标准C++中的string类的用法总结

#include <string>

using std::string; using std::wstring; 或using namespace std;

string 类的构造函数：

string(const char \*s);    //用c字符串s初始化

string(int n,char c);     //用n个字符c初始化

此外，string类还支持默认构造函数和复制构造函数，如string s1；string s2=”hello”；都是正确的写法。当构造的string太长而无法表达时会抛出length\_error异常 ；

string 类的字符操作：

const char &operator[](int n)const;

const char &at(int n)const;

char &operator[](int n);

char &at(int n);

operator[]和at()均返回当前字符串中第n个字符的位置，但at函数提供范围检查，当越界时会抛出out\_of\_range异常，下标运算符[]不提供检查访问。

const char \*data()const;//返回一个非null终止的c字符数组

const char \*c\_str()const;//返回一个以null终止的c字符串

int copy(char \*s, int n, int pos = 0) const;//把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目

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

string类的输入输出操作:

string类重载运算符operator>>用于输入，同样重载运算符operator<<用于输出操作。

函数getline(istream &in,string &s);用于从输入流in中读取字符串到s中，以换行符’\n’分开。

string的赋值：

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

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串

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

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

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>时返回1，<时返回-1，==时返回0

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

string的交换：

void swap(string &s2);    //交换当前字符串与s2的值

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

string 类的替换函数：

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

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

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

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

rbegin和rend用于从后向前的迭代访问，通过设置迭代器string::reverse\_iterator,string::const\_reverse\_iterator实现

字符串流处理：

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

例如：

string input("hello,this is a test");

    istringstream is(input);

    string s1,s2,s3,s4;

    is>>s1>>s2>>s3>>s4;//s1="hello,this",s2="is",s3="a",s4="test"

    ostringstream os;

    os<<s1<<s2<<s3<<s4;

    cout<<os.str();