第八章 模板和容器

C++简明双链教程（李昕著，清华大学出版社）

作者：李昕

PPT制作者：韩睿毅

目录

01 泛型编程

泛型编程

1. 泛型编程定义

泛型编程指在书写代码时，不考虑具体数据类型，而模板是泛型编程的基础。C++中的泛型编程主要分为模板函数和模板类。

2. 标准模板库

面向对象和泛型编程的目的就是提升复用性，C++的标准模板库（STL）提供了大体分为六大组件，分别是：容器、算法、迭代器、仿函数、适配器（配接器）、空间配置器。

泛型编程

容器	各种数据结构，如vector、list、deque、set、map等,用来存放数据。
算法	各种常用的算法，如sort、find、copy、for_each等。
迭代器	扮演了容器与算法之间的胶合剂。
仿函数	行为类似函数，可作为算法的某种策略
适配器	用来修饰容器或者仿函数或迭代器接口。
空间配置器	负责空间的配置与管理。

1.1 模板函数

模板函数提供一个抽象的函数，并不具体指定其中数据的类型，而是某个虚拟类型代替。只提供基本的功能。其具体的数据类型，只在其被调用时视具体情况实例化。

样例输入	样例输入
(无)	Max(i, j): 39 Max(i, j):20.7 Max(i, j): World

```cpp
#include <iostream>
#include <string>
using namespace std;

template <typename T>                    //模板函数声明与定义
T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}
int main ()
{
    int i = 39, j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;
    double f1 = 13.5, f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;
    string s1 = "Hello", s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;
    return 0;
}
```

1.1 模板函数

虚拟类型可以有多个

样例输入	样例输入
(无)	test(10,5)=15 test(5,'A')=F test(10,5.5) =15.5 test(5.5,10) =15

因为返回类型为**T2**，因此第10行的结果为字符类型，第11行结果为浮点类型，第12行结果被取整。

```
1   #include <iostream>
2   using namespace std;
3   template <typename T1, typename T2> //模板函数声明与定义
4   T2 test(T1 tmp1, T2 tmp2) {
5       T2 tmp = tmp1 + tmp2;
6       return tmp;
7   }
8   int main(void) {
9       cout << "test(10,5)=" << test(10,5) << endl;
10      cout << "test(5,'A')=" << test(5,'A') << endl ;
11      cout << "test(10,5.5) =" << test(10,5.5) << endl;
12      cout << "test(5.5,10) =" << test(5.5,10) << endl;
13      return 0;
14  }
```

1.2 模板类

与模板函数类似，可以构建模板类，不指定具体数据类型。

样例输入	样例输入
(无)	8 15

```
1   #include <iostream>
2   using namespace std;
3   template<class type> class Container {
4   private:
5       type data;
6   public:
7       Container(type d) { this->data = d; }
8       type operator+(const Container<type>& t){
9           return this->data + t.data;
10      }
11  };
12  int main() {
13      Container<int> ia(3), ib(5);
14      cout << ia + ib << endl;
15      Container<string> as("abc"), bs("def");
16      cout << as + bs << endl;
17      return 0;
18  }
```

- 第3行定义了一个模板类
- 第13行将数据类型指定的为int
- 第15行调用时，数据类型指定为string。
- 具体执行时，就会显示不同的类型的具体操作

知识点

索引	要点	正链	反链
T811	掌握模板函数，能够自定义简单的模板函数	T26B	T812B
T812	理解模板类，会用模板类执行基本操作。	T811	

02 STL容器

STL容器

STL容器就是将运用最广泛的一些数据结构实现出来，常用的数据结构：数组, 链表,树, 栈, 队列, 集合, 映射表 等。这些容器分为序列式容器和关联式容器两种。C++11新加入4种容器，主要结构采用哈希函数，因此也称为无序容器。

ps：参考文档链接：

https://cplusplus.com/reference/stl/

容器种类	功能
序列容器	主要包括 vector 向量容器、list 列表容器以及 deque 双端队列容器。元素在容器中的位置同元素的值无关，即容器不是排序的
关联式容器（排序容器）	包括 set 集合容器、multiset多重集合容器、map映射容器以及 multimap 多重映射容器。排序容器中的元素默认是由小到大排序好的。
哈希容器（无序容器）	分别是unordered_set 哈希集合、unordered_multiset 哈希多重集合、unordered_map 哈希映射以及 unordered_multimap 哈希多重映射。哈希容器中的元素是未排序的，元素的位置由哈希函数确定。

2.1 容器的分类

1. 序列容器

序列容器，是以线性排列来存储某一指定类型（例如 int、double 等）的数据，每个元素均有固定的位置。

容器	描述	增加或删除元素	随机访问
数组容器 array<T,N>	表示可以存储 N 个 T 类型的元素，是 C++ 本身提供的一种容器。	长度固定，不能增加或删除元素	Y
向量容器 vector<T>	长度可变，即在存储空间不足时，会自动申请更多的内存。	尾部增删效率O(1) 其他位置增删效率O(n)	Y
双端队列容器 deque<T>	和 vector 相似，头部和尾部插入和删除元素都非常高效。	头部尾部增删效率O(1) 其他位置增删效率O(n)	Y
链表容器 list<T>	长度可变，由 T 类型元素组成的序列，以双向链表形式组织元素	任意位置增删效率O(1)	N
正向链表容器 forward_list<T>	以单链表的形式组织元素，它内部的元素只能从第一个元素开始访问，比链表容器快、更节省内存。	任意位置增删效率O(1)	N
堆栈 stack<T>	在deque<T>的基础上形成，只能在尾部进行增加删除，实现先进后出	尾部增删效率O(1) 其他位置不能增删	N
单向队列 queue<T>	在deque<T>的基础上形成，只能在尾部新增，头部弹出，实现先进先出	尾部增加效率O(1)， 头部弹出效率O(1)， 其他位置不能增删	N

2.1 容器的分类

2. 关联式容器

关联式容器底层采用二叉树结构，更确切的说是红黑树结构，各元素之间没有严格的物理顺序关系。

关联式容器名称	特点
map	定义在 <map> 头文件中，使用该容器存储的数据，其各个元素的键必须是唯一的（即不能重复），该容器会根据各元素键的大小，默认进行升序排序（调用 std::less<T>）。
set	定义在 <set> 头文件中，使用该容器存储的数据，各个元素键和值完全相同，且各个元素的值不能重复（保证了各元素键的唯一性）。该容器会自动根据各个元素的键（其实也就是元素值）的大小进行升序排序（调用 std::less<T>）。
multimap	定义在 <map> 头文件中，和 map 容器唯一的不同在于，multimap 容器中存储元素的键可以重复。
multiset	定义在 <set> 头文件中，和 set 容器唯一的不同在于，multiset 容器中存储元素的值可以重复（一旦值重复，则意味着键也是重复的）。

2.1 容器的分类

3. 无序容器

无序容器仅是在前面提到的 4 种关联式容器名称的基础上，添加了 "unordered_"。关联式容器会对存储的键值进行排序，但是无序容器不会。

无序容器	特点
unordered_map	存储键值对 <key, value> 类型的元素，其中各个键值对键的值不允许重复，且该容器中存储的键值对是无序的。
unordered_multimap	和 unordered_map 唯一的区别在于，该容器允许存储多个键相同的键值对。
unordered_set	不再以键值对的形式存储数据，而是直接存储数据元素本身（当然也可以理解为，该容器存储的全部都是键 key 和值 value 相等的键值对，正因为它们相等，因此只存储 value 即可）。另外，该容器存储的元素不能重复，且容器内部存储的元素也是无序的。
unordered_multiset	和 unordered_set 唯一的区别在于，该容器允许存储值相同的元素。

- 如果涉及大量遍历容器的操作，建议首选关联式容器；
- 反之，如果更多的操作是通过键获取对应的值，则应首选无序容器。

知识点

索引	要点	正链	反链
T821	了解STL容器的分类		T823

2.2 迭代器

迭代器的底层实际就是一个指针，通过迭代器可以指向容器中的某个元素。***迭代器名** 就表示迭代器指向的元素。不同容器的迭代器功能强弱程度也有所不同。主要分为前向迭代器、双向迭代器和随机访问迭代器。

2.2 迭代器

- 前向迭代器的功能被所有类型迭代器兼容，包括++操作，即一次前向移动一个位置；复制或赋值；还可以用 == 和 != 运算符进行比较。C++中采用begin()指向首元素，用end()指向尾后元素，即最后一个有效元素后面的元素。

- 双向迭代器比正向迭代器多支持一个--操作，即一次后向移动一个位置。

- 随机访问迭代器支持的功能最多，除了以上提到的功能，它还支持加上任意偏移量并得到新的迭代器；通过下标形式访问元素；用 <、>、<=、>= 运算符进行比较；另外，两个迭代器的减法操作表示二者所指向元素的序号之差。

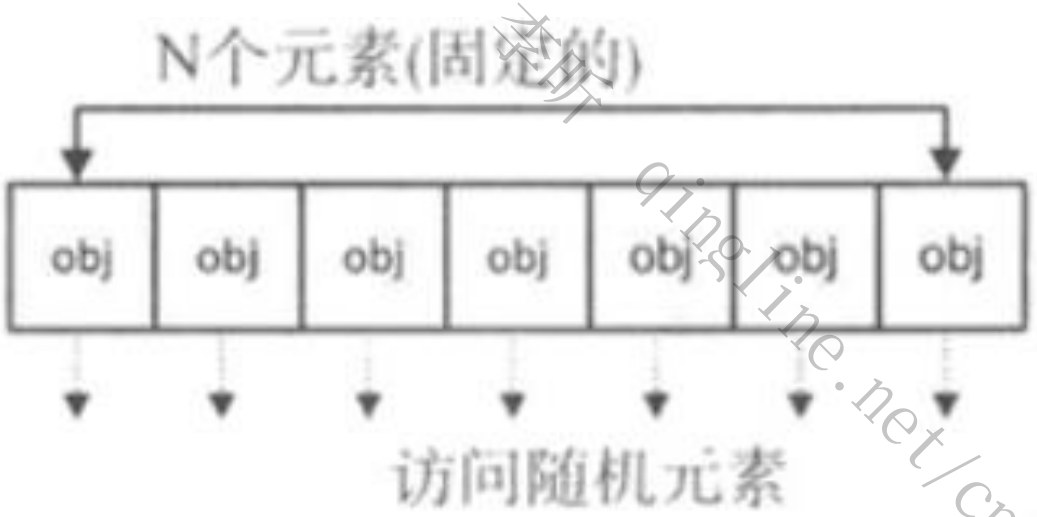
知识点

索引	要点	正链	反链
T822	迭代器是容器访问的主要方式，其本质就是通过类封装进行功能限定的指针	T791	T831
T823	能够清晰掌握不同类型迭代器和不同类型容器直接的对应关系，并理解造成这些异同的原因	T621, T821	
T824	双向迭代器和前向迭代器只能逐个遍历元素，终止判断只能采用!=运算		
T825	resize, reserve, insert, erase, assign, push_back等底层空间操作都会造成空间重新分配，进而导致迭代器的失效，因此要对迭代器进行重新赋值	T542	

03 向量vector

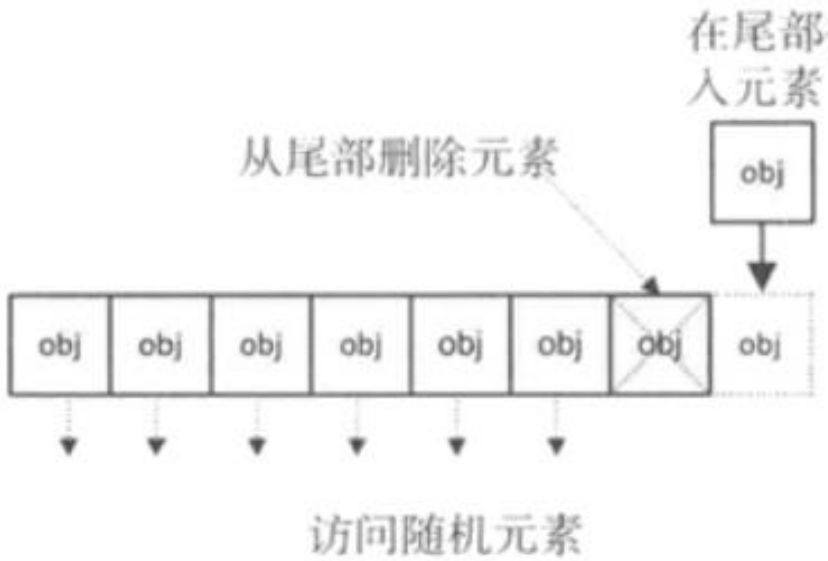
模板数组array

STL中提供了模板数组array，用于优化原生数组的使用。与原生数组相比，模板数组更安全、更便利，在进行随机范围时，除了重载操作符[]通过下标访问之外，还提供了函数at进行下标访问。



向量vector

STL提供了向量vector类型，由头文件<vector>引入，其工作方式与数组类似，但是容量可以根据需要自动伸缩。



vector在进行扩展时，先找更大的内存空间，然后将原数据拷贝新空间，释放原空间。

3.1 遍历

以下的遍历方式适用于所有使用随机访问迭代器的容器，例如：vector, string等。

样例输入	样例输出
(无)	0 1 2 3 0 1 1 3 3 2 1 0 false true true false

```cpp
#include <iostream>
#include <vector> //需要引入 vecotr 头文件
using namespace std;
int main()
{
    vector<int> v1(4);                          //创建长度为4的vector
    for (size_t i = 0; i < v1.size(); i++)      //设定 values 容器为 {0,1,2,3}
        v1.at(i) = i;
    auto v2 = v1;
    v2[2] = 1;
    if (!v1.empty()) {      //如果容器不为空，则输出容器中所有的元素
        for (auto it = v1.begin(); it < v1.end(); it++)
            cout << *it << " ";
    }
    cout << endl;
    for(auto e:v2)
        cout << e << " ";
    cout << endl;
    for (auto it = v1.rbegin(); it < v1.rend(); it++) //使用反向迭代器遍历容器
        cout << *it << " ";
    cout << endl << boolalpha;
    cout << (v1 == v2) << '\t' << (v1 != v2)<< endl;
    cout << (v1 > v2) << '\t' << (v1 < v2)<< endl;
}
```

知识点

索引	要点	正链	反链
T831	掌握容器遍历的方式，empty是最高效的容器判定为空的方法	T822	

3.2 vector的典型操作

样例输入	样例输出
(无)	4 S T L L L S L C T S L C S

```cpp
#include<iostream>
#include<vector>
#include<algorithm>                    //for_each和copy
#include<iterator>                     //ostream_iterator
using namespace std;
int main()
{
    vector<char> v;                    //初始化一个空vector容量
    string s = "LTSA";
    for(auto e:string("LTSA")) v.emplace_back(e);     //或 v.assign(s.begin(), s.end());
    cout << v.size() << endl;          //容器中的元素个数
    v.pop_back();
    for (auto it = v.rbegin(); it < v.rend(); it++) //使用反向迭代器遍历容器
        cout << *it << " ";
    cout << endl;
    cout << v.at(0) << '\t'<<v.front()<<'\t'<<v.back()<< endl;
    v.emplace(v.begin()+1, 'C');       //在距离首元素偏移为1的位置插入新字符，也可以使用insert
    for_each(v.begin(), v.end(), [](auto &elem) { cout << elem << ' '; });
    cout << endl;
    v.erase(v.begin() + 2);            //删除距离首元素偏移为2的元素
    copy(v.begin(), v.end(), ostream_iterator<char>(cout, " "));
    return 0;
}
```

随堂练习 8.1

输入一个超大的正整数n，$n \le 10^{30}$，将n逐位保存到一个整型vector中。

知识点

索引	要点	正链	反链
T832	掌握向量vector的典型操作，这是STL中最常用的容器	T541	

3.3 查找重复元素

找出数组中重复的数字。在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

样例输入	样例输出
6 2 5 4 5 3 4	5

```cpp
#include <iostream>
#include <vector>
using namespace std;
class Solution {
public:
    int findRepeatNumber(vector<int>& nums) {
        vector<int> ret(nums.size(),0);
        for(auto e:nums)
        {
            if(ret[e]!=0)  return e;
            ret[e]++;
        }
        return 0;
    }
};

int main ()
{
    Solution s;
    size_t n;
    cin >> n;
    vector<int> nums(n);
    for (size_t i = 0;i<n;++i){
        cin >> nums[i];
    }
    cout << s.findRepeatNumber(nums) << endl;
}
```

- sort排序
- 遍历数组，前面的值和后面的值相等即为答案。

知识点

索引	要点	正链	反链
T833	掌握使用vector代替原生数组，理解vector比原生数组的易用性	T526	

04 高级应用

4.1降序排序

sort函数默认是采用升序排序，第五章提到可以将升序排序的结果调用reverse函数，形成降序。

降序排序

结合迭代器或仿函数，可以直接进行降序排序。

样例输入	样例输出
4 3 5 1 7	7 5 3 1

排序时采用反向迭代器，利用rbegin和rend两个函数，其中的大小比较与前向迭代正好相反，因此可以形成逆序效果。

```
1  #include<iostream>
2  #include<vector>
3  #include<algorithm>
4  using namespace std;
5
6  int main ()
7  {
8      size_t n;
9      cin>>n;
10     vector<int> nums(n);
11     for (size_t i = 0;i<n;++i){
12         cin>>nums[i];
13     }
14     sort(nums.rbegin(),nums.rend());//或sort(nums.begin(),nums.end(),greater<int>());
15     for (size_t i = 0;i<n;++i){
16         cout<<nums[i]<<' ';
17     }
18 }
```

第14行注释的结果，是将仿函数greater作为比较器，大的元素在前，因此也可以达到降序的目的。

知识点

索引	要点	正链	反链
T841	掌握逆序排序的方法	T831	

4.2 全部删除指定元素

例题8.2

据说2011年11月11日是百年光棍节。这个日期写成字符串是“20111111”，有6个1连续出现，小明把这样的字符串（有6个1连续出现，但可以在1之间有空格间隔）叫做光棍串，即“2011 11 11”也是光棍串。

样例输入	样例输出
2 2011111 2011 11 11	No Yes

在字符串中查找是否存在6个1的子串，通过find去掉所有空格。

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;
int main()
{
    int n;
    cin >> n;
    cin.ignore();
    string s;
    while(n--){
        getline(cin,s);
        auto it = remove(s.begin(),s.end(),' ');
        s.resize(it-s.begin());
        //s.erase(remove(s.begin(),s.end(),' '),s.end());
        size_t pos = s.find("111111");
        cout<<(pos==-1?"No":"Yes")<<endl;
    }
    return 0;
}
```

如果不是删除无效元素，而是将无效元素全部置为空格，可以采用如下语句 fill(remove(s.begin(),s.end(),' '),s.end(),' ');

知识点

索引	要点	正链	反链
T842	掌握全部删除指定元素的方法	T541	

4.3 for_each算法*

例题8.3

要求编写程序，将英文字母替换加密。变换规则是：将明文中的所有英文字母替换为字母表中的后一个字母，同时将小写字母转换为大写字母，大写字母转换为小写字母。例如，字母a->B、b->C、…、z->A、A->b、B->c、…、Z->a。输入一行字符，将其中的英文字母按照以上规则转换后输出，其他字符按原样输出。

样例输入	样例输出
Reold Z123?	sFPME a123?

采用简单循环可以达成目标。使用<algorithm>库中for_each算法，可以对容器中的每个元素做相同的处理。

```
#include <iostream>
#include <algorithm>
using namespace std;

void encode(char& ch)
{
    if(ch=='Z')
        ch='a';
    else if(ch=='z')
        ch='A';
    else if(islower(ch))
        ch=char(toupper(ch)+1);
    else if(isupper(ch))
        ch=char(tolower(ch)+1);
}
int main()
{
    string s;
    getline(cin,s);
    for_each(s.begin(),s.end(),encode);
    cout<<s;
    return 0;
}
```

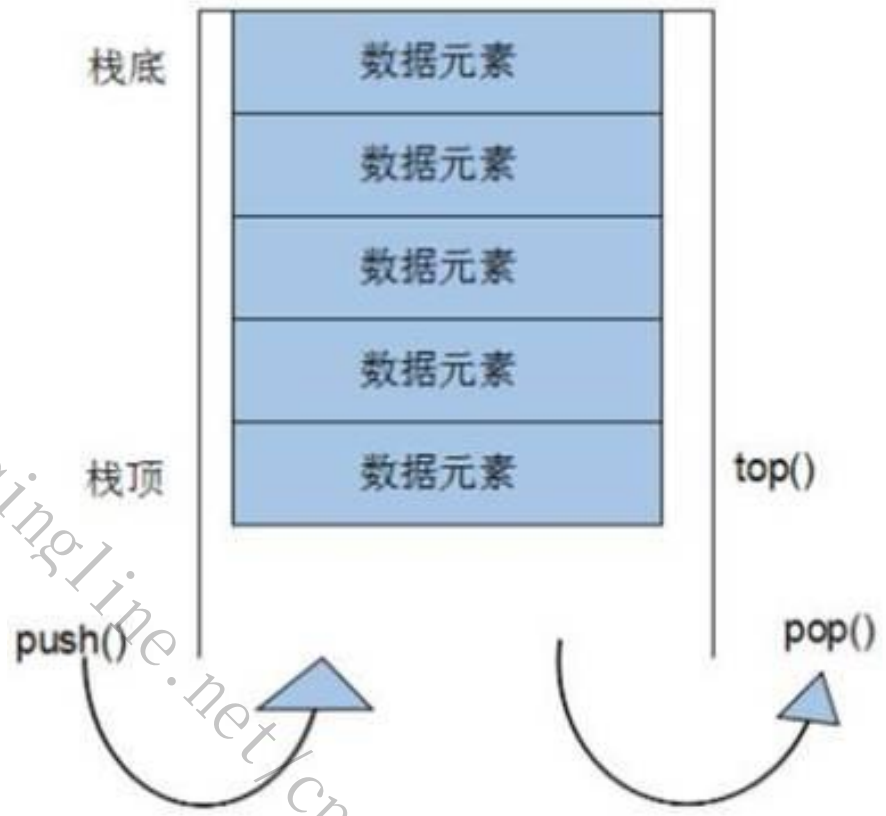
知识点

索引	要点	正链	反链
T843	掌握for_each算法，了解把一个函数作为另外一个函数参数的形式		

05 堆栈stack

堆栈

堆栈stack是先进后出的数据结构，它只能在**尾部**添加或删除，栈中只有**顶端**的元素才可以被外界使用。



例题8.4

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。有效字符串需满足：1）左括号必须用相同类型的右括号闭合。2）左括号必须以正确的顺序闭合。

样例输入	样例输出
()[]{}	true
([)]	false

```cpp
#include <iostream>
#include <stack>
using namespace std;
class Solution {
public:
    bool isValid(string s) {
        stack<char> st;
        for(auto e:s){
            if(e=='('||e=='['||e=='{')
                st.push(e);
            else if(!st.empty()&&abs(e-st.top())<=2)    //有效括号对的ASCII码值不超过2
                st.pop();
            else
                return false;
        }
        return st.empty();                              //左括号有残留
    }
};
int main ()
{
    Solution s;
    string str;
    cin >> str;
    cout <<boolalpha<< s.isValid(str) << endl;
}
```

当字符串遍历结束时，**堆栈应该为空**，否则意味着有残留的左括号未找到匹配的右括号。

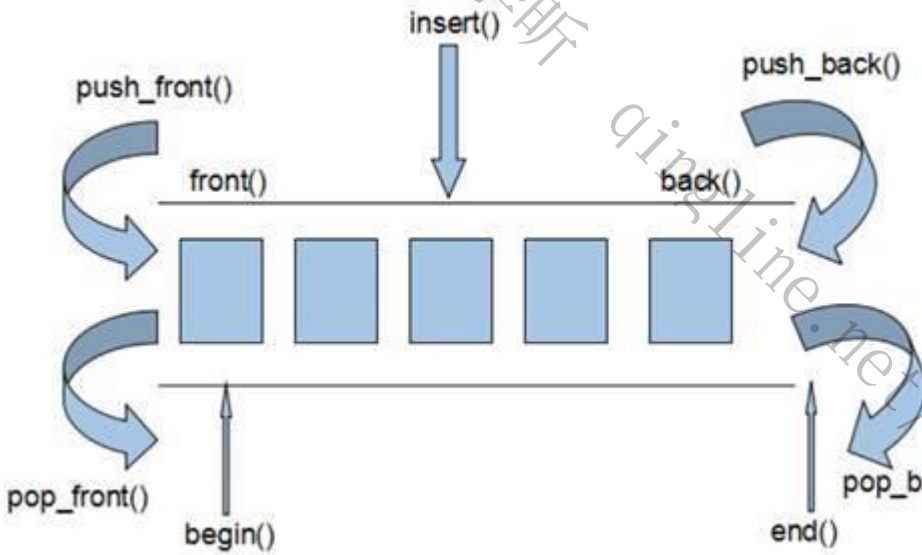
知识点

索引	要点	正链	反链
T851	掌握堆栈stack的用法，学会堆栈增删元素的特点，主要解决匹配问题	T241, T341	

06 其他典型序列容器

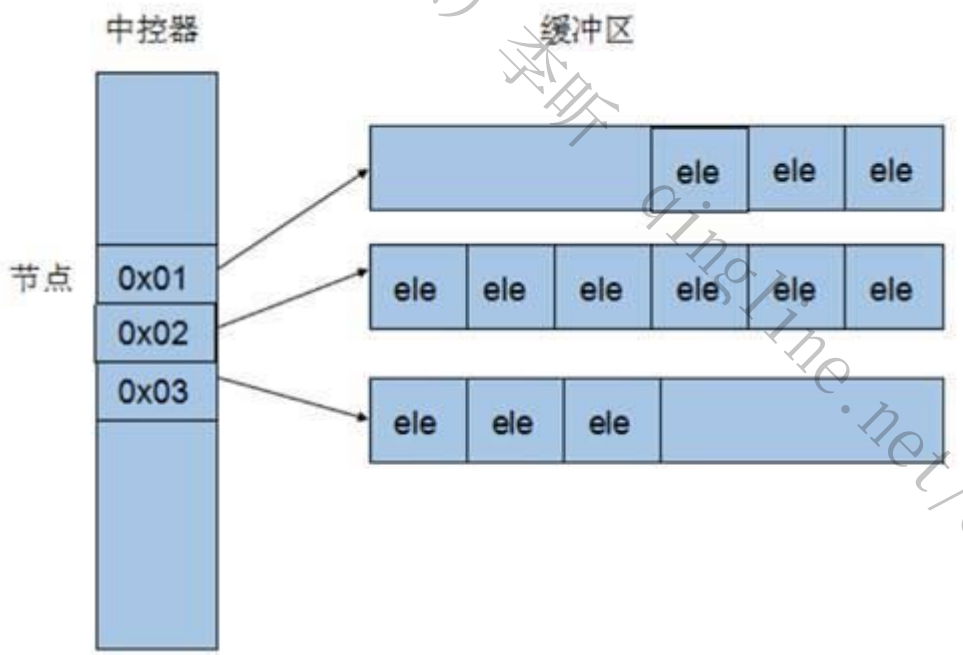
6.1 双向队列

双向队列deque的绝大部分操作都与vector相同，但是可以在两端进行增删操作。



6.1 双向队列

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支持随机访问。



例题8.5

给定一个整数数组 mums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 k 个数字。滑动窗口每次只向右移动一位。返回滑动窗口中的最大值 。

样例输入	样例输出
nums = [1,3,-1,-3,5,3,6,7]	[3,3,5,5,6,7]

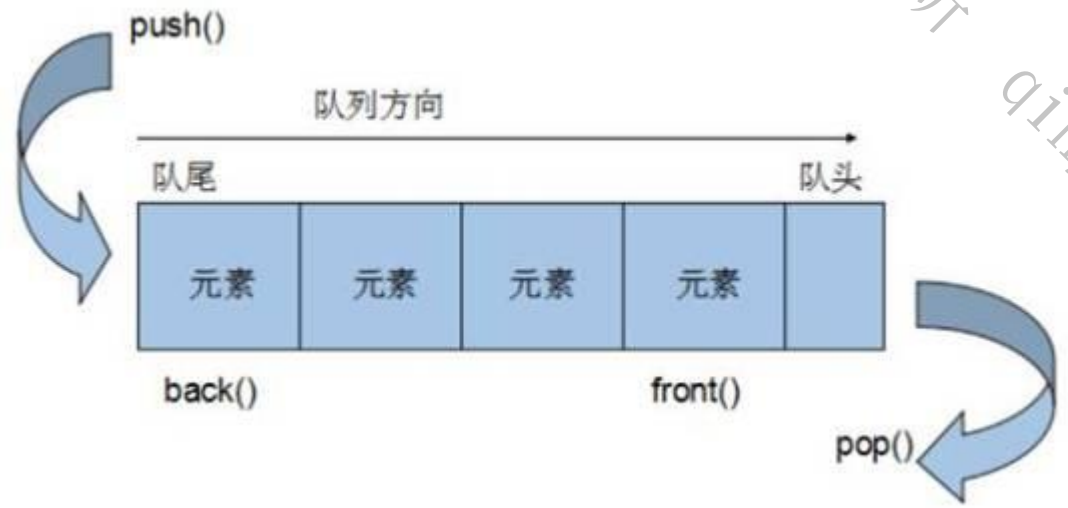
```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, size_t k) {
        vector<int> ret;
        deque<int> qe;
        if(nums.empty())     return ret;
        for(size_t i=0;i<nums.size();i++) {
            //新来的元素>队列之中的元素，说明最大值发生了变化
            while(!qe.empty()&&nums[i]>=nums[qe.back()])
                qe.pop_back();
            //窗口之中保存的是下标
            //i-qe.front()>=k，说明这个元素不在窗口之中了
            while(!qe.empty()&&i-qe.front()>=k)
                qe.pop_front();
            qe.push_back(i);//将下标存入数组中，因为要比较元素是否"过期"
            if(i+1>=k)//经过的元素可以满足一个窗口了
                ret.push_back(nums[qe.front()]);
        }
        return ret;
    }
};
```

知识点

索引	要点	正链	反链
T861	掌握双向队列deque的用法，主要解决在算法中需要设定滑动窗口的问题		T862

6.2 单向队列queue

单向队列queue是在双向队列deque基础上完成的，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。



- queue的在尾部添加函数为**push()**
- 在头部删除函为**pop()**
- 它只有一个push和一个pop操作，这与deque需要区分头部和尾部不同。

例题8.6

给定一个字符串 s ， 请你找出其中不含有重复字符的最长子串的长度。

样例输入	样例输出
akgekwelkrjlkjfka sdfashdfkladfald	7

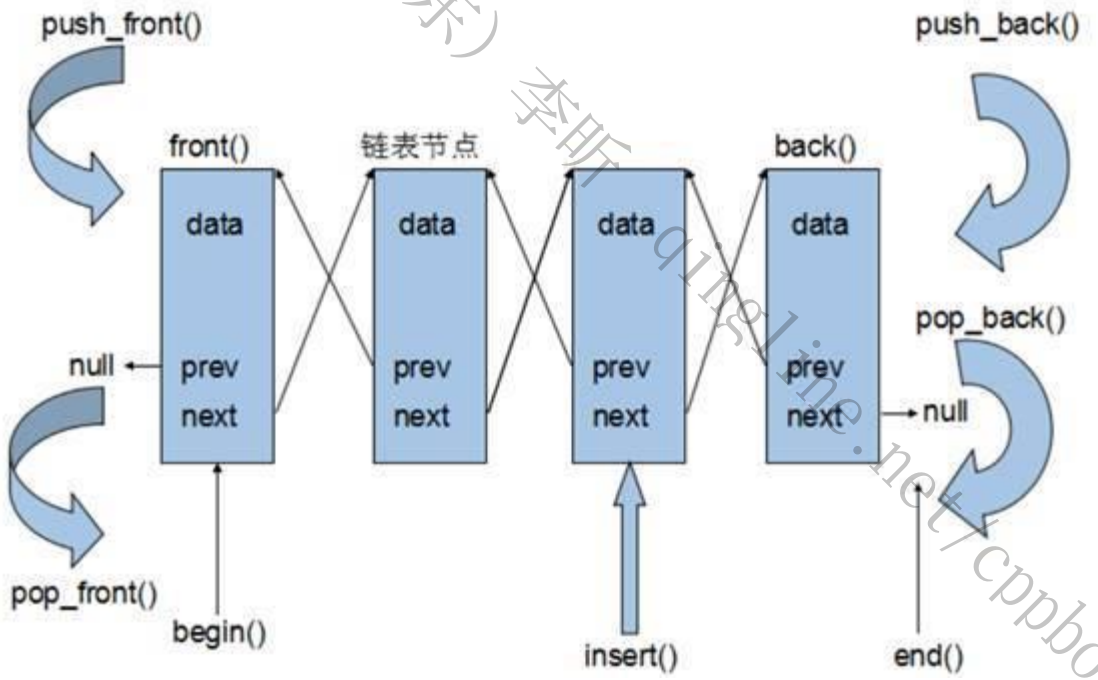
```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        queue<char>de;//滑动窗口
        vector<bool> arr(200,false);            //统计数组
        size_t maxsize=0;                       //记录最长的长度
        for(size_t i=0;i<s.size();i++){
            if(arr[s[i]]){                      //表示不是第一次出现
                maxsize=max(maxsize,de.size()); //先将最长的长度保存下来
                while(arr[s[i]]){               //不为false说明还有重复字符
                    arr[de.front()]=false;      //标记数组对应的字符为清除状态
                    de.pop();                   //删掉
                }
            }
            //此时说明删掉了开始部分重复的字符串，或新出现的字符没有出现过，直接进入滑动窗口
            de.push(s[i]);
            arr[s[i]]=true;                     //标记数组，表示出现过
        }
        maxsize=max(maxsize,de.size());
        return maxsize;
    }
};
```

知识点

索引	要点	正链	反链
T862	掌握单向队列deque的用法，主要解决在算法中需要设定滑动窗口的问题	T861	

6.3 链表list

链表list是一种物理存储单元上非连续的存储结构，数据元素的逻辑顺序是通过链表中的指针链接实现。由一系列结点组成。



6.3 链表list

链表list中的迭代器只支持前移和后移，属于双向迭代器。

优点：

- 采用**动态存储分配**，不会造成内存浪费和溢出；
- 链表执行**插入**和**删除**操作十分方便，修改指针即可，不需要移动大量元素。

缺点：

- 空间(指针域) 和 时间（遍历）额外耗费较大；
- **插入操作和删除操作都不会造成原有list迭代器的失效。**

例题8.7

n 个人围成一圈，从第一个人开始报数,数到 *m* 的人出列，再由下一个人重新从 11 开始报数，数到 m*m* 的人再出圈，依次类推，直到所有的人都出圈，请输出依次出圈人的编号。

【输入】

输入两个整数 n,m。

【输出】

输出一行 n个整数，按顺序输出每个出圈人的编号。

样例输入	样例输出
10 3	3 6 9 2 7 1 8 5 10 4

```cpp
#include<iostream>
#include<list>
using namespace std;
int main(){
    int m,n;
    cin>>m>>n;
    list<int> ls;
    for(int i=0;i<m;i++)
        ls.push_back(i+1);          //构建初始的list
    auto it=ls.begin();
    int i=0;
    while(!ls.empty()){
        i = (i+1)%n;
        auto next=++it;             //备份下一个节点的迭代器
        if(!i) {
            cout<<*(--it)<<' ';     //返回要删除的节点并输出
            ls.erase(it);           //删除节点，迭代器it失效
        }
        it = next;                  //返回到下一个节点
        if(it==ls.end())            //形成循环访问
            it=ls.begin();
    }
    return 0;
}
```

最重要的是形成一个环，然后能在任意位置形成高效的删除操作。list正好符合这样的要求。

例题8.8

写一个程序完成以下命令：

new id ——新建一个指定编号为id的序列(id<10000)

add id num——向编号为id的序列加入整数num

merge id1 id2——合并序列id1和id2中的数，并将id2清空

unique id——去掉序列id中重复的元素

out id ——从小到大输出编号为id的序列中的元素，以空格隔开。

第9行创建了list的数组，数组中的每个元素都是一个list。

```
1   #include<iostream>
2   #include<list>
3   #include<algorithm>      //find_if
4   #include<iterator>       //ostream_iterator
5   using namespace std;
6
7   int main()
8   {
9       list<int> ls[10005];
10      int n,id1,id2, num;
11      cin >> n;
12      char str[100];
```

例题8.8

样例输入	样例输出
16 new 1 new 2 add 1 4 add 1 2 add 1 3 add 2 3 add 2 2 add 2 5 add 2 4 out 1 out 2 merge 1 2 out 1 out 2 unique 1 out 1	2 3 4 2 3 4 5 2 2 3 3 4 4 5 2 3 4 55

第17行的匿名函数中，使用了[&num]，表示引用局部变量num。也可也使用[&]表示引用当前范围内的任何变量，[num]通过传值方式使用num，[=]通过传值方式使用当前范围内的任意变量。

```
13      while(n--){
14          cin >> str;
15          if(str[0]=='a'){
16              cin >> id1 >> num;
17              auto it=find_if(ls[id1].begin(),ls[id1].end(),[&num](int v) {return v>num;});
18              ls[id1].insert(it,num);
19          }else if(str[0]=='n'){
20              cin >> id1;
21          }else if(str[0]=='m'){
22              cin >> id1 >> id2;
23              ls[id1].merge(ls[id2]);          //合并的两个序列必须有序
24          }else if(str[0]=='u'){
25              cin >>id1;
26              ls[id1].unique ();               //唯一化处理前，list必须有序
27          }else if(str[0]=='o'){
28              cin >> id1;
29              copy(ls[id1].begin(), ls[id1].end(), ostream_iterator<int>(cout," "));
30              cout << endl;
31          }
32      }
33      return 0;
34  }
```

仿函数

仿函数（Functor）又称为函数对象（Function Object）是一个能行使函数功能的类。仿函数本质上就是通过重载运算符()，将一个类对象按照函数形式进行访问，其使用方法与普通函数相同。以下定义了一个仿函数comp：

```
1   class comp
2   {
3   public:
4       comp(int t):num(t){}//显式构造函数
5       //const放前面表示这个函数的返回值是不可修改的，放后面表示这个函数不修改当前对象的成员。
6       bool operator()(int v) const{
7           return v>num;
8       }
9   private:
10      const int num;
11  };
```

- 第4行的num(t)表示用形参t对成员属性num进行初始化。
- 第6-8行重载了操作符()，使comp的对象能够像函数一样被调用。

知识点

索引	要点	正链	反链
T863	掌握链表list的用法，这是一个典型的节点空间不连续容器，迭代器终止判断不能用>或<，只能用!=。	T791	

07 字典

7.1 关联容器字典map

字典map是关联容器的典型代表，所有元素都是键值对，在C++中用pair实现。pair中第一个元素是first，作为**key（键值）**，起到索引作用，第二个元素为second，作为**value（实值）**，所有元素都会根据元素的键值自动排序，可以根据key值快速找到value值。

map属于关联式容器，底层结构是用红黑树实现，查找复杂度为**O(log_2n)**，其中n是元素的数量。

例题8.9

有n根可以忽视粗细的棒子。第i棒的长度是ai。 有人想从这些棒子中选出4个棒子，用这些棒子做1个矩形(包括正方形)。 求最大可以制作的矩形面积。

【输入格式】第一行数量n。 第二行n个棒子的长度。$4 \le n \le 10^5, 1 \le a_i \le 10^9$

【输出格式】最大矩形面积，如果无法组成矩形，输出0。

样例输入	样例输出
6 3 1 2 4 2 1	2
10 3 3 3 3 4 4 4 5 5 5	20
4 1 2 3 4	0

因为棒子的总数量相对比较小，可以采用map记录存在的棒子，这样键的总量就会大幅减少，map是按照键值**从小到大**排序的，因此从尾部寻找符合要求的棒子组成矩形即可。

```cpp
#include <iostream>
#include <map>
using namespace std;
int main ()
{
    map<int,int> a;
    int n;
    cin>>n;
    for(int i=0;i<n;++i){
        int val;
        cin>>val;
        if(a.count(val))
        {   if(a[val]<4)   a[val]++; }
        else
            a[val]=1;              //或写为a.insert(pair<int,int>(val,1));
    }
```

例题8.9

有n根可以忽视粗细的棒子。第i棒的长度是ai。 有人想从这些棒子中选出4个棒子，用这些棒子做1个矩形(包括正方形)。 求最大可以制作的矩形面积。

【输入格式】第一行数量n。 第二行n个棒子的长度。$4 \le n \le 10^5, 1 \le a_i \le 10^9$

【输出格式】最大矩形面积，如果无法组成矩形，输出0。

样例输入	样例输出
6 3 1 2 4 2 1	2
10 3 3 3 3 4 4 4 5 5 5	20
4 1 2 3 4	0

第18行的循环逆序遍历map，因为迭代器书写比较复杂，所以**auto**自动构建数据类型书写更方便，而且可以减少语法关键词的记忆。

```
17      int l1=0;
18      for(auto rit=a.rbegin();rit!=a.rend();rit++){
19          if(rit->second>=2 && l1>0){                 //当前棒子数量大于2，并且找到过一对棒子
20              cout<<l1*rit->first<<endl;
21              return 0;
22          }else if(rit->second>=4)                    //当前棒子数量大于4，直接构建方形
23          {
24              cout<<rit->first*rit->first<<endl;
25              return 0;
26          }else if(rit->second>=2 && l1==0){          //找到一对棒子，记录并寻找下一对棒子
27              l1 = rit->first;
28          }
29      }
30      cout<<0<<endl;                                  //没有找到合适的棒子构成矩形
31      return 0;
32  }
```

知识点

索引	要点	正链	反链
T871	掌握关联容器字典map的用法。可以认为这是打表法的高级用法，当数据量比较大但是比较稀疏时，可以用字典代替打表法 注意map的键是从小到大天然排序的	T526	

7.2 无序容器字典unordered_map

无序容器与关联容器的最大区别在于：关联容器底层采用**红黑树**，其所有元素按照键进行排序，当需要进行有序遍历时，非常用用；无序容器底层采用的是**哈希表**，当需要进行随机访问某个键时，**访问速度为常量级，即O(1)**，当需要**频繁进行快速定位**时，无序容器就显示出了它的效率优势。

例题8.10

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target的那两个整数，并返回它们的数组下标。假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。可以按任意顺序返回答案。

【接口声明】

vector<int> twoSum(vector<int>& nums, int target)

【数据范围】

$2 \le nums.length \le 10^4$,

$-10^9 \le nums[i] \le 10^9$,

$-10^9 \le target \le 10^9$

只会存在一个有效答案

nums.length的数值范围决定了键的数量不会太多，哈希map既可以类似打表法中数组的快速定位，也可以避免无效空间的浪费。

```
1  class Solution {
2  public:
3      vector<int> twoSum(vector<int>& nums, int target) {
4          unordered_map<int,size_t> m;
5          for (size_t i = 0; i < nums.size();++i)
6              m[nums[i]] = i;                        //反向记录第i个数值对应的下标
7          for (auto it1 = nums.begin(); it1 < nums.end() - 1; it1++)
8          {
9              if (m.count(target - *it1))            //若存在互补数
10             {
11                 int first = it1 - nums.begin(); //当前数的序号
12                 int second = m[target - *it1];  //互补数的序号
13                 if(first!=second){                  //如果不是同一个元素
14                     return {first,second};          //用两个元素初始化构建一个列表
15                 }
16             }
17         }
18         return vector<int>();//返回一个空的vector，保证语法正确，本题保证不会执行到这里
19     }
20 };
```

例题8.10

样例输入	样例输出
nums = [2,7,11,15], target = 9	[0,1]
nums = [3,2,4], target = 6	[1,2]
nums = [3,3], target = 6	[0,1]

很多文献上使用哈希map时使用头文件<hash_map>中的hash_map，这是一个非标准库，正确方式应该使用头文件<unordered_map>中的**unordered_map**。

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int,size_t> m;
        for (size_t i = 0; i < nums.size();++i)
            m[nums[i]] = i;                         //反向记录第i个数值对应的下标
        for (auto it1 = nums.begin(); it1 < nums.end() - 1; it1++)
        {
            if (m.count(target - *it1))             //若存在互补数
            {
                int first = it1 - nums.begin(); //当前数的序号
                int second = m[target - *it1];  //互补数的序号
                if(first!=second){                  //如果不是同一个元素
                    return {first,second};          //用两个元素初始化构建一个列表
                }
            }
        }
        return vector<int>();//返回一个空的vector，保证语法正确，本题保证不会执行到这里
    }
};
```

知识点

索引	要点	正链	反链
T872	掌握无序容器的用法，重点掌握利用unordered_map构建大且稀疏数据的打表法	T526	

7.3 无序容器字典unordered_set

可以认为字典是一种下标为任意类型的特殊数组，**因为字典可以遍历，因此可以和循环联动**，简化书写。

例题8.11

给你一个整数数组 nums 和一个整数 k ，判断数组中是否存在两个 不同的索引 i 和 j ，满足 nums[i] == nums[j] 且 abs(i - j) <= k 。如果存在，返回 true ；否则，返回 false 。

【接口声明】

bool containsNearbyDuplicate(vector<int>& nums, int k)

【数据范围】

1 <= nums.length <= 10^5

-10^9 <= nums[i] <= 10^9

0 <= k <= 10

【题目解析】可以通过暴力解决，但效率比较低。以下引入滑动窗口的概念，只在长度为k的范围内进行查找，并且利用哈希表查找复杂度为O(1)的特点，加快查找的效率。

```
1  class Solution {
2  public:
3      bool containsNearbyDuplicate(vector<int>& nums, int k) {
4          unordered_set<int> s;
5          for (int i = 0; i < nums.size(); i++) {
6              if (i > k) {
7                  s.erase(nums[i - k - 1]);        //删除超过距离k的元素
8              }
9              if (s.count(nums[i])) {              //哈希查找
10                 return true;
11             }
12             s.emplace(nums[i]);                  //添加新元素
13         }
14         return false;
15     }
16 };
```

例题8.11

样例输入	样例输出
nums = [1,2,3,1], k = 3	true
nums = [1,0,1,1], k = 1	true
nums = [1,2,3,1,2,3], k = 2	false

建立一个unordered_set对象，其中只保留k个元素，相当于一个滑动窗口。当距离超过k时，将元素从窗口中删除，第12行将新元素添加到窗口中。第9行利用哈希O(1)的复杂度进行判断是否存在。如果存在则返回true。

```
1  class Solution {
2  public:
3      bool containsNearbyDuplicate(vector<int>& nums, int k) {
4          unordered_set<int> s;
5          for (int i = 0; i < nums.size(); i++) {
6              if (i > k) {
7                  s.erase(nums[i - k - 1]);        //删除超过距离k的元素
8              }
9              if (s.count(nums[i])) {              //哈希查找
10                 return true;
11             }
12             s.emplace(nums[i]);                  //添加新元素
13         }
14         return false;
15     }
16 };
```

知识点

索引	要点	正链	反链
T873	重点掌握利用unordered_set构建滑动窗口，理解哈希表查找复杂度为O(1)的特性，利用这一特性，可以代替数组的打表法，尤其对稀疏或非数值数据具有良好的效果。	T526	

7.4 字典与循环的联动

可以认为字典是一种下标为任意类型的特殊数组，**因为字典可以遍历，因此可以和循环联动**，简化书写。

例题8.12

罗马数字包含以下七种字符：

I， V， X， L，C，D 和 M

字符	数值	字符	数值	字符	数值	字符
I	1	V	5	X	10	L
数值	字符	数值	字符	数值	字符	数值
50	C	100	D	500	M	1000

例如， 罗马数字2写做 II，即为两个并列的1。12写做XII，即为X+II。 27写做XXVII，即为XX+V+II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。

X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。

C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。

```cpp
class Solution {
public:
    int romanToInt(string s) {
        int sum=0;
        for(int i=0;i<s.size();++i){
            if(s[i]=='I'){
                if(s[i+1]=='V'){++i;sum+=4;}
                else if(s[i+1]=='X'){++i;sum+=9;}
                else sum++;
            }
            else if(s[i]=='X'){
                if(s[i+1]=='L'){++i;sum+=40;}
                else if(s[i+1]=='C'){++i;sum+=90;}
                else sum+=10;
            }
            else if(s[i]=='C'){
                if(s[i+1]=='D'){++i;sum+=400;}
                else if(s[i+1]=='M'){++i;sum+=900;}
                else sum+=100;
            }
            else if(s[i]=='V'){sum+=5;}
            else if(s[i]=='L'){sum+=50;}
            else if(s[i]=='D'){sum+=500;}
            else if(s[i]=='M'){sum+=1000;}
        }
        return sum;
    }
};
```

可以通过多重判断解决。
但是书写比较复杂。

例题8.12

【接口声明】

int romanToInt(string s)

【数据范围】

1 <= s.length <= 15

s 仅含字符 ('I', 'V', 'X', 'L', 'C', 'D', 'M')

题目数据保证 s 是一个有效的罗马数字，且表示整数在范围 [1, 3999] 内

题目所给测试用例皆符合罗马数字书写规则，不会出现跨位等情况。

样例输入	样例输出
s = "III"	3
s = "LVIII"	58
s = "MCMXCIV"	1994

造成以上代码多重判断的复杂性根源在于键是一系列不规律的字符，可以将这些特殊键构造成字典，从而简化循环书写逻辑。

```
1  class Solution {
2  public:
3      int romanToInt(string s) {
4          int sum=0;
5          unordered_map<char,int> m1={{'I',1},{'V',5},{'X',10},{'L',50},
6                                      {'C',100},{'D',500},{'M',1000}};
7          unordered_map<string,int> m2={{"IV",4},{"IX",9},{"XL",40},
8                                        {"XC",90},{"CD",400},{"CM",900}};
9          for(int i=0;i<s.size();++i){
10             if(m2.count(s.substr(i,2))) sum+=m2[s.substr(i++,2)];
11             else sum+=m1[s[i]];
12         }
13         return sum;
14     }
15 };
```

第10行判断双字符键是否存在，如果存在则增加对应的值。特别注意**i++**，因为是双字符键，需要跨越两个字符，因此sum累加后，要将i增加1。

第11行对单字符键进行累加操作。

随堂练习 8.2

仿照C++的定义对可能含有转义序列的字符串进行转换，输出转换后的结果。**只需实现**：\n, \t, \?, \', \", \\即可。注意根据知识点T274，当输入中有转义字符时，不会认为是转义字符，而会逐个字符处理。

样例输入	样例输出
new\nline	new line
T\tAB	T AB
\?\'\"\\	?'"\

知识点

索引	要点	正链	反链
T874	掌握字典与循环联动的方法，理解字典在书写上类似特殊下标的数组	T522	

THANKS