C++常用STL讲解

1、STL是什么?

搬砖工具

STL 是一个 C++ 库，包含算法、容器、函数、迭代器

2、字符串（string)

参考文档：[C++ string 类详解 - tongye - 博客园 (cnblogs.com)]

字符串是存储在内存的连续字节中的一系列字符。C++ 处理字符串的方式有两种，一种来自 C 语言，常被称为 C-风格字符串，另一种是基于 string 类库的字符串处理方式。这里，我们主要学习string类对字符串的处理。

```
#include <string>   //要包含string头文件
using namespace std;//在命名空间std里
创建string类型变量
string str;
string str(10,'a');创建一个包含 10 个元素的 string 对象，其中每个元素都被初始化为字符 'a'
string str = "chdacm"; 创建string时直接用字符串内容对其赋值，注意字符串要用双引号""
Input:
cin >> str;
getline (cin, str); 按行读入
Output:
cout << str << endl; 输出字符串

赋值：= 将后面的字符串赋值给前面的字符串O(n)
比较：== != < <= > >= 比较的是两个字符串的字典序大小 O(n)
连接：+ += 将一个运算符加到另一个运算符后面O(n)
用 string.append() 函数来在一个 string 对象后面附加一个 string 对象或 C 风格的字符串
str1.append("CHDACM");
str.push_back('a');使用 string.push_back() 函数来在一个 string 对象后面附加一个字符
str[idx] 返回字符串s中下标为idx的字符，string中下标也是从0开始O(1)
str.substr(p, n) 返回从s的下标p开始的n个字符组成的字符串，如果n省略就取到底O(n)
str.length()返回字符串的长度O(1)
str.empty()判断s是否为空，空返回1，不空返回0，O(1)
str.erase(p0, len) 删除s中从p0开始的len个字符，如果len省略就删到底O(n)
str.erase(s.begin() + i) 删除下标为i的字符 O(n)
str1.insert(p0, str2, pos, len) 后两个参数截取s2，可以省略 O(n)
str.insert(p0, n, c) 在p0处插入n个字符c O(n)
str1.replace(p0, len0, str2, pos, len) 删除p0开始的len0个字符，然后在p0处插入串str2中从pos开始的len个字符，后两个参数可以省略 O(n)
str.reverse(s.begin(), s.end());反转字符串

str1.find(str2) 从前往后，查找成功时返回第一次出现的下标，失败返回-1 O(n*m)
str1.rfind(str2, pos) 从pos开始从后向前查找字符串str2中字符串在当前串后边第一次出现的下标 O(n*m)
```

3、动态数组（vector）

参考文档:C++ STL vector容器详解 (biancheng.net)

在c++中，vector是一个十分有用的容器

作用：它能够像容器一样存放各种类型的对象，简单地说，vector是一个能够存放任意类型的动态数组，能够增加和压缩数据。

vector在C++标准模板库中的部分内容，它是一个多功能的，能够操作多种数据结构和算法的模板类和函数库。

```
vector<type>v 创建动态数组v
type v[index] 获取v中第 index 个元素 O(1)
v.push_back(type item) 向v后面添加一个元素item O(1)
v.pop_back() 删除 v 最后一个元素 O(1)
v.size() 获取 v 中元素个数 O(1)
v.resize(int n) 把 v 的长度设定为 n 个元素 O(n)
v.empty() 判断 v 是否为空 O(1)
v.clear() 清空 v 中的元素
v.insert(iterator it, type x)向迭代器it指向元素前增加一个元素x,O(n)
v.erase(iterator it) 删除向量中迭代器指向元素,O(n)
v.front()返回首元素的引用O(1)
v.back()返回尾元素的引用O(1)
v.begin()返回首迭代器，指向第一个元素O(1)
v.end()返回尾迭代器，指向向量最后一个元素的下一个位置O(1)

//vector的创建
#include <vector>
using namespace std;
int main()
{
    vector<int> v1;
    vector<double> v2{1, 1, 2, 3, 5, 8};
    vector<long long> v3(20);
    vector<string>v4(20, "chdacm");
    //创建一个存string类型的动态数组，长度为20，存的都是"chdacm"
    vector<int>v5[3];
    vector<vector<int>> v5{{1, 2}, {1}, {1, 2, 3}};
    //存int的二维数组，行和列都可变，初始状态
    return 0;
}
```

```
int main()
{
    vector<int>v;
    for(int i = 1; i <= 5; i ++)
        v.push_back(i);//向动态数组中插入1~5
    cout << v.size() << endl;//输出数组的大小，有几个值
    for(int i = 0; i < 5; i ++)
        cout << v[i] << " ";//输出v[0]~v[4]，也就是1~5
    cout << endl;
    v.clear();//将v清空，此时size为0
    v.resize(10);//为v重新开辟大小为10的空间，初始为0
    for(int i = 0; i < v.size(); i ++)
        cout << v[i] << " ";
    while(!v.empty())
```

```
        v.pop_back();
    return 0;
}
```

```
int main()
{
    vector<int> v{0, 1, 2, 3, 4};
    v.erase(v.begin() + 3);//删除v[3]，v变为{0, 1, 2, 4}
    v.insert(v.begin() + 3, 666);//在v[3]前加上666，v变成{0, 1, 2, 666, 4}
    v.front() = 10;//将v[0]改成10，等同于v[0]=10;
    v.back() = 20;//将v[4]改成20等同于v[v.size()-1]=20;
    for(int i = 0; i < v.size(); i ++)
        cout << v[i] << " ";//使用下标访问的方法遍历v
    cout << endl;
    for(auto i = v.begin(); i != v.end(); i ++)
        cout << *i << " ";//使用迭代器，从v.begin()到v.end()-1
    cout << endl;
    for(auto i : v)
        cout << i << " ";
    cout << endl;
    return 0;
}
```

4、*关于迭代器（iterator）

```
vector<int>::iterator it = v.begin();
或者 auto it = v.begin();
cout << *it << endl;//相当于cout << v[0] << endl;
it ++;
cout << *it << endl;
```

5、队列和栈（queue and stack）

queue只能在容器的末尾添加新元素，只能从头部移除元素。(先进先出)

stack 只能在栈顶插入元素，也只能在栈顶取出元素（先进后出）

```
创建方法：
queue<type> q; 建立一个存放数据类型为type的队列q
使用方法：
▸ q.push(item)：在 q 的最后添加一个type类型元素item O(1)
▸ q.pop()：使 q 最前面的元素出队 O(1)
▸ q.front()：获取 q 最前面的元素 O(1)
▸ q.size()：获取 q 中元素个数 O(1)
▸ q.empty()：判断 q 是否为空，空返回1，不空返回0 O(1)

stack<type>stk;与queue类似
```

6、优先队列（priority_queue）

priority_queue中出队顺序与插入顺序无关，与数据优先级有关，本质是一个堆

```
创建方法：
priority_queue<Type, Container, Functional>
// Type：数据类型
// Container：存放数据的容器，默认用的是vector
// Functional：排序方法
使用方法
 pq.push(item)：在 pq 中添加一个元素 O(logn)
 pq.top()：获取 pq 最大的元素 O(1)
 pq.pop()：使 pq 中最大的元素出队 O(logn)
 pq.size()：获取 pq 中元素个数 O(1)
 pq.empty()：判断 pq 是否为空 O(1)
```

priority_queue中存的元素如果是结构体这样无法进行比较的类型，必须要重载运算符<，相当于先使得优先队列中的元素可以进行比较再建立pq，否则直接建优先队列是会报错的

```
#include <iostream>
#include <queue>
using namespace std;
struct Node{
    int x, y;
};
bool operator<(Node a, Node b){
    return a.x < b.x;
}
int main(){
    priority_queue<Node> pq;
    pq.push({1, 3});
    pq.push({3, 2});
    pq.push({2, 1});
    while(!pq.empty())
    {
        cout << pq.top().x << ' ' << pq.top().y << endl;
        pq.pop();
    }
    return 0;
}
```

7、集合（set）

集合(set)是一种包含对象的容器，可以快速地（logn）查询元素是否在已知几集合中。

set 中所有元素是有序地，且只能出现一次，因为 set 中元素是有序的，所以存储的元素必须已经定义过 < 运算符（因此如果想在 set 中存放 struct 的话必须手动重载 < 运算符，和优先队列一样）

与set类似的还有

- multiset元素有序可以出现多次
- unordered_set元素无序只能出现一次
- unordered_multiset元素无序可以出现多次

```
建立方法:
set<Type>s;
multiset<Type>s;
unorded_set<Type>s;
unorded_multiset<Type>s;
如果Type无法进行比较，还需要和优先队列一样定义 < 运算符
遍历方法:
for(auto i : s) cout << i << " ";
//和vector的类似
```

```
使用方法:
s.insert(item)：在 s 中插入一个元素 O(logn)
s.size()：获取 s 中元素个数 O(1)
s.empty()：判断 s 是否为空 O(1)
s.clear()：清空 s O(n)
s.find(item)：在 s 中查找一个元素并返回其迭代器，找不到的话返回 s.end() O(logn)
s.begin()：返回 s 中最小元素的迭代器，注意set中的迭代器和vector中的迭代器不同，无法直接加上某个数，因此要经常用到--和++O(1)
s.end()：返回 s 中最大元素的迭代器的后一个迭代器 O(1)
s.count(item)：返回 s 中 item 的数量。在set中因为所有元素只能在 s 中出现一次，所以返回值只能是 0 或 1，在multiset中会返回存的个数 O(logn)
s.erase(position)：删除 s 中迭代器position对应位置的元素O(logn)
s.erase(item)：删除 s 中对应元素 O(logn)
s.erase(pos1, pos2)：删除 [pos1, pos2) 这个区间的位置的元素 O(logn + pos2 - pos1)
s.lower_bound(item)：返回 s 中第一个大于等于item的元素的迭代器，找不到就返回
s.end() O(logn)
s.upper_bound(item)：返回 s 中第一个大于item的元素的迭代器，找不到就返回s.end() O(logn)
```

8、映射（map）

map 是照特定顺序存储由 key 和 value 的组合形成的元素的容器， map 中元素按照 key 进行排序，每个 key 都是唯一的，并对应着一个value，value可以重复

- 与map类似的还有unordered_map(哈希表)，区别在于key不是按照顺序排序

```
建立方法:
map<key,value> mp;
unordered_map<key,value> mp;
遍历方法:
for(auto i:mp)
    cout << i.first << ' ' << i.second << endl;
```

```
mp.size()：获取 mp 中元素个数 O(1)
mp.empty()：判断 mp 是否为空 O(1)
mp.clear()：清空 mp O(1)
mp.begin()：返回 mp 中最小 key 的迭代器，和set一样，只可以用到--和++操作 O(1)
mp.end()：返回 mp 中最大 key 的迭代器的后一个迭代器 O(1)
mp.find(key)：在 mp 中查找一个 key 并返回其 iterator，找不到的话返回 s.end() O(logn)
mp.count(key)：在 mp 中查找 key 的数量，因为 map中 key 唯一，所以只会返回 0 或 1 O(logn)
mp.erase(key)：在 mp 中删除 key 所在的项，key和value都会被删除 O(logn)
mp.lower_bound(item)：返回 mp 中第一个key大于等于item的迭代器，找不到就返回
mp.end() O(logn)
mp.upper_bound(item)：返回 mp 中第一个key大于item的迭代器，找不到就返回mp.end() O(logn)
mp[key]：返回 mp 中 key 对应的 value。如果key 不存在，则返回 value 类型的默认构造器
(defaultconstructor)所构造的值，并该键值对插入到 mp 中O(logn)
mp[key] = xxx：如果 mp 中找不到对应的 key 则将键值对 (key: xxx) 插入到 mp 中，如果存
在 key 则将这个 key 对应的值改变为 xxx O(logn)
```

[3581. 单词识别 - AcWing题库]

题目大意：输入一行字符串，输出每个单词和他的出现次数，格式word:times；

AC代码：

```
#include <string>
#include <map>
#include <iostream>
#include <vector>
using namespace std;
map<string, int> used;
string s;
int main(){
    getline(cin, s);
    string now = "";
    for(auto x : s)
    {
        if(x == ' ' || x == ',' || x == '.')
        {
            if(now != "")
            {
                used[now] ++;
                now = "";
            }
        }
        else
        {
            now += tolower(x);
        }
    }
    if(now != "") used[now] ++;
    for(auto &[x,y] : used)
    {
        cout<<x<<':'<<y<<endl;
    }
    return 0;
}
```

9、常用函数

- max(val1, val2)：返回更大的数
- min(val1, val2)：返回更小的数
- swap(type, type)：交换两者的值，可以是两个值也可以是两个STL的容器

sort(first, last, compare)

- first：排序起始位置（指针或 iterator）
- last：排序终止位置（指针或 iterator）
- compare：比较方式，可以省略，省略时默认按升序排序，如果排序的元素没有定义比较运算（如结构体），必须有compare
- sort 排序的范围是 [first, last)，时间为 O(nlogn)

```
#include <algorithm>
#include <iostream>
using namespace std;
bool cmp(int a, int b){return a > b;}
int main()
{
    int arr[]={3,2,5,1,4};
    sort(arr, arr+5);
    sort(arr, arr+5,greater<int>());//从大到小排
    sort(arr, arr+5,cmp);
    sort(arr, arr+5,[](int a, int b){return a > b;});
    return 0;
}
```

去重函数（unique）

```
unique(first, last)：

▸ [first, last)范围内的值必须是一开始就提前排好序的
▸ 移除 [first, last) 内连续重复项
▸ 去重之后的返回最后一个元素的下一个地址（迭代器）

#include <algorithm>
using namespace std;
int main()
{
    int arr[] = {3, 2, 2, 1, 2}, n;
    sort(arr, arr + 5);//需要先排序
    n=unique(arr, arr + 5) - arr;//n是去重后的元素个数
    return 0;
}
```

二分函数（lower_bound/upper_bound）

lower_bound(first, last, value)
▸ first：查找起始位置（指针或 iterator）
▸ last：查找终止位置（指针或 iterator）
▸ value：查找的值
▸ lower_bound 查找的范围是 [first, last)，返回的是序列中第一个大于等于 value 的元素的位置，时间为 O(logn)
▸ [first, last)范围内的序列必须是提前排好序的，不然会错
▸ 如果序列内所有元素都比 value 小，则返回last
upper_bound(first, last, value)
▸ upper_bound 与 lower_bound 相似，唯一不同的地方在于upper_bound 查找的是序列中第一个大于 value 的元素

```
int main()
{
    int arr[] = {3, 2, 5, 1, 4};
    sort(arr, arr + 5);//需要先排序
    cout << *lower_bound(arr, arr + 5, 3);//输出数组中第一个大于等于3的值
    return 0;
}
```

10、总结

C++ STL简介

```
#include <vector>
vector，变长数组
    size()  返回元素个数
    empty()  返回是否为空
    clear()  清空
    front()/back()
    push_back()/pop_back()
    begin()/end()
    []
```

```
pair<int, int>
    first，第一个元素
    second，第二个元素
    支持比较运算，以first为第一关键字，以second为第二关键字
```

```
#include <string>
string，字符串
    szie()/length()  返回字符串长度
    empty()
    clear()
    substr(起始下标，(子串长度))  返回子串
    c_str() 转化成C语言风格字符串
```

```
#include <queue>
queue, 队列
    size()
    empty()
    push()  向队尾插入一个元素
    front()  返回队头元素
    back()  返回队尾元素
    pop()  弹出队头元素
```

```
#include <queue>
priority_queue, 优先队列，默认是大根堆
    push()  插入一个元素
    top()  返回堆顶元素
    pop()  弹出堆顶元素
    定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;
```

```
#include <stack>
stack, 栈
    size()
    empty()
    push()  向栈顶插入一个元素
    top()  返回栈顶元素
    pop()  弹出栈顶元素
```

```
#include <dqueue>
deque, 双端队列
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []
```

```
#include <set>
#include <map>
set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
    clear()
    begin()/end()
    ++, -- 返回前驱和后继，时间复杂度 O(logn)
```

```
set/multiset
    insert()  插入一个数
    find()  查找一个数
    count()  返回某一个数的个数
    erase()
        (1) 输入是一个数x，删除所有x   O(k + logn)
        (2) 输入一个迭代器，删除这个迭代器
    lower_bound()/upper_bound()
        lower_bound(x)  返回大于等于x的最小的数的迭代器
        upper_bound(x)  返回大于x的最小的数的迭代器
```

```
map/multimap
    insert()  插入的数是一个pair
    erase()  输入的参数是pair或者迭代器
    find()
    []   时间复杂度是 O(logn)
    lower_bound()/upper_bound()
```

```
unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound()， 迭代器的++，--
```

```
bitset, 压位
    bitset<10000> s;
    ~, &, |, ^
    >>, <<
    ==, !=
    []
    count()  返回有多少个1
    any()  判断是否至少有一个1
    none()  判断是否全为0
    set()  把所有位置成1
    set(k, v)  将第k位变成v
    reset()  把所有位变成0
    flip()  等价于~
    flip(k) 把第k位取反
```