常用 STL 总结（长网文）

一、什么是 STL？

STL（Standard Template Library），即标准模板库，是一个具有工业强度的，高效的 C++程序库。它被容纳于 C++标准程序库（C++ Standard Library）中，是 ANSI/ISO C++标准中最新的也是极具革命性的一部分。该库包含了诸多在计算机科学领域里所常用的基本数据结构和基本算法。为广大 C++程序员们提供了一个可扩展的应用框架，高度体现了软件的可复用性。

二、为什么 STL 重要？

1、原因

C 语言是一门很有用的语言，但在算法竞赛中却不流行，原因在于它太底层，缺少一 些“实用的东西”。例如，在 2013 年 ACM/ICPC 世界总决赛中，有 1347 份用 C＋＋提交，323 份 用 Java 提交，但一份用 C 提交的都没有。虽然 C 语言的主要内容，已经足以应付许多算法竞赛的题目了。然而，“能 写”并不代表“好写”，有些题目虽然可以用 C 语言写出来，但是用 C＋＋写起来往往会更快， 而且更不容易出错。例如 cin 和 cout，但是运行时间却不如 C 语言，using namespace std 语句，万能头文件，数组大小可以使用 const 声明的常数,以及黑科技 STL...

2、STL 的作用

STL 的作用:加快书写速度，例如 sort 使用 unique 函数 这些可即以简化书写，而且运行速度和二分这些算法运行速度差不多。你可以用它来操作几乎任何数据集合，包括链表，容器和数组.vector 容器简直就是数组加强版，它的作用非常强大

一定要学，听我的，比如你要查一个数组大小，你该不会还要 count 一下，然后再 for 遍历一下？？ string，deque,太多了，不多赘述请往下看 …

三、STL 知识点总结

0.使用说明书

首先先收藏这篇文章，STL 确实有点多，第一次可以看代码自己敲一遍，然后今后用到忘记了查就行，主要还是要多用，用多了自然就会了，STL 中六大组件：容器、迭代器、算法、仿函数、迭代适配器、空间配制器。本文章主要涉及前三个，另外会有一些使用小技巧，和实战习题。

1、vector 【不定长数组】

你说它是数组吧，是，但又不完全是，还比数组好用

①头文件：

```
#include<vector>
```

②初始化：

这个初始化比较详细，后面一些容器用法类似

```
#include<iostream>
#include<vector>
using namespace std;
int main () {
    //几种初始化的方法
    vector<int> a;//定义一个 vector 未初始化 输出》 0
    vector<int> a(3);//定义一个长度为 3 的 vector 未初始化 输出》0
0 0
    vector<int> a(10, 3); //定义一个长度为 10，且每个数赋值为 3

    //将向量 b 中从下标 0 1 2（共三个）的元素赋值给 a，a 的类型为 int
型
    vector<int>a(b.begin(),b.begin+3);
```

```
    //从数组中获得初值
    int b[7]={1,2,3,4,5,6,7};
    vector<int> a(b,b+7）;

    for(auto x : a) {//遍历输出
        cout << x << " ";
    }
    return 0;
}
```

③size()

```
a.size()//返回元素个数
```

④a.resize

```
a.resize( )//改变大小
```

⑤empty()

```
a.empty();
//判断 a 是否为空，空则返回 true，非空则返回 false
```

⑥front()和 back()

```
a.front();  //返回 a 的第 1 个元素,当且仅当 a 存在
a.back(); //返回 vector 的最后一个数
```

⑦倍增的思想

[C++]系统为某一程序分配空间的所需时间，与空间大小无关，与申请次数有关如

申请一个空间为 1000 和 空间为 1 申请 1000 次的所需时间差别是很大的，申请次数越多，越耗时间

⑧clear()

```
a.clear(); //清空 a 中的元素
```

⑨支持比较运算

比较操作==，！=，<，<，<=，>，>=

```
int main () {
    //支持比较运算
    vector<int> a(4, 3), b(3, 4);
    //a: 3 3 3 3 b:4 4 4

    //比较原理字典序 (根据最前面那个判断，如果一样就往后比较)
    if (a < b) {
        puts("a < b");
    }
    return 0;
}
```

⑩push_back()和 pop_back();

```
a.pop_back(); //删除 a 向量的最后一个元素
a.push_back(5); //在 a 的最后一个向量后插入一个元素，其值为 5
```

⑪begin()和 end()

```
a.begin();// vector 的第 0 个数 a.end();// vector 的最后一个的数的后面一个数
//通常与 for 循环结合使用
```

⑫遍历 vector 的三种方法

```
int main () {
    vector<int> a;
    for (int i = 0; i < 10; i ++) {
        a.push_back(i);
    }

    //三种遍历 vector 的方法
    for (int i = 0; i < a.size(); i ++) {
        cout << a[i] << ' ';
    }
    cout << endl;

    for (auto i = a.begin(); i != a.end(); i ++) {
        cout << *i << ' ';
    }
    cout << endl;

    //C++11 的新语法
    for (auto x : a) {
        cout << x << ' ';
    }
    cout << endl;
```

```
    return 0;
}
```

⑬结合算法 erase() reverse()

```
#include<algorithm>
```

a.erase(p)//从 a 中删除迭代器 p 指定的元素，p 必须指向 c 中的一个真实元素，不能是最后一个元素 end()

a.erase(b,e)//从 a 中删除迭代器对 b 和 e 所表示的范围中的元素，返回 e

```
vector<int> a={1,2,3,4,5};
```

reverse(a.begin(),a.end());//a 的值为 5，4，3，2，1 倒置

2、pair 【套娃模拟器】

可以理解为(x,y)数学中的坐标表示

小技巧：使用 typedef 定义 typedef pair<int,int>PII

①头文件

```
 #include<utility>
```

②初始化

使用：pair<first 数据类型,second 的数据类型>元素名；

pair 中只有两个元素，first 和 second。

```
//俩种方法初始化
pair<string,int>p("hello",1);
```

```
p =make_pair("hello",1);
```

③first() 和 second()

```
p("hello",1);
```

```
p.first;//第一个元素 =hello
p.second;//第二个元素 = 1
```

④嵌套（套娃）

```
vector< vector<pair<int,int>>>//与 vector 结合【再写个 vector 结合即可】

//套娃操作 用 pair 存储 3 个数据
pair<int, pair<int,int>>p(1,{2,3});
```

⑤实战题

可以做下这道题离散化 AcWing 802

3、string

支持比较操作符 >,>=,<,<=,==,!=

①头文件

```
#include<string>
```

②初始化

```
string a ="ac";
```

③ substr()

```
#include<iostream>
#include<string>

using namespace std;

int main () {
    string a = "ac";

    a += "w";//支持比较操作符>,>=,<,<=,==,!=
    cout << a << endl; //输出子串 a :acw

    a += "ing";
```

```
    cout << a << endl;

    //以字符串数组理解
    cout << a.substr(0, 3) << endl; //当第一个数是 0 则后一位数:输出从头开始的长度为 3 的子串
    cout << a.substr(0, 3) << endl; //当第一个数是 1 则输出下标为 1 到下标为 3 的子串
    cout << a.substr(0, 9) << endl;//如果超出长度范围 则输出原子串
    cout << a.substr(1) << endl; //从下标为 1 开始输出
    cout << a.substr(0) << endl; //原子串

    printf("%s\n", a.c_str());//如果用 printf 输出

    return 0;
}
```

④ c_str()

```
// 返回这个 string 对应的字符数组的头指针
string s ="Hello World!";

printf("%s", s.c_str());//输出 "Hello World!"
```

⑤push_back() 和 insert()

```
// 尾插一个字符
a.push_back('a');

// insert(pos,char):在制定的位置 pos 前插入字符 char

a.insert(a.begin(),'1');
```

⑥empty()

判断 a 是否为空，空则返回 true，非空则返回 false

⑦size() length()

都是返回字母个数

```
string s ="cpt";
cout << a.size()<< endl;//输出 3
```

```
printf("%s", a.length());//输出 3
```

⑧clear()

把字符串清空

可以发现 string 和 vector 这些还是有很多共同的函数的

4、queue【队列】 和 priority_queue 【优先队列、堆】

队列是一种数据结构原理：先进先出，元素从一端入队，从另一端出队，就像是排队。

优先队列和队列特性不同：按优先级排序 和 获取

①头文件

```
#include< queue >//都在这个头文件
```

②初始化

```
//queue <类型> 变量名
//priority_queue <类型> 变量名;

queue <int> q; //定义一个名为 q 队列

priority_queue <int> q; //默认是大根堆

//定义小根堆
```

小根堆：`priority_queue <类型,vecotr <类型>,greater <类型>>` 变量名

```
q.size();// 这个队列的长度
q.empty();//用于判断这个队列是否为空，空则返回 true，非空则返回 false

q.push();//往队尾插入一个元素

q.pop();//队列：把队头弹出 优先队列 ：弹出堆顶元素
```

④区别

队列：

```
q.front();// 返回队头元素
q.back(); //返回队尾元素
```

优先队列：

```
q.top();// 返回堆顶元素
```

⑤清空

注意：队列和堆没有 clear 函数

所以清空的方法就是重新初始化

```
q = queue <int>();
```

5、stack 【栈】

①头文件

```
#include<stack>
```

②初始化

```
//stack<类型> 名字;
stack<int> s;
```

③size()

返回这个栈的长度

④push()

向栈顶插入一个元素

⑤top()

返回栈顶元素

⑥pop()

弹出栈顶元素

6、deque【双向队列】

好用，几乎其他容器的都有，就是慢一点

①头文件

```
#include<deque>
```

②初始化

```
deque<int> dq;//定义一个 int 类型的双向队列
```

③常用函数

```
dq.size(); //返回这个双端队列的长度

dq.empty(); //返回这个队列是否为空，空则返回 true，非空则返回 false

dq.clear(); //清空这个双端队列

dq.front(); //返回第一个元素

dq.back(); //返回最后一个元素

dq.push_back(); //向最后插入一个元素

dq.pop_back(); //弹出最后一个元素

dq.push_front(); //向队首插入一个元素

dq.pop_front();//弹出第一个元素

dq.begin(); //双端队列的第 0 个数

dq.end(); //双端队列的最后一个的数的后面一个数
```

7、set 【集合】和 multiset

集合与映射也是两个常用的容器,set 类似于数学上的集合

①头文件

```
#include<set>
```

②初始化

```
set<string> s;//string 集合
```

③区别

set 不允许元素重复，如果有重复就会被忽略，但 multiset 允许.

④常用函数

```
size();// 返回元素个数
empty(); //返回 set 是否是空的

clear(); //清空

begin(); //第 0 个数，支持++或--，返回前驱和后继

end(); //最后一个的数的后面一个数，支持++或--，返回前驱和后继

insert(); //插入一个数

find(); //查找一个数

count(); //返回某一个数的个数

erase(x); //删除所以 x 时间复杂度 O(k + logn)

erase(s.begin(),s.end());//删除一个迭代器
```

⑤核心函数

```
lower_bound(x); //返回大于等于 x 的最小的数的迭代器 核心操作
upper_bound(x);//返回大于 x 的最小的数的迭代器 不存在返回 end()
```

8、map 【映射】 /multimap

map 就是从键（key）到值（value）的映射。因为重载了[ ]运算符，map 像是数组的“高 级版”。例如可以用一个 `map<string,int>month_name` 来表示“月份名字到月份编号”的映射， 然后用 `month_name["July"]=7` 这样的方式来赋值

①头文件

```
#include <map>
```

②初始化

这个初始化有点不同 还是小技巧搭配 typedef 简化

```
map<string,int> m ={"A",10};
```

③常用函数

```
insert(); //插入一个数，插入的数是一个 pair
erase();

//（1）输入是 pair

//（2）输入一个迭代器，删除这个迭代器

find(); //查找一个数

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
```

④ 映射[]

时间复杂度 O(logn)

```
#include <iostream>
#include <string>
#include<map>

using namespace std;
```

```
int main(){
    map<string,int>a;
    a["abc"] = 1;//把字符串"abc" 映射为 1
    cout << a["abc"] << endl; //查找 abc 程序输出 1

    return 0;
}
```

⑤应用

```
#include <iostream>
#include <string>
#include<map>

using namespace std;

typedef pair<string,int> PSI;

int main()
{
    map<string,int> mp;
    mp.insert(make_pair("heihei",5));
    mp.insert(PSI("haha",10));//简化

    map<string,int>::iterator it=mp.begin();//迭代器：map<int,
char>::iterator it
    for(;it!=mp.end();it++)
    cout<<it->first<<" "<<it->second<<endl;

    return 0;
}
```

9、哈希表

①头文件

```
unordered_set，unordered_map，
unordered_muliset,unordered_multimap
//头文件就是加上对应名称
```

②优势

和上面 map 和 set 类似，增删改查的时间复杂度是 O(1)

③缺点

不支持 lower_bound()和 upper_bound()

10、bitset 【压位】

它是一种类似数组的结构，它的每一个元素只能是 0 或 1，每个元素仅用 1bit 空间,用于节省空间，

并且可以直接用 01 串赋值，可以理解为一个二进制的数组

①头文件

```
#include<bitset>

bitset<4> bs;//无参构造，长度为 4 ，默认每一位为 0
bitset<8>b(12);//长度为 8 ，二进制保存，前面用 0 补充
string s ="100101";//01 串赋值
bitset<10>bs(s);//长度为 10，前面用 0 补充
```

③支持操作

- ~取反，&与，|与或，^异或
- >>，<< 移动
- ==，!=
- [ ] 取 0/1

④常用函数

```
count(); //返回 1 的个数
any(); //判断是否至少有一个 1

none(); //判断是否全为 0
```

```
set(); //把所有位置赋值为 1

set(k,v); //将第 k 位变成 v

reset(); //把所有位变成 0

flip(); //把所有位取反，等价于~

flip(k); //把第 k 位取反
```