

**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 26日**

**计算机科学与技术学院**

**目 录**

[3 基于链序存储结构的线性表实现 2](#_Toc498625610)

[23.1 问题描述 2](#_Toc498625611)

[3.2 基本运算定义 2](#_Toc498625612)

[3.3 系统设计 3](#_Toc498625612)

[3.3.1 系统总体设计 3](#_Toc498625613)

[3.3.2 算法设计 4](#_Toc498625615)

[3.3 顺序表演示系统实现与测试 7](#_Toc498625616)

[3.3.1 系统实现 7](#_Toc498625617)

[3.3.2 算法测试 30](#_Toc498625618)

[3.3.3 界面测试 31](#_Toc498625618)

[3.4 实验小结 31](#_Toc498625619)

[参考文献 32](#_Toc498625620)

# 3基于二叉链表的二叉树实现

**3.1 实验目的**

通过实验达到⑴加深对二叉树的概念、基本运算的理解；⑵熟练掌握二叉树的逻辑结构与物理结构的关系；⑶以二叉链表作为物理结构，熟练掌握二叉树基本运算的实现。

**3.2 系统设计**

3.2.1 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是一个由ccgen.py生成代码的parser(即reflection,C++20标准库提供了原生功能)，负责将输入翻译到下一层即relected\_impl。它将请求进一步解释，并与后端数据结构进行交互，获取返回值，被rfaketerm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。为了美观，rfaketerm默认情况下会把所有异常抓下并打印错误信息到stdout。

User Manual在 rfaketerm中使用help命令即可获得。为了便于GUI下的使用，rfaketerm启动时会自动模拟执行help命令。

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)，其负责管理数据结构对象hust\_xxxx::unordered\_btree。为了实现对多个线性表的管理，只需使用std::vector<btree>即可。

该演示系统提供的操作有：初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树和求二叉树深度等20种基本运算和Select, List等用于在多个树间切换的操作，详见help。

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。

3.2.2 算法设计

依据最小完备性和常用性相结合的原则，以函数形式定义了二叉树的初始化二叉树、销毁二叉树、创建二叉树、清空二叉树、判定空二叉树和求二叉树深度等20种基本运算，具体运算功能定义如下。

⑴初始化二叉树：函数名称是InitBiTree(T)；初始条件是二叉树T不存在；操作结果是构造空二叉树T。

⑵销毁二叉树：树函数名称是DestroyBiTree(T)；初始条件是二叉树T已存在；操作结果是销毁二叉树T。

⑶创建二叉树：函数名称是CreateBiTree(T,definition)；初始条件是definition 给出二叉树T的定义；操作结果是按definition构造二叉树T。

⑷清空二叉树：函数名称是ClearBiTree (T)；初始条件是二叉树T存在； 操作结果是将二叉树T清空。

⑸判定空二叉树：函数名称是BiTreeEmpty(T)；初始条件是二叉树T存在；操作结果是若T为空二叉树则返回TRUE，否则返回FALSE。

⑹求二叉树深度：函数名称是BiTreeDepth(T)；初始条件是二叉树T存在；操作结果是返回T的深度。

⑺获得根结点：函数名称是Root(T)；初始条件是二叉树T已存在；操作结果是返回T的根。

⑻获得结点：函数名称是Value(T,e)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是返回e的值。

⑼结点赋值：函数名称是Assign(T,&e,value)；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是结点e赋值为value。

⑽获得双亲结点：函数名称是Parent(T,e) ；初始条件是二叉树T已存在，e是T中的某个结点；操作结果是若e是T的非根结点，则返回它的双亲结点指针，否则返回NULL。

⑾获得左孩子结点：函数名称是LeftChild(T,e)；初始条件是二叉树T存在，e是T中某个节点；操作结果是返回e的左孩子结点指针。若e无左孩子，则返回NULL。

⑿获得右孩子结点：函数名称是RightChild(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右孩子结点指针。若e无右孩子，则返回NULL。

⒀获得左兄弟结点：函数名称是LeftSibling(T,e)；初始条件是二叉树T存在，e是T中某个结点；操作结果是返回e的左兄弟结点指针。若e是T的左孩子或者无左兄弟，则返回NULL。

⒁获得右兄弟结点：函数名称是RightSibling(T,e)；初始条件是二叉树T已存在，e是T中某个结点；操作结果是返回e的右兄弟结点指针。若e是T的右孩子或者无有兄弟，则返回NULL。

⒂插入子树：函数名称是InsertChild(T,p,LR,c)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1，,非空二叉树c与T不相交且右子树为空；操作结果是根据LR为0或者1，插入c为T中p所指结点的左或右子树，p 所指结点的原有左子树或右子树则为c的右子树

⒃删除子树：函数名称是DeleteChild(T.p.LR)；初始条件是二叉树T存在，p指向T中的某个结点，LR为0或1。 操作结果是根据LR为0或者1，删除c为T中p所指结点的左或右子树。

⒄前序遍历：函数名称是PreOrderTraverse(T,Visit())；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果：先序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒅中序遍历：函数名称是InOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是中序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒆后序遍历：函数名称是PostOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是后序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

⒇按层遍历：函数名称是LevelOrderTraverse(T,Visit))；初始条件是二叉树T存在，Visit是对结点操作的应用函数；操作结果是层序遍历t，对每个结点调用函数Visit一次且一次，一旦调用失败，则操作失败。

**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 3.10.1 GNU Make 4.2.1 GNU ld 2.29.1 GNU ar 2.29.1 kernel 4.14.5-1-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)。

使用了gc库。

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 := btree.hpp

#ifndef HUST\_BTREE\_HPP\_

#define HUST\_BTREE\_HPP\_

//#include <gc.h> //You cannot compile it and it doesn't matter.

#include <rlib/require/cxx11>

#include <stdexcept>

#include <exception>

#include <functional>

#include <rlib/string/string.hpp>

#include <rlib/stdio.hpp>

namespace hust\_xxxx {

enum class foreach\_rule {LEFT\_MIDDLE\_RIGHT, LEFT\_RIGHT\_MIDDLE, MIDDLE\_LEFT\_RIGHT};

template<typename data\_t>

class [[deprecated/\*, "fatal memory bug, invalid algo, extremely bad design."\*/]] unordered\_btree {

struct node {

data\_t payload;

node \*left = nullptr;

node \*right = nullptr;

node \*parent = nullptr;

size\_t depth = 0; //Root

node() = delete;

node(const data\_t &payload, node \*parent) : payload(payload), parent(parent), depth(parent?parent->depth+1:0) {}

void for\_each(foreach\_rule rule, std::function<void(node &)> func) {

if(rule == foreach\_rule::MIDDLE\_LEFT\_RIGHT) func(\*this);

if(left) left->for\_each(rule, func);

if(rule == foreach\_rule::LEFT\_MIDDLE\_RIGHT) func(\*this);

if(right) right->for\_each(rule, func);

if(rule == foreach\_rule::LEFT\_RIGHT\_MIDDLE) func(\*this);

}

};

public:

using nlang = std::string;

unordered\_btree() {}

bool empty() const {

return root == nullptr;

}

bool clear() {

root = nullptr;

}

size\_t depth() {

size\_t max\_depth = 0;

this->for\_each([&max\_depth](node &n){

max\_depth = n.depth>max\_depth ? n.depth : max\_depth;

});

return max\_depth;

}

nlang \_root() {

return std::move(nlang(""));

}

data\_t get(const nlang &pos) {

auto n = nlang\_translate(pos);

if(!n)

throw std::runtime\_error("Trying to access an empty node.");

return std::move(n->payload);

}

void set(const nlang &pos, const data\_t &payload) {

auto iter = nlang\_translate(pos);

if(iter)

iter->payload = payload;

else

nlang\_translate(pos, true, payload);

}

nlang parent(nlang pos) {

rlib::replaceSubString(pos, " ", "");

return pos.empty() ? pos : pos.substr(0, pos.size()-1);

}

nlang lchild(const nlang &pos) {

return pos + 'L';

}

nlang rchild(const nlang &pos) {

return pos + 'R';

}

void for\_each(std::function<void(node &)> func, typename hust\_xxxx::foreach\_rule rule = foreach\_rule::LEFT\_MIDDLE\_RIGHT) {

if(root) root->for\_each(rule, func);

}

void level\_for\_each(std::function<void(node &)> func) {

size\_t curr\_depth = 0;

while(true) {

bool must\_break = true;

this->for\_each([&, \_curr\_depth=curr\_depth](node &n){

if(n.depth == \_curr\_depth) {

func(n);

must\_break = false;

}

});

if(must\_break) break;

++curr\_depth;

}

}

static void printer(node &n) {rlib::print(n.payload, "");}

void merge(unordered\_btree &another, const nlang &where, bool right) {

auto n = nlang\_translate(where);

if(right) n->right = another.root;

else n->left = another.root;

another.root = nullptr;

}

void drop(const nlang &where, bool right) {

auto n = nlang\_translate(where);

if(right) n->right = nullptr;

else n->left = nullptr;

}

private:

node \*nlang\_translate(const nlang &lang, bool newIfNull = false, const data\_t &newPayload = data\_t()) {

node \*curr = root;

for(auto act : lang) {

if(!curr)

throw std::runtime\_error("invalid nlang to this tree. Too many null in path.");

switch(act) {

case 'L':

if(!curr->left)

curr->left = new node(newPayload, curr);

curr = curr->left;

break;

case 'R':

if(!curr->right)

curr->right = new node(newPayload, curr);

curr = curr->right;

break;

case ' ':

break;

default:

throw std::runtime\_error("invalid nlang to this tree.");

}

}

if(!curr && newIfNull) //Create root.

root = new node(newPayload, nullptr);

return curr;

}

node \*root = nullptr;

};

}

#endif//FileName := ccgen.py

#!/usr/bin/python3

src = 'reflected\_impl.hpp'

mode = 'gen\_code'

#mode = 'gen\_help'

# DO NOT use macro in func\_name! It'll gen wrong code!

macro\_list = [

('nlangref','nlang'),

('nlang','NodeLanguage'),

('dataref\_t','data\_t'),

('void','null'),

]

size\_arg = ['size\_t']

int\_arg = ['int', 'data\_t']

string\_arg = ['NodeLanguage']

void\_ret = ['void', 'null']

def gen\_code(line):

line = line.replace('\t','').replace('\r', '').strip()

if len(line) == 0:

return

ret\_type = line.split(' ')[0]

funcAndArgs = line[len(ret\_type):].strip().split('(')

func\_name, args = funcAndArgs[0], funcAndArgs[1].split(')')[0]

print('//\_\_ccgen\_debug\_\_: `ret name(args)` is `{} {}({})`'.format(ret\_type, func\_name, args))

args\_string = []

for arg in args.split(','):

arg\_type = arg.strip().split(' ')[0].replace(' ','')

if len(arg\_type) == 0:

continue

if arg\_type in size\_arg:

args\_string.append('SIZE\_ARG({})'.format(len(args\_string)+1)) # start from one

elif arg\_type in int\_arg:

args\_string.append('INT\_ARG({})'.format(len(args\_string)+1)) # start from one

elif arg\_type in string\_arg:

args\_string.append('STRING\_ARG({})'.format(len(args\_string)+1)) # start from one

else:

raise RuntimeError('Unclassed arg left here. line={}|arg\_type={}'.format(line, arg\_type))

args\_size = len(args\_string)

args\_string = ', '.join(args\_string)

print(' IFCMD("{}") {{'.format(func\_name))

print(' WANT\_ARG({})'.format(args\_size))

if ret\_type not in void\_ret:

print(' HAVE\_RETURN\_VALUE')

print(' impl.{}({});'.format(func\_name, args\_string))

if ret\_type not in void\_ret:

print(' PRINT\_RETURN\_VALUE')

print(' }')

def gen\_help(line):

line = line.replace('\t','').replace('\r', '').strip()

if len(line) == 0:

return

ret\_type = line.split(' ')[0]

funcAndArgs = line[len(ret\_type):].strip().split('(')

func\_name, args = funcAndArgs[0], funcAndArgs[1].split(')')[0]

# print('//\_\_ccgen\_debug\_\_: `ret name(args)` is `{} {}({})`'.format(ret\_type, func\_name, args))

if len(args) == 0:

print('{} -> {}'.format(func\_name, ret\_type))

else:

print('{} [{}] -> {}'.format(func\_name, args, ret\_type))

if mode == 'gen\_code':

fuck\_a\_line = gen\_code

print('//Code generated by ccgen.py below. Do not edit them by hand.')

else:

fuck\_a\_line = gen\_help

print('FuncName [Argument ...] -> ReturnValue # Instructions')

with open(src) as fd:

cont = fd.read()

working = False

for line in cont.split('\n'):

if -1 != line.find('\_\_py\_ccgen\_begin\_\_'):

working = True

continue

if -1 != line.find('\_\_py\_ccgen\_end\_\_'):

working = False

continue

if working:

for \_from, \_to in macro\_list:

line = line.replace(\_from, \_to)

fuck\_a\_line(line)

if mode != 'gen\_code':

exit(0)

print('''

IFCMD("exit") {

rlib::println("bye~");

::std::exit(0);

}

IFCMD("help") {

help\_msg();

}

//impl.debug();

//Code generated by ccgen.py ahead. Do not edit them by hand.

''')//FileName := cmake\_clean.sh

#!/bin/bash

make clean

rm -rf cmake-build-debug/ cmake\_install.cmake Makefile CMakeFiles CMakeCache.txt

//FileName := CMakeLists.txt

cmake\_minimum\_required(VERSION 3.2)

project(hust\_\_)

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")

include\_directories("/usr/include")

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

include\_directories(".")

### create a custom target called build\_libr that is part of ALL

### and will run each time you type make

##add\_custom\_target(build\_libr ALL

## COMMAND make

## WORKING\_DIRECTORY rlib

## COMMENT "Calling rlib makefile to build libr.a")

add\_library(r STATIC rlib/libr.cc)

set(BUILD\_SRC main.cc reflected\_impl.hpp btree.hpp general\_ui.hpp parser.hpp)

add\_executable(exp3 ${BUILD\_SRC})

##add\_dependencies(exp3 build\_libr)

target\_link\_libraries(exp3 r)//FileName := general\_ui.hpp

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

#define HUST\_\_\_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::vector<std::string>)>;

static void showError(const std::string &msg) {

rlib::printfln("{}{}Error{}{}: {}{}", color\_t::red, font\_t::bold, clear, color\_t::lightgray, msg, clear);

}

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

callback(splitString("help"));

while(true) {

prompt();

try {

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

}

catch(std::exception &e) {

showError(e.what());

}

if(std::cin.eof())

std::exit(0);

}

}

private:

static void prompt() {

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

rlib::printf("rfaketerm 0.2 ~");

}

else {

rlib::printf("{}rfaketerm 0.2{} {}~{} ", color\_t::green, clear, font\_t::bold, clear);

}

}

};

#endif

//FileName := input

Assign 1

Assign L 4

Assign R 2

Assign LR 32

Assign LL 22

Assign RL 21

Assign LRL 324

......省略大约500行

InOrderTraverse

CreateBiTree

Select 1

Assign 10

Assign L 40

Assign R 20

Assign LR 320

Assign LL 220

Assign RL 210

Assign LRL 3240

......省略大约500行

Select 0

InsertChild LL 1 0

PreOrderTraverse

//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\_\_\_PARSER\_HPP

#define \_HUST\_\_\_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 void help\_msg() {

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

rfaketerm 0.2 HUST\_xxxx special edition

>>> Usage: <Command> [args ...]

>>> Command List:

CommandName [Arguments ...] -> ReturnValue # Instructions

help -> null # Show this message

exit -> null # exit politely

Select [int i] -> null # Select which btree to use (Select 0 by default, index starts from zero)

List -> null # List how many btree is working currently

InitBiTree -> null

DestroyBiTree -> null

CreateBiTree -> null

ClearBiTree -> null

BiTreeEmpty -> bool

BiTreeDepth -> int

Root -> NodeLanguage

Value [NodeLanguage n] -> data\_t

Assign [NodeLanguage n, data\_t val] -> null

Parent [NodeLanguage n] -> NodeLanguage

LeftChild [NodeLanguage n] -> NodeLanguage

RightChild [NodeLanguage n] -> NodeLanguage

LeftSibling [NodeLanguage n] -> NodeLanguage

RightSibling [NodeLanguage n] -> NodeLanguage

InsertChild [NodeLanguage n, int toInsert, int LR] -> null # toInsert is index of btree to insert, start from zero, in `List`

DeleteChild [NodeLanguage n, int LR] -> null

PreOrderTraverse -> null

InOrderTraverse -> null

PostOrderTraverse -> null

LevelOrderTraverse -> null

>>> What's NodeLanguage?

NodeLanguage is a string language, with which you can appoint a node in a tree easily and quickly.

Example: assume you have a tree like this now,

A

/ \

B C

/ \ \

F G H

/ /

J X

Then you can use NodeLanguage to represent every node:

A = ""

B = "L"

C = "R"

F = "LL"

G = "LR"

H = "RR"

J = "LRL"

X = "RRL"

Every 'L' and 'R' represents a step, and you can reach the node step by step.

You can also appoint a not existing node, sothat you can insert a node here. But all node in the path must exists, here're examples:

Assign(Y, "RRLR"); //Good

Assign(D, "RLL"); //Bad, "RL" not exist

Assign(M, "L"); //Valid, B is erased and M is assigned

Assign(N , " LR L L "); //Valid, extra spaces are allowed in NodeLanguage

So you can build a tree quickly in my terminal like this:

rfaketerm ~ Assign 1

rfaketerm ~ Assign L 3

rfaketerm ~ Assign R 22

rfaketerm ~ Assign LR 11

)\_STR\_";

rlib::println(msg);

}

public:

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

if(to\_parse.empty())

return;

rlib::print(std::boolalpha);

#define AREA\_BEGIN if(to\_parse.begin()->empty()) {}

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

#define AREA\_END else

#define WANT\_ARG(n) if(to\_parse.size() != n+1) {throw std::runtime\_error(rlib::format\_string("{} arguments wanted but {} provided.", n, to\_parse.size()-1));}

#define STRING\_ARG(n) to\_parse[n]

#define SIZE\_ARG(n) std::stoul(to\_parse[n])

#define INT\_ARG(n) std::stoi(to\_parse[n])

#define HAVE\_RETURN\_VALUE auto ret =

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

AREA\_BEGIN

//Code generated by ccgen.py below. Do not edit them by hand.

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null Select(size\_t i)`

IFCMD("Select") {

WANT\_ARG(1)

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

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null List()`

IFCMD("List") {

WANT\_ARG(0)

impl.List();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null InitBiTree()`

IFCMD("InitBiTree") {

WANT\_ARG(0)

impl.InitBiTree();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null DestroyBiTree()`

IFCMD("DestroyBiTree") {

WANT\_ARG(0)

impl.DestroyBiTree();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null CreateBiTree()`

IFCMD("CreateBiTree") {

WANT\_ARG(0)

impl.CreateBiTree();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null ClearBiTree()`

IFCMD("ClearBiTree") {

WANT\_ARG(0)

impl.ClearBiTree();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `bool BiTreeEmpty()`

IFCMD("BiTreeEmpty") {

WANT\_ARG(0)

HAVE\_RETURN\_VALUE

impl.BiTreeEmpty();

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `size\_t BiTreeDepth()`

IFCMD("BiTreeDepth") {

WANT\_ARG(0)

HAVE\_RETURN\_VALUE

impl.BiTreeDepth();

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage Root()`

IFCMD("Root") {

WANT\_ARG(0)

HAVE\_RETURN\_VALUE

impl.Root();

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `data\_t Value(NodeLanguage n)`

IFCMD("Value") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.Value(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null Assign(NodeLanguage n, data\_t val)`

IFCMD("Assign") {

WANT\_ARG(2)

impl.Assign(STRING\_ARG(1), INT\_ARG(2));

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage Parent(NodeLanguage n)`

IFCMD("Parent") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.Parent(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage LeftChild(NodeLanguage n)`

IFCMD("LeftChild") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.LeftChild(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage RightChild(NodeLanguage n)`

IFCMD("RightChild") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.RightChild(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage LeftSibling(NodeLanguage n)`

IFCMD("LeftSibling") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.LeftSibling(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `NodeLanguage RightSibling(NodeLanguage n)`

IFCMD("RightSibling") {

WANT\_ARG(1)

HAVE\_RETURN\_VALUE

impl.RightSibling(STRING\_ARG(1));

PRINT\_RETURN\_VALUE

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null InsertChild(NodeLanguage n, size\_t toInsert, size\_t LR)`

IFCMD("InsertChild") {

WANT\_ARG(3)

impl.InsertChild(STRING\_ARG(1), SIZE\_ARG(2), SIZE\_ARG(3));

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null DeleteChild(NodeLanguage n, size\_t LR)`

IFCMD("DeleteChild") {

WANT\_ARG(2)

impl.DeleteChild(STRING\_ARG(1), SIZE\_ARG(2));

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null PreOrderTraverse()`

IFCMD("PreOrderTraverse") {

WANT\_ARG(0)

impl.PreOrderTraverse();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null InOrderTraverse()`

IFCMD("InOrderTraverse") {

WANT\_ARG(0)

impl.InOrderTraverse();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null PostOrderTraverse()`

IFCMD("PostOrderTraverse") {

WANT\_ARG(0)

impl.PostOrderTraverse();

}

//\_\_ccgen\_debug\_\_: `ret name(args)` is `null LevelOrderTraverse()`

IFCMD("LevelOrderTraverse") {

WANT\_ARG(0)

impl.LevelOrderTraverse();

}

IFCMD("exit") {

rlib::println("bye~");

::std::exit(0);

}

IFCMD("help") {

help\_msg();

}

//Code generated by ccgen.py ahead. Do not edit them by hand.

AREA\_END {

throw std::invalid\_argument("Invalid argument. Try to type `help` to get helped.");

}

}

};

#endif //\_HUST\_\_\_PARSER\_HPP

//FileName := reflected\_impl.hpp

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

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

#include <utility>

#include <functional>

#include <algorithm>

#include <vector>

#include "btree.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 \_\_\_IndexPlusOne) {

// auto index = \_\_\_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 \_\_\_IndexPlusOne, data\_t elem) {

// auto index = \_\_\_IndexPlusOne - 1;

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

// container.insert(iter, elem);

// }

// data\_t ListDelete(size\_t \_\_\_IndexPlusOne) {

// auto index = \_\_\_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;

//};

using hust\_xxxx::unordered\_btree;

class reflected\_impl {

public:

using data\_t = int;

using dataref\_t = const data\_t &;

using nlang = std::string;

using nlangref = const nlang &;

reflected\_impl() : containers(1), current(containers.begin()) {}

//\_\_py\_ccgen\_begin\_\_

void Select(size\_t i) {current = containers.begin() + i;}

void List() {rlib::printfln("You have {} btree now, selecting {}.", containers.size(), current - containers.begin());}

void InitBiTree() {}

void DestroyBiTree() {containers.erase(current); current = containers.begin();}

void CreateBiTree() {containers.push\_back(unordered\_btree<data\_t>());}

void ClearBiTree() {current->clear();}

bool BiTreeEmpty() {return current->empty();}

size\_t BiTreeDepth() {return current->depth();}

nlang Root() {return current->\_root();}

data\_t Value(nlangref n) {return current->get(n);}

void Assign(nlangref n, dataref\_t val) {return current->set(n, val);}

nlang Parent(nlangref n) {return current->parent(n);}

nlang LeftChild(nlangref n) {return current->lchild(n);}

nlang RightChild(nlangref n) {return current->rchild(n);}

nlang LeftSibling(nlangref n) {return current->lchild(current->parent(n));}

nlang RightSibling(nlangref n) {return current->rchild(current->parent(n));}

void InsertChild(nlangref n, size\_t toInsert, size\_t LR) {return current->merge(containers[toInsert], n, LR==1);}

void DeleteChild(nlangref n, size\_t LR) {return current->drop(n, LR==1);}

void PreOrderTraverse() {current->for\_each(unordered\_btree<data\_t>::printer, hust\_xxxx::foreach\_rule::MIDDLE\_LEFT\_RIGHT);}

void InOrderTraverse() {current->for\_each(unordered\_btree<data\_t>::printer, hust\_xxxx::foreach\_rule::LEFT\_MIDDLE\_RIGHT);}

void PostOrderTraverse() {current->for\_each(unordered\_btree<data\_t>::printer, hust\_xxxx::foreach\_rule::LEFT\_RIGHT\_MIDDLE);}

void LevelOrderTraverse() {current->level\_for\_each(unordered\_btree<data\_t>::printer);}

//\_\_py\_ccgen\_end\_\_

private:

std::vector<unordered\_btree<data\_t>> containers;

decltype(containers.begin()) current;

};

extern reflected\_impl impl;

#endif

//FileName := rlib

cat: rlib: 是一个目录

3.3.2 算法测试

由于本次实验测试程序并不好写，也没有提前写好的测试程序(各种库的树实现当然都是平衡树)，只用复制粘贴的方法生成了1000多个测试样例，可以初步说明程序的鲁棒性。

`cmake . -DCMAKE\_BUILD\_TYPE=Release ; and make ; and ./exp3 < input`

其中input的内容为

Assign 1

Assign L 4

Assign R 2

Assign LR 32

Assign LL 22

Assign RL 21

Assign LRL 324

Assign L 1

Assign LL 4

Assign LR 2

Assign LLR 32

Assign LLL 22

Assign LRL 21

Assign LLRL 324

Assign RL 4

Assign RR 2

Assign RLR 32

Assign LRLRLL 22

Assign LRLRRL 21

Assign LRLRLRL 324

Assign LRL 1

Assign LRLL 4

Assign LRLR 2

Assign LRLLR 32

Assign LRLLL 22

Assign LRLRL 21

Assign LRLLRL 324

Assign LRLL 1

Assign LRLLL 4

Assign LRLLR 2

Assign LRLLLR 32

Assign LRLLLL 22

Assign LRLLRL 21

Assign LRLLLRL 324

Assign RLRLRLRL 4

Assign RLR

此处省略1100行 内容见原文件

LRLLRRLRRRL 210

Assign LRLRRLRLRLLRRLRRLRL 3240

Assign LRLRRLRLRLLRLLL 10

Assign LRLRRLRLRLLRLLLL 40

Assign LRLRRLRLRLLRLLLR 20

Assign LRLRRLRLRLLRLLLLR 320

Assign LRLRRLRLRLLRLLLLL 220

Assign LRLRRLRLRLLRLLLRL 210

Assign LRLRRLRLRLLRLLLLRL 3240

Assign LRLRRLRLRLLRLLLL 10

Assign LRLRRLRLRLLRLLLLL 40

Assign LRLRRLRLRLLRLLLLR 20

Assign LRLRRLRLRLLRLLLLLR 320

Assign LRLRRLRLRLLRLLLLLL 220

Assign LRLRRLRLRLLRLLLLRL 210

Assign LRLRRLRLRLLRLLLLLRL 3240

Assign LRLRRLRLRLLRLLLRL 40

Assign LRLRRLRLRLLRLLLRR 20

Assign LRLRRLRLRLLRLLLRLR 320

Assign LRLRRLRLRLLRLLLLRLRLL 220

Assign LRLRRLRLRLLRLLLLRLRRL 210

Assign LRLRRLRLRLLRLLLLRLRLRL 3240

Assign LRLRRLRLRLLRLLLLRL 10

Assign LRLRRLRLRLLRLLLLRLL 40

Assign LRLRRLRLRLLRLLLLRLR 20

Assign LRLRRLRLRLLRLLLLRLLR 320

Assign LRLRRLRLRLLRLLLLRLLL 220

Assign LRLRRLRLRLLRLLLLRLRL 210

Assign LRLRRLRLRLLRLLLLRLLRL 3240

Assign LRLRRLRLRLLRLLLLRLL 10

Assign LRLRRLRLRLLRLLLLRLLL 40

Assign LRLRRLRLRLLRLLLLRLLR 20

Select 0

InsertChild LL 1 0

PreOrderTraverse

程序进行这些操作并没有出现错误。由于输出过长，此处无法展示，请直接运行测试。

3.3.3 界面测试

rfaketerm和中间的每一层中间层都复用了过去的框架，采用了较好的实现方式和架构，同时使用了代码自动生成，其已经经历多次实验的考验。简单的测试表明，界面的正确性没有问题。

**2.4 实验小结**

此次实验相比上次做了以下更新：

rlib更新了stdio.hpp，加入了更接近python的fmt风格。进行了重构，由纯头库变为含部分静态库，解决了符号冲突的隐患。对所有子模块完善了对不同C++的检测，增加编译的鲁棒性。以及其他小的修复和接口更新。

实验程序框架方面，增加了简单的代码生成器和帮助信息生成器，可以增加开发速度和程序可靠性。完善了异常处理，增加了用户的异常时体验。完善了对信号和EOF的处理规则更新了，使得rfaketerm成为一个支持简单脚本的shell，极大的便利了从外部的自动化测试和自动化任务。其他大量的不完善细节。

本次实验加深了对二叉树的概念、基本运算的理解，掌握了二叉树的基本预算的实现。熟练了二元树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 |  |  |  |  |  |  |  |