知乎地址：https://zhuanlan.zhihu.com/c_1101089619118026752
作者：小哲
github：https://github.com/lxztju/notes
微信公众号：小哲AI

C++ STL 总结

STL算法

```
#include<algorithm>
```

非变序算法

计数算法

```
//count
//count_if
//用法类似于下文中的find函数
```

搜索算法

```
//search
//search_n
//find
//find_if
//find_end


//find
int num = 5;
vector<int>::const_iterator findnum= find(VectorInts.begin(), VectorInts.end(),
num);  //这里可以采用auto定义比较简单
cout << *findnum <<endl;
//findif
auto findnum2 = find_if( VectorInts.begin(), VectorInts.end(),
                        [](int element){return (element %2 ==0);});
cout << *findnum2 <<endl;
cout << "position is : " << distance(VectorInts.begin(), findnum2);

//search
auto searchnums = search(VectorInts.begin(), VectorInts.end(),
                         findVector.begin(), findVector.end());
```

比较算法

```
//equal
//mismatch


```

变序算法

初始化算法

```
//fill
//fill_n
//generate
//generate_n
```

修改

```
//for_each
//trasnsform

//for_each对指定范围内的元素应用一元函数

//transform 可以应用一元或者二元函数
//tolower
//toupper
```

删除

```
//remove
//remove_if
//remove_copy
//remove_copy_if
//unique
//unique_copy
```

替换

```
//replace
//replace_if
```

排序

```
//sort
//stable_sort
//partial_sort
//partial_sort_copy
```

分区

```
//partition
//stable_partition
```

排序容器

```
//binary_search
//lower_bound
//upper_bound
```

STL容器类

1. STL string类

```
#include <string>

//初始化

const char* str1 = "Hello World";
string str2 (str1);
cout << str2<<endl;


// 拼接字符串
//采用+= 或者append
string str1 ("hello");
string str2 ("world");
str1.append(str2);

// 查找字符或者字符串
//find 第二个参数为起始的位置,返回索引的位置, 如果没有找到返回-1,或者用string::npos来进行判断
position = str1.find('ll', 0)


//截短字符串
// erase(index, num)删除索引元素开始的num个字符,
str1.erase(3, 5)

//字符串反转
reverse(str1.begin(), str1.end());
```

2. STL vector类

动态数组

```
#include <vector>
// 初始化
vector <int> vectorint (10, 3); //10个数字,初始值为3,
vector<vector<int>> vectvect (4, vector<int>(5, 6)); // 4行5列初始值为6

//push_back()加入元素
```

```
vectorint.push_back(35);

// pop_back() 移除末尾元素
vectorint.pop_back()


//insert()
vectorint.insert(vectorint.begin(), 3, 19);//表示在开头插入3个19

//size()
vectorint.size()//返回vector中元素的数目
```

3. STL deque类

```
#include <deque>

//初始化
deque<int> testdeque(10, 3);

//push_back()
//popback()
//push_front()
//pop_front()
// 访问两种方式
testdeque.push_back(4);
testdeque.push_back(4);
testdeque.push_front(5);
testdeque.push_front(5);
for (size_t i=0; i <testdeque.size(); i++){
    cout << i << ": "<<testdeque[i]<<endl;
}
testdeque.pop_back();
testdeque.pop_front();
for (auto e = testdeque.begin(); e != testdeque.end(); e++){
    cout << *e<< endl;
}
```

4. STL list类

```
    //初始化
    list<int> listTest (10, 3);
    //push_back()
    //push_front()
    listTest.push_back(5);
    listTest.push_back(5);
    listTest.push_front(4);
    listTest.push_front(4);
    cout <<"initialize"<<endl;
    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;
```

```
    //pop_back() pop_front()
    listTest.pop_front();
    listTest.pop_back();
    cout <<" pop: "<<endl;
    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;

    //insert()
    listTest.insert(listTest.begin(), 6); //开头插入6

    listTest.insert(listTest.end(), 7); //末尾插入7

    listTest.insert(listTest.begin(), 5, 8);//开头插入5个8

    listTest.insert(listTest.begin(), listTest.begin(), listTest.end()); //开头插
入listtest中元素

    cout<<"insert: "<<endl;
    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;

    //erase()
    auto e = listTest.begin();
    e++;
    e++;
    listTest.erase(listTest.begin(), e);  //接收两个迭代器,删除两个迭代器中间元素

    cout<< "erease: "<<endl;
    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;


    //reverse()
    listTest.reverse();

    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;

    //sort()
    listTest.sort(); //排序整个链表

    //remove()
    listTest.remove(8); //删除某个元素

    for (auto e = listTest.begin(); e!= listTest.end(); e++)
        cout <<*e << " " ;
    cout <<endl;
```

5. STL forward_list类

这是一个单向链表,不同于list的双向链表,只能push_front(),不能采用push_back(),insert可以使用.

6. STL 集合

集合类

set, multiset有序的, 查询时间复杂度为log

unordered_set, unordered_multiset无序,查询时间复杂度为常数

```
#include <set>

template <typename T>
struct SortDescending
{
    bool operator() (const T& lhs, const T& rhs) const
    {
        return (lhs > rhs);
    }
};


template <typename T>
void DisplayConstents(const T& input){
    for (auto e= input.begin(); e!= input.end(); e++){
        cout << *e << ' ';
    }
    cout <<endl;
}

//初始化
    set<int> setInteger1;
    multiset<int> msetInteger1;
    set<int, SortDescending<int>> setInteger2;
    multiset<int, SortDescending<int>> msetInteger2;

    //插入元素
    setInteger1.insert(30);
    setInteger1.insert(20);
    setInteger2.insert(10);
    setInteger2.insert(15);
    DisplayConstents(setInteger1);
    DisplayConstents(setInteger2);

    msetInteger1.insert(setInteger1.begin(), setInteger1.end());
    DisplayConstents(msetInteger1);

    //查找find()
    auto find_integer1 = setInteger1.find(20);
    cout << (find_integer1 == setInteger1.end())<<endl;
```

```
    auto find_integer2 = setInteger1.find(70);
    cout << (find_integer2 == setInteger1.end())<<endl;

    cout << typeid(find_integer1).name() <<endl;

    // 删除元素
    msetInteger2.insert(43);
    msetInteger2.insert(23);
    msetInteger2.insert(34);
    msetInteger2.insert(45);
    msetInteger2.insert(56);
    int msetsize = msetInteger2.size();
    cout <<"size: " << msetsize<<endl;

    msetInteger2.erase(23);
    DisplayConstents(msetInteger2);
```

7. STL映射

利用键值对进行查找

map, multimap　有序的map

unordered_map, unordered_multimap　无序的map

```
#include <map>
    //初始化
    map<int, string> mapinttoString1;
    multimap<int, string> mmapinttoString1;

    map<int, string, SortDescending<int>> mapinttostring2;
    multimap<int, string, SortDescending<int>> mmapinttostring2;


    //插入元素
    mapinttoString1.insert(map<int, string>::value_type(3, "Three"));

    mapinttoString1.insert(make_pair(-1, "minus one"));

    mapinttoString1.insert(make_pair(-1, "minus 1"));  //插入无效

    mapinttoString1.insert(pair<int, string> (100, "handred"));

    mapinttoString1[200] = "Two hundred";

    mapinttoString1[200] = "Two hundreds";  //覆盖之前的元素
    DisplayConstents(mapinttoString1);

    mmapinttoString1.insert(map<int, string>::value_type(3, "Three"));

    mmapinttoString1.insert(make_pair(-1, "minus one"));

    mmapinttoString1.insert(make_pair(-1, "minus 1"));  //插入无效

    mmapinttoString1.insert(pair<int, string> (100, "handred"));
```

```
    //查找find()

    auto findmap = mapinttoString1.find(-1);
    if (findmap == mapinttoString1.end())
    {
        cout <<"not" <<endl;
    }
    else
        cout << findmap->first << "   " << findmap->second <<endl;


    auto findmmap = mmapinttoString1.find(-1);

    //在multimap中查找
    size_t num = mmapinttoString1.count(-1);
    cout <<num<<endl;
    for (size_t i=0; i<num; i++)
    {
        cout << findmmap->first << ": "<< findmmap->second << endl;
        findmmap++;
    }


    // 删除元素

    mapinttoString1.erase(-1);
    mmapinttoString1.erase(100);
    DisplayConstents(mapinttoString1);
    DisplayConstents(mmapinttoString1);
```

8. 栈

```
#include <stack>
    // 初始化
//pop
//push
//top
//empty
//size
    stack <int> stackInts;
    stack <double > stackDoubles;
    vector<int> vectorInts(10, 3);
    stackInts.push(5);
    stackInts.push(3);
    while (stackInts.size() !=0 ) {
        cout << stackInts.top() << endl;
        stackInts.pop();
    }
```

9. 队列

```
#include <queue>
//只允许末尾插入,头部删除

//push 队尾插入
//pop 队首删除
//front  返回队尾元素
//back 返回队首元素
//empty
//size
    queue<int> queueInts;
    queueInts.push(10);
    queueInts.push(3);
    queueInts.push(4);
    int queueFronts = queueInts.front();
    cout<<queueFronts<<endl;
    queueFronts = -1;
    cout<<queueFronts<<endl;
    while (queueInts.size() !=0 ) {
        cout << queueInts.front() << endl;
        queueInts.pop();
    }


// priority_queue 优先级队列
//pop  删除队首元素(最大的元素)
//push 在队列中插入一个元素
//top 返回队首元素(即最大的元素)
//empty 判断是否为空
//size 队列中元素的个数


    // 初始化
    priority_queue<int> queueInts;
//      priority_queue<int, deque<int>,greater <int>> queueInts;
//  使用greater来使得队列有小到大排序,队首存放最小,使用deque作为内部容器,默认情况下采用
vector,这里也可以指定vector
    queueInts.push(1);
    queueInts.push(100);
    queueInts.push(3);
    queueInts.push(4);
    queueInts.push(30);
    int queueFronts = queueInts.top();
    cout<<queueFronts<<endl;
    queueFronts = -1;

    cout<<queueFronts<<endl;
    while (queueInts.size() !=0 ) {
        cout << queueInts.top() << endl;
        queueInts.pop();
    }
```

###