STL

任轩笛

绍兴市第一中学

2016年1月2日

Contents

1. 总体介绍
   - STL简介
   - 左闭右开
   - 迭代器
   - <号的定义
2. 常见的STL
   - algorithm
   - pair
   - string
   - vector
   - queue与priority_queue
   - set与multiset
   - map
3. 奇怪的错误
   - == or not?

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

什么是STL

什么是STL

Standard Template Library,标准模板库。

什么是STL

Standard Template Library,标准模板库。

即C++对于一些数据结构有专门的库，可以极大地精简代码，减少出错率。这是我看来C++优于PASCAL最大之处。

STL不是万能的!

STL不是万能的!

常数比较大，一般常数在纯手写相同数据结构的2倍以上，有时甚至会被卡到10多倍。

STL不是万能的!

常数比较大，一般常数在纯手写相同数据结构的2倍以上，有时甚至会被卡到10多倍。 在不开-O2的情况下慎用！尤其是不要过分依赖STL。

STL不是万能的!

常数比较大，一般常数在纯手写相同数据结构的2倍以上，有时甚至会被卡到10多倍。 在不开-O2的情况下慎用！尤其是不要过分依赖STL。

能开数组的地方最好不要用vector,能sort的地方最好不要用set,能Hash的地方最好不要用map。

STL不是万能的!

常数比较大，一般常数在纯手写相同数据结构的2倍以上，有时甚至会被卡到10多倍。 在不开-O2的情况下慎用！尤其是不要过分依赖STL。

能开数组的地方最好不要用vector,能sort的地方最好不要用set,能Hash的地方最好不要用map。

另外，不要以为有了STL就可以不学习堆、快排之类的东西了，这些算法的思想精髓仍然是非常重要的!

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

左闭右开

左闭右开

STL中的东西基本上都是左闭右开的，而且区间用地址来表示。

左闭右开

STL中的东西基本上都是左闭右开的，而且区间用地址来表示。举个简单的例子，对A[1..n]排序是sort(A+1,A+n+1)。

左闭右开

STL中的东西基本上都是左闭右开的，而且区间用地址来表示。举个简单的例子，对A[1..n]排序是sort(A+1,A+n+1)。

如果有返回值的话一般返回的也是地址。

左闭右开

STL中的东西基本上都是左闭右开的，而且区间用地址来表示。举个简单的例子，对A[1..n]排序是sort(A+1,A+n+1)。

如果有返回值的话一般返回的也是地址。

比如lower_bound(A+1,A+n+1,x)-A表示在1..n中找第一个 $\geq x$ 的值的下标。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

iterator

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。 但是迭代器不支持加减操作，只能++或者--。

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。但是迭代器不支持加减操作，只能++或者--。迭代器也不支持<和>比较，只有==和!=。

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。但是迭代器不支持加减操作，只能++或者--。迭代器也不支持<和>比较，只有==和!=。

通过迭代器可以遍历STL容器。

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。但是迭代器不支持加减操作，只能++或者--。迭代器也不支持<和>比较，只有==和!=。

通过迭代器可以遍历STL容器。

某种STL容器的begin(),end()都是该容器类型的迭代器，左闭右开。

iterator

某种STL容器的iterator可以理解为一种变量类型，它和指针的功能差不多。但是迭代器不支持加减操作，只能++或者--。迭代器也不支持<和>比较，只有==和!=。

通过迭代器可以遍历STL容器。

某种STL容器的begin(),end()都是该容器类型的迭代器，左闭右开。

```
1  for (set<int>::iterator i=S.begin();i!=S.end();i++) printf("%d\n",*i);
```

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

<号的定义

<号的定义

像sort,set,map这种需要依赖有序性的东西，使用时要定义<号。它所有的比较运算都是通过<来比较的。

<号的定义

像sort,set,map这种需要依赖有序性的东西，使用时要定义<号。它所有的比较运算都是通过<来比较的。

C++中已有的变量类型都是自带<号的。基本上都是按数值大小来比较的。string是按字典序比较的。

<号的定义

像sort,set,map这种需要依赖有序性的东西，使用时要定义<号。它所有的比较运算都是通过<来比较的。

C++中已有的变量类型都是自带<号的。基本上都是按数值大小来比较的。string是按字典序比较的。

怎么给结构体定义<呢?

<号的定义

像sort,set,map这种需要依赖有序性的东西，使用时要定义<号。它所有的比较运算都是通过<来比较的。

C++中已有的变量类型都是自带<号的。基本上都是按数值大小来比较的。string是按字典序比较的。

怎么给结构体定义<呢?

```
1  struct hahaha
2  {
3      int x;
4      bool operator < (const hahaha& B) const {return x<B.x;}
5  };
```

<号的定义

像sort,set,map这种需要依赖有序性的东西，使用时要定义<号。它所有的比较运算都是通过<来比较的。

C++中已有的变量类型都是自带<号的。基本上都是按数值大小来比较的。string是按字典序比较的。

怎么给结构体定义<呢?

```
1  struct hahaha
2  {
3      int x;
4      bool operator < (const hahaha& B) const {return x<B.x;}
5  };
```

或者还有一些别的方法。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

algorithm

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

__gcd(a,b):返回a和b的最大公约数。它们类型必须相同。(听说NOIp用下划线开头的函数会爆蛋……不推荐使用)

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

__gcd(a,b):返回a和b的最大公约数。它们类型必须相同。(听说NOIp用下划线开头的函数会爆蛋……不推荐使用)

swap(a,b):交换a和b，类型必须相同。

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

__gcd(a,b):返回a和b的最大公约数。它们类型必须相同。(听说NOIp用下划线开头的函数会爆蛋……不推荐使用)

swap(a,b):交换a和b，类型必须相同。

min(a,b)及max(a,b):返回较小值/较大值，都需要定义<号而不是≤或>号。它们类型必须相同。

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

__gcd(a,b):返回a和b的最大公约数。它们类型必须相同。(听说NOIp用下划线开头的函数会爆蛋……不推荐使用)

swap(a,b):交换a和b，类型必须相同。

min(a,b)及max(a,b):返回较小值/较大值，都需要定义<号而不是≤或>号。它们类型必须相同。

sort(A,A+n):对相应区间排序。待排序的类型必须定义了<号。

algorithm

这个词是算法的意思。这个库提供了一些简单的函数。

abs(x):返回x的绝对值。

__gcd(a,b):返回a和b的最大公约数。它们类型必须相同。(听说NOIp用下划线开头的函数会爆蛋……不推荐使用)

swap(a,b):交换a和b，类型必须相同。

min(a,b)及max(a,b):返回较小值/较大值，都需要定义<号而不是≤或>号。它们类型必须相同。

sort(A,A+n):对相应区间排序。待排序的类型必须定义了<号。或者用sort(A,A+n,cmp)，它将根据cmp(x,y)函数的布尔返回值来比较。返回1表示x < y。

unique(A,A+n):将对应区间去重，必须定义==号，或者用cmp来返回==关系。这个函数返回去完重后末元素+1的地址。

unique(A,A+n):将对应区间去重，必须定义==号，或者用cmp来返回==关系。这个函数返回去完重后末元素+1的地址。 注意它实际上是检查所有相邻的元素是否相同，所以用它离散的话要先排序。

unique(A,A+n):将对应区间去重，必须定义==号，或者用cmp来返回==关系。这个函数返回去完重后末元素+1的地址。 注意它实际上是检查所有相邻的元素是否相同，所以用它离散的话要先排序。

lower_bound(A,A+n,x):返回区间内第一个$\geq x$的值的地址。必须定义<号。序列必须有序。

unique(A,A+n):将对应区间去重，必须定义==号，或者用cmp来返回==关系。这个函数返回去完重后末元素+1的地址。 注意它实际上是检查所有相邻的元素是否相同，所以用它离散的话要先排序。

lower_bound(A,A+n,x):返回区间内第一个$\geq x$的值的地址。必须定义<号。序列必须有序。

upper_bound(A,A+n,x):返回区间内第一个$> x$的值的地址。必须定义<号。序列必须有序。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

pair

pair

有序对。用来把几个变量组合起来。能实现struct的部分功能。

pair

有序对。用来把几个变量组合起来。能实现struct的部分功能。

比如$pair < int, char >$。比较<的方式是先比较第一个，如果相同再比较第二个。

pair

有序对。用来把几个变量组合起来。能实现struct的部分功能。

比如$pair < int, char >$。比较<的方式是先比较第一个，如果相同再比较第二个。

只有两种元素的<都定义了，这个pair的<才算定义了。

pair

有序对。用来把几个变量组合起来。能实现struct的部分功能。

比如$pair < int, char >$。比较<的方式是先比较第一个，如果相同再比较第二个。

只有两种元素的<都定义了，这个pair的<才算定义了。

用p.first和p.second访问两个元素。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

string

string

C++很难处理字符串模拟题?

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持：

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持:

s[x]:返回该字符串的第x个字符的引用。

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持:

s[x]:返回该字符串的第x个字符的引用。

s.empty():返回是否为空。

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持:

s[x]:返回该字符串的第x个字符的引用。

s.empty():返回是否为空。

s.length()或s.size():返回该字符串的长度。复杂度O(1)。

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持:

s[x]:返回该字符串的第x个字符的引用。

s.empty():返回是否为空。

s.length()或s.size():返回该字符串的长度。复杂度O(1)。

s.substr(pos,len):返回从s的第pos位开始，长len位的一个子串。长度溢出没有关系。

string

C++很难处理字符串模拟题?

不不不，C++ string比PASCAL的string好用到不知道哪里去了。

string是一种变量类型。相当于一个结构体。以下全都是左闭右开的。用双引号表示常量string。string支持:

s[x]:返回该字符串的第x个字符的引用。

s.empty():返回是否为空。

s.length()或s.size():返回该字符串的长度。复杂度O(1)。

s.substr(pos,len):返回从s的第pos位开始，长len位的一个子串。长度溢出没有关系。

s.erase(pos,len):删除从pos开始长为len的串，长度溢出也没有关系。

string

string

s.insert(pos,...):在pos位置之前插入一个字符串或者一个字符。

string

s.insert(pos,...):在pos位置之前插入一个字符串或者一个字符。

s.find(...):参数是字符串或者一个字符，返回第一个匹配到的位置。匹配不到返回-1。

string

s.insert(pos,...):在pos位置之前插入一个字符串或者一个字符。

s.find(...):参数是字符串或者一个字符，返回第一个匹配到的位置。匹配不到返回-1。 还有从后往前查找、查找不在s中的字符、从后往前找不在s中的字符等一系列命令，有兴趣同学自行百度。

string

s.insert(pos,...):在pos位置之前插入一个字符串或者一个字符。
s.find(...):参数是字符串或者一个字符，返回第一个匹配到的位置。匹配不到返回-1。 还有从后往前查找、查找不在s中的字符、从后往前找不在s中的字符等一系列命令，有兴趣同学自行百度。
s+t:返回t连接到s后面形成的一个string。两者都可以是string或char。

读入与输出

读入与输出

string是不支持scanf读入、printf输出的。

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

1.使用cin。cin可以直接读取string。读到空格或回车为止。请注意不关同步的cin、cout效率。

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

1.使用cin。cin可以直接读取string。读到空格或回车为止。请注意不关同步的cin、cout效率。

2.搞一个char t[...];scanf读入这个t，然后s=string(t)。

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

1.使用cin。cin可以直接读取string。读到空格或回车为止。请注意不关同步的cin、cout效率。

2.搞一个char t[...];scanf读入这个t，然后s=string(t)。

常见的输出方法有：

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

1.使用cin。cin可以直接读取string。读到空格或回车为止。请注意不关同步的cin、cout效率。

2.搞一个char t[...];scanf读入这个t，然后s=string(t)。

常见的输出方法有：

1.使用cout。

读入与输出

string是不支持scanf读入、printf输出的。

常见的读入方法有：

1.使用cin。cin可以直接读取string。读到空格或回车为止。请注意不关同步的cin、cout效率。

2.搞一个char t[...];scanf读入这个t，然后s=string(t)。

常见的输出方法有：

1.使用cout。

2.s.c_str()是一个只读的字符数组，可以printf这个东西。

一些(伪)黑科技?

一些(伪)黑科技?

怎么把数字转成字符串或者把字符串转成数字?

一些(伪)黑科技?

怎么把数字转成字符串或者把字符串转成数字?

实际上string，甚至是char数组都可以轻松完成这些功能。

一些(伪)黑科技?

怎么把数字转成字符串或者把字符串转成数字?

实际上string，甚至是char数组都可以轻松完成这些功能。

```cpp
#include<cstdio>
#include<string>
#include<sstream>
#include<cstring>
#include<iostream>
#include<algorithm>

#define For(i,x,y) for (int i=x;i<y;i++)
using namespace std;

typedef long long LL;
typedef double Db;

stringstream ss;
string s,t;

int main()
{
    ss<<5;ss>>s;
    cout<<s<<endl;
    ss.clear();
    ss<<"orzJPY"<<333;
    ss>>s;
    cout<<s<<t<<endl;
}
```

一些(伪)黑科技?

怎么把数字转成字符串或者把字符串转成数字?

实际上string，甚至是char数组都可以轻松完成这些功能。

```cpp
#include<cstdio>
#include<string>
#include<sstream>
#include<cstring>
#include<iostream>
#include<algorithm>

#define For(i,x,y) for (int i=x;i<y;i++)
using namespace std;

typedef long long LL;
typedef double Db;

stringstream ss;
string s,t;

int main()
{
    ss<<5;ss>>s;
    cout<<s<<endl;
    ss.clear();
    ss<<"orzJPY"<<333;
    ss>>s;
    cout<<s<<t<<endl;
}
```

```cpp
#include<cstdio>
#include<string>
#include<sstream>
#include<cstring>
#include<iostream>
#include<algorithm>

#define For(i,x,y) for (int i=x;i<y;i++)
using namespace std;

typedef long long LL;
typedef double Db;

char ss[200];
char s[20];

int main()
{
    sprintf(ss,"%d",5);
    sscanf(ss,"%s",s);
    printf("%s\n",s);
    sprintf(ss,"orzJPY333");
    sscanf(ss,"%s",s);
    printf("%s\n",s);
}
```

一些(伪)黑科技?

怎么把数字转成字符串或者把字符串转成数字?

实际上string，甚至是char数组都可以轻松完成这些功能。

```cpp
#include<cstdio>
#include<string>
#include<sstream>
#include<cstring>
#include<iostream>
#include<algorithm>

#define For(i,x,y) for (int i=x;i<y;i++)
using namespace std;

typedef long long LL;
typedef double Db;

stringstream ss;
string s,t;

int main()
{
    ss<<5;ss>>s;
    cout<<s<<endl;
    ss.clear();
    ss<<"orzJPY"<<333;
    ss>>s;
    cout<<s<<t<<endl;
}
```

```cpp
#include<cstdio>
#include<string>
#include<sstream>
#include<cstring>
#include<iostream>
#include<algorithm>

#define For(i,x,y) for (int i=x;i<y;i++)
using namespace std;

typedef long long LL;
typedef double Db;

char ss[200];
char s[20];

int main()
{
    sprintf(ss,"%d",5);
    sscanf(ss,"%s",s);
    printf("%s\n",s);
    sprintf(ss,"orzJPY333");
    sscanf(ss,"%s",s);
    printf("%s\n",s);
}
```

```
C:\Users\RXDoi\Des
5
orzJPY333

--------------
Process exited
Press any key t
```

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

vector

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

v.push_back(...):往后面塞一个元素。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

v.push_back(...):往后面塞一个元素。

v.pop_back(...):弹出最后那个元素。还有push_front(),pop_front()等,但是是O(n)的。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

v.push_back(...):往后面塞一个元素。

v.pop_back(...):弹出最后那个元素。还有push_front(),pop_front()等,但是是O(n)的。

v.resize(x):把这个vector的大小重新定为x。注意左闭右开。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

v.push_back(...):往后面塞一个元素。

v.pop_back(...):弹出最后那个元素。还有push_front(),pop_front()等,但是是O(n)的。

v.resize(x):把这个vector的大小重新定为x。注意左闭右开。

由于内存是连续的，所以还支持sort、unique、lower_bound、upper_bound等操作。

vector

相当于一个可变长数组。以$vector < int >$为例。同样都是左闭右开的。内存是连续的!

v.size():返回长度(元素个数)。 v.empty():返回是否为空。

v.clear():清空所有元素。

v[x]:返回下标为x的元素的引用。

v.push_back(...):往后面塞一个元素。

v.pop_back(...):弹出最后那个元素。还有push_front(),pop_front()等,但是是O(n)的。

v.resize(x):把这个vector的大小重新定为x。注意左闭右开。

由于内存是连续的,所以还支持sort、unique、lower_bound、upper_bound等操作。 遍历vector可以直接当成数组遍历,也可以用iterator遍历。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

queue

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。

为什么说近似呢？因为它的内存不是随时释放的。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。 由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。 由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。 下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):在队首加入一个元素。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):在队首加入一个元素。

q.pop():弹出队尾元素。

queue

队列。占用内存近似是O(元素个数)的，因此可以代替循环队列。为什么说近似呢？因为它的内存不是随时释放的。由于队列这种东西实在是太好写了，没必要无谓加大常数。不推荐使用。下面也以$queue < int >$为例。

q.size():返回长度(元素个数)。q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):在队首加入一个元素。

q.pop():弹出队尾元素。

q.front():访问队尾元素。

priority_queue

priority_queue

优先队列。有时可以替代堆。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):把一个元素加入优先队列。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):把一个元素加入优先队列。

q.pop():弹出优先级最大的元素。

priority_queue

优先队列。有时可以替代堆。 当时限紧时慎用!

元素类型需要定义<号，比较大的会先出队。。

比如$priority_queue < int >$或

者$priority_queue < int, vector < int >, greater < int >>$。

q.size():返回长度(元素个数)。 q.empty():返回是否为空。

q.clear():清空所有元素。

q.push(...):把一个元素加入优先队列。

q.pop():弹出优先级最大的元素。

q.front():访问优先级最大的元素。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

set

set

集合。元素类型需定义<号。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

s.find(...):返回查找的东西的迭代器。如果找不到返回end()。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

s.find(...):返回查找的东西的迭代器。如果找不到返回end()。

s.insert(...):插入一个元素

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

s.find(...):返回查找的东西的迭代器。如果找不到返回end()。

s.insert(...):插入一个元素

s.erase(...):删除一个元素，注意不管是set还是multiset都是把相同的东西全部删除!

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

s.find(...):返回查找的东西的迭代器。如果找不到返回end()。

s.insert(...):插入一个元素

s.erase(...):删除一个元素，注意不管是set还是multiset都是把相同的东西全部删除！ 如果想让multiset只删除其中一个，

要s.erase(s.find(...))。

set

集合。元素类型需定义<号。

set会把两个插入的相同的值当成一个，multiset会把它们区别开。

其内部是有序的。

s.size():返回元素个数。 s.empty():返回是否为空。 s.clear():清空所有元素。

s.find(...):返回查找的东西的迭代器。如果找不到返回end()。

s.insert(...):插入一个元素

s.erase(...):删除一个元素，注意不管是set还是multiset都是把相同的东西全部删除！ 如果想让multiset只删除其中一个，

要s.erase(s.find(...))。 事实上，传入迭代器的话multiset就会只删一个。

set

set

set是有序的但不是连续的!

set

set是有序的但不是连续的!

lower_bound(s.begin(),s.end(),...)是$O(n)$的!

set

set是有序的但不是连续的!

lower_bound(s.begin(),s.end(),...)是$O(n)$的!

要用s.lower_bound(...)，这是$O(\log n)$的。

set

set是有序的但不是连续的!

lower_bound(s.begin(),s.end(),...)是$O(n)$的!

要用s.lower_bound(...)，这是$O(\log n)$的。

遍历set只能用迭代器。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

map

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。 m.empty():返回是否为空。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。 m.empty():返回是否为空。 m.clear():清空所有元素。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。 m.empty():返回是否为空。 m.clear():清空所有元素。

m[...]:返回map中相应的第一种元素的值的引用。注意它会新建这个元素！

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。 m.empty():返回是否为空。 m.clear():清空所有元素。

m[...]:返回map中相应的第一种元素的值的引用。注意它会新建这个元素！ m.count(...):返回map中是否有第一种元素为...的值。

map

类似于重载了[]运算符的$set < pair < ..., ... > >$

比如$map < string, int >$。前面那种东西必须定义<。然后相当于一个以前一种元素为下标，后一种元素为值的数组。访问元素是$O(\log n)$的。

m.size():返回元素个数。 m.empty():返回是否为空。 m.clear():清空所有元素。

m[...]:返回map中相应的第一种元素的值的引用。注意它会新建这个元素！ m.count(...):返回map中是否有第一种元素为...的值。

遍历map要用迭代器。迭代器相当于一个pair，first是下标，second是值。

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

Contents

1 总体介绍
- STL简介
- 左闭右开
- 迭代器
- <号的定义

2 常见的STL
- algorithm
- pair
- string
- vector
- queue与priority_queue
- set与multiset
- map

3 奇怪的错误
- == or not?

== or not?

== or not?

问题背景大概是这样的：有两个东西，每个东西都有一种颜色和一个值。颜色相同时值越大越好，颜色不同时不能比较。

== or not?

问题背景大概是这样的：有两个东西，每个东西都有一种颜色和一个值。颜色相同时值越大越好，颜色不同时不能比较。

小明对于这个问题写了一个map：

== or not?

问题背景大概是这样的：有两个东西，每个东西都有一种颜色和一个值。颜色相同时值越大越好，颜色不同时不能比较。

小明对于这个问题写了一个map：

```cpp
#include<map>
using namespace std;

struct node
{
    int col,key;
    bool operator < (const node& B) const
    {
        if (col==B.col) return key<B.key;
        return 0;
    }
};map<node,int> M;

int main()
{
}
```

== or not?

问题背景大概是这样的：有两个东西，每个东西都有一种颜色和一个值。颜色相同时值越大越好，颜色不同时不能比较。

小明对于这个问题写了一个map：

```cpp
#include<map>
using namespace std;

struct node
{
    int col,key;
    bool operator < (const node& B) const
    {
        if (col==B.col) return key<B.key;
        return 0;
    }
};map<node,int> M;

int main()
{
}
```

请思考这份代码有什么隐患。或者说，当他把全部代码写完后，可能会出现什么问题。

== or not?

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

所以呢?

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

所以呢?

小明的map会把所有颜色不相同的两个东西都当成是相等的!

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

所以呢?

小明的map会把所有颜色不相同的两个东西都当成是相等的!

这样带来的后果，轻则没什么事，重则会引发答案错误，甚至内存无法回收等一系列问题!

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

所以呢?

小明的map会把所有颜色不相同的两个东西都当成是相等的!

这样带来的后果，轻则没什么事，重则会引发答案错误，甚至内存无法回收等一系列问题!

这不是危言耸听，而是切身经历。

== or not?

在map内部，只定义了<号，是怎么判断两个东西相等的?

$!(a < b)\&\&!(b < a)$

所以呢?

小明的map会把所有颜色不相同的两个东西都当成是相等的!

这样带来的后果，轻则没什么事，重则会引发答案错误，甚至内存无法回收等一系列问题!

这不是危言耸听，而是切身经历。

请务必保证如果两个值不相等，要么$a < b$要么$b < a$。

当$a == b$时，<要返回0! 用字典序比较是一个比较好的选择。

Thanks for your listening.Questions are welcomed.