# STL

## 基本概念

standard template library

为数据管理，数据操作而操作

函数模板一般都是对应算法的。

## 类模板

成为**容器**，包括**标准容器**和**容器适配器**，也叫**特殊容器**，还有一部分称为**函数对象**（function），也叫**仿函数**，以及还包括**内存分配器（allocator）**（默认使用new来分配内存）。

标准容器包括 **序列式容器**以及**关联式容器**。标准容器的内部类型叫**迭代器**（iterator）。

迭代器封装**指针**，支持*，->，以及++,==,!=。

在数组里面，指针是最原始的迭代器，零封装。

各个容器一般都提供一个迭代器提示终点。

### 标准容器

#### 标准容器的共性

构造函数：无参构造，拷贝构造，区间构造（标准容器中用迭代器表示位置，区间就是用两个迭代器表示的两个位置）。

迭代器相关：.begin(),.end(),.rbegin(),.rend() 正向迭代器：iterator 反向迭代器：reverse_iterator 只读迭代器：const_iterator const_reverse_iterator

插入：insert（pos,element),pos是个迭代器

删除：erase（pos） erase(pos_begin,pos_end)(区间删除，含头不含尾)

清除：clear()

大小：.size() .max_size() .empty()

交换：.swap(c2) swap(c1,c2)

运算符：=,>,<,>=,<=,==,!=

### 序列式容器

vector，deque(双端队列),list

#### 序列式容器的共性

构造函数 初始化默认零初始化

插入 .insert(pos,n,element) .insert(pos,pos_beg,pos_end)

赋值 assign(n,element) .assign(pos_beg,pos_end)

调整 .resize(n,element = 零初始化)

首尾 .front() .back() 返回的是引用，所以可以修改。

增删 .push_back(element) .pop_back()

#### 序列式容器的个性

**vector的个性：**

**vector的个性：**

- .capacity() **当前容量**
- .reserve(n) **约定容量**
- .operator(不检查越界) .at(i)（越界会抛出异常） **下标[]**

**deque的个性**

- .operator(不检查越界) .at(i)（越界会抛出异常） **下标[]**
- .push_front(element) .pop_front() **增删**

**list的个性**

- .push_front(element) .pop_front() .remove(element) //不支持下标[]**增删**
- .unique() **除重** 相邻的重复元素只保留一个
- .sort() **排序** 默认用小于符号比较，从小到大排序
- .reverse() **逆序** 颠倒链表中的元素顺序
- .splice(pos,list2) .splice(pos,list2,pos) .splice(pos,list2,pos_beg,pos_end) **转移**
- .merge(list2) **归并**

### 关联式容器(都是用二叉查找树来实现)

数据集--set(不允许重复) multiset(允许重复) map(关键字不允许重复) multimap（允许重复关键字）

**关联式容器的共性**

- 都是使用二叉树查找，都自动根据关键字排序，都是默认升序
- **查找** .find(key)返回一个迭代器指向找到的第一个元素，没找到的话就返回一个指向无效位置的迭代器，返回.end()
- **统计** .count(key)统计关键字等于k的元素的个数
- **删除** 删除关键字等于k的所有元素。
- **区间** .lower_bound(key)取得关键字为key的第一个元素的位置
  .upper_bound(key) 取得关键字为key的最后一个元素之后的位置
  .equal_range(key) 一次取得关键字为key的元素的区间，返回的是一个pair，用起来比较不方便。
- **插入** .insert(element)

## 函数模板

称为**通用算法**，一般需要的话从资料上查询就可以了，包括很多模板。它是以来与迭代器来实现的。

**区间**由起始位置以及超越终点的位置组成，半开半闭区间，含头不含尾。

---

## 实例代码

第一个是一个对于排序算法的使用

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

class Person{
    string name;
    int age;

    public:
        Person(const char* name,int age):name(name),age(age){}
        friend ostream& operator<<(ostream& o,const Person& p){
            return o<<p.name<<':'<<p.age;
        }
        friend bool operator<(const Person& a,const Person& b){
            return a.age<b.age;
        }
```

```
}

template<typename T>
void print(T b,T e)
{
    while(b != e)
        cout << *b++ <<' ';
}
int main()
{
    int a[6] = {8,1,6,3,2,5};
    double b[4] = {5.5,3.3,6.6,2.2};
    string c[5] = {"nice","to","see","you","all"};
    Person d[3] ={Person("wuwnexiao",18),Person("shenlin",20),Person("jianguo",33)};
    sort(a,a+6);sort(b,b+4);sort(c,c+5);sort(d,d+3);
    print(a,a+6);print(b,b+4);print(c,c+5);print(d,d+3);

}
```

第二个对容器共性的使用

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

int main()
{
    int a[5] = {33,11,55,22,44};
    vector<int> vi(a,a+5);

    cout << vi.size() << endl;
    cout << vi.begin() << endl;
    sort(vi.begin(),vi.end());
    vector<int> iterator b = vi.begin();
    while(b ! = vi.end())
    {
        cout << *b << ' ';
    }
    cout << endl;

}
```

第三个对序列式共性的使用

```
#include <iostream>
#include <deque>
using namespace std;
#include "print.h"
#include <string>

int main()
{
  deque<string> ds;
  //deque<vector<int> >;

  //ds.push_back("曾文武");
  ds.push_back("ss");
  ds.push_back("学校圈");
  ds.push_back("高上");

  print(ds.begin(),ds.end());

  ds.insert(++++ds.begin(),2,"芙蓉");

  print(ds.begin(),ds.end());

  string s[3] = {"事实上","迭代","对方是否"};

  ds.insert(----ds.end(),s,s+3);
```

```
  print(ds.begin(),ds.end());
  cout << "front" <<ds.front()<< ",back:"<<ds.back()<<endl;

  ds.pop_back();
  ds.pop_back();

  ds.resize(12,"过撒旦");

  print(ds.begin(),ds.end());

  ds.assign(5,"康森铃");
  print(ds.begin(),ds.end());
  ds.front() = "孙东东";
  ds.back() = "两证";
  print(ds.begin(),ds.end());
}
```

第四个 对vector个性的使用

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

#include <vector>
#include "print.h"
#include <exception>
#include <typeinfo>
void print(const vector<int>& v);
int main()
{
  vector<double> vd,vv;

  for(int i = 0;i<9;i++)
  {
    vd.push_back(i+0.1);
    cout << &*vd.begin()<< ":";
    cout << vd.size() << "/" << vd.capacity() << endl;
  }
  cout << "----------"<< endl;
  vv.reserve(9);
  for(int i = 0;i<9;i++)
  {
    vd.push_back(i+0.1);
    cout << vv.size() << "/" << vv.capacity() << endl;
  }

  vd[3] = 123.45;
  vv.at(5) = 67.8;

  for(int i = 0;i<vd.size();i++)
  {
    cout << vd[i]<< " ";
  }
  cout << endl;
  try{
  for(int i = 0;i<=vd.size();i++)
  {
    cout << vd.at(i)<< " ";
  }
  cout << endl;
  }
  catch(exception& e)
  {
    cout << "异常"<<e.what() << endl;
    cout << "类型"<<typeid(e).name()<<endl;
  }
  int m = 3,n=5;
  vector<vector<int> >vvi(m,vector<int>(n));
  //vector<vector<int> >ivv;
  vvi.resize(m+3);
```

```
    vvi[1].assign(9,1);
    vvi[5].assign(4,5);
    print(vvi);
  }

  void print(const vector<vector<int> >& v)
  {
    for(int i = 0;i<v.size();i++)
    {
      for(int j = 0;j<v[i].size();j++)
      {
        cout << v[i][j] <<" ";
      }
      cout << endl;
    }
  }
```

第五个 对deque的个性的使用

```
#include <iostream>
using namespace std;
#include <deque>
#include "print.h"

int main()
{
  deque<char> dc;
  dc.push_back(97);
  dc.push_back('c');
  dc.push_front('s');
  dc.push_front('d');
  dc.push_back('k');
  dc.push_front('$');
  print(dc.begin(),dc.end());
  dc[1] = 't';
  for(int i = 0;i<dc.size();i++)
  {
    cout << dc[i]<<" ";
  }
  cout << endl;

  dc.pop_back();
  dc.pop_front();
  print(dc.begin(),dc.end());
}
```

第六个 对list个性的使用

```
#include <iostream>
using namespace std;
#include <list>
#include  <print.h>
#include <cassert>
bool compare(int x,int y)
{
  x%=3;y%=3;
  return x<y;
}
int main()
{
  int a[10] = {3,8,8,8,5,5,1,8,8,6},b[6] = {9,3,5,2,7,6};
  list<int> li(a,a+10),lili;
  print(li.begin(),li.end());

  li.unique();
  print(li.begin(),li.end());

  li.sort();
  print(li.begin(),li.end());
```

```
  li.unique();
  print(li.begin(),li.end());

  li.reverse();
  print(li.begin(),li.end());

  li.splice(li.begin(),lili);

  print(li.begin(),li.end());

  assert(lili,empty());

  li.remove(5);
  print(li.begin(),li.end());

  li.sort();
  print(li.begin(),li.end());

  lili.push_back(5);
  lili.push_back(4);
  lili.push_back(7);

  print(lili.begin(),lili.end());
  li.merge(lili);
  print(li.begin(),li.end());

  lili.sort(greater<int>())
  lili.sort(compare());
  print(lili.begin(),lili.end());
}
```