STL(Standard Template Library)的中心思想是将算法和数据结构分离，彼此独立设计，最后再用iterator将他们结合在一起，获得最大的适配性。

C++ STL(标准模板库)是一套功能强大的 C++ 模板类，提供了通用的模板类和函数，这些模板类和函数可以实现多种流行和常用的算法和数据结构，如向量、链表、队列、栈。

C++ 标准模板库的核心包括以下三个组件：

组件	描述
容器(Containers)	容器是用来管理某一类对象的集合。C++ 提供了各种不同类型的容器，比如 deque、list、vector、map 等。
算法(Algorithms)	算法作用于容器。它们提供了执行各种操作的方式，包括对容器内容执行初始化、排序、搜索和转换等操作。
迭代器(iterators)	迭代器用于遍历对象集合的元素。这些集合可能是容器，也可能是容器的子集。

在C++标准中，STL被组织为下面的13个头文件：

- 容器：<vector>、<list>、<deque>、<set>、<map>、<stack>、<queue>
- 算法：<algorithm>、<numeric>、<functional>
- 迭代器：<iterator>、<memory>、<utility>

其中各个容器的主要作用如下：

- 向量(vector) 连续存储的元素 <vector>
- 列表(list) 由节点组成的双向链表，每个结点包含着一个元素 <list>
- 双队列(deque) 连续存储的指向不同元素的指针所组成的数组 <deque>
- 集合(set) 由节点组成的红黑树，每个节点都包含着一个元素，节点之间以某种作用于元素对的谓词排列，没有两个不同的元素能够拥有相同的次序 <set>
- 多重集合(multiset) 允许存在两个次序相等的元素的集合 <set>
- 栈(stack) 后进先出的值的排列 <stack>
- 队列(queue) 先进先出的执的排列 <queue>
- 优先队列(priority_queue) 元素的次序是由作用于所存储的值对上的某种谓词决定的的一种队列 <queue>
- 映射(map) 由{键，值}对组成的集合，以某种作用于键对上的谓词排列 <map>
- 多重映射(multimap) 允许键对有相等的次序的映射 <map>

一、容器(Containers)

string(字符串)

性质：一种存储字符的特殊容器，和vector相似。支持快速随机访问，能够在末尾快速插入和删除元素。

```
#include<string>

//1.定义
string str1, str2, str3;

//2.访问：下标和迭代器
str1[i];                  //通过下标访问

//3.常用函数
str3 = str1 + str2;                          //将str1和str2拼接，赋值给str3
if(str1 < str2)    {}                        //将str1和str2进行比较(字典序)
str1.size();                                     //返回str1内的字符个数，时间复杂度为O(1)
str1.insert(3, "hello");              //在str1[3]处插入"opq"
str1.insert(str1.begin + 3, str2.begin(), str2.end());    //在str1[3]处插入str2
str1.erase(str1.begin + 3);          //删除str1[3]
str1.erase(str1.begin + 3, str1.begin() + 5);      //删除str1[3], str1[4]
str1.clear();                                    //删除str1中的所有字符
str1.substr(pos, len);                       //返回从pos位置开始，长度为len的子串
string::npos; //作为find函数失配时的返回值。本身值为-1，但是是unsigned类型，使用可以认为是unsigned类型的最大值。
str1.find(str2);                               //当str2是str1的子串时，返回其在str中第一次出现的位置
str1.find(str2, pos);                        //从位置pos开始寻找
str1.replace(pos, len, str2);    //把str1的子串替换为str2
stoi(string str);                            //将string转化为int
```

vector(向量)

性质：变长数组，可随机访问

主要用途：

1. 做变长数组使用
2. 使用vector实现邻接表

```
#include<vector>

//1.定义
vector<int> vec(10);    //定义了10个整型元素的向量
vector<int> v[10];        //定义一个包含10个vector<int>的数组

//2.访问：下标和迭代器
vec[i];                                  //通过下标访问

//3.常用函数
vec.push_back(x);          //在vec中尾部加入一个元素x，时间复杂度为O(1)
vec.pop_back();              //在vec中删除最后一个元素，时间复杂度为O(1)
vec.size();                          //返回vec中元素的个数，返回值为unsigned类型，时间复杂度为O(1)
vec.clear();                       //清空vec中所有元素，时间复杂度为O(N)
vec.insert(vec.begin() + 2, -1);    //将-1插入vec[2]的位置，时间复杂度为O(N)
vec.erase(vec.begin() + 3);             //删除元素vec[3]，时间复杂度为O(N)
vec.erase(vec.begin() + 1, vec.begin() + 4);    //删除v[1]～v[3]，不包括v[4]，时间复杂度为O(N)
```

vector主要用途：

queue(队列)

性质：先进先出

主要用途：

1. 广度优先搜素
2. 模拟队列类问题

注意：在使用front()和pop()前，必须使用empty()判空，否则可能会因为队空而出现错误。

```
#include<queue>

//1.定义
queue<int> q;


//2.访问
q.front();    //访问队首
q.back();        //访问队尾

//3.常用函数
q.push(x);    //入队，时间复杂度为O(1)
q.pop();    //出队，时间复杂度为O(1)
q.empty();//判空，时间复杂度为O(1)
q.size();    //返回队内元素个数，时间复杂度为O(1)
```

priority_queue(优先队列)

优先队列，底层用堆实现。在优先队列中，队首元素一定是当前队列中优先级最高的那一个。(元素为整数时默认数字越大优先级越高)

```
#include <queue>

//1.定义
priority_queue <int> q;

//2.访问
top();    //获得队首元素(即堆顶元素)

//3.主要函数
q.push(x);    //入队,时间复杂度为O(logN)
q.pop();    //出队，时间复杂度为O(logN)
q.empty(); //判空，时间复杂度为O(1)
q.size();     //返回元素个数，时间复杂度为O(1)
```

优先级设置

1. 基本数据类型的优先级设置

```
priority_queue<int, vector<int>, less<int> > q;    //数字大的优先级大
priority_queue<int, vector<int>, greater<int> > q;    //数字小的优先级大
```

1. 结构体的优先级设置

```
//默认情况下，水果价格高的优先级越高
struct fruit {
  string name;
  int price;
  friend bool operator < (fruit f1, fruit f2) {
    return f1.price < f2.price;
  }
}

//重载小于号之后，水果价格低的优先级越高
struct fruit {
  string name;
  int price;
  friend bool operator < (fruit f1, fruit f2) {
    return f1.price > f2.price;
  }
}
```

注意：重载大于号会编译错误，因为从数学上来讲只需要重载小于号。优先队列中的函数和sort中的cmp函数是相反的。

1. 在结构体外面修改优先级

```
struct cmp{
  bool operator () (fruit f1, fruit f2) {
    return f1.price > f2.price;
  }
}

//定义时要用以下方式
priority_queue<fruit, vector<fruit>, cmp> q;
```

1. 如果结构体内的数据较为庞大(比如出现了字符串或者数组)，建议使用引用来提高效率

```
friend bool operator < (const fruit f1, const fruit f2) {
  return f1.price > f2.price;
}

bool operator () (const fruit f1, const fruit f2) {
```

```
    return f1.price > f2.price;
  }
```

主要用途：

1. 解决一些贪心问题
2. 对Dijkstra算法进行优化(因为优先队列的本质是堆)

【注意】在使用top()和pop()前，必须使用empty()判空，否则可能会因为队空而出现错误。

map(映射)

关键字唯一，且一个关键字只能对应一个值。以关键词按从大到小排序。

map内部也是使用红黑树实现的。

它的特点是增加和删除节点对迭代器的影响很小，除了那个操作节点，对其他的节点都没有什么影响。对于迭代器来说，可以修改实值，而不能修改key。

```
#include<map>

//1.定义
map<int, string> mp;         //定义一个空的map
map<string, string> authors = {     //定义一个非空的map
  {"Joyce","James"},{"Austen","Jane"},{"Dickens","Charles"}
}

//2.访问
map['c'];     //通过下标访问
map<char, int>::iterator it = mp.begin();                          //通过迭代器访问
printf("%c %d\n", it -> first, it -> second);

//3.常用函数
mp.find();
mp.erase();
mp.size();
mp.clear();
m.insert({"Who","Ever"});
```

map的主要用途

1. 需要建立字符(或字符串)与整数之间映射的题目，使用map可以减少代码量。
2. 判断大整数或者其他类型数据是否存在的题目，可以把map当bool数组用
3. 字符串和字符串的映射也有可能会遇到

需要注意的是，map的键和值是唯一的，而如果一个键需要对应多个值，就只能用multimap。此外，C++11标准中还增加了unordered_map，以散列代替map内部的红黑树实现，使其可以只处理映射而不对key进行排序，速度要比map快的多。

set(集合)

在set中每个元素的值都唯一，而且系统能根据元素的值自动进行升序排序。

set容器实现了红黑树(Red-Black Tree)的平衡二叉检索树的数据结构。

set中数元素的值不能直接被改变。

unorder_set以散列代替set内部的红黑树，用于处理只去重不排序的需求，速度要比set快得多

```
#include<set>

//1.定义
set<string> s;         //定义一个空的set
set<string> s={"the","but","and","or","a"};         //定义一个非空的set

//2.set只能通过迭代器访问

//3.常用函数
s.insert(k);     //在s中添加关键词为k的元素，时间复杂度为O(logN)
set<int>::iterator it = s.find(k);         //返回set中对应值为k的迭代器，时间复杂度为O(logN)
s.erase(s.find(k))     //用迭代器删除关键词为k的元素
s.erase(k);                       //从s中删除关键词为k的元素，时间复杂度为O(logN)
s.erase(s.find(k), s.end());     //从s中删除从k到末尾的所有值，时间复杂度最大为O(N)
s.size();                          //返回set内的元素个数，时间复杂度为O(1)
s.clear();                       //删除set中的所有元素，时间复杂度为O(N)
```

set的主要用途：

1. 自动去重并升序排序
2. 需要注意的是，set中的元素是唯一的，如果需要处理不唯一的情况，则需要用multiset。此外，C++11标准中还增加了unordered_set，以散列代替set内部的红黑树实现，使其可以用来处理只去重但不排序的需求，速度要比map快的多。

pair(对组)

pair是一个很实用的“小玩意”，当想要将两个元素绑在一起作为一个合成元素、又不想要因此定义结构体的时候，使用pair可以很方便的作为一个替代品。应届生说，pair实际上可以看成是一个内部有两个元素的结构体，而且这两个元素的类型是可以指定的。

```
struct pair {
    typeName1 first;
  typeName2 second;
}
```

```
#include <utility>
//或者是#include<map>,因为map的内部实现中涉及pair，所以添加map头文件的时候会自动添加utility头文件。

//1.定义
pair<string, int> p;
pair<string, int> p("haha", 5);     //初始化

//2.临时创建一个pair
p = pair<string, int>("haha", 5);
p = make_pair("haha", 5);

//3.访问
cout << p.first;
cout << p.second;
```

比较操作数

两个pair类型数据可以直接用==、!=、<、<=、>、>=比较大小。比较规则是先以first的大小作为标准，只有当first相等时才去判别second的大小。

主要用途

1. 用来代替二元结构体及其构造函数，可以节省编码时间
2. 作为map的键值对来进行插入。

```
mp.insert(pair<string, int>("haha", 5));
mp.insert(make_pair("xixi", 10));
```

二、算法(algorithm)

```
#include <algorithm>

max(x, y);                  //返回两个数的最大值,参数只能是两个
max(x, max(y, z));//返回三个数的最大值
min(x, y);                  //返回两个数的最小值
abs(x);                       //返回x的绝对值
swap(x,y);                  //用来交换x和y的值
reverse(it, it2);     //将数组或者容器内的范围内的元素进行反转
next_permutation(a, a+3); //给出一个序列在全排列中的下一个序列
fill(a, a + 5, -1);      }//把数组或者容器中的某一段赋为某个相同的值。(可以是任意值)
sort(a, a + 5, cmp);     //按cmp顺序排序
lower_bound(first, last, val);     //寻找数组或容器范围内第一个大于等于val的元素的位置，如果是数组，返回改位置的指针，如果是容器，则返回该位置的迭代器
upper_bound(first, last, val);     //寻找数组或容器范围内第一个大于val的元素的位置
```

三、迭代器(iterations)

提供一种方法，使之能依次访问容器内的各个元素，而又不暴露该聚合物内部的表达方式。

迭代器通常是左闭右开。

vec.end()指向vec的最后一个元素的后一位，vec[i]和*(vi.begin()+i)等价。

除了vector和string以外的STL容器都不支持*(it + i)的访问方式。

```
vector<int> vec;
for (vector<int>::iterator it = vec.begin(); it != vec.end(); it++){
  printf("%d", *it);
}
```