e

目录

[1. 小技巧 1](#_Toc535336112)

[1.1 64位整形的定义与输出 1](#_Toc535336113)

[1.2 快读 1](#_Toc535336114)

[1.3 计算程序运行时间 1](#_Toc535336115)

[1.4 double类型的输出 2](#_Toc535336116)

[2. 优先队列 2](#_Toc535336117)

[3. C++无序容器(hash表) 3](#_Toc535336118)

[4. C++ map，multimap 3](#_Toc535336119)

[5. C++set ，multiset 4](#_Toc535336120)

[6. C++istreamstring，ostreamstring 5](#_Toc535336121)

[7. C 字符串格式化输入输出 6](#_Toc535336122)

[7.1 sprintf()//格式化输出到字符串，针对字符数组 6](#_Toc535336123)

[7.2 sscanf()//从字符串中格式化输入，针对字符数组 7](#_Toc535336124)

[8. string成员函数 7](#_Toc535336125)

[8.1 string的特性描述: 7](#_Toc535336126)

[8.2 string的赋值： 7](#_Toc535336127)

[8.3 string的连接： 8](#_Toc535336128)

[8.4 string的比较： 8](#_Toc535336129)

[8.5 string的子串： 8](#_Toc535336130)

[8.7 string类的查找函数： 9](#_Toc535336131)

[8.7 string替换函数 //自行实现比较方便 10](#_Toc535336132)

[8.8 string类的插入函数： 10](#_Toc535336133)

[8.9 string类的删除函数 10](#_Toc535336134)

[8.10 string类的迭代器处理： 11](#_Toc535336135)

[8.11字符串流处理： 11](#_Toc535336136)

[9. string.h部分成员函数 11](#_Toc535336137)

[9.1 strcpy() 字符串拷贝 11](#_Toc535336138)

[9.2 strcat() 字符串拼接 11](#_Toc535336139)

[9.3 strcmp() 字符串比较 12](#_Toc535336140)

[9.4 strlen() 字符串长度 12](#_Toc535336141)

[9.5 strchr() 查询字符 12](#_Toc535336142)

[9.6 strstr() 查询子串 12](#_Toc535336143)

[9.7 strtok()分解字符串 str 为一组字符串，delim 为分隔符。 12](#_Toc535336144)

[10. c部分函数 13](#_Toc535336145)

[10.1 getline() 从流中输入，针对string类使用 头文件在string中 13](#_Toc535336146)

[10.2 cin.getline() 从流中输入，针对字符数组使用 13](#_Toc535336147)

[10.3 fgets() 从流中输入,针对字符数组 14](#_Toc535336148)

[11. c++部分函数（STL） 15](#_Toc535336149)

[11.1 lower\_bound（）和upper\_bound() 15](#_Toc535336150)

[11.2 unique()去重函数 15](#_Toc535336151)

**注：函数只是辅助工具，竞赛中思路才是王道。**

# 小技巧

## 1.1 64位整形的定义与输出

#include<stdio.h>

#include<iostream>

using namespace std;

int main()

{

/\*mingw32-g++环境下\*/

long long a=1e10;//long long形式定义一个a

\_\_int64 b=1e10;//定义一个64位整形

printf("a=%lld\n",a);//输出long long形式的

printf("b=%I64d\n",b);//输出64位整形的

cout<<"a="<<a<<" b="<<b<<endl;

return 0;//

}

## 1.2 快读

这里是一个读入int的示例。

inline int read(){

int s=0,w=1;

char ch=getchar();//多组输入eof返回-1

while(ch<'0'||ch>'9'){if(ch=='-')w=-1;ch=getchar();}

while(ch>='0'&&ch<='9') s=s\*10+ch-'0',ch=getchar();

return s\*w;

}

## 1.3 计算程序运行时间

#include<iostream>

#include<time.h>

using namespace std;

int main()

{

double Begtime,Endtime;

Begtime=(double)clock()/CLOCKS\_PER\_SEC;//单位是秒

/\*程序运行\*/

Endtime=(double)clock()/CLOCKS\_PER\_SEC;

cout<<"Runtime:"<<Endtime-Begtime<<" s"<<endl;

}

## 1.4 double类型的输出

POJdouble输出用%f，至于这点找资料到时候补充

# 优先队列

#include<cstdio>

#include<queue>

using namespace std;

int main()

{

/\*

普通队列是严格按照队列数据结构来的。

优先队列与普通队列不同，

1.优先队列队首元素总是最大的，因此执行pop操作时，删除的总是最大的元素

2.执行top操作时返回的总是最大元素的引用

3.默认比较器为less<T>

4.虽然每次取出的序列是有序的 内部使用堆排序实现，但是比multise/set略快一点

\*/

priority\_queue<int> mmp;/\*可以适当的对运算符号进行重载\*/

mmp.push(1);

mmp.push(2);

mmp.push(3);

mmp.push(4);

while(!mmp.empty())

{

printf("%d ",mmp.top());

mmp.pop();

}

/\*输出4 3 2 1\*/

return 0;

}

# C++无序容器(hash表)

C++11新增了四种所谓的”无序容器”，分别是unordered\_map、unordered\_set、unordered\_multimap、unordered\_multise，他们的实现都是hash表。查找时间比关联容器快，大多数情况下时间复杂度都是O（1）,无序容器的用法和关联容器相似。

例如下面关于unordered\_map的程序

#include<unordered\_map>

#include<iostream>

#include<string>

using namespace std;

int main()

{

unordered\_map<string,int> turingWinner;//图灵奖获奖名单

turingWinner.insert(make\_pair("Dijkstra",1972));

turingWinner.insert(make\_pair("Scott",1976));

turingWinner.insert(make\_pair("Wilkes",1967));

turingWinner.insert(make\_pair("Hamming",1968));

turingWinner["Ritchie"]=1983;

string name;

cin>>name;//输入姓名

unordered\_map<string ,int>::iterator p=turingWinner.find(name);

//根据姓名查找获奖时间

if(p!=turingWinner.end())

cout<<p->second<<endl;

else

cout<<"Not Found!"<<endl;

return 0;

}

# C++ map，multimap

map要求key值唯一，multimap key值可以重复，下面是map的使用方法

#include<iostream>

#include<algorithm>

#include<cstring>

#include<cstdio>

#include<map>

using namespace std;

typedef map<int,int> mymap;

int main()

{

map<int,int> mmp;//需要对第一个元素类型的运算符 < 号定义

/\* 插入\*/

mmp.insert(pair<int,int>(1,10));

mmp.insert(mymap::value\_type(2,10));

mmp.insert(make\_pair(5,10));

mmp[100]=10;

/\*遍历\*/

mymap::iterator it;

for(it=mmp.begin();it!=mmp.end();++it)

cout<< it->first <<":"<< it->second <<" ";//输出 1:10 2:10 5:10 100:10

cout<<endl;

/\*输出元素个数\*/

cout<<mmp.size()<<endl;//输出 4

/\*查找\*/

it=mmp.find(1);

cout<< it->second<<endl;//输出 10

/\*删除\*/

it=mmp.erase(it);//删除key值为1的

//mmp.erase(1);同样作用

/\*清空容器\*/

mmp.clear();

return 0;

}

成员函数还有lower\_bound()和upper\_bound()

# C++set ，multiset

set的使用方法multiset与值使用相同，set不允许元素重复，multiset允许元素重复

#include<iostream>

#include<algorithm>

#include<cstring>

#include<cstdio>

#include<set>

using namespace std;

int main()

{

set<int> a;

/\*插入\*/

a.insert(10);

a.insert(5);

a.insert(12);

a.insert(10);

/\*遍历容器\*/

set<int>::iterator it;//定义set<int> 类型的迭代器

for(it=a.begin(); it!=a.end(); ++it) //遍历容器a

cout<<\*it<<" ";

cout<<endl;//输出5 10 12

/\*容器内元素个数\*/

cout<<a.size()<<endl;//输出 3

/\*查找\*/

it=a.find(10);//查找10在容器a的位置

if(it==a.end())//

cout<<"Not exist\n"<<endl;

else

cout<<\*it<<endl;//输出10

/\*--删除--\*/

a.erase(10);

it=a.find(10);

if(it!=a.end())

a.erase(it);

/\*清空容器\*/

a.clear();

return 0;

}

成员函数还有lower\_bound()和upper\_bound()

# C++istreamstring，ostreamstring

C++字符串输入输出流

#include<cstdio>

#include<sstream>

#include<string>

#include<iostream>

using namespace std;

int main()

{

string src("Avatar 123 5.2 Titanic K");

istringstream istrStream(src);//建立src到istristrStream的联系，也可以使用istrStream.str()对其赋值

string s1,s2;

int n;double d;char c;

istrStream>>s1>>n>>d>>s2;//把src的内容当作输入流进行读取

ostringstream ostrStream;

ostrStream<<s1<<endl<<s2<<endl<<n<<endl<<d<<endl<<c<<endl;

cout<<ostrStream.str();//输出ostrStream的字符串，可使用成员函数clear()清空

return 0;

}

/\*

程序输出结果是：

Avatar

Titanic

123

5.2

K

\*/

# C 字符串格式化输入输出

## 7.1 sprintf()//格式化输出到字符串，针对字符数组

#include<cstdio>

using namespace std;

int main()

{

char s[100];

/\*

int sprintf( char \*buffer, const char \*format, [ argument] … );

printf为输出到屏幕上，此函数改为输出到指定字符串里。使用方法与printf相同

\*/

sprintf(s,"%.\*s",4,"abcded");

printf("%s\n",s);/\*abcd\*/

return 0;

}

%.\*s 表示有两项， 第一项指定了长度，第二项则是%s的内容，所以取前三位

## 7.2 sscanf()//从字符串中格式化输入，针对字符数组

#include<cstdio>

using namespace std;

int main()

{

char s[100]="100 50";

/\*

int sscanf(const char \*buffer,const char \*format, [ argument ] ... );

scanf为标准输入，此函数改为从到指定字符串里格式化输入。使用方法与scanf相同

\*/

int a,b;

sscanf(s,"%d %d",&a,&b);

printf("a=%d,b=%d\n",a,b);/\*输出a=100,b=50\*/

return 0;

}

# string成员函数

## 8.1 string的特性描述:

**int capacity()const;** //返回当前容量（即string中不必增加内存即可存放的元素个数）

**int max\_size()const;** //返回string对象中可存放的最大字符串的长度

**int size()const;** //返回当前字符串的大小

**int length()const;** //返回当前字符串的长度

**bool empty()const;** //当前字符串是否为空

**void resize(int len,char c);**//把字符串当前大小置为len，并用字符c填充不足的部分

## 8.2 string的赋值：

string &assign(const char \*s);//用c类型字符串s赋值

string &assign(const char \*s,int n);//用c字符串s开始的n个字符赋值

string &assign(const string &s);//把字符串s赋给当前字符串

string &assign(int n,char c);//用n个字符c赋值给当前字符串

string &assign(const string &s,int start,int n);//把字符串s中从start开始的n个字符赋给当前字符

string &assign(const\_iterator first,const\_itertor last); //把first和last迭代器之间的部分赋给字符串

**用重载运算符 ”=” 号赋值**

## 8.3 string的连接：

**string &operator+=(const string &s);//把字符串s连接到当前字符串的结尾**

string &append(const char \*s); //把c类型字符串s连接到当前字符串结尾

string &append(const char \*s,int n);//把c类型字符串s的前n个字符连接到当前字符串结尾

string &append(const string &s); //同operator+=()

string &append(const string &s,int pos,int n);//把字符串s中从pos开始的n个字符连接到当前字符串的结尾

string &append(int n,char c); //在当前字符串结尾添加n个字符c

string &append(const\_iterator first,const\_iterator last);//把迭代器first和last之间的部分连接到当前字符串的结尾

## 8.4 string的比较：

**bool operator==(const string &s1,const string &s2)const;//比较两个字符串是否相等　运算符">","<",">=","<=","!="均被重载用于字符串的比较；**

int compare(const string &s) const;//比较当前字符串和s的大小

int compare(int pos, int n,const string &s)const;//比较当前字符串从pos开始的n个字符组成的字符串与s的大小

int compare(int pos, int n,const string &s,int pos2,int n2)const;//比较当前字符串从pos开始的n个字符组成的字符串与s中//pos2开始的n2个字符组成的字符串的大小

int compare(const char \*s) const;

int compare(int pos, int n,const char \*s) const;

int compare(int pos, int n,const char \*s, int pos2) const;//compare函数在大于(>)时返回1，小于(<)时返回-1，等于(==)时返回0

## 8.5 string的子串：

**string substr(int pos = 0,int n = npos) const;//返回pos开始的n个字符组成的字符串**

8.6 **string的交换**

void swap(string &s2); //交换当前字符串与s2的值

## 8.7 string类的查找函数：

**int find(char c, int pos = 0) const;//从pos开始查找字符c在当前字符串的位置**

**int find(const char \*s,int pos = 0) const;//从pos开始查找字符串s在当前串中的位置**

int find(const char \*s, int pos, int n) const;//从pos开始查找字符串s中前n个字符在当前串中的位置

**int find(const string &s,int pos = 0) const;//从pos开始查找字符串s在当前串中的位置　//查找成功时返回所在位置，失败返回string::npos的值**

int rfind(char c, int pos = npos) const;//从pos开始从后向前查找字符c在当前串中的位置

int rfind(const char \*s, int pos = npos) const;

int rfind(const char \*s, int pos, int n = npos) const;

int rfind(const string &s,int pos = npos) const;　//从pos开始从后向前查找字符串s中前n个字符组成的字符串在当前串中的位置，成功返回所在位置，失败时返回string::npos的值

int find\_first\_of(char c, int pos = 0) const;//从pos开始查找字符c第一次出现的位置

int find\_first\_of(const char \*s, int pos = 0) const;

int find\_first\_of(const char \*s, int pos, int n) const;

int find\_first\_of(const string &s,int pos = 0) const;　//从pos开始查找当前串中第一个在s的前n个字符组成的数组里的字符的位置。查找失败返回string::npos

int find\_first\_not\_of(char c, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos = 0) const;

int find\_first\_not\_of(const char \*s, int pos,int n) const;

int find\_first\_not\_of(const string &s,int pos = 0) const;　//从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos

int find\_last\_of(char c, int pos = npos) const;

int find\_last\_of(const char \*s, int pos = npos) const;

int find\_last\_of(const char \*s, int pos, int n = npos) const;

int find\_last\_of(const string &s,int pos = npos) const;

int find\_last\_not\_of(char c, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos = npos) const;

int find\_last\_not\_of(const char \*s, int pos, int n) const;

int find\_last\_not\_of(const string &s,int pos = npos) const;　//find\_last\_of和find\_last\_not\_of与find\_first\_of和find\_first\_not\_of相似，只不过是从后向前查找。

## 8.7 string替换函数 //自行实现比较方便

string &replace(int p0, int n0,const char \*s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const char \*s, int n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符

string &replace(int p0, int n0,const string &s);//删除从p0开始的n0个字符，然后在p0处插入串s

string &replace(int p0, int n0,const string &s, int pos, int n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符

string &replace(int p0, int n0,int n, char c);//删除p0开始的n0个字符，然后在p0处插入n个字符c

string &replace(iterator first0, iterator last0,const char \*s);//把[first0，last0）之间的部分替换为字符串s

string &replace(iterator first0, iterator last0,const char \*s, int n);//把[first0，last0）之间的部分替换为s的前n个字符。

string &replace(iterator first0, iterator last0,const string &s);//把[first0，last0）之间的部分替换为串s

string &replace(iterator first0, iterator last0,int n, char c);//把[first0，last0）之间的部分替换为n个字符c

string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串。

## 8.8 string类的插入函数：

string &insert(int p0, const char \*s);

string &insert(int p0, const char \*s, int n);

string &insert(int p0,const string &s);

string &insert(int p0,const string &s, int pos, int n);　//前4个函数在p0位置插入字符串s中pos开始的前n个字符

string &insert(int p0, int n, char c);//此函数在p0处插入n个字符c

iterator insert(iterator it, char c);//在it处插入字符c，返回插入后迭代器的位置

void insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符

void insert(iterator it, int n, char c);//在it处插入n个字符c

## 8.9 string类的删除函数

iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置。

iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置。

string &erase(int pos = 0, int n = npos);//删除pos开始的n个字符，返回修改后的字符串。

## 8.10 string类的迭代器处理：

string类提供了向前和向后遍历的迭代器iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。  
　　用string::iterator或string::const\_iterator声明迭代器变量，const\_iterator不允许改变迭代的内容。常用迭代器函数有：

const\_iterator begin()const;　iterator begin(); //返回string的起始位置

const\_iterator end()const;　iterator end(); //返回string的最后一个字符后面的位置

const\_iterator rbegin()const;　iterator rbegin(); //返回string的最后一个字符的位置

const\_iterator rend()const;　iterator rend(); //返回string第一个字符位置的前面

## 8.11字符串流处理：

通过定义ostringstream和istringstream变量实现，在#include <sstream>头文件中。

# string.h部分成员函数

## 9.1 strcpy() 字符串拷贝

原型：char \*strcpy(char\* dest, const char \*src);

功能：把从 src 地址开始且含有 NULL 结束符的字符串复制到以 dest 开始的地址空间。

说明：dest 必须有足够的空间来容纳 src 的字符串，返回指向 dest 的指针。

## 9.2 strcat() 字符串拼接

原型：char \*strcat(char \*dest, const char \*src);

功能：把 src 所指字符串添加到 dest 结尾处(覆盖 dest 结尾处的'\0')。

说明：dest 必须有足够的空间来容纳 src 的字符串，返回指向 dest 的指针。

## 9.3 strcmp() 字符串比较

原型: int strcmp(char \*str1, char \*str2);

功能：两串自左向右逐字符比较，str1>str2，返回正数；str1==str2，返回 0

## 9.4 strlen() 字符串长度

原型: size\_t strlen(const char \*string);

功能：从字符串的首地址开始到遇到第一个'\0'停止计数。

## 9.5 strchr() 查询字符

原型：char \*strchr(const char \*s,char c);

功能：查找字符串 s 中首次出现字符 c 的位置。 返回首次出现 s 位置的指针，返回的地址是被查找字符串指针开始的第一个与 c 相 同字符的指针，如果 s 中不存在 c 则返回 NULL。

## 9.6 strstr() 查询子串

原型：char \*strstr(char \*str1, const char \*str2);

功能：用于判断字符串 str2 是否是 str1 的子串。如果是，则该函数返回 str2 在 str1 中第 一次出现的地址；否则，返回 NULL。

## 9.7 strtok()分解字符串 str 为一组字符串，delim 为分隔符。

原型：char \*strtok(char s[], const char \*delim);

功能：分解字符串为一组字符串。s为要分解的字符串，delim为分隔符字符串（分隔符字符串中的每个字符都会当作分隔符）。

说明：strtok()用来将字符串分割成一个个片段。参数s指向欲分割的字符串，参数delim则为分割字符串中包含的所有字符。当strtok()在参数s的字符串中发现参数delim中**包含的分割字符**时,**则会将该字符改为\0 字符**。在第一次调用时，strtok()必需给予参数s字符串，往后的调用则将参数s设置成NULL。每次调用成功则返回指向被分割出片段的指针。strtok函数会破坏被分解字符串的完整，调用前和调用后的s已经不一样了。如果要保持原字符串的完整，可以使用strchr和sscanf的组合等。

# c部分函数

## 10.1 getline() 从流中输入，针对string类使用 头文件在string中

#include<iostream>

#include<string>

using namespace std;

int main()

{

/\*

string中的函数

函数原型：istream& getline (istream& is, string& str, char delim='\n');

解释:is为输入流，读取的字符串保存在string类型的str，delim为结束符

功能：若先碰见delim，则停止吸收字符，并将该delim字符转换成'\0',注意delim已经被吸收

\*/

string str;

char s[100];

getline(cin,str,'c');/\*abcdef\*/

cout<<str.length()<<" "<<str<<endl;/\*2 ab\*/

getline(cin,str);/\*abcd\*/

cout<<str.length()<<" "<<str<<endl;/\*4 abcd\*/

return 0;

}

## 10.2 cin.getline() 从流中输入，针对字符数组使用

#include<iostream>

#include<string>

using namespace std;

int main()

{

/\*

cin的成员函数:

函数原型：istream& getline (char\* s, streamsize n, char delim='\n');

解释:s为对应的字符串，streamsizes字符串要接受的字符数量(包括字符串结束符号),

delim为结束符(遇见结束符即停止)

功能：若先碰见delim，则停止吸收字符，并将该delim字符转换成'\0',注意delim已经被吸收

或者吸收n-1个字符，

\*/

char s[100];

cin.getline(s,5);/\*输入:abcdefg\*/

cout<<s<<endl;/\*输出:abcd\*/

cin.getline(s,5,'c');/\*输入:abcdefg\*/

cout<<s<<endl;/\*输出:ab\*/

return 0;

}

## 10.3 fgets() 从流中输入,针对字符数组

#include<iostream>

#include<string>

#include<cstring>

#include<cstdio>

using namespace std;

int main()

{

/\*

函数原型：

char \*fgets(char \*buf, int bufsize, FILE \*stream);

从stream中吸收bufsize-1个字符,或者遇见回车结束

注意：遇见回车结束时会把回车也加到s里面

\*/

char s[100];

fgets(s,5,stdin);/\*输入:ab回车cd\*/

cout<<strlen(s)<<" "<<s<<endl;/\*3 ab回车\*/

return 0;

}

# c++部分函数（STL）

## 11.1 lower\_bound（）和upper\_bound()

**可以对数组使用，也可以对顺序容器使用,需要对<号重载,时间复杂度O(logn)级别**

lower\_bound()返回一个 iterator 它指向在[first,last)标记的**有序序列**中可以插入value，而不会破坏容器顺序的第一个位置，而这个位置标记了一个**不小于**value 的值。该函数为[C++](https://baike.baidu.com/item/C%2B%2B) [STL](https://baike.baidu.com/item/STL/70103)内的函数。（即第一个>=val的位置，若不存在则返回last）

upper\_bound()返回一个 iterator 它指向在[first,last)标记的**有序序列**中可以插入value，而不会破坏容器顺序的第一个位置，而这个位置标记了一个**不小于**value 的值。该函数为[C++](https://baike.baidu.com/item/C%2B%2B) [STL](https://baike.baidu.com/item/STL/70103)内的函数。(第一个>val的位置，若不存在则返回 last)

#include<algorithm>

#include<stdio.h>

using namespace std;

int main()

{

int a[]={1,2,3,4,4,4,5,6};

printf("第一个不小于4的位置为:%d\n",lower\_bound(a,a+8,4)-a);/\*输出：3 \*/

printf("第一个大于4的位置为：%d\n",upper\_bound(a,a+8,4)-a);/\*输出:6 \*/

return 0;

}

## 11.2 unique()去重函数

unique()返回一个 iterator 它指向在[first,last)标记的**有序序列，他会把不重复的元素移动到**[first,iterator) ,作用为去重函数，它并不会把重复的元素删除，而是把重复的移动到[iterator,last).时间复杂度为O(n)级别

#include <iostream>

#include <algorithm>

using namespace std;

int main(void)

{

int a[8] = {2,2,2,4,4,6,7,8};

int c;

//sort(a, a + 8); //对于无序的数组需要先排序

c=(unique(a,a+8)-a);

cout<<c<<endl;

for(int i=0;i<8;++i)

cout<<a[i]<<" ";

cout<<endl;

/\*

输出:

5

2 4 6 7 8 6 7 8

\*/

return 0;

}