1="http://www.sex.777.sex.999.sex.com";
    while(1)
    {
        int ret = str1.find("sex");
        if(ret == -1)
            break;

        str1.replace(ret,3,"***");
    }

    cout<<str1<<endl;
}

```



5、string比较操作

```


1  > < == != 运算符 可用
2  /*
3  compare函数在>时返回 1,<时返回 -1,==时返回 0。
4  比较区分大小写,比较时参考字典顺序,排越前面的越小。
5  大写的A比小写的a小。
6  */
7  int compare(const string &s) const;//与字符串s比较
8  int compare(const char *s) const;//与字符串s比较

```

```

void test05()
{
    string str1 ="hehe";
    string str2 = "haha";
    if(str1 > str2)
    {
        cout<<"大于"<<endl;
    }
    else
    {
        cout<<"不大于"<<endl;
    }
}

```



```

void test05()
{
    string str1 = "hehe";
    string str2 = "haha";
    if(str1.compare(str2) > 0 )
    {
        cout<<"大于"<<endl;
    }
    else if(str1.compare(str2) == 0)
    {
        cout<<"等于"<<endl;
    }
    else if(str1.compare(str2) < 0)
    {
        cout<<"小于"<<endl;
    }
}

```



6、提取string子串

1 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

```

1 void test06()
2 {
3     string str1="hehehe:hahaha:xixixi:lalala";
4     int pos = 0;
5     while(1)
6     {
7         int ret = str1.find(":", pos);
8         if(ret < 0)
9         {
10            string tmp = str1.substr(pos, str1.size()-pos);
11            cout<<tmp<<endl;
12            break;
13        }
14
15        string tmp = str1.substr(pos, ret-pos);
16        cout<<tmp<<endl;
17
18        pos = ret+1;
19    }
20 }

```



```
hehehe
hahaha
xixixi
lalala
```

7、string插入和删除操作

```
1 string& insert(int pos, const char* s); //插入字符串
2 string& insert(int pos, const string& str); //插入字符串
3 string& insert(int pos, int n, char c); //在指定位置插入n个字符c
4 string& erase(int pos, int n = npos); //删除从Pos开始的n个字符
```

```
void test07()
{
    string str1="hello";
    str1.insert(2,"###");
    cout<<str1<<endl;
    str1.erase(2, 3);
    cout<<str1<<endl;
    str1.erase(0,str1.size());
    cout<<str1<<endl;
}
```

删除整个字符串

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
he###llo
hello
```

8、string和c-style字符串转换

```
void test08()
{
    //char * 转换成 string （默认支持）
    string str1;
    str1 = (string)("hello");
    cout<<str1<<endl;

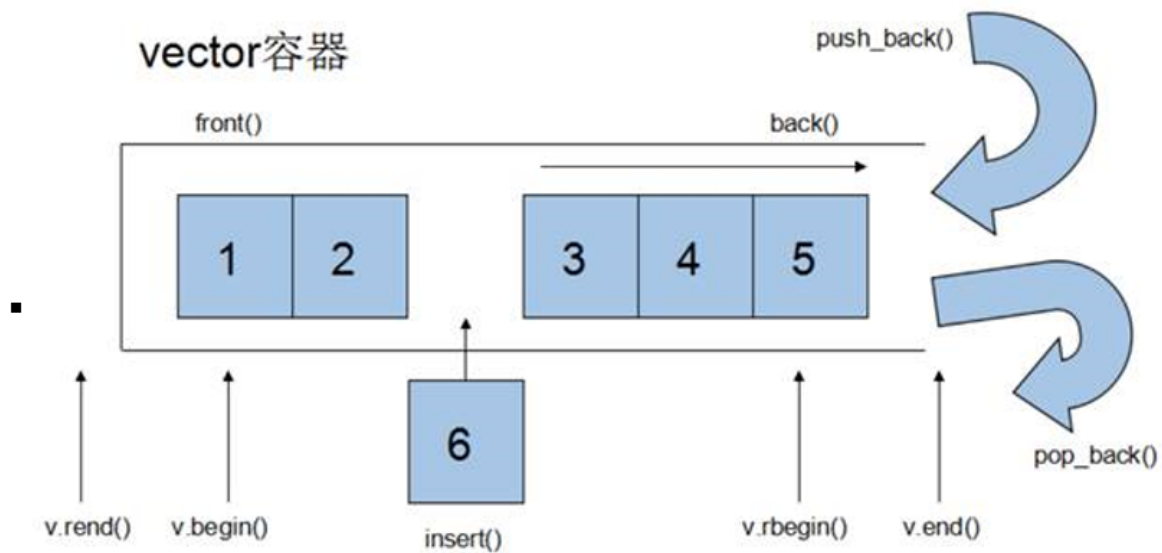
    //string 转成 char * 必须使用 成员函数c_str
    string str2="hello world";
    // char *p = const_cast<char *>(str2.c_str());
    char *p = (char *)str2.c_str();
    cout<<p<<endl;
}
```

```
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreat
hello
hello world
```

知识点3 【vector容器】（重要）

1、vector的概述

vector容器：单端动态数组容器



`push_back`尾部插入元素、`pop_back`尾部删除元素

`front()`头元素、`back()`尾元素

`begin()`得到的是 容器的 起始迭代器（首元素的位置）

`end()` 得到的是 结束迭代器（尾元素的下一个元素位置）

必须包含头文件：`#include <vector>`

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  void test01()
5  {
6      vector<int> v1;
7      v1.push_back(10);
8      v1.push_back(30);
9      v1.push_back(50);
10     v1.push_back(20);
11     v1.push_back(40);
12
13     //遍历容器
14     //定义一个迭代器iterator 保存是元素的位置
15     vector<int>::iterator it=v1.begin();
16     for(;it!=v1.end(); it++)
17     {
18         /*it == int
19         cout<<*it<<" ";
20     }
21     cout<<endl;
22 }
```

```

23
24 int main(int argc, char *argv[])
25 {
26     test01();
27     return 0;
28 }

```

```

10 30 50 20 40

```

vector的末雨绸缪机制:

```

1 void test01()
2 {
3     vector<int> v1;
4     cout<<"容量:"<<v1.capacity()<<" 大小:"<<v1.size()<<endl;
5     vector<int>::iterator it;
6     int i=0;
7     int count = 0;
8     for(i=1;i<=1000;i++)
9     {
10        v1.push_back(1);
11        if(it != v1.begin())
12        {
13            count++;
14            cout<<"第"<<count<<"次开辟空间, 容量为: "<<v1.capacity()<<endl;
15            it=v1.begin();
16        }
17    }
18 }

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
容量:0 大小:0
第1次开辟空间, 容量为: 1
第2次开辟空间, 容量为: 2
第3次开辟空间, 容量为: 4
第4次开辟空间, 容量为: 8
第5次开辟空间, 容量为: 16
第6次开辟空间, 容量为: 32
第7次开辟空间, 容量为: 64
第8次开辟空间, 容量为: 128
第9次开辟空间, 容量为: 256
第10次开辟空间, 容量为: 512
第11次开辟空间, 容量为: 1024
```

2、vector API

```
1 3.2.4.1 vector构造函数
2 vector<T> v; //采用模板实现类实现, 默认构造函数
3 vector(v.begin(), v.end()); //将v[begin(), end())区间中的元素拷贝给本身。
4 vector(n, elem); //构造函数将n个elem拷贝给本身。
5 vector(const vector &vec); //拷贝构造函数。
6
7 //例子 使用第二个构造函数 我们可以...
8 int arr[] = {2,3,4,1,9};
9 vector<int> v1(arr, arr + sizeof(arr) / sizeof(int));
10 3.2.4.2 vector常用赋值操作
11 assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。
12 assign(n, elem); //将n个elem拷贝赋值给本身。
13 vector& operator=(const vector &vec); //重载等号操作符
14 swap(vec); // 将vec与本身的元素互换。
15 3.2.4.3 vector大小操作
16 size(); //返回容器中元素的个数
17 empty(); //判断容器是否为空
18 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。
```

```

19 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。
20 capacity(); //容器的容量
21 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。
22 3.2.4.4 vector数据存取操作
23 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。
24 operator[]; //返回索引idx所指的数据，越界时，运行直接报错
25 front(); //返回容器中第一个数据元素
26 back(); //返回容器中最后一个数据元素
27 3.2.4.5 vector插入和删除操作
28 insert(const_iterator pos, int count, ele); //迭代器指向位置pos插入count个元
    素ele.
29 push_back(ele); //尾部插入元素ele
30 pop_back(); //删除最后一个元素
31 erase(const_iterator start, const_iterator end); //删除迭代器从start到end之
    间的元素
32 erase(const_iterator pos); //删除迭代器指向的元素
33 clear();

```

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 void test01()
5 {
6     vector<int> v1;
7     v1.reserve(1000); //预留空间
8
9     cout<<"容量:"<<v1.capacity()<<" 大小:"<<v1.size()<<endl;
10    vector<int>::iterator it;
11    int i=0;
12    int count = 0;
13    for(i=1;i<=1000;i++)
14    {
15        v1.push_back(1);
16        if(it != v1.begin())
17        {
18            count++;
19            cout<<"第"<<count<<"次开辟空间，容量为: "<<v1.capacity()<<endl;
20            it=v1.begin();
21        }
22    }

```

```
23 }
24 void printVectorInt(vector<int> &v)
25 {
26     vector<int>::iterator it;
27     for(it=v.begin(); it!=v.end();it++)
28     {
29         cout<<*it<<" ";
30     }
31     cout<<endl;
32 }
33
34 void test02()
35 {
36     vector<int> v1(5,100);
37     printVectorInt(v1);
38
39     vector<int> v2 = v1;
40     printVectorInt(v2);
41
42     vector<int> v3(v1.begin(), v1.end());
43     printVectorInt(v3);
44
45     vector<int> v4;
46     // v4 = v3;
47     v4.assign(10,10);
48     printVectorInt(v4);
49
50     //交换
51     v3.swap(v4);
52     printVectorInt(v3);
53     printVectorInt(v4);
54
55     cout<<"大小:"<<v4.size()<<" 容量:"<<v4.capacity()<<endl;
56     //容器是否为空
57     vector<int> v5;
58     if(v5.empty())
59     {
60         cout<<"空"<<endl;
61     }
62     else
```

```

63  {
64  cout<<"非空"<<endl;
65  }
66
67  vector<int> v6(10, 30);
68  cout<<"大小:"<<v6.size()<<" 容量:"<<v6.capacity()<<endl;
69  printVectorInt(v6);
70  //v6.resize(20);//过大补0
71  //v6.resize(20, 50);//过大补50
72  v6.resize(32);
73  cout<<"大小:"<<v6.size()<<" 容量:"<<v6.capacity()<<endl;
74  printVectorInt(v6);
75  }
76
77  int main(int argc, char *argv[])
78  {
79  test01();
80  return 0;
81  }

```

```

1  void test03()
2  {
3  vector<int> v1;
4  v1.push_back(10);
5  v1.push_back(20);
6  v1.push_back(30);
7  v1.push_back(40);
8  v1.push_back(50);
9
10 cout<<"头元素:"<<v1.front()<<" 尾元素:"<<v1.back()<<endl;
11 //at越界抛出异常 【】越界不会抛出异常
12 cout<<v1.at(1)<<" "<<v1[1]<<endl;
13 v1.at(1)=200;
14 v1[3]=300;
15 printVectorInt(v1);//10 200 30 300 50
16
17 v1.pop_back();//尾删
18 printVectorInt(v1);//10 200 30 300
19 v1.insert( v1.begin()+2, 3, 500 );
20 printVectorInt(v1);//10 200 500 500 500 30 300

```

```

21  v1.erase(v1.begin()+2, v1.begin()+5 );
22  printVectorInt(v1);//10 200 30 300
23  v1.clear();
24  cout<<"大小:"<<v1.size()<<" 容量:"<<v1.capacity()<<endl;
25  }

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

```

头元素:10    尾元素:50
20 20
10 200 30 300 50
10 200 30 300
10 200 500 500 500 30 300
10 200 30 300
大小:0 容量:8

```

3、巧用swap收缩空间

```

1  void test04()
2  {
3      vector<int> v1;
4      v1.reserve(1000);
5      v1.assign(5,100);
6      cout<<"大小:"<<v1.size()<<" 容量:"<<v1.capacity()<<endl;
7      //v1.resize(3);
8      vector<int>(v1).swap(v1);
9      cout<<"大小:"<<v1.size()<<" 容量:"<<v1.capacity()<<endl;
10 }

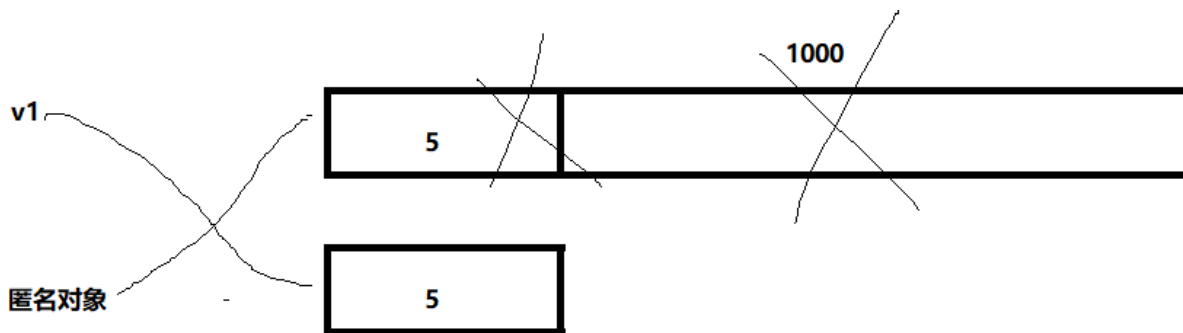
```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\q

```

大小:5 容量:1000
大小:5 容量:5

```

4、vector容器 嵌套 容器

```
1 void test05()
2 {
3     vector<int> v1(5,10);
4     vector<int> v2(5,100);
5     vector<int> v3(5,1000);
6
7     //需求: 定义一个容器 存放v1 v2 v3
8     vector< vector<int> > v;
9     v.push_back(v1);
10    v.push_back(v2);
11    v.push_back(v3);
12
13    //遍历
14    vector< vector<int> >::iterator it;
15    for(it=v.begin(); it!=v.end(); it++)
16    {
17        /*it == vector<int>
18        vector<int>::iterator mit;
19        for(mit=(*it).begin();mit!=(*it).end();mit++ )
20        {
21            /*mit == int
22            cout<<*mit<<" ";
23        }
24        cout<<endl;
25    }
26 }
```

```
10 10 10 10 10
100 100 100 100 100
1000 1000 1000 1000 1000
```

5、使用算法 对 vector容器排序

```
1 #include<algorithm>//算法头文件
2 bool myCompare(int value1, int value2)
3 {
4     return value1<value2;
5 }
6 void test06()
7 {
8     vector<int> v1;
9     v1.push_back(20);
10    v1.push_back(60);
11    v1.push_back(30);
12    v1.push_back(50);
13    v1.push_back(40);
14    v1.push_back(10);
15    printVectorInt(v1);
16
17    //sort算法排序
18    sort(v1.begin(), v1.end());
19    //sort(v1.begin(), v1.end(), greater<int>());
20    //sort(v1.begin(), v1.end(), myCompare);
21    printVectorInt(v1);
22 }
```

```
20 60 30 50 40 10
10 20 30 40 50 60
```

6、vector存放自定义数据类型

```
1 class Person
2 {
3     friend void printVectorPerson(vector<Person> &v);
4     friend bool myComparePerson(const Person &ob1, const Person &ob2);
```

```

5 private:
6     int num;
7     string name;
8     float score;
9 public:
10    Person(){}
11    Person(int num, string name, float score)
12    {
13        this->num = num;
14        this->name = name;
15        this->score = score;
16    }
17    #if 0
18        //方法2: 重载自定义数据的<运算符
19        bool operator<(const Person &ob)
20        {
21            return this->num < ob.num;
22        }
23    #endif
24 };
25 void printVectorPerson(vector<Person> &v)
26 {
27     vector<Person>::iterator it;
28     for(it=v.begin(); it!=v.end(); it++)
29     {
30         /*it == Person
31         cout<<(*it).num<<" "<<(*it).name<<" "<<(*it).score<<endl;
32     }
33 }
34 //方法1: 对于自定义容器排序 必须实现 排序规则
35 bool myComparePerson(const Person &ob1, const Person &ob2)
36 {
37     if(ob1.num == ob2.num)
38         return ob1.score<ob2.score;
39     return ob1.num > ob2.num;
40 }
41
42 void test07()
43 {
44     vector<Person> v;

```

```

45
46 v.push_back(Person(100, "lucy", 88.8f));
47 v.push_back(Person(103, "bob", 99.8f));
48 v.push_back(Person(103, "tom", 77.8f));
49 v.push_back(Person(103, "德玛", 88.8f));
50 v.push_back(Person(101, "小法", 66.8f));
51
52 printVectorPerson(v);
53 //方法1: 对于自定义容器排序 必须实现 排序规则
54 sort(v.begin(), v.end(), myComparePerson);
55 //方法2: 重载自定义数据的<运算符
56 //sort(v.begin(), v.end());
57 cout<<"-----"<<endl;
58 printVectorPerson(v);
59 }

```

```

100 lucy 88.8
103 bob 99.8
103 tom 77.8
103 德玛 88.8
101 小法 66.8

```

```

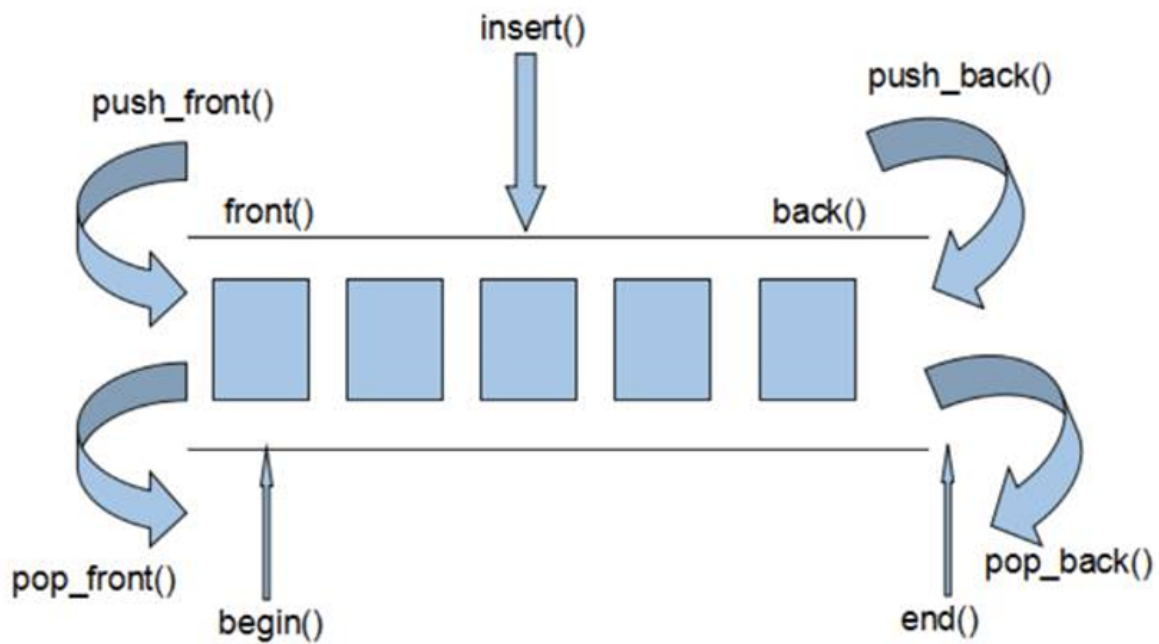
-----
103 tom 77.8
103 德玛 88.8
103 bob 99.8
101 小法 66.8
100 lucy 88.8

```

知识点4【deque容器】（了解）

1、deque概述

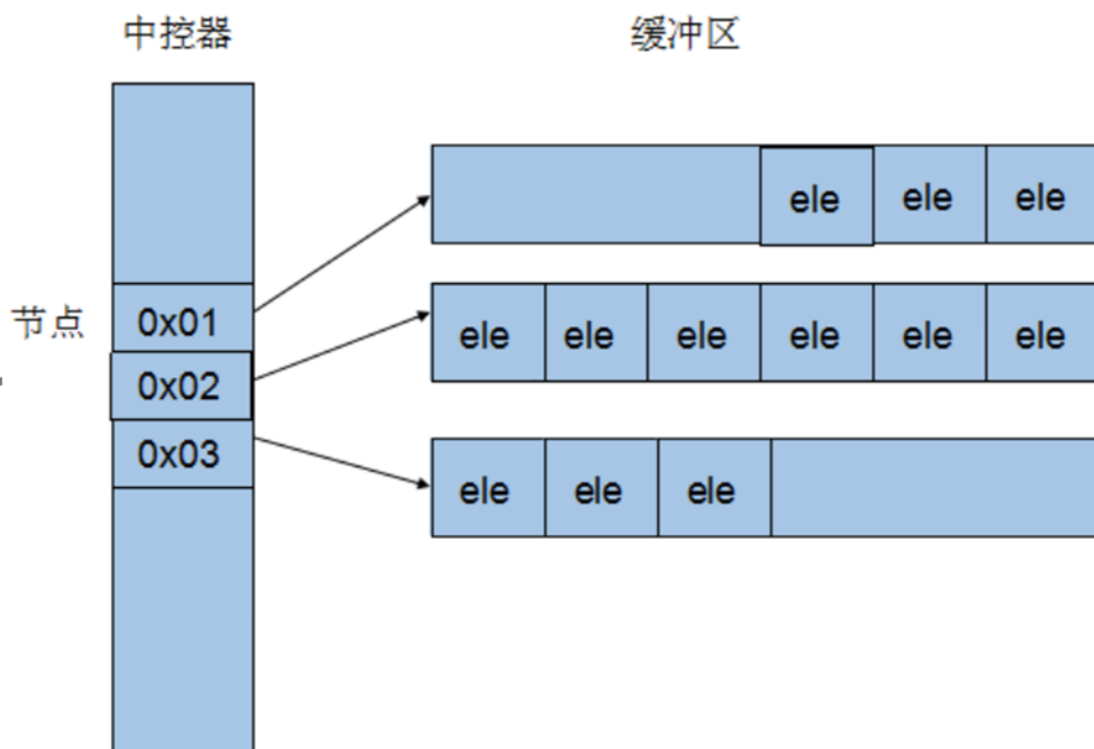
deque:双端动态数组



Deque容器和vector容器最大的差异,

一在于deque允许使用**常数项时间**对头端进行元素的插入和删除操作。

二在于deque没有**容量**的概念。



←

2、deque的API

如果迭代器能+1 那么该迭代器 为**随机访问**迭代器

- 1 3.3.3.1 deque构造函数
- 2 `deque<T> deqT;` //默认构造形式
- 3 `deque(beg, end);` //构造函数将[beg, end)区间中的元素拷贝给本身。
- 4 `deque(n, elem);` //构造函数将n个elem拷贝给本身。

```

5 deque(const deque &deq); //拷贝构造函数。
6 3.3.3.2 deque赋值操作
7 assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。
8 assign(n, elem); //将n个elem拷贝赋值给本身。
9 deque& operator=(const deque &deq); //重载等号操作符
10 swap(deq); // 将deq与本身的元素互换
11 3.3.3.3 deque大小操作
12 deque.size(); //返回容器中元素的个数
13 deque.empty(); //判断容器是否为空
14 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。
15 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。
16 3.3.3.4 deque双端插入和删除操作
17 push_back(elem); //在容器尾部添加一个数据
18 push_front(elem); //在容器头部插入一个数据
19 pop_back(); //删除容器最后一个数据
20 pop_front(); //删除容器第一个数据
21 3.3.3.5 deque数据存取
22 at(idx); //返回索引idx所指的数据,如果idx越界,抛出out_of_range。
23 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。
24 front(); //返回第一个数据。
25 back(); //返回最后一个数据
26 3.3.3.6 deque插入操作
27 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。
28 insert(pos, n, elem); //在pos位置插入n个elem数据,无返回值。
29 insert(pos, beg, end); //在pos位置插入[beg, end)区间的数据,无返回值。
30 3.3.3.7 deque删除操作
31 clear(); //移除容器的所有数据
32 erase(beg, end); //删除[beg, end)区间的数据,返回下一个数据的位置。
33 erase(pos); //删除pos位置的数据,返回下一个数据的位置

```

```

1 #include <iostream>
2 #include <deque>
3 using namespace std;
4 void printfDequeInt(deque<int> &d)
5 {
6     deque<int>::iterator it;
7     for(it=d.begin(); it!=d.end(); it++)
8     {

```

```

9  cout<<*it<<" ";
10 }
11 cout<<endl;
12 }
13
14 void test01()
15 {
16     deque<int> d1;
17     d1.push_back(1);
18     d1.push_back(2);
19     d1.push_back(3);
20     d1.push_front(4);
21     d1.push_front(5);
22     d1.push_front(6);
23     printfDequeInt(d1);//6 5 4 1 2 3
24
25     cout<<"大小:"<<d1.size()<<endl;
26     d1.pop_front();
27     printfDequeInt(d1);//5 4 1 2 3
28     d1.pop_back();
29     printfDequeInt(d1);//5 4 1 2
30     d1.insert(d1.begin()+1,3, 100);
31     printfDequeInt(d1);//5 100 100 100 4 1 2
32 }
33
34 int main(int argc, char *argv[])
35 {
36     test01();
37     return 0;
38 }

```

3、deque容器的案例

vector存放的数据 没有多大的规律 只是纪录数据。

deque容器：用于类似竞技的数据

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。

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

```
1  #include<vector>
2  class Player
3  {
4  public:
5      string name;
6      float score;
7  public:
8      Player(){}
9      Player(string name,float score=0.0f)
10     {
11         this->name = name;
12         this->score=score;
13     }
14 };
15 void createPlayer(vector<Player> &v)
16 {
17     string seedName = "ABCDE";
18     int i=0;
19     for(i=0;i<5;i++)
20     {
21         string tmpName = "选手";
22         tmpName+=seedName[i];
23
24         v.push_back(Player(tmpName));
25     }
26 }
27 #include<stdlib.h>
28 #include<time.h>
29 #include<algorithm>
30 void playGame(vector<Player> &v)
31 {
32     //设置随机数 种子
33     srand(time(NULL));
34     //每名选手都要参加
35     vector<Player>::iterator it;
36     for(it=v.begin(); it!=v.end();it++)
37     {
```

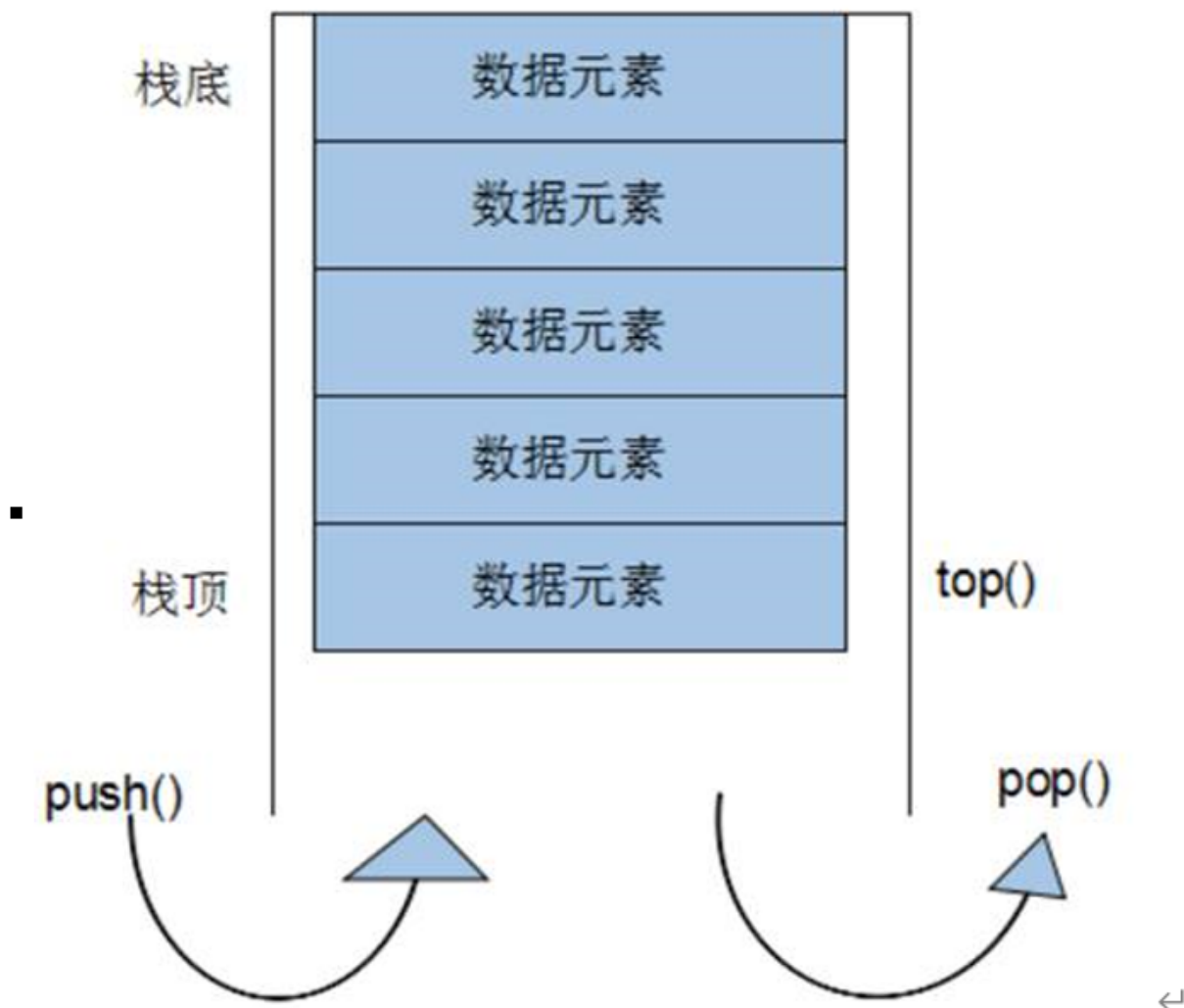


```
38 //10个评委打分
39 deque<float> d;
40 int i=0;
41 for(i=0;i<10;i++)
42 {
43     d.push_back(rand()%41+60);
44 }
45
46 // 对d容器排序
47 sort(d.begin(),d.end());
48
49 //去掉最高分
50 d.pop_back();
51 //去掉最低分
52 d.pop_front();
53
54 //求总分数
55 (*it).score = accumulate(d.begin(),d.end(), 0)/d.size();
56 }
57 }
58 void showScore(vector<Player> &v)
59 {
60     vector<Player>::iterator it;
61     for(it=v.begin(); it!=v.end();it++)
62     {
63         cout<<(*it).name<<"所得分数:"<<(*it).score<<endl;
64     }
65 }
66 void test02()
67 {
68     //创建5名选手 放入vector容器中
69     vector<Player> v;
70     createPlayer(v);
71
72     //开始比赛
73     playGame(v);
74
75     //公布成绩
76     showScore(v);
77 }
```

选手A所得分数:81
选手B所得分数:84
选手C所得分数:87
选手D所得分数:80
选手E所得分数:81

知识点5【stack栈容器】（了解）

stack是一种先进后出(First In Last Out,FILO)的数据结构。



操作数据的一端 叫栈顶。

top永远指向栈顶元素。

栈容器没有迭代器。不支持遍历行为。

1 3.4.3.1 stack构造函数

2 `stack<T> stkT;` //stack采用模板类实现， stack对象的默认构造形式：

```

3  stack(const stack &stk); // 拷贝构造函数
4  3.4.3.2 stack赋值操作
5  stack& operator=(const stack &stk); // 重载等号操作符
6  3.4.3.3 stack数据存取操作
7  push(elem); // 向栈顶添加元素
8  pop(); // 从栈顶移除第一个元素
9  top(); // 返回栈顶元素
10 3.4.3.4 stack大小操作
11 empty(); // 判断堆栈是否为空
12 size(); // 返回堆栈的大小

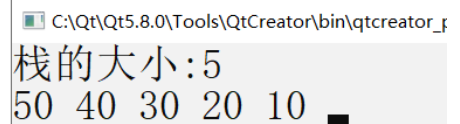
```

```

#include <stack>
using namespace std;
void test01()
{
    stack<int> s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);
    s.push(50);

    if(!s.empty())
    {
        cout<<"栈的大小:"<<s.size()<<endl;
        while(!s.empty())
        {
            cout<<s.top()<<" ";
            s.pop();
        }
    }
}

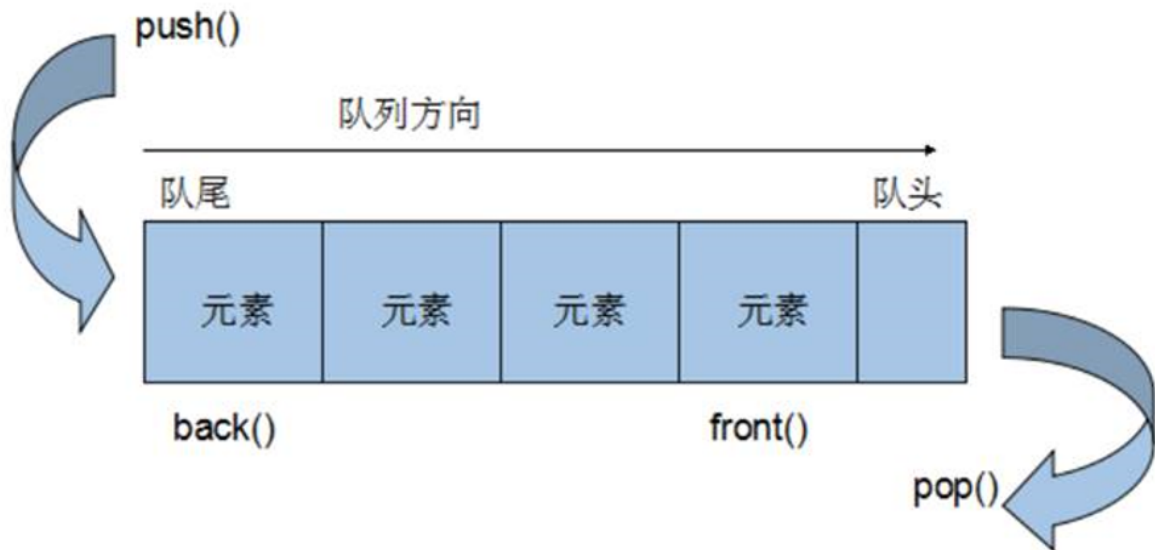
```



C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_g
 栈的大小:5
 50 40 30 20 10

知识点6 【queue队列容器】（了解）

Queue是一种先进先出(First In First Out, FIFO)的数据结构



出数据的一方叫队头，入数据的一方叫队尾。

queue容器没有迭代器 不支持遍历行为。

```
1 queue<T> queT; //queue采用模板类实现，queue对象的默认构造形式：
2 queue(const queue &que); //拷贝构造函数
3 3.5.3.2 queue存取、插入和删除操作
4 push(elem); //往队尾添加元素
5 pop(); //从队头移除第一个元素
6 back(); //返回最后一个元素
7 front(); //返回第一个元素
8 3.5.3.3 queue赋值操作
9 queue& operator=(const queue &que); //重载等号操作符
10 3.5.3.4 queue大小操作
11 empty(); //判断队列是否为空
12 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);
    q.push(50);

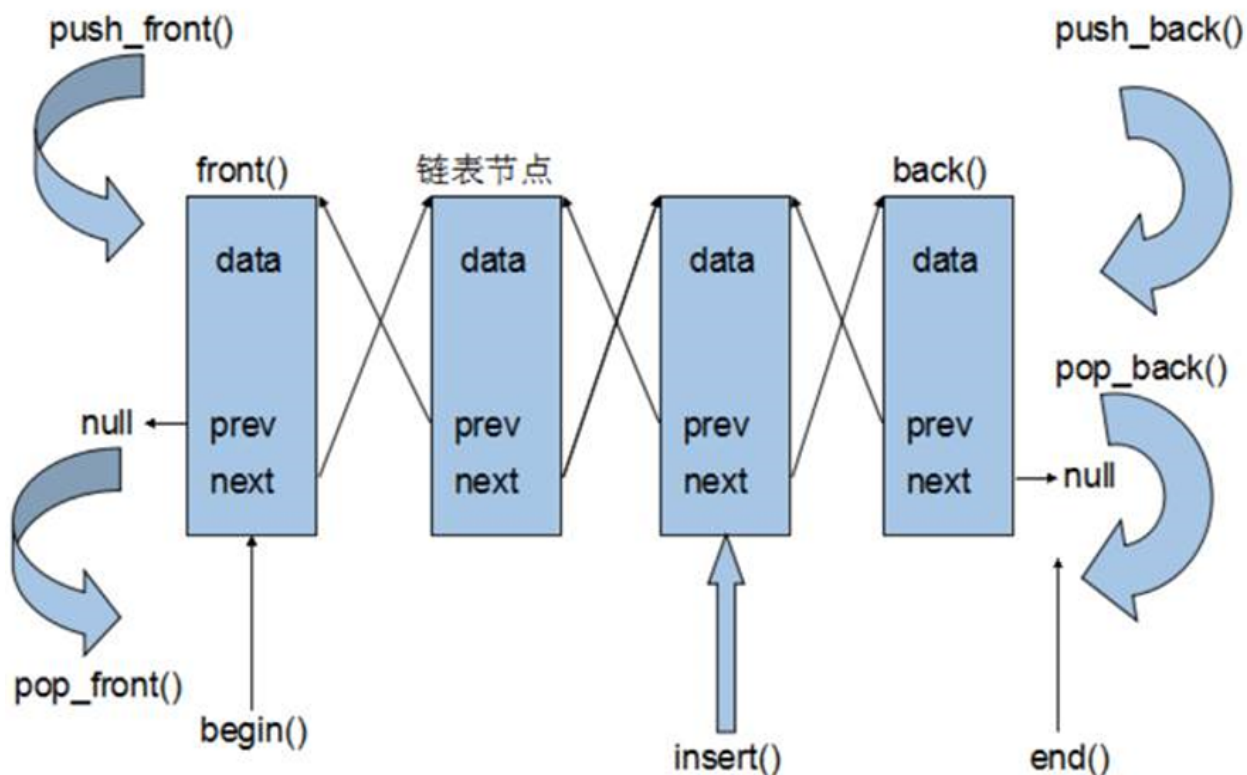
    if(!q.empty())
    {
        cout<<"栈的大小:"<<q.size()<<endl;
        while(!q.empty())
        {
            cout<<q.front()<<" ";
            q.pop();
        }
    }
}

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator
 栈的大小:5
 10 20 30 40 50

知识点7 【list链表容器】（了解）

list是双向循环链表



list容器的迭代器是 双向迭代器。

- 3.6.4.1 list构造函数
- `list<T> lstT;` //list采用模板类实现,对象的默认构造形式:

```

3 list(beg, end); //构造函数将[beg, end)区间中的元素拷贝给本身。
4 list(n, elem); //构造函数将n个elem拷贝给本身。
5 list(const list &lst); //拷贝构造函数。
6 3.6.4.2 list数据元素插入和删除操作
7 push_back(elem); //在容器尾部加入一个元素
8 pop_back(); //删除容器中最后一个元素
9 push_front(elem); //在容器开头插入一个元素
10 pop_front(); //从容器开头移除第一个元素
11 insert(pos, elem); //在pos位置插elem元素的拷贝，返回新数据的位置。
12 insert(pos, n, elem); //在pos位置插入n个elem数据，无返回值。
13 insert(pos, beg, end); //在pos位置插入[beg, end)区间的数据，无返回值。
14 clear(); //移除容器的所有数据
15 erase(beg, end); //删除[beg, end)区间的数据，返回下一个数据的位置。
16 erase(pos); //删除pos位置的数据，返回下一个数据的位置。
17 remove(elem); //删除容器中所有与elem值匹配的元素。
18 3.6.4.3 list大小操作
19 size(); //返回容器中元素的个数
20 empty(); //判断容器是否为空
21 resize(num); //重新指定容器的长度为num，
22 若容器变长，则以默认值填充新位置。
23 如果容器变短，则末尾超出容器长度的元素被删除。
24 resize(num, elem); //重新指定容器的长度为num，
25 若容器变长，则以elem值填充新位置。
26 如果容器变短，则末尾超出容器长度的元素被删除。
27 3.6.4.4 list赋值操作
28 assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。
29 assign(n, elem); //将n个elem拷贝赋值给本身。
30 list& operator=(const list &lst); //重载等号操作符
31 swap(lst); //将lst与本身的元素互换。
32 3.6.4.5 list数据的存取
33 front(); //返回第一个元素。
34 back(); //返回最后一个元素。
35 3.6.4.6 list反转排序
36 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。
37 sort(); //list排序

```

```

1 #include <iostream>
2 #include <list>
3 #include <algorithm>

```

```
4 using namespace std;
5 void printListInt(list<int> &l)
6 {
7     list<int>::iterator it;
8     for(it=l.begin(); it!=l.end();it++)
9     {
10         cout<<*it<<" ";
11     }
12     cout<<endl;
13 }
14
15 void test01()
16 {
17     list<int> l1;
18     l1.push_back(10);
19     l1.push_back(20);
20     l1.push_back(30);
21     l1.push_front(40);
22     l1.push_front(50);
23     l1.push_front(60);
24
25     printListInt(l1);//60 50 40 10 20 30
26     //list容器 是双向迭代器 不支持+2 支持++
27     list<int>::iterator it=l1.begin();
28     it++;
29     it++;
30     l1.insert(it, 3, 100);
31     printListInt(l1);//60 50 100 100 100 40 10 20 30
32
33     //删除所有100
34     l1.remove(100);
35     printListInt(l1);//60 50 40 10 20 30
36
37     //对链表排序
38     //STL提供的算法 只支持 随机访问迭代器，而list是双向迭代器 所以sort不支持list
39     // l1.sort(greater<int>());
40     l1.sort();
41     printListInt(l1);//10 20 30 40 50 60
42 }
43
```

```

44 int main(int argc, char *argv[])
45 {
46     test01();
47     return 0;
48 }
49

```

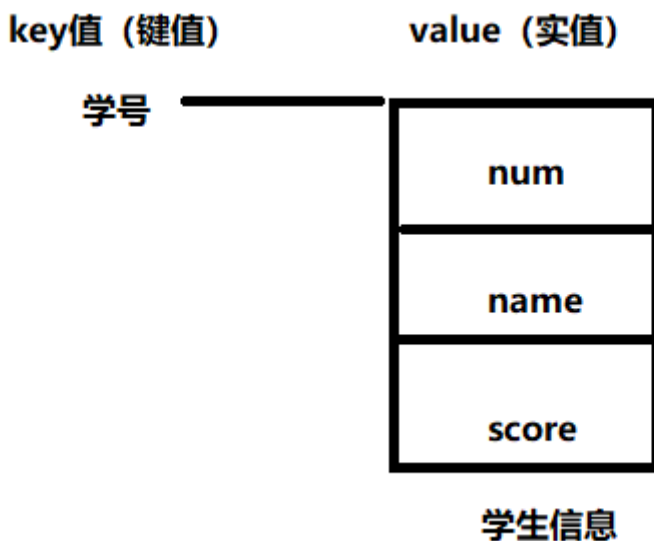
```

60 50 40 10 20 30
60 50 100 100 100 40 10 20 30
60 50 40 10 20 30
10 20 30 40 50 60

```

知识点8【set容器】（了解）

1、set容器概述



但是set容器 只有键值，在插入数据的时候 自动根据 **键值** 排序。不允许有**相同**的键值。**不能修改**set容器的元素值，会破坏set的数据结构。set容器的迭代器是**只读迭代器** (const_iterator) 。

```

1 3.7.2.1 set构造函数
2 set<T> st; //set默认构造函数:
3 multiset<T> mst; //multiset默认构造函数:
4 set(const set &st); //拷贝构造函数
5 3.7.2.2 set赋值操作
6 set& operator=(const set &st); //重载等号操作符

```



```

7  swap(st); //交换两个集合容器
8  3.7.2.3 set大小操作
9  size(); //返回容器中元素的数目
10 empty(); //判断容器是否为空
11 3.7.2.4 set插入和删除操作
12 insert(elem); //在容器中插入元素。
13 clear(); //清除所有元素
14 erase(pos); //删除pos迭代器所指的元素，返回下一个元素的迭代器。
15 erase(beg, end); //删除区间[beg,end)的所有元素，返回下一个元素的迭代器。
16 erase(elem); //删除容器中值为elem的元素。
17 3.7.2.5 set查找操作
18 find(key); //查找键key是否存在,若存在，返回该键的元素的迭代器；若不存在，返回set.end();
19 count(key); //查找键key的元素个数
20 lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。
21 upper_bound(keyElem); //返回第一个key>keyElem元素的迭代器。
22 equal_range(keyElem); //返回容器中key与keyElem相等的上下限的两个迭代器

```

```


1  #include <iostream>
2  #include<set>
3  using namespace std;
4  void printSetInt(set<int> &s)
5  {
6      set<int>::iterator it;
7      for(it=s.begin(); it!=s.end(); it++)
8      {
9          cout<<*it<<" ";
10     }
11     cout<<endl;
12 }
13
14 void test01()
15 {
16     set<int> s1;
17     s1.insert(30);
18     s1.insert(10);
19     s1.insert(20);
20     s1.insert(20);
21     s1.insert(40);
22     printSetInt(s1);

```

```

23 }
24
25 int main(int argc, char *argv[])
26 {
27     test01();
28     return 0;
29 }

```

 C:\Qt\Qt5.8.0\Tools\QtCreator\b

10 20 30 40

2、更改set容器的排序规则（定义set容器时 修改）

```
1 set<int, 排序规则类> s1;
```

一般都是通过 “仿函数” 修改set容器的排序规则。

```

1 #include <iostream>
2 #include<set>
3 using namespace std;
4 //仿函数
5 class MyCompare
6 {
7 public:
8     bool operator()(int v1, int v2)
9     {
10         return v1>v2;
11     }
12 };
13 void printSetInt(set<int> &s)
14 {
15     set<int>::iterator it;
16     for(it=s.begin(); it!=s.end();it++)
17     {
18         cout<<*it<<" ";
19     }
20     cout<<endl;
21 }
22
23 void printSetInt(set<int,MyCompare> &s)
24 {

```

```

25  set<int,MyCompare>::iterator it;
26  for(it=s.begin(); it!=s.end();it++)
27  {
28      cout<<*it<<" ";
29  }
30  cout<<endl;
31  }
32
33
34  void test01()
35  {
36      set<int,MyCompare> s1;
37      s1.insert(30);
38      s1.insert(10);
39      s1.insert(20);
40      s1.insert(20);
41      s1.insert(40);
42      printSetInt(s1);
43  }
44
45  int main(int argc, char *argv[])
46  {
47      test01();
48      return 0;
49  }

```

C:\Qt\Qt5.8.0\Tools\QtCreat

40 30 20 10

3、如果set容器存放自定义数据 必须更改排序规则

```

1  class Person
2  {
3      friend class MyComparePerson;
4      friend ostream& operator<<(ostream &out, Person ob);
5  private:
6      int num;
7      string name;
8      float score;

```

```

9 public:
10     Person(){}
11     Person(int num,string name, float score)
12     {
13         this->num = num;
14         this->name = name;
15         this->score = score;
16     }
17 };
18 ostream& operator<<(ostream &out, Person ob)
19 {
20     out<<ob.num<<" "<<ob.name<<" "<<ob.score<<endl;
21     return out;
22 }
23
24 class MyComparePerson
25 {
26 public:
27     bool operator()(Person ob1, Person ob2)
28     {
29         return ob1.num < ob2.num;
30     }
31 };
32 void printSetInt(set<Person,MyComparePerson> &s)
33 {
34     set<Person,MyComparePerson> ::iterator it;
35     for(it=s.begin(); it!=s.end();it++)
36     {
37         cout<<(*it);
38     }
39     cout<<endl;
40 }
41 void test02()
42 {
43     set<Person,MyComparePerson> s1;
44     s1.insert(Person(100,"lucy", 88.8f));
45     s1.insert(Person(104,"bob", 99.8f));
46     s1.insert(Person(103,"tom", 77.8f));
47     s1.insert(Person(101,"德玛", 66.8f));
48     s1.insert(Person(105,"寒冰", 55.8f));

```

```
49  printSetInt(s1);
50  }
```

C:\Qt\Qt5.6.0\Tools\QtCreator\bin\qtcrea

```
100  lucy  88.8
101  德玛  66.8
103  tom   77.8
104  bob   99.8
105  寒冰  55.8
```

4、set的API

```
1  void test03()
2  {
3      set<int> s1;
4      s1.insert(10);
5      s1.insert(30);
6      s1.insert(50);
7      s1.insert(70);
8      s1.insert(90);
9      printSetInt(s1);
10
11     set<int>::const_iterator ret;
12     ret = s1.find(50);
13     if(ret != s1.end())
14     {
15         cout<<"找到的结果:"<<*ret<<endl;
16     }
17
18     //count(key); //查找键key的元素个数 set容器的结果只能是0或1
19     cout<<s1.count(50)<<endl;
20 }
```

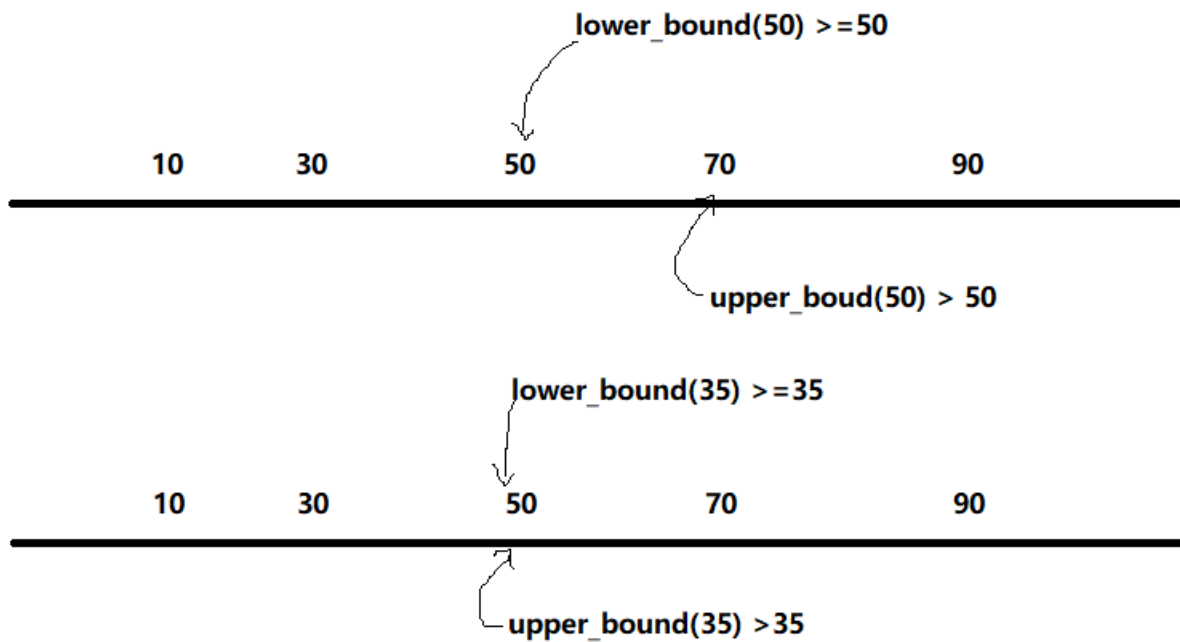
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreatc

10 30 50 70 90

找到的结果:50

1

5、查找元素的上下限




```
1 void test04()
2 {
3     set<int> s1;
4     s1.insert(10);
5     s1.insert(30);
6     s1.insert(50);
7     s1.insert(70);
8     s1.insert(90);
9     printSetInt(s1);
10
11     set<int>::const_iterator ret;
12     //lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。(下限)
13     ret = s1.lower_bound(50);
14     if(ret != s1.end())
15     {
16         cout<<"下限为:"<<*ret<<endl;
```

```

17  }
18  //upper_bound(keyElem);//返回第一个key>keyElem元素的迭代器（上限）
19  ret = s1.upper_bound(50);
20  if(ret !=s1.end())
21  {
22  cout<<"上限为:"<<*ret<<endl;
23  }
24  //equal_range(keyElem);//返回容器中key与keyElem相等的上下限的两个迭代器
25  //返回值类型为pair 对组
26  pair< set<int>::const_iterator, set<int>::const_iterator> p;
27  p = s1.equal_range(50);
28  if(p.first != s1.end())
29  {
30  cout<<"下限为:"<<*(p.first)<<endl;
31  }
32  if(p.second != s1.end())
33  {
34  cout<<"上限为:"<<*(p.second)<<endl;
35  }
36  }

```

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_

```

10 30 50 70 90
下限为:50
上限为:70
下限为:50
上限为:70

```

案例2:

```

1 void test05()
2 {
3  set<int> s1;
4  pair<set<int>::const_iterator, bool> ret;
5  ret = s1.insert(10);
6  if(ret.second == true)
7  {

```

```

8  cout<<"第一次插入成功"<<endl;
9  }
10 else
11 {
12  cout<<"第一次插入失败"<<endl;
13  }
14
15  ret = s1.insert(10);
16  if(ret.second == true)
17  {
18  cout<<"第二次插入成功"<<endl;
19  }
20 else
21 {
22  cout<<"第二次插入失败"<<endl;
23  }
24 }

```

第一次插入成功
第二次插入失败

知识点9【multiset容器】（了解）

set容器：键值不允许重复

multiset容器：键值可以重复

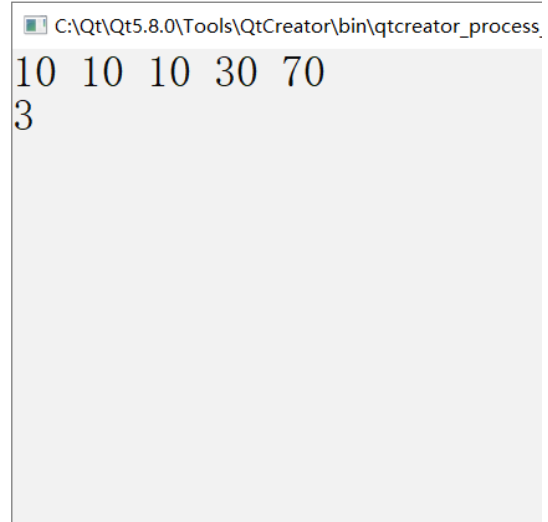

```

void printSetInt(multiset<int> &s)
{
    multiset<int>::iterator it;
    for(it=s.begin(); it!=s.end(); it++)
    {
        cout<<(*it)<<" ";
    }
    cout<<endl;
}

void test06()
{
    multiset<int> s1;
    s1.insert(10);
    s1.insert(30);
    s1.insert(10);
    s1.insert(70);
    s1.insert(10);
    printSetInt(s1);

    cout<<s1.count(10)<<endl;
}

```



C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process

```

10 10 10 30 70
3

```

知识点10 【pair对组】（了解）

对组(pair)将一对值组合成一个值，这一对值可以具有不同的数据类型，两个值可以分别用pair的两个公有属性first和second访问

```

void test07()
{
    //方式1
    pair<int, string> p1(10086, "移动");
    pair<int, string> p2(10010, "联通");
    pair<int, string> p3(10000, "电信");

    //方式2: (推荐)
    pair<int, string> p4=make_pair(9527, "星爷");

    cout<<p4.first<<" "<<p4.second<<endl;
}

```



C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcre

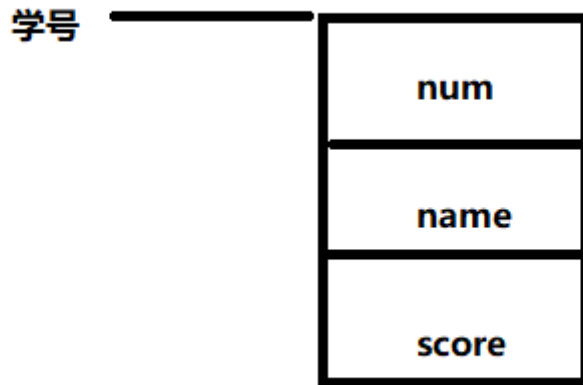
```

9527 星爷

```

知识点11 【map容器】（重要）

key值 (键值) value (实值)



map容器：每个元素都是 键值-实值 成对存储，自动根据键值排序，键值不能重复，不能修改。

```
1 map构造函数
2 map<T1, T2> mapTT; //map默认构造函数:
3 map(const map &mp); //拷贝构造函数
4 3.8.2.2 map赋值操作
5 map& operator=(const map &mp); //重载等号操作符
6 swap(mp); //交换两个集合容器
7 3.8.2.3 map大小操作
8 size(); //返回容器中元素的数目
9 empty(); //判断容器是否为空
10 3.8.2.4 map插入数据元素操作
11 map.insert(...); //往容器插入元素，返回pair<iterator, bool>
12 map<int, string> mapStu;
13 // 第一种 通过pair的方式插入对象
14 mapStu.insert(pair<int, string>(3, "小张"));
15 // 第二种 通过pair的方式插入对象
16 mapStu.inset(make_pair(-1, "校长"));
17 // 第三种 通过value_type的方式插入对象
18 mapStu.insert(map<int, string>::value_type(1, "小李"));
19 // 第四种 通过数组的方式插入值
20 mapStu[3] = "小刘";
21 mapStu[5] = "小王";
22 3.8.2.5 map删除操作
23 clear(); //删除所有元素
24 erase(pos); //删除pos迭代器所指的元素，返回下一个元素的迭代器。
25 erase(beg, end); //删除区间[beg, end)的所有元素，返回下一个元素的迭代器。
26 erase(keyElem); //删除容器中key为keyElem的对组。
```

27 3.8.2.6 map查找操作

```
28 find(key); //查找键key是否存在,若存在,返回该键的元素的迭代器; /若不存在,返回m  
ap.end();  
29 count(keyElem); //返回容器中key为keyElem的对组个数。对map来说,要么是0,要么是  
1。对multimap来说,值可能大于1。  
30 lower_bound(keyElem); //返回第一个key>=keyElem元素的迭代器。  
31 upper_bound(keyElem); //返回第一个key>keyElem元素的迭代器。  
32 equal_range(keyElem); //返回容器中key与keyElem相等的上下限的两个迭代器
```

案例1：键值的map容器设计

```
1 #include <iostream>  
2 #include <map>  
3 using namespace std;  
4 class Person  
5 {  
6     friend void test01();  
7     friend void printMapAll(map<int, Person> &m);  
8 private:  
9     int num;  
10    string name;  
11    float score;  
12 public:  
13    Person(){}  
14    Person(int num,string name, float score);  
15 };  
16 void printMapAll(map<int, Person> &m)  
17 {  
18     map<int, Person>::const_iterator it;  
19     for(it=m.begin(); it!=m.end();it++)  
20     {  
21         ((*it) ==pair<int, Person>  
22         cout<<"学号:"<<(*it).first<<" 姓名:"<<(*it).second.name<<" \  
23         分数:"<<(*it).second.score<<endl;  
24     }  
25 }  
26  
27 void test01()  
28 {  
29     map<int, Person> m;  
30     //方式1:  
31     m.insert(pair<int,Person>(103, Person(103,"lucy", 88.8f)));
```

```

32 //方式2: 推荐
33 m.insert(make_pair(101,Person(101,"bob", 77.7f)));
34 //方式3:
35 m.insert( map<int, Person>::value_type( 102 , Person(102,"tom",
36 66.6f)));
37 //方式4:
38 m[104] = Person(104,"德玛", 99.9f);
39
40 printMapAll(m);
41
42 //假如key值存在 m[key]代表对应的实值
43 cout<< m[107].num<<" "<<m[107].name<<" "<<m[107].score<<endl;
44
45 cout<<"-----"<<endl;
46 printMapAll(m);
47
48 m.erase(104);
49 cout<<"-----"<<endl;
50 printMapAll(m);
51
52 //查找key为103的数据
53 map<int, Person>::const_iterator ret;
54 ret = m.find(103);
55 if(ret != m.end())
56 {
57     /*ret == pair<int,Person>
58     cout<<(*ret).first<<" "<<(*ret).second.name<<" "
59     <<(*ret).second.score<<endl;
60 }
61 }
62
63 int main(int argc, char *argv[])
64 {
65     test01();
66     return 0;
67 }
68
69 Person::Person(int num, string name, float score)
70 {
71     this->num = num;
72     this->name = name;

```

```

71  this->score = score;
72  }

```

multimap允许键值重复

```

C:\Qt\5.9.0\tools\qtcreator\bin\qtcreator_process_stable.exe
学号:101 姓名:bob 分数:77.7
学号:102 姓名:tom 分数:66.6
学号:103 姓名:lucy 分数:88.8
学号:104 姓名:德玛 分数:99.9
0 0
-----
学号:101 姓名:bob 分数:77.7
学号:102 姓名:tom 分数:66.6
学号:103 姓名:lucy 分数:88.8
学号:104 姓名:德玛 分数:99.9
学号:107 姓名: 分数:0
-----
学号:101 姓名:bob 分数:77.7
学号:102 姓名:tom 分数:66.6
学号:103 姓名:lucy 分数:88.8
学号:107 姓名: 分数:0
103 lucy 88.8

```

multimap: 键值可以重复

知识点12 【multimap案例】

公司今天招聘了5个员工，5名员工进入公司之后，需要指派员工在那个部门工作 人员信息有: 姓名 年龄 电话 工资等组成 通过Multimap进行信息的插入 保存 显示 分部门显示员工信息 显示全部员工信息

```

1  #define SALE_DEPATMENT 1 //销售部门
2  #define DEVELOP_DEPATMENT 2 //研发部门
3  #define FINACIAL_DEPATMENT 3 //财务部门

```

```

1  #include <iostream>
2  #include <vector>

```

```

3  #include <string>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <map>
7  using namespace std;
8  class Person
9  {
10     friend void showDepartmentPerson(multimap<int,Person> &m);
11     friend void personJoinDepartment(vector<Person> &v,
multimap<int,Person> &m);
12 private:
13     string name;
14     int age;
15     int money;
16     string tel;
17 public:
18     Person(){}
19     Person(string name, int age, int money, string tel);
20 };
21
22 void createVectorPerson(vector<Person> &v);
23 void personJoinDepartment(vector<Person> &v, multimap<int,Person> &m);
24 void showDepartmentPerson(multimap<int,Person> &m);
25 int main(int argc, char *argv[])
26 {
27     //创建vector容器 存放 员工
28     vector<Person> v;
29     createVectorPerson(v);
30
31     //5名员工加入部门
32     multimap<int, Person> m;
33     personJoinDepartment(v, m);
34
35     //显示部门员工
36     showDepartmentPerson(m);
37
38     return 0;
39 }
40
41 Person::Person(string name, int age, int money, string tel)
42 {

```

```

43  this->name = name;
44  this->age = age;
45  this->money = money;
46  this->tel = tel;
47  }
48
49  void createVectorPerson(vector<Person> &v)
50  {
51      //设置随机数种子
52      srand(time(NULL));
53      int i=0;
54      for(i=0;i<5; i++)
55      {
56          string seedName="ABCDE";
57          string tmpName = "员工";
58          tmpName += seedName[i];
59          int age = 20+i;
60          int money = 15000+rand()%10*1000;
61          string tel = to_string(rand());
62
63          v.push_back(Person(tmpName,age,money,tel));
64      }
65      return;
66  }
67
68  void personJoinDepartment(vector<Person> &v, multimap<int,Person> &m)
69  {
70      vector<Person>::iterator it;
71      for(it=v.begin(); it != v.end(); it++)
72      {
73          /*it == Person
74          cout<<"请输入["<<(*it).name<<"]加入的部门0(销售)、1(研发)、2(财务):";
75          int op = 0;
76          cin>>op;
77
78          m.insert(make_pair(op, *it));
79          //(m[op]) = (*it);
80      }
81  }
82

```

```

83 void showDepartmentPerson(multimap<int,Person> &m)
84 {
85     cout<<"请选择你要显示的部门0(销售)、1(研发)、2(财务):";
86     int op = 0;
87     cin>>op;
88
89     switch (op) {
90     case 0:
91         cout<<"-----销售部门员工如下-----"<<endl;
92         break;
93     case 1:
94         cout<<"-----研发部门员工如下-----"<<endl;
95         break;
96     case 2:
97         cout<<"-----财务部门员工如下-----"<<endl;
98         break;
99     }
100
101     //0 1 1 1 2
102     //寻找op的位置
103     multimap<int,Person>::const_iterator ret;
104     ret = m.find(op);
105     if(ret == m.end())
106         return;
107     //统计op的个数
108     int count = m.count(op);
109
110     //从op的位置 按照个数 逐个遍历
111     int i=0;
112     for(i=0;i<count; i++, ret++)
113     {
114         /*ret == pair<int, Person>
115         cout<<"\t"<<(*ret).second.name<<" "<<(*ret).second.age<<" "\
116         <<(*ret).second.money<<" "<<(*ret).second.tel<<endl;
117     }
118     return;
119 }

```



```
C:\Qt\Qt5.6.0\tools\QtCreator\bin\qtcreator_process_std.exe
请输入[员工A]加入的部门0(销售)、1(研发)、2(财务):0
请输入[员工B]加入的部门0(销售)、1(研发)、2(财务):1
请输入[员工C]加入的部门0(销售)、1(研发)、2(财务):1
请输入[员工D]加入的部门0(销售)、1(研发)、2(财务):2
请输入[员工E]加入的部门0(销售)、1(研发)、2(财务):2
请选择你要显示的部门0(销售)、1(研发)、2(财务):1
-----研发部门员工如下-----
      员工B 21 22000 20069
      员工C 22 18000 15951
```

知识点13 【容器的调用时机】（了解）

3.9 STL 容器使用时机

	vector	deque	list	set	multiset	map	multimap
典型内存结构	单端数组	双端数组	双向链表	二叉树	二叉树	二叉树	二叉树
可随机存取	是	是	否	否	否	对 key 而言：不是	否

	vector	deque	list	set	multiset	map	multimap
元素搜寻速度	慢	慢	非常慢	快	快	对 key 而言：快	对 key 而言：快
元素安插移除	尾端	头尾两端	任何位置	-	-	-	-

- vector的使用场景：比如软件历史操作记录的存储
- deque的使用场景：比如排队购票系统，支持头端的快速移除，尾端的快速添加
- list的使用场景：比如公交车乘客的存储，随时可能有乘客下车，支持频繁的不确实位置元素的移除插入
- set的使用场景：比如对手机游戏的个人得分记录的存储，存储要求从高分到低分
- map的使用场景：比如按ID号存储十万个用户，想要快速要通过ID查找对应的用户