12.1答：b1=4,b1=0;

12.2答：加上const就可以了

12.3答：不需要，const成员是const对象对外唯一借口，而常对象也不允许被修改，const成员函数也不会修改对象的数据的。

12.4答：可以，size\_type可以保证是大于等于0的数。

12.5答：容易使用，不过会给调试带来难度。 可以直接初始化。

12.6-7答：#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

shared\_ptr<vector<int>> vector\_generator()

{

shared\_ptr<vector<int>> ptr\_v = make\_shared<vector<int>>();

return ptr\_v;

}

shared\_ptr<vector<int>> vector\_process(shared\_ptr<vector<int>> ptr\_v)

{

int i;

while (cin >> i)

{

ptr\_v->push\_back(i);

}

return ptr\_v;

}

void vector\_printer(shared\_ptr<vector<int>> ptr\_v)

{

for (const auto &e : \*ptr\_v)

cout << e << " ";

cout << "\n";

}

int main()

{

shared\_ptr<vector<int>> my\_ptr = vector\_generator();

vector\_process(my\_ptr);

vector\_printer(my\_ptr);

return 0;

}

12.8答：转换成bool型，内存没被释放，会造成内存泄露。

12.9答：r=q之后造成内存泄露

12.10答：#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

void process(std::shared\_ptr<int> ptr)

{

std::cout << "inside the process function:" << ptr.use\_count() << endl;

}

int main()

{

std::shared\_ptr<int> p(new int(42));

process(std::shared\_ptr<int>(p));//2

std::cout << p.use\_count() << endl;//1

auto q = p;

std::cout << p.use\_count() << endl;//2

std::cout << "the int p now points to is:" << \*p << endl;//42

return 0;

}

12.11#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

void process(std::shared\_ptr<int> ptr)

{

std::cout << "inside the process function:" << ptr.use\_count() << endl;

}

int main()

{

std::shared\_ptr<int> p(new int(42));

process(std::shared\_ptr<int>(p.get()));

return 0;

}

12.12答：合法、不合法、不合法、合法

12.13答：#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

int main()

{

auto sp = std::make\_shared<int>(1);

auto p = sp.get();

cout << \*sp << endl;

cout << \*p << endl;

//delete p;

return 0;

}运行发生错误

12.14答：

#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

struct destination{};

struct connection{};

connection connect(destination\*);

void disconnect(connection);

void f(destination &d);

void end\_connection(connection \*p);

int main()

{

return 0;

}

connection connect(destination \*)

{

connection conn;

return conn;

}

void f(destination &d)

{

connection conn = connect(&d);

std::shared\_ptr<connection> p\_conn(&conn, end\_connection);

}

void end\_connection(connection \*p)

{

disconnect(\*p);

}

12.15答：

#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

struct destination{};

struct connection{};

connection connect(destination\*);

void disconnect(connection);

void f(destination &d);

void end\_connection(connection \*p);

int main()

{

return 0;

}

connection connect(destination \*)

{

connection conn;

return conn;

}

void f(destination &d)

{

connection conn = connect(&d);

std::shared\_ptr<connection> p\_conn(&conn, end\_connection);

}

void end\_connection(connection \*p)

{

disconnect(\*p);

}

void lamda\_f(destination &d)

{

connection conn = connect(&d);

std::shared\_ptr<connection> p\_conn(&d, [](connection \*p){disconnect(\*p); });

}

12.16答：

#include <iostream>

#include <vector>

#include <string>

#include <memory>

using namespace std;

int main()

{

unique\_ptr<string> p1(new string("chenxun"));

unique\_ptr<string>p2(p1);

}

12.17答：（a）编译的时候出现错误（b）运行时出现错误（c）

12.18答：不知道为什么没有。哈哈。请自己Google吧

12.19答：#include <iostream>

#include <vector>

#include <string>

#include <memory>

#include <string>

using namespace std;

class StrBlobPtr;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlob{

friend class StrBlobPtr;

public:

typedef std::vector<std::string>::size\_type size\_type;

StrBlob() :data(make\_shared<vector<string>>()){}

StrBlob(initializer\_list<string> il) :data(make\_shared<vector<string>>(il)){}

size\_type size() const { return data->size(); }

bool empty() const { return data->empty(); }

void push\_back(const std::string &t) { data->push\_back(t); }

void pop\_back();

std::string &front();

std::string &back();

const std::string& front() const;

const std::string& back() const;

/\*\*

\* @attention these two members must defined after wy\_StrBlobPtr's

\* declararion and before its definition.

\*/

StrBlobPtr begin();

StrBlobPtr end();

private:

std::shared\_ptr<std::vector<std::string>> data;

void check(size\_type i, const std::string &msg) const;

};

void StrBlob::check(size\_t i, const string &msg) const

{

if (i >= data->size())

throw out\_of\_range(msg);

}

string& StrBlob::front()

{

check(0, "front on empty StrBlob");

return data->front();

}

string& StrBlob::back()

{

check(0,"back on empty StrBlob");

return data->back();

}

void StrBlob::pop\_back()

{

check(0, "pop\_back on empty StrBlob");

data->pop\_back();

}

const std::string& StrBlob::front() const

{

check(0, "front on empty StrBlob");

return data->front();

}

const std::string& StrBlob::back() const

{

check(0, "back on empty StrBlob");

return data->back();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlobPtr{

public:

StrBlobPtr() :curr(0){};

StrBlobPtr(StrBlob &a, rsize\_t sz= 0):wptr(a.data),curr(sz){ };

std::string &deref() const;

StrBlobPtr &incr();//前缀递增

private:

std::shared\_ptr < std::vector < std::string >>

check(std::size\_t, const std::string&) const;

std::weak\_ptr<std::vector<std::string>> wptr;

std::size\_t curr;

};

std::shared\_ptr < std::vector < std::string >> StrBlobPtr::check(std::size\_t i, const std::string& msg) const

{

auto ret = wptr.lock();//vector是否存在

if (!ret)

throw std::runtime\_error("unbound StrBlobPtr");

if (i >= ret->size())

throw std::out\_of\_range(msg);

return ret;

}

std::string& StrBlobPtr::deref() const

{

auto p= check(curr,"dereference past end");

return (\*p)[curr];

}

StrBlobPtr & StrBlobPtr::incr()

{

check(curr, "increment past end of StrBlobPtr");

++curr;

return \*this;

}

StrBlobPtr StrBlob::begin(){ return StrBlobPtr(\*this); }

StrBlobPtr StrBlob::end()

{

auto ret = StrBlobPtr(\*this, data->size());

return ret;

}

12.20答：

#include <iostream>

#include <vector>

#include <string>

#include <memory>

#include <string>

#include <fstream>

using namespace std;

class StrBlobPtr;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlob{

friend class StrBlobPtr;

public:

typedef std::vector<std::string>::size\_type size\_type;

StrBlob() :data(make\_shared<vector<string>>()){}

StrBlob(initializer\_list<string> il) :data(make\_shared<vector<string>>(il)){}

size\_type size() const { return data->size(); }

bool empty() const { return data->empty(); }

void push\_back(const std::string &t) { data->push\_back(t); }

void pop\_back();

std::string &front();

std::string &back();

const std::string& front() const;

const std::string& back() const;

/\*\*

\* @attention these two members must defined after wy\_StrBlobPtr's

\* declararion and before its definition.

\*/

StrBlobPtr begin();

StrBlobPtr end();

private:

std::shared\_ptr<std::vector<std::string>> data;

void check(size\_type i, const std::string &msg) const;

};

void StrBlob::check(size\_t i, const string &msg) const

{

if (i >= data->size())

throw out\_of\_range(msg);

}

string& StrBlob::front()

{

check(0, "front on empty StrBlob");

return data->front();

}

string& StrBlob::back()

{

check(0,"back on empty StrBlob");

return data->back();

}

void StrBlob::pop\_back()

{

check(0, "pop\_back on empty StrBlob");

data->pop\_back();

}

const std::string& StrBlob::front() const

{

check(0, "front on empty StrBlob");

return data->front();

}

const std::string& StrBlob::back() const

{

check(0, "back on empty StrBlob");

return data->back();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlobPtr{

public:

StrBlobPtr():curr(0){};

StrBlobPtr(StrBlob &a, rsize\_t sz= 0):wptr(a.data),curr(sz){ };

std::string &deref() const;

StrBlobPtr &incr();//前缀递增

private:

std::shared\_ptr < std::vector < std::string >>

check(std::size\_t, const std::string&) const;

std::weak\_ptr<std::vector<std::string>> wptr;

std::size\_t curr;

};

std::shared\_ptr < std::vector < std::string >> StrBlobPtr::check(std::size\_t i, const std::string& msg) const

{

auto ret = wptr.lock();//vector是否存在

if (!ret)

throw std::runtime\_error("unbound StrBlobPtr");

if (i >= ret->size())

throw std::out\_of\_range(msg);

return ret;

}

std::string& StrBlobPtr::deref() const

{

auto p= check(curr,"dereference past end");

return (\*p)[curr];

}

StrBlobPtr & StrBlobPtr::incr()

{

check(curr, "increment past end of StrBlobPtr");

++curr;

return \*this;

}

StrBlobPtr StrBlob::begin(){ return StrBlobPtr(\*this); }

StrBlobPtr StrBlob::end()

{

auto ret = StrBlobPtr(\*this, data->size());

return ret;

}

int main()

{

std::string line;

std::ifstream fin("chenxun.txt");

StrBlob sb;

StrBlobPtr wp(sb);

while (std::getline(fin, line))

{

sb.push\_back(line);

std::cout << wp.deref() << endl;

wp.incr();

}

return 0;

}

12.21答：可读性差

12.22答：修改其构造函数加入const属性

12.23答：#include<iostream>

#include<string>

using namespace std;

char \*my\_concat1(const char\* const s1, const char \*const s2)

{

char \*const result = new char[sizeof(s1)+sizeof(s2)+5];

char \*my\_result = result;

const char\*my\_s1 = s1;

const char\*my\_s2 = s2;

while (\*my\_s1!='\0')

{

\*my\_result = \*my\_s1;

++my\_result;

++my\_s1;

}

while (\*my\_s2 != '\0')

{

\*my\_result = \*my\_s2;

++my\_result;

++my\_s2;

}

\*my\_result = '\0';

return result;

}

char \*my\_concat2(const std::string s1, const std::string s2)

{

char \*const result = new char[sizeof(s1)+sizeof(s2)+5];

char \*my\_result = result;

for (const auto &s : s1)

{

\*my\_result = s;

++my\_result;

}

for (const auto &s : s2)

{

\*my\_result = s;

++my\_result;

}

\*my\_result = '\0';

return result;

}

int main()

{

std::string s1, s2;

std::cin >> s1 >> s2;

char\*my\_char = my\_concat2(s1, s2);

cout << my\_char << endl;

delete[] my\_char;

return 0;

}

12.24答：如上题所示

12.25答：delete [] pa;

12.26答：#include<iostream>

#include<string>

#include<memory>

using namespace std;

int main()

{

std::allocator<std::string> alloc;

std::string \*const p = alloc.allocate(10);

std::string \*q = p;

std::string word;

while (cin >> word&&q != p + 2)

{

alloc.construct(q, word);

++q;

}

q = p;

while (q!=p+2)

{

cout << \*q++ << endl;

}

while (q != p)

alloc.destroy(--q);

return 0;

}

12.27答：#include<iostream>

#include<string>

#include<memory>

#include<vector>

#include<map>

#include<set>

#include<fstream>

#include<sstream>

using namespace std;

class queryResult;

class textQuery

{

friend class queryResult;

public:

textQuery() = default;

textQuery(std::ifstream &fin);

const queryResult query(const std::string &word);

private:

std::shared\_ptr<std::vector<std::string>> file;

std::shared\_ptr<std::map<std::string, std::set<int>>> my\_map;

};

textQuery::textQuery(std::ifstream &fin) :

file(std::make\_shared<std::vector<std::string>>()),

my\_map(std::make\_shared<std::map<std::string, std::set<int>>>())

{

std::string line;

while (std::getline(fin, line))

{

(\*file).push\_back(line);

}

}

class queryResult{

friend void print(std::ostream &os, const queryResult &qr);

public:

queryResult() = default;

queryResult(const queryResult &qr);

queryResult(std::size\_t c, const std::string &s,

const std::shared\_ptr<std::vector<std::string>> &sp\_f,

const std::shared\_ptr<std::map<std::string, std::set<int>>> &sp\_m);

std::size\_t getCounter() const { return counter; }

std::string getQueryWord() const { return queryWord; }

std::shared\_ptr<std::vector<std::string>> get\_file() const { return f; }

std::shared\_ptr<std::map<std::string, std::set<int>>> get\_my\_map() const { return m; }

private:

//! number of occurrence

std::size\_t counter = 0;

//! the word being searched

std::string queryWord = "";

//! smart pointer to a vector to be storing a file.

std::shared\_ptr<std::vector<std::string>> f = nullptr;

//! smart pointer to a map to be storing results of querries.

std::shared\_ptr<std::map<std::string, std::set<int>>> m = nullptr;

};

const queryResult textQuery::query(const std::string &qWord)

{

//! storing the amount of occurrence

std::size\_t counter = 0;

//! loop through each line

for (std::size\_t i = 0; i != file->size(); ++i)

{

//! break into each word

std::stringstream lineStream((\*file)[i]);

std::string word;

while (lineStream >> word)

{

if (!word.compare(qWord))

{

++counter;

//! add the index of the line into the result map

(\*my\_map)[qWord].insert(i);

}

}

}

//! creare a object holding the result .

queryResult qResult(counter, qWord, file, my\_map);

return qResult;

}

//copy construct

inline queryResult::queryResult(const queryResult &qr) :counter(qr.getCounter()),

queryWord(qr.getQueryWord()),

f(qr.get\_file()),

m(qr.get\_my\_map())

{

cout << 2 << endl;

}

//construct

queryResult::queryResult(std::size\_t c, const std::string &str,

const shared\_ptr<std::vector<std::string>> &sp\_f,

const shared\_ptr<std::map<std::string, std::set<int>>> &sp\_m) :

counter(c), queryWord(str), f(sp\_f), m(sp\_m)

{

cout << 1 << endl;

}

//! a non-member function printing the result of a query.

void print(std::ostream &os, const queryResult &qr)

{

//! fetch the word being queried.

const std::string queryWord = qr.getQueryWord();

//! print the word and occurrence times

os << "The word ["

<< queryWord

<< "] occurs "

<< qr.getCounter()

<< " times :\n";

//! fetch smart pointers to the map and the file

auto sp\_m = qr.get\_my\_map();

auto sp\_f = qr.get\_file();

//! print each line in which it appears and the corresponding index.

for (const auto &index : (\*sp\_m)[queryWord])

std::cout << "\n(Line "

<< index

<< ") "

<< (\*sp\_f)[index]

<< "\n\n";

}

void runQueries(ifstream &infile)

{

textQuery tq(infile);

while (true)

{

cout << "enter word to look for, or q to quit:";

string word;

if (!(cin >> word) || word == "q")

break;

print(cout, tq.query(word));

}

}

int main()

{

ifstream infile("chenxun.txt");

runQueries(infile);

return 0;

}

12.28答：

、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、

12.30答：#include<iostream>

#include<string>

#include<memory>

#include<vector>

#include<map>

#include<set>

#include<fstream>

#include<sstream>

using namespace std;

string make\_plural(size\_t ctr, const string word, const string &ending)

{

return (ctr > 1) ? word + ending : word;

}

class QueryRsult;

class TextQuery

{

public:

using line\_no = std::vector<std::string>::size\_type;

TextQuery(std::ifstream &);

QueryRsult query(const std::string&) const;

private:

std::shared\_ptr<std::vector<std::string>> file;

std::map<std::string, std::shared\_ptr<std::set<line\_no>>> wm;

};

TextQuery::TextQuery(ifstream &is) :file(new vector<string>)

{

string text;

while (getline(is, text))

{

file->push\_back(text);

int n = file->size() - 1;

istringstream line(text);

string word;

while (line >> word)

{

auto &lines = wm[word];

//shared\_ptr<std::set<line\_no>> &lines = wm[word];

if (!lines)

lines.reset(new set<line\_no>);

lines->insert(n);

}

}

}

class QueryRsult{

friend std::ostream &print(std::ostream &, const QueryRsult &);

public:

using line\_no = std::vector<std::string>::size\_type;

QueryRsult(std::string s,

std::shared\_ptr<std::set<line\_no>> p,

std::shared\_ptr<std::vector<std::string>> f) :

sought(s), lines(p), file(f){}

private:

std::string sought;

std::shared\_ptr<std::set<line\_no>> lines;

std::shared\_ptr<std::vector<string>> file;

};

QueryRsult TextQuery::query(const string &sought) const

{

static shared\_ptr<std::set<line\_no>> nodata(new set<line\_no>);

auto loc = wm.find(sought);

if (loc == wm.end())

return QueryRsult(sought, nodata, file);

else

return QueryRsult(sought, loc->second, file);

}

ostream &print(ostream &os, const QueryRsult &qr)

{

os << qr.sought << "occurs" << qr.lines->size() << " "

<< make\_plural(qr.lines->size() ,"time","s") << endl;

for (auto num : \*qr.lines)

os << num + 1 << " " << \*(qr.file->begin() + num) << endl;

return os;

}

void runQueries(ifstream &infile)

{

TextQuery tq(infile);

while (true)

{

cout << "enter word to look for, or q to quit:";

string word;

if (!(cin >> word) || word == "q")

break;

print(cout, tq.query(word)) << endl;

}

}

int main()

{

ifstream infile("chenxun.txt");

runQueries(infile);

return 0;

}

、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、

12.32答：#include<iostream>

#include<string>

#include<memory>

#include<vector>

#include<map>

#include<set>

#include<fstream>

#include<sstream>

#include"strblob.h"

using namespace std;

string make\_plural(size\_t ctr, const string word, const string &ending)

{

return (ctr > 1) ? word + ending : word;

}

class QueryRsult;

class TextQuery

{

public:

using line\_no = std::vector<std::string>::size\_type;

TextQuery(std::ifstream &);

QueryRsult query(const std::string&) const;

private:

StrBlob file;

std::map<std::string, std::shared\_ptr<std::set<line\_no>>> wm;

};

TextQuery::TextQuery(ifstream &is) :file(StrBlob())

{

string text;

while (getline(is, text))

{

file.push\_back(text);

int n = file.size() - 1;

istringstream line(text);

string word;

while (line >> word)

{

//auto &lines = wm[word];

shared\_ptr<std::set<line\_no>> &lines = wm[word];

if (!lines)

lines.reset(new set<line\_no>);

lines->insert(n);

}

}

}

class QueryRsult{

friend std::ostream &print(std::ostream &, const QueryRsult &);

public:

using line\_no = std::vector<std::string>::size\_type;

QueryRsult( std::string s,

std::shared\_ptr<std::set<line\_no>> p,

StrBlob f ):

sought(s), lines(p), file(f){}

private:

std::string sought;

std::shared\_ptr<std::set<line\_no>> lines;

StrBlob file;

};

QueryRsult TextQuery::query(const string &sought) const

{

static shared\_ptr<std::set<line\_no>> nodata(new set<line\_no>);

auto loc = wm.find(sought);

if (loc == wm.end())

return QueryRsult(sought, nodata, file);

else

return QueryRsult(sought, loc->second, file);

}

ostream &print(ostream &os,const QueryRsult &qr)

{

os << qr.sought << "occurs" << qr.lines->size() << " "

<< make\_plural(qr.lines->size(), "time", "s") << endl;

for (auto num : \*qr.lines)

{

os << num + 1 << " ";

const StrBlobPtr wp(qr.file, num);

os << wp.deref() << "\n";

}

return os;

}

void runQueries(ifstream &infile)

{

TextQuery tq(infile);

while (true)

{

cout << "enter word to look for, or q to quit:";

string word;

if (!(cin >> word) || word == "q")

break;

print(cout, tq.query(word)) << endl;

}

}

int main()

{

ifstream infile("chenxun.txt");

runQueries(infile);

return 0;

}

、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、、

#ifndef STRBLOB\_H

#define STRBLOB\_H

#include <iostream>

#include <vector>

#include <string>

#include <memory>

#include <string>

#include <fstream>

using namespace std;

class StrBlobPtr;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlob{

friend class StrBlobPtr;

public:

typedef std::vector<std::string>::size\_type size\_type;

StrBlob() :data(make\_shared<vector<string>>()){}

StrBlob(initializer\_list<string> il) :data(make\_shared<vector<string>>(il)){}

size\_type size() const { return data->size(); }

bool empty() const { return data->empty(); }

void push\_back(const std::string &t) { data->push\_back(t); }

void pop\_back();

std::string &front();

std::string &back();

const std::string& front() const;

const std::string& back() const;

/\*\*

\* @attention these two members must defined after wy\_StrBlobPtr's

\* declararion and before its definition.

\*/

StrBlobPtr begin();

StrBlobPtr end();

private:

std::shared\_ptr<std::vector<std::string>> data;

void check(size\_type i, const std::string &msg) const;

};

void StrBlob::check(size\_t i, const string &msg) const

{

if (i >= data->size())

throw out\_of\_range(msg);

}

string& StrBlob::front()

{

check(0, "front on empty StrBlob");

return data->front();

}

string& StrBlob::back()

{

check(0, "back on empty StrBlob");

return data->back();

}

void StrBlob::pop\_back()

{

check(0, "pop\_back on empty StrBlob");

data->pop\_back();

}

const std::string& StrBlob::front() const

{

check(0, "front on empty StrBlob");

return data->front();

}

const std::string& StrBlob::back() const

{

check(0, "back on empty StrBlob");

return data->back();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class StrBlobPtr{

public:

StrBlobPtr() :curr(0){};

StrBlobPtr(StrBlob &a, rsize\_t sz = 0) :wptr(a.data), curr(sz){ };

StrBlobPtr(const StrBlob &a, const size\_t sz = 0) : wptr(a.data), curr(sz) { }

std::string &deref() const;

StrBlobPtr &incr();//前缀递增

private:

std::shared\_ptr < std::vector < std::string >>

check(std::size\_t, const std::string&) const;

std::weak\_ptr<std::vector<std::string>> wptr;

std::size\_t curr;

};

std::shared\_ptr < std::vector < std::string >> StrBlobPtr::check(std::size\_t i, const std::string& msg) const

{

auto ret = wptr.lock();//vector是否存在

if (!ret)

throw std::runtime\_error("unbound StrBlobPtr");

if (i >= ret->size())

throw std::out\_of\_range(msg);

return ret;

}

std::string& StrBlobPtr::deref() const

{

auto p = check(curr, "dereference past end");

return (\*p)[curr];

}

StrBlobPtr & StrBlobPtr::incr()

{

check(curr, "increment past end of StrBlobPtr");

++curr;

return \*this;

}

StrBlobPtr StrBlob::begin(){ return StrBlobPtr(\*this); }

StrBlobPtr StrBlob::end()

{

auto ret = StrBlobPtr(\*this, data->size());

return ret;

}

#endif STRBLOB\_H