

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

**专业班级： 计算机科学与技术1601班**

**学 号： U201614531**

**姓 名： 刘本嵩**

**指导教师： 周时阳**

**报告日期： 2017年 12月 12日**

**计算机科学与技术学院**

**目 录**

[2 基于链序存储结构的线性表实现 2](#_Toc498625610)

[2.1 问题描述 2](#_Toc498625611)

[2.2 基本运算定义 2](#_Toc498625612)

[2.3 系统设计 3](#_Toc498625612)

[2.3.1 系统总体设计 3](#_Toc498625613)

[2.3.2 算法设计 4](#_Toc498625615)

[2.3 顺序表演示系统实现与测试 7](#_Toc498625616)

[2.3.1 系统实现 7](#_Toc498625617)

[2.3.2 算法测试 30](#_Toc498625618)

[2.3.3 界面测试 31](#_Toc498625618)

[2.4 实验小结 31](#_Toc498625619)

[参考文献 32](#_Toc498625620)

# 2基于链式存储结构的线性表实现

## 2.1 实验目的

通过实验达到⑴加深对线性表的概念、基本运算的理解；⑵熟练掌握线性表的逻辑结构与物理结构的关系；⑶物理结构采用单链表,熟练掌握线性表的基本运算的实现。

## 2.2 线性表基本运算定义

依据最小完备性和常用性相结合的原则，以函数形式定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体运算功能定义如下。

⑴初始化表：函数名称是InitaList(L)；初始条件是线性表L不存在已存在；操作结果是构造一个空的线性表。

⑵销毁表：函数名称是DestroyList(L)；初始条件是线性表L已存在；操作结果是销毁线性表L。

⑶清空表：函数名称是ClearList(L)；初始条件是线性表L已存在；操作结果是将L重置为空表。

⑷判定空表：函数名称是ListEmpty(L)；初始条件是线性表L已存在；操作结果是若L为空表则返回TRUE,否则返回FALSE。

⑸求表长：函数名称是ListLength(L)；初始条件是线性表已存在；操作结果是返回L中数据元素的个数。

⑹获得元素：函数名称是GetElem(L,i,e)；初始条件是线性表已存在，1≤i≤ListLength(L)；操作结果是用e返回L中第i个数据元素的值。

⑺查找元素：函数名称是LocateElem(L,e,compare())；初始条件是线性表已存在；操作结果是返回L中第1个与e满足关系compare（）关系的数据元素的位序，若这样的数据元素不存在，则返回值为0。

⑻获得前驱：函数名称是PriorElem(L,cur\_e,pre\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是第一个，则用pre\_e返回它的前驱，否则操作失败，pre\_e无定义。

⑼获得后继：函数名称是NextElem(L,cur\_e,next\_e)；初始条件是线性表L已存在；操作结果是若cur\_e是L的数据元素，且不是最后一个，则用next\_e返回它的后继，否则操作失败，next\_e无定义。

⑽插入元素：函数名称是ListInsert(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)+1；操作结果是在L的第i个位置之前插入新的数据元素e。

⑾删除元素：函数名称是ListDelete(L,i,e)；初始条件是线性表L已存在且非空，1≤i≤ListLength(L)；操作结果：删除L的第i个数据元素，用e返回其值。

⑿遍历表：函数名称是ListTraverse(L,visit())，初始条件是线性表L已存在；操作结果是依次对L的每个数据元素调用函数visit()。

**2.2 系统设计**

2.2.1 系统总体设计

本系统采用顺序表作为线性表的物理结构，实现线性表的基本运算。遵守C++14标准。

系统具有一个Terminal风格交互界面，称为rfaketerm，在general\_ui.hpp中实现。fake\_terminal::go会阻塞主线程，接收输入，简单parse之后通过callback函数进行处理。callback是一个更高级的parser，负责将输入翻译到C++函数地址并执行std::invoke，获取返回值，即时打印到stdout。在程序发生未定义行为时，会通过std::exception向自身发送*SIGABRT*信号，这有利于通用调试工具的应用。

在主程序中完成函数调用所需实参值的准备和函数执行结果的显示，并给出适当的操作提示显示。

系统定义一个reflection\_impl(作为本题要求的接口和容器库普遍承认的接口之间的wrapper)，其中含有一个核心链表对象Lab::list。如果需要实现对多个线性表的管理，只需使用std::deque<reflection\_impl>即可进行管理。

该演示系统提供的操作有：表的初始化、销毁、清空、判空，求表长、获取数据元素、查找数据元素、获得前驱、获得后继、插入数据元素、删除数据元素、表的遍历、表的选择。

在程序中实现消息处理和操作提示，包括数据的输入和输出，错误操作提示、程序的退出。

2.2.2 算法设计

（1）InitList(SqList \* L)

设计：分配存储空间，并初始化表长为0，表容量为LIST\_INIT\_SIZE。每次创建表时新建的表位序为最大。例如当前有6个表，执行此函数后创建的新表为表7

操作结果：构造一个空的线性表。

（2）DestroyList(SqList \* L)

设计：释放存储空间，每次操作当前线性表，销毁后当前线性表之后的线性表左移一个位序。例如当前操作表2，销毁表2后原表3左移成为表2，以此类推

操作结果：销毁线性表L。

（3）ClearList(SqList \* L)

设计：线性表L的长度赋值为0

操作结果：将L重置为空表。

（4）ListEmpty(SqList L)

设计：根据表长判断表是否为空

操作结果：若L为空表，则返回TRUE,否则返回FALSE。

（5）ListLength(SqList L)

设计：返回表长

操作结果：返回L中数据元素的个数。

（6）GetElem(SqList L, int i, ElemType \* e)

设计：根据位序找到第i个元素的地址并将其值赋值给指针e指向的元素

操作结果：用指针e指向的元素返回L中第i个数据元素的值。

（7）LocateElem(SqList L, ElemType e)

设计：遍历线性表找到第一个和元素e的相等的元素

操作结果：返回L中第1个与e相等的的数据元素的位序，若这样的数据元素不存在，则返回值为0。

（8）PriorElem(SqList L, ElemType cur, ElemType \* pre\_e)

设计：遍历线性表找到第一个和元素cur的相等的元素，如果其有前驱，用pre\_e返回，函数返回TRUE；否则函数返回FALSE，pre\_e无意义

操作结果：若cur是L的数据元素，且不是第一个，则用pre\_e返回它的 前驱，否则操作失败，pre\_e无定义。

（9）NextElem（L，cur\_e，&next\_e）

设计：遍历线性表找到第一个和元素cur的相等的元素，如果其有后继，用next\_e返回，函数返回TRUE；否则函数返回FALSE，next\_e无意义

操作结果：若cur是L的数据元素，且不是最后一个，则用next\_e返回它 的后继，否则操作失败，next\_e无定义。

（10）ListInsert(SqList \* L, int i, ElemType e)

设计：如果线性表已满，重新分配存储空间。将线性表指针L指向的线性表第i个元素之后的元素都右移一个位序，之后将e插入第i个位序

操作结果：在L的第i个位置之前插入新的数据元素e，L的长度加1

（11）ListDelete(SqList \* L, int i, ElemType \* e)

设计：将第i个位序的值赋给指针e指向的变量，之后第i个位序之后的元素全部左移一个位序

操作结果：删除L的第i个数据元素，用e返回其值，L的长度减1.

（12）ListTraverse(SqList L)

设计：遍历并输出表L中的每个元素值，返回表长

操作结果：依次输出表L中的每个变量的值

**2.3 顺序表演示系统实现与测试**

2.3.1 系统实现

编程环境：Linux x86\_64 ARCH gcc 8.0.0 cmake 3.10.0 GNU Make 4.2.1 GNU ld 2.29.1 GNU ar 2.29.1 kernel 4.13.12-1-ARCH 其他环境设定均在CMakeLists.txt进行了说明。

为Windows进行了交叉编译，使用cmake 3.10.0 mingw-gcc 6.3.1 nmake Windows 10 1709 (summer creator update) 静态编译使用mingw-gcc 6.3.1提供的libstdc++。

Windows版本缺失部分功能(界面美化)。

下面是src目录下的hpp/cc/CMakeLists.txt文件清单：依赖于rlib，此库被打包进源码目录，库内容均为原创。其中包含了测试所用代码。

//////////// FileName := CMakeLists.txt

cmake\_minimum\_required(VERSION 3.5)

project(hust\_shit)

set(CMAKE\_CXX\_STANDARD 14)

set(CMAKE\_C\_STANDARD 11)

set(CMAKE\_VERBOSE\_MAKEFILE ON)

set(CMAKE\_CXX\_FLAGS\_DEBUG "-g -DMALLOC\_CHECK\_=2")

set(CMAKE\_CXX\_FLAGS\_RELEASE "-O3")

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -msse4.2")

include\_directories("/usr/include")

include\_directories("/usr/local/include")

include\_directories(".")

set(BUILD\_SRC main.cc reflected\_impl.hpp lab\_list.hpp labafx.hpp general\_ui.hpp parser.hpp)

add\_executable(exp2 ${BUILD\_SRC})

//////////// FileName := general\_ui.hpp

#ifndef HUST\_SHIT\_GENERAL\_UI\_HPP\_

#define HUST\_SHIT\_GENERAL\_UI\_HPP\_

#include <functional>

#include <string>

#include <iostream>

#include <list>

#include <rlib/stdio.hpp>

#include <rlib/terminal.hpp>

#include <rlib/string/string.hpp>

#include <rlib/sys/os.hpp>

using namespace rlib::terminal;

using rlib::splitString;

class fake\_terminal {

public:

using callback\_t = std::function<void (std::list<std::string>)>;

[[noreturn]] static void go(const callback\_t &callback) {

while(true) {

prompt();

callback(splitString(rlib::io::scanln()));

}

}

private:

static void prompt() {

if constexpr(rlib::OSInfo::os == rlib::OSInfo::os\_t::WINDOWS) {

rlib::io::print(color\_t::green, "rfaketerm 0.0", clear, font\_t::bold, "~", clear);

}

else {

rlib::io::print(color\_t::green, "rfaketerm 0.0", clear, font\_t::bold, "~", clear);

}

}

};

#endif

//////////// FileName := labafx.hpp

#ifndef LAB\_AFX\_HPP\_

#define LAB\_AFX\_HPP\_

#include <cstddef>

#include <nmmintrin.h>

// typedef struct lab\_\_pair\_st Pair;

namespace LabUtils

{

template<typename ForwardIterator>

size\_t distance(ForwardIterator a, ForwardIterator b)

{

size\_t dist = 0;

for (; true; ++dist, ++a)

{

if (a == b) break;

}

return dist;

}

template<typename ForwardIterator>

ForwardIterator advance(ForwardIterator a, size\_t n)

{

for (size\_t cter = 0; cter < n; ++cter)

{

++a;

}

return a;

}

}

namespace Lab

{

////////////////////////////////////////////////

///////// SECTION TO IGNORE BEGINS /////////////

////////////////////////////////////////////////

constexpr unsigned INIT\_HASH\_VALUE = 0x01234567;

unsigned int naive\_hash(const void \*data, int size)

{

// work only for Plain Old Data (POD)

// stupid but efficient for random data

// unsafe for attack, but security is NOT required

auto crc = INIT\_HASH\_VALUE;

unsigned char \*data\_ = (unsigned char \*) data;

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

{

crc = \_mm\_crc32\_u8(crc, data\_[i]);

}

return crc;

}

template<typename T>

unsigned int hast\_f(const T &s)

{

// optimised for base type

// faster than pure naive\_hash

return naive\_hash(&s, sizeof(s));

}

// the following is for speedups

template<>

unsigned int hast\_f(const unsigned long long &s)

{

return \_mm\_crc32\_u64(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const long long &s)

{

return \_mm\_crc32\_u64(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const double &s)

{

union

{

double f;

unsigned long long i;

} u;

u.f = s;

return \_mm\_crc32\_u64(INIT\_HASH\_VALUE, u.i);

}

template<>

unsigned int hast\_f(const float &s)

{

union

{

float f;

unsigned int i;

} u;

u.f = s;

return \_mm\_crc32\_u32(INIT\_HASH\_VALUE, u.i);

}

template<>

unsigned int hast\_f(const unsigned &s)

{

return \_mm\_crc32\_u32(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const int &s)

{

return \_mm\_crc32\_u32(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const unsigned short &s)

{

return \_mm\_crc32\_u16(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const short &s)

{

return \_mm\_crc32\_u16(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const signed char &s)

{

return \_mm\_crc32\_u8(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const unsigned char &s)

{

return \_mm\_crc32\_u8(INIT\_HASH\_VALUE, s);

}

template<>

unsigned int hast\_f(const char &s)

{

return \_mm\_crc32\_u8(INIT\_HASH\_VALUE, s);

}

////////////////////////////////////////////////

////////// SECTION TO IGNORE ENDS //////////////

////////////////////////////////////////////////

// work only for Plain Old Data (POD)

// otherwise correctness is not guaranteed

// stupid but efficient for random data

// unsafe for attack, since security is NOT required

unsigned int naive\_hash(const void \*data, int size);

// Lab::hash<T> simulates std::hash<T>

// usage: hash\_result = hash<T>()(item\_to\_hash);

template<typename T>

class hash

{

public:

unsigned int operator()(const T &s) { return hash\_f(s); }

};

template<typename T1, typename T2>

struct pair

{

T1 first;

T2 second;

};

// USE Lab::make\_pair LIKE std::make\_pair

template<typename T1, typename T2>

pair<T1, T1> make\_pair(const T1 &first, const T2 &second)

{

return pair<T1, T2>{first, second};

};

// usage:

// auto comp = less<T>();

// comp(a, b) == a < b;

// OR

// less<T>()(a, b) == a < b;

template<typename T>

class less

{

public:

bool operator()(const T &a, const T &b) { return a < b; }

};

}

#endif

//////////// FileName := lab\_list.hpp

#ifndef LAB\_LIST\_HPP\_

#define LAB\_LIST\_HPP\_

#include <cstddef>

#include <iterator>

namespace Lab

{

template<typename Type>

class list

{

private:

struct node

{

node() = default;

node(const Type &data, node \*pre, node \*next) : data(data), pre(pre), next(next) {}

Type data = 0;

node \*pre = nullptr;

node \*next = nullptr;

};

public:

class iterator :

public std::iterator<std::bidirectional\_iterator\_tag, Type, Type, const Type \*, Type &>

{

public:

iterator(node \*currentTmp) { current = currentTmp; }

Type &operator\*() { return current->data; }

const Type &operator\*() const { return current->data; }

iterator &operator++()

{

current = current->next;

return \*this;

}

iterator &operator--()

{

current = current->pre;

return \*this;

}

iterator &operator++(int)

{

auto restore = \*this;

current = current->next;

return restore;

}

iterator &operator--(int)

{

auto restore = \*this;

current = current->pre;

return restore;

}

bool operator!=(const iterator &another) const

{

return another.current != current;

}

bool operator==(const iterator &another) const

{

return another.current == current;

}

node \*current;

};

void push\_back(const Type &elem);

void push\_front(const Type &elem);

iterator begin();

iterator end();

size\_t size() const;

void pop\_front();

void pop\_back();

void insert(iterator iter, const Type &elem);

void erase(iterator iter);

void clear();

~list();

private:

node \*beg = nullptr;

node \*en = nullptr;

size\_t length = 0;

};

template<typename Type>

typename list<Type>::iterator list<Type>::begin()

{

return iterator(beg);

}

template<typename Type>

typename list<Type>::iterator list<Type>::end()

{

return iterator(en);

}

template<typename Type>

void list<Type>::push\_back(const Type &elem)

{

node \*newNode = new node;

newNode->data = elem;

newNode->pre = en;

newNode->next = nullptr;

if (en)

{

en->next = newNode;

}

en = newNode;

if (!length)

{

beg = newNode;

}

length++;

}

template<typename Type>

void list<Type>::push\_front(const Type &elem)

{

node \*newNode = new node;

newNode->data = elem;

newNode->next = beg;

newNode->pre = nullptr;

if (beg)

{

beg->pre = newNode;

}

beg = newNode;

if (!length)

en = newNode;

length++;

}

template<typename Type>

size\_t list<Type>::size() const { return length; }

template<typename Type>

void list<Type>::pop\_front()

{

node \*newNode = new node;

newNode = beg;

beg = beg->next;

if (beg)

beg->pre = nullptr;

length--;

delete newNode;

}

template<typename Type>

void list<Type>::pop\_back()

{

node \*newNode = new node;

length--;

newNode = en;

en = en->pre;

if (en)

en->next = nullptr;

delete newNode;

}

template<typename Type>

void list<Type>::insert(iterator iter, const Type &elem)

{

if(iter == this->end()) return this->push\_back(elem);

if(iter == this->begin()) return this->push\_front(elem);

node \*newNode = new node{elem, iter.current->pre, iter.current};

iter.current->pre->next = newNode;

iter.current->pre = newNode;

length++;

}

template<typename Type>

void list<Type>::erase(iterator iter)

{

// node \*newNode = iter.current->pre;

if (iter.current->pre)

iter.current->pre->next = iter.current->next;

if (iter.current->next)

iter.current->next->pre = iter.current->pre;

delete iter.current;

length--;

}

template<typename Type>

void list<Type>::clear()

{

while (beg != en)

{

node \*newNode = beg->next;

delete beg;

beg = newNode;

length--;

}

delete beg;

beg = en = nullptr;

length = 0;

}

template<typename Type>

list<Type>::~list()

{

this->clear();

}

} // namespace Lab

#endif

//////////// FileName := list\_test.cc

/\*\*

\* By recolic, Nov 10.

\*/

#include <chrono>

#include <iostream>

#include <random>

#include <functional>

#include "test\_utils.hpp"

using println = rlib::io::println;

std::default\_random\_engine rand\_eng(810);

std::uniform\_real\_distribution<double> distribution(0, 100);

double m\_rand() {return distribution(rand\_eng);}

template <class operation\_t, typename... args\_t>

void timed\_func(const std::string &info, std::function<operation\_t> f, args\_t... args)

{

println(info, "launched.");

auto begin = std::chrono::high\_resolution\_clock::now();

f(args ...);

auto end = std::chrono::high\_resolution\_clock::now();

println(info, "used", std::chrono::duration<double>(end - begin).count(), "s");

}

template <class operation\_t, typename... args\_t>

void repeat(size\_t count, std::function<operation\_t> f, args\_t... args)

{

for(size\_t cter = 0; cter < count; ++cter)

f(args ...);

}

int main()

{

using data\_t = double;

Lab::list<data\_t> lsa;

std::list<data\_t> lsb;

using op\_arg1\_t = Lab::list<data\_t> &;

using op\_arg2\_t = std::list<data\_t> &;

#define op\_args\_t op\_arg1\_t, op\_arg2\_t

using operation\_t = void(op\_args\_t);

auto co\_push\_back = [](auto &bufa, auto &bufb){

auto val = m\_rand();

bufa.push\_back(val);

bufb.push\_back(val);

};

auto co\_push\_front = [](auto &bufa, auto &bufb){

auto val = m\_rand();

bufa.push\_front(val);

bufb.push\_front(val);

};

auto co\_pop\_front = [](auto &bufa, auto &bufb){

bufa.pop\_front();

bufb.pop\_front();

};

auto co\_pop\_back = [](auto &bufa, auto &bufb){

bufa.pop\_back();

bufb.pop\_back();

};

auto co\_erase = [](auto &bufa, auto &bufb){

bufa.erase(++bufa.begin());

bufb.erase(++bufb.begin());

};

auto co\_clear = [](auto &bufa, auto &bufb){

bufa.clear();

bufb.clear();

};

using namespace std::placeholders;

#define TEST(count, operation, desc) LIST\_ASSERT\_EQUIVALENCE(lsa, lsb, std::function<operation\_t>( \

std::bind(timed\_func<operation\_t, op\_args\_t>, desc, \

std::function<operation\_t>(std::bind(repeat<operation\_t, op\_args\_t>, count, operation, \_1, \_2)), \

\_1, \_2)))

TEST(1000, co\_push\_back, "push1");

TEST(10000000, co\_push\_back, "push2");

TEST(9999000, co\_pop\_back, "pop1");

TEST(54320, co\_push\_back, "push3");

TEST(123, co\_pop\_back, "pop2");

TEST(1, co\_erase, "erase1");

TEST(66, co\_push\_back, "push4");

TEST(543, co\_erase, "erase2");

TEST(2, co\_clear, "clear1");

TEST(3456, co\_push\_back, "push5");

println("s/back/front/g and retest...");

TEST(1000, co\_push\_front, "push1");

TEST(10000000, co\_push\_front, "push2");

TEST(9999000, co\_pop\_front, "pop1");

TEST(54320, co\_push\_front, "push3");

TEST(123, co\_pop\_front, "pop2");

TEST(1, co\_erase, "erase1");

TEST(66, co\_push\_front, "push4");

TEST(543, co\_erase, "erase2");

TEST(2, co\_clear, "clear1");

TEST(3456, co\_push\_front, "push5");

println("All tests done.");

return 0;

}

//////////// FileName := main.cc

#include <general\_ui.hpp>

#include <parser.hpp>

reflected\_impl impl;

int main() {

fake\_terminal::go(parser::parse);

}

//////////// FileName := parser.hpp

#ifndef \_HUST\_SHIT\_PARSER\_HPP

#define \_HUST\_SHIT\_PARSER\_HPP 1

#include <reflected\_impl.hpp>

#include <list>

#include <string>

#include <iomanip>

#include <rlib/stdio.hpp>

#include <rlib/terminal.hpp>

using namespace rlib::terminal;

class parser {

private:

static std::string getArg(const std::list<std::string> &ls, size\_t n) {

auto iter = ls.cbegin();

for(size\_t cter = 0; cter < n; ++cter) {

++iter;

}

return std::move(\*iter);

}

static void help\_msg() {

std::string msg = R"\_STR\_(

rfaketerm 0.0 shit specially edition

Usage: <Command> [args ...]

Command List:

help : Show this message.

exit : exit politely.

InitList

DestroyList

ClearList

ListEmpty

ListLength

GetElem <size\_t positionPlusOne>

LocateElem <data\_t elemValue>

PriorElem <data\_t elemValue>

NextElem <data\_t elemValue>

ListInsert <size\_t positionPlusOne> <data\_t elemValue>

ListDelete <size\_t positionPlusOne>

ListTraverse

)\_STR\_";

rlib::io::println(msg);

}

public:

static void parse(const std::list<std::string> &to\_parse) {

if(to\_parse.empty())

return;

rlib::io::print(std::boolalpha);

#define IFCMD(str) if(\*to\_parse.begin() == str)

#define WANT\_ARG(n) if(to\_parse.size() != n+1) {rlib::io::println(color\_t::red, font\_t::bold, "Error:", clear, color\_t::lightgray, n, "arguments wanted but", to\_parse.size()-1, "provided.", clear); return;}

#define SIZE\_ARG(n) std::stoul(getArg(to\_parse, n))

#define DATA\_ARG(n) std::stoi(getArg(to\_parse, n))

#define HAVE\_RETURN\_VALUE auto ret =

#define PRINT\_RETURN\_VALUE rlib::io::println(ret);

IFCMD("InitList") {

WANT\_ARG(0)

impl.InitList();

}

IFCMD("DestroyList") {

WANT\_ARG(0)

impl.DestroyList();

}

IFCMD("ClearList") {

WANT\_ARG(0)

impl.ClearList();

}

IFCMD("ListEmpty") {

WANT\_ARG(0)

HAVE\_RETURN\_VALUE

impl.ListEmpty();

PRINT\_RETURN\_VALUE

}

IFCMD("ListLength") {

WANT\_ARG(0)

HAVE\_RETURN\_VALUE

impl.ListLength();

PRINT\_RETURN\_VALUE

}

IFCMD("GetElem") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.GetElem(SIZE\_ARG(1));

PRINT\_RETURN\_VALUE

}

IFCMD("LocateElem") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.LocateElem(DATA\_ARG(1));

PRINT\_RETURN\_VALUE

}

IFCMD("PriorElem") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.PriorElem(DATA\_ARG(1));

PRINT\_RETURN\_VALUE

}

IFCMD("NextElem") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.NextElem(DATA\_ARG(1));

PRINT\_RETURN\_VALUE

}

IFCMD("ListInsert") {

WANT\_ARG(2)

impl.ListInsert(SIZE\_ARG(1), DATA\_ARG(2));

}

IFCMD("ListDelete") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.ListDelete(SIZE\_ARG(1));

PRINT\_RETURN\_VALUE

}

IFCMD("ListTraverse") {

WANT\_ARG(0)

impl.ListTraverse();

}

// Shits done.

IFCMD("exit") {

rlib::io::println("bye~");

::std::exit(0);

}

IFCMD("help") {

help\_msg();

}

//impl.debug();

}

};

#endif //\_HUST\_SHIT\_PARSER\_HPP

//////////// FileName := reflected\_impl.hpp

#ifndef HUST\_SHIT\_REFLECTED\_IMPL\_HPP\_

#define HUST\_SHIT\_REFLECTED\_IMPL\_HPP\_

/\*

\* You should NEVER use this code in ANY consequence,

\* as these code is just to make hust happy.

\*/

#include <utility>

#include <functional>

#include <algorithm>

#include "lab\_list.hpp"

#include "labafx.hpp"

#include <rlib/stdio.hpp>

class reflected\_impl {

public:

using data\_t = int;

using BooleanAsserter = std::function<bool(const data\_t &)>;

using OperationVisiter = std::function<void(const data\_t &)>;

void InitList() const {}

void DestroyList() {container.clear();}

void ClearList() {container.clear();}

bool ListEmpty() const {return container.size() == 0;}

size\_t ListLength() const {return container.size();}

data\_t GetElem(size\_t \_shit\_IndexPlusOne) {

auto index = \_shit\_IndexPlusOne - 1;

auto iter = container.begin();

for(size\_t cter = 0; cter < index; ++cter) {

++iter;

}

return std::move(\*iter);

}

size\_t \_LocateElem(const BooleanAsserter &comparer) {

auto iter = std::find\_if(container.begin(), container.end(), comparer);

if(iter == container.end()) {

return 0;

}

return LabUtils::distance(container.begin(), iter);

}

size\_t LocateElem(data\_t val) {

auto comparer = BooleanAsserter([v=val](const data\_t &dat){

return dat == v;

});

return \_LocateElem(comparer);

}

data\_t PriorElem(data\_t tofind) {

auto pos = std::find(container.begin(), container.end(), tofind);

if(pos == container.end() || pos == container.begin()) {

throw std::runtime\_error("ElemError: You told me that it's undefined, so I do it.");

}

return \*(--pos);

}

data\_t NextElem(data\_t tofind) {

auto pos = std::find(container.begin(), container.end(), tofind);

if(pos == container.end() || pos == --container.end()) {

throw std::runtime\_error("ElemError: You told me that it's undefined, so I do it.");

}

return \*(++pos);

}

void ListInsert(size\_t \_shit\_IndexPlusOne, data\_t elem) {

auto index = \_shit\_IndexPlusOne - 1;

auto iter = LabUtils::advance(container.begin(), index);

container.insert(iter, elem);

}

data\_t ListDelete(size\_t \_shit\_IndexPlusOne) {

auto index = \_shit\_IndexPlusOne - 1;

auto iter = LabUtils::advance(container.begin(), index);

auto to\_return = \*iter;

container.erase(iter);

return std::move(to\_return);

}

void \_ListTraverse(const OperationVisiter &visiter) {

std::for\_each(container.begin(), container.end(), visiter);

}

void ListTraverse() {

\_ListTraverse(OperationVisiter([](const auto &val){rlib::io::print(val, " ");}));

rlib::io::println("");

}

void debug() {

rlib::io::println\_iter(container);

rlib::io::println(container.size());

}

private:

Lab::list<data\_t> container;

};

extern reflected\_impl impl;

#endif

//////////// FileName := rlib

cat: rlib: 是一个目录

//////////// FileName := test\_utils.hpp

#include <cstdlib>

#include <rlib/stdio.hpp>

#include <rlib/traits.hpp>

#define dynamic\_assert(cond, message) do { \

if(!cond) { \

rlib::io::println("dynamic assertion failed:", message); \

std::exit(2); \

} \

} while(false)

// -- operation must be a templated callable object, usually templated lambda.

// NEW: operation must fuck two buf at same time.

#define ASSERT\_EQUIVALENCE(bufA, bufB, operation, equal\_checker) \

do { \

static\_assert(std::is\_same<rlib::is\_callable<decltype(equal\_checker<double>)>::type, \

std::true\_type>::value, \

"equal\_checker is not callable"); \

dynamic\_assert(equal\_checker(bufA, bufB), "given buf is not equal."); \

operation(bufA, bufB); \

dynamic\_assert(equal\_checker(bufA, bufB), "operation failed."); \

} while(false)

/\*

//vector

#include "lab\_vector.hpp"

#include <vector>

template<typename data\_t>

bool vector\_equal(const Lab::vector<data\_t> &vcta, const std::vector<data\_t> &vctb)

{

if(vcta.size() != vctb.size()) return false;

Lab::vector<data\_t> &fake\_vcta = const\_cast<Lab::vector<data\_t> &>(vcta);

for(auto ia = fake\_vcta.begin(), ib = vctb.begin();

ia != fake\_vcta.end() && ib != vctb.end();

++ia, ++ib)

{

if(\*ia != \*ib) return false;

}

return true;

}

#define VECTOR\_ASSERT\_EQUIVALENCE(bufA, bufB, operation) ASSERT\_EQUIVALENCE(bufA, bufB, operation, vector\_equal)

\*/

//list

#include "lab\_list.hpp"

#include <list>

template<typename data\_t>

bool list\_equal(const Lab::list<data\_t> &bufa, const std::list<data\_t> &bufb)

{

if(bufa.size() != bufb.size()) return false;

Lab::list<data\_t> &fake\_bufa = const\_cast<Lab::list<data\_t> &>(bufa);

for(auto ia = fake\_bufa.begin(), ib = bufb.begin();

ia != fake\_bufa.end() && ib != bufb.end();

++ia, ++ib)

{

if(\*ia != \*ib) return false;

}

return true;

}

#define LIST\_ASSERT\_EQUIVALENCE(bufA, bufB, operation) ASSERT\_EQUIVALENCE(bufA, bufB, operation, list\_equal)

/\*

//set

#include "lab\_set.hpp"

#include <set>

template<typename data\_t>

bool set\_equal(const Lab::set<data\_t> &bufa, const std::set<data\_t> &bufb)

{

if(bufa.size() != bufb.size()) return false;

Lab::set<data\_t> &fake\_bufa = const\_cast<Lab::set<data\_t> &>(bufa);

for(auto ia = fake\_bufa.begin(), ib = bufb.begin();

ia != fake\_bufa.end() && ib != bufb.end();

++ia, ++ib)

{

if(\*ia != \*ib) return false;

}

return true;

}

#define SET\_ASSERT\_EQUIVALENCE(bufA, bufB, operation) ASSERT\_EQUIVALENCE(bufA, bufB, operation, set\_equal)

//priority\_queue

#include "lab\_priority\_queue.hpp"

#include <queue>

template<typename data\_t>

bool priority\_queue\_equal(const Lab::priority\_queue<data\_t> &bufa, const std::priority\_queue<data\_t> &bufb)

{

return true;

}

#define PRIORITY\_QUEUE\_ASSERT\_EQUIVALENCE(bufA, bufB, operation) ASSERT\_EQUIVALENCE(bufA, bufB, operation, priority\_queue\_equal)

template<typename data\_t>

bool priority\_queue\_destroy\_and\_check(Lab::priority\_queue<data\_t> &bufa, std::priority\_queue<data\_t> &bufb)

{

if(bufa.size() != bufb.size()) return false;

while(bufb.size())

{

if(bufa.top() != bufb.top()) return false;

bufa.pop();

bufb.pop();

}

return true;

}

//unordered\_map

#include "lab\_unordered\_map.hpp"

#include <unordered\_map>

template<typename key\_t, typename data\_t>

bool unordered\_map\_equal(const Lab::unordered\_map<key\_t, data\_t> &bufa, const std::unordered\_map<key\_t, data\_t> &bufb)

{

if(bufa.size() != bufb.size()) return false;

Lab::unordered\_map<key\_t, data\_t> &fake\_bufa = const\_cast<Lab::unordered\_map<key\_t, data\_t> &>(bufa);

for(auto ia = fake\_bufa.begin(), ib = bufb.begin();

ia != fake\_bufa.end() && ib != bufb.end();

++ia, ++ib)

{

if(\*ia != \*ib) return false;

if(fake\_bufa.find((\*ib).first) != ia) return false;

}

return true;

}

template<typename key\_data\_t>

bool \_unordered\_map\_equal(const Lab::unordered\_map<key\_data\_t, key\_data\_t> &bufa, const std::unordered\_map<key\_data\_t, key\_data\_t> &bufb)

{

return unordered\_map\_equal(bufa, bufb);

}

#define UNORDERED\_MAP\_ASSERT\_EQUIVALENCE(bufA, bufB, operation) ASSERT\_EQUIVALENCE(bufA, bufB, operation, \_unordered\_map\_equal)

\*/

2.3.2 算法测试

直接通过测试程序对算法部分可靠性进行测试。

插入/删除测试各20000000次，其他测试分必要性共几千次，测试结果完全正确(和std::list进行严格的表现比较)。

push1 launched.

push1 used 0.000312113 s

push2 launched.

push2 used 2.07433 s

pop1 launched.

pop1 used 0.89098 s

push3 launched.

push3 used 0.00973691 s

pop2 launched.

pop2 used 2.5136e-05 s

erase1 launched.

erase1 used 3.297e-06 s

push4 launched.

push4 used 4.5379e-05 s

erase2 launched.

erase2 used 0.00011576 s

clear1 launched.

clear1 used 0.00816342 s

push5 launched.

push5 used 0.0015594 s

s/back/front/g and retest...

push1 launched.

push1 used 0.000389672 s

push2 launched.

push2 used 1.93912 s

pop1 launched.

pop1 used 0.983706 s

push3 launched.

push3 used 0.01539 s

pop2 launched.

pop2 used 7.5143e-05 s

erase1 launched.

erase1 used 3.21e-06 s

push4 launched.

push4 used 0.000109478 s

erase2 launched.

erase2 used 0.000324628 s

clear1 launched.

clear1 used 0.0244477 s

push5 launched.

push5 used 0.000904992 s

All tests done.

2.3.3 界面测试

简单的测试表明，界面的正确性没有问题。

**2.4 实验小结**

本次实验加深了对线性表的概念、基本运算的理解，掌握了线性表的基本预算的实现。熟练了线性表的逻辑结构和物理结构之间的关系。今后的学习过程中应当多从数据结构的角度分析如何进行数据的处理、存储以方便问题的解决，并要勤加练习达到熟能生巧的地步。

参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

[3] ISO/IEC 14882:2014(E)

[4] ISO/IEC 14882:2011

指导教师评定意见

一、对实验报告的评语

|  |
| --- |
|  |

二、对实验报告评分

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 评分项目  (分值) | 程序内容  (36.8分) | 程序规范  (9.2分) | 报告内容  (36.8分) | 报告规范  (9.2分) | 考勤  （8分） | 逾期扣分 | 合 计  (100分) |
| 得分 |  |  |  |  |  |  |  |