

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсивная обработка иерархических списков

Студент гр. 9303

Алексеевко Б.

Преподаватель

Филатов Ар. Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомится с одной из часто используемых нелинейных конструкций, способами ее организации и рекурсивной обработки. Получить навыки решения задач обработки иерархических списков, как с использованием базовых функций рекурсивной обработки, так и без использования рекурсии.

Задание.

1) Подсчитать общий вес заданного бинарного коромысла `bk`, т. е. суммарный вес его гирек. Для этого ввести рекурсивную функцию

```
unsigned int W (const БинКор bk) .
```

Основные теоретические положения.

Бинарное коромысло устроено так, что у него есть два *плеча*: *левое* и *правое*. Каждое плечо представляет собой (невесомый) стержень определенной *длины*, с которого свисает либо *гирька*, либо еще одно бинарное коромысло, устроенное таким же образом.

В соответствии с данным выше рекурсивным определением бинарного коромысла представим бинарное коромысло (`БинКор`) списком из двух элементов

$$\text{БинКор} ::= (\text{Плечо} \text{ Плечо}) ,$$

где первое плечо является левым, а второе – правым. В свою очередь `Плечо` будет представляться списком из двух элементов

$$\text{Плечо} ::= (\text{Длина} \text{ Груз}) ,$$

где `Длина` есть натуральное число, а `Груз` представляется вариантами

$$\text{Груз} ::= \text{Гирька} \mid \text{БинКор} ,$$

Выполнение работы.

Для создания бинарного коромысла были реализованы следующие классы:

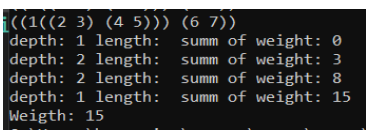
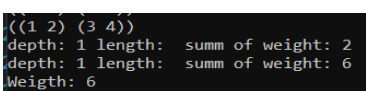
- `binTree` – класс бинарного коромысла, у которого есть два плеча (`left`, `right`) указатели на тип класса `Arm`.
- `Arm` – плечо коромысла, у которого есть поле длины (`length`), поле `tag` типа `bool`, которое определяет есть продолжение у плеча или нет и поле указатель на тип класса `Node`.
- `Node` – класс, который, в зависимости от значения `tag` (поле класса `Arm`), хранит в поле `weight` вес гирьки либо указатель на следующее бинарное коромысло.
- `treeWork` – класс, в котором реализованы методы инициализации бинарного коромысла и его обработки.

Функция `unsigned int W(binTree* bk)` считает общую массу всех грузовиков. Эта функция вызывает два взаимно-рекурсивных метода класса `treeWork`: `tourBin` и `tourArm`, которые обходят все плечи коромысла.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<code>((1((2 3) (4 5))) (6 7))</code>	 <p>Рисунок 1 – Тестирование программы</p>	Ожидаемый ответ.
2.	<code>((1 2) (3 4))</code>	 <p>Рисунок 2 – Тестирование программы</p>	Ожидаемый ответ.

Выводы.

В работе реализован иерархический список и функции для работы с ним. Был реализован выбор потока для ввода данных. Были реализованы классы, которые представляют собой бинарное коромысло, которое можно будет использовать в последствии.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "treeWork.h"

int main() {
    treeWork tree;
    tree.startReading();
    tree.writeBin(tree.getTree());
    std::cout << std::endl;
    std::cout << "Weigth: " << tree.W(tree.getTree());
    return 0;
}
```

Название файла: treeWork.h

```
#pragma once
#include "arm.h"
#include "binTree.h"
#include <string>
#include <fstream>
#include <iostream>

class treeWork {
public:
    void startReading();
    void readBin(binTree* tree, char sym); // Взаимно
рекурсивные
    arm* readArm(arm* arm_);
    void readBinFile(binTree* tree, char sym, std::ifstream&
in);
    arm* readArmFile(arm* arm_, std::ifstream& in);
    void writeArm(arm* arm_);
    void writeBin(binTree* tree);
    void outPut(int level, int weight_, int allWeigth);
    void tourArm(arm* arm_, int& level, unsigned int& weight_);
    void tourBin(binTree* tree, int& level, unsigned int&
weight_);
    binTree* getTree();
    unsigned int W(binTree* bk);
private:
    binTree* tree_;
};
```

Название файла: treeWork.cpp

```
#include "treeWork.h"
```

```

void treeWork::startReading() {
    std::cout << "Will you use file input or from consol?(f - file /
c - console)" << std::endl;
    char choice;
    std::cin >> choice;
    switch (choice) {
    case 'f': {
        std::cout << "Enter file's name" << std::endl;
        std::string name_;
        while (std::cin.get() != '\n');
        getline(std::cin, name_);
        std::ifstream in(name_);
        char cur;
        in.get(cur);
        binTree* tree = new binTree;
        this->tree_ = tree;
        if (cur == '(') {
            readBinFile(tree, cur, in);
        }
        break;
    }
    case 'c': {
        std::cout << "Enter input data" << std::endl;
        char cur;

        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

        std::cin.get(cur);
        binTree* tree = new binTree;
        this->tree_ = tree;
        if (cur == '(') {
            readBin(tree, cur);
        }
        break;
    }
}

void treeWork::readBin(binTree* tree, char sym) {
    if (sym == '('){
        arm* left = new arm;
        left = readArm(left);
        if (!left) {
            std::cout << "error in bin left" << std::endl;
            exit(1);
        }
        tree->setLeft(left);
    }
}

```

```

char cur;
std::cin.get(sym);
if (sym != ' ') {
    std::cout << "error in bin" << std::endl;
    exit(1);
}
arm* right = new arm;
right = readArm(right);
if (!right) {
    std::cout << "error in bim right" << std::endl;
    exit(1);
}
tree->setRight(right);
std::cin.get(sym);
if (sym != ')') {
    std::cout << "error in ')' bin" << std::endl;
    exit(1);
}
}
else {
    std::cout << "ERROR" << std::endl;
    exit(1);
}
}

arm* treeWork::readArm(arm* arm_) {
char cur;
std::cin.get(cur);
if (cur == '(') {
    int length;
    std::cin >> length;
    arm_->setLength(length);
    std::cin.get(cur);
    if (cur == '(') {
        arm_->setTag(true);
        node* node_ = new node;
        binTree* bin_ = new binTree;
        node_->setBinTree(bin_);
        arm_->setNode(node_);
        readBin(bin_, cur);
    }
    else if (cur == ' ') {
        arm_->setTag(false);
        node* node_ = new node;
        int weigth;
        std::cin >> weigth;
        node_->setWeight(weigth);
        arm_->setNode(node_);
    }
}
}

```



```

    }
    std::cin.get(cur);
    if (cur != ')') {
        std::cout << "error ')' in arm" << std::endl;
        return nullptr;
    }
}
else {
    std::cout << "error arm" << std::endl;
    return nullptr;
}
return arm_;
}

void treeWork::readBinFile(binTree* tree, char sym,
std::ifstream& in) {
    if (sym == '(') {
        arm* left = new arm;
        left = readArmFile(left, in);
        if (!left) {
            std::cout << "error in bin left" << std::endl;
            exit(1);
        }
        tree->setLeft(left);
        char cur;
        in.get(sym);
        if (sym != ' ') {
            std::cout << "error in bin" << std::endl;
            exit(1);
        }
        arm* right = new arm;
        right = readArmFile(right, in);
        if (!right) {
            std::cout << "error in bim right" << std::endl;
            exit(1);
        }
        tree->setRight(right);
        in.get(sym);
        if (sym != ')') {
            std::cout << "error in ')' bin" << std::endl;
            exit(1);
        }
    }
    else {
        std::cout << "ERROR";
        exit(1);
    }
}

```

```

arm* treeWork::readArmFile(arm* arm_, std::ifstream& in) {
    char cur;
    in.get(cur);
    if (cur == '(') {
        int length;
        in >> length;
        arm_>setLength(length);
        in.get(cur);
        if (cur == '(') {
            arm_>setTag(true);
            node* node_ = new node;
            binTree* bin_ = new binTree;
            node_>setBinTree(bin_);
            arm_>setNode(node_);
            readBinFile(bin_, cur, in);
        }
        else if (cur == ' ') {
            arm_>setTag(false);
            node* node_ = new node;
            int weigth;
            in >> weigth;
            node_>setWeight(weigth);
            arm_>setNode(node_);
        }
        in.get(cur);
        if (cur != ')') {
            std::cout << "error3";
            return nullptr;
        }
    }
    else {
        std::cout << "error4";
        return nullptr;
    }
    return arm_;
}

void treeWork::writeBin(binTree* tree) {

    if (!tree) {
        std::cout << "()" << std::endl;
        return;
    }

    std::cout << '(';
    writeArm(tree->getLeft());
    std::cout << ' ';

```

```

writeArm(tree->getRight());
std::cout << ')';
}

void treeWork::writeArm(arm* arm_) {
node* node_ = arm_->getNode();
if (!arm_) {
    std::cout << "()";
    return;
}
std::cout << '(';
std::cout << arm_->getLength();
if (!arm_->getTag()) {
    std::cout << ' ';
    std::cout << node_->getWeight();
}
else {
    writeBin(node_->getBinTree());
}
std::cout << ')';
}

binTree* treeWork::getTree() {
return this->tree_;
}

unsigned int treeWork::W(binTree* bk) {
unsigned int weighth_ = 0;
int level = 0;
tourBin(bk, level, weighth_);
return weighth_;
}

void treeWork::tourBin(binTree* tree, int& level, unsigned int&
weight_) {
    tourArm(tree->getLeft(), level, weight_);
    tourArm(tree->getRight(), level, weight_);
}

void treeWork::tourArm(arm* arm_, int& level, unsigned int&
weight_) {
    level++;
    node* node_ = arm_->getNode();
    weight_ += node_->getWeight();
    outPut(level, node_->getWeight(), weight_);
    if (arm_->getTag()) {
        tourBin(node_->getBinTree(), level, weight_);
    }
}

```

```

    level--;
}

void treeWork::outPut(int level, int weight_, int allWeigth) {
    std::cout << "depth: " << level << " length: " << " summ of
weight: " << allWeigth << std::endl;
}

```

Название файла: binTree.h

```

#pragma once
#include "arm.h"
class arm;
class binTree {
public:
    ~binTree();
    void setLeft(arm* new_);
    void setRight(arm* new_);
    arm* getLeft();
    arm* getRight();

private:
    arm* left = nullptr;
    arm* right = nullptr;
};

```

Название файла binTree.cpp:

```

#include "binTree.h"
void binTree::setLeft(arm* new_) {
    this->left = new_;
}

void binTree::setRight(arm* new_) {
    this->right = new_;
}

arm* binTree::getLeft() {
    return this->left;
}

arm* binTree::getRight() {
    return this->right;
}

binTree::~~binTree() {
    delete this->left;
    delete this->right;
}

```

Название файла arm.h:

```

#pragma once

```

```

#include "node.h"
class node;
class arm {
private:
int length = 0;
bool tag = false;
node* node_ = nullptr;
public:
~arm();
void setLength(int new_);
void setTag(bool new_);
void setNode(node* new_);
int getLength();
bool getTag();
node* getNode();
};

```

Название файла arm.cpp:

```

#include "arm.h"

void arm::setLength(int new_) {
this->length = new_;
}

void arm::setTag(bool new_) {
this->tag = new_;
}

void arm::setNode(node* new_) {
this->node_ = new_;
}

int arm::getLength() {
return this->length;
}

bool arm::getTag() {
return this->tag;
}

node* arm::getNode() {
return this->node_;
}

arm::~~arm() {
delete this->node_;
}

```

Название файла node.h:

```

#pragma once
#include "binTree.h"
class binTree;
class node {
private:
int weight = 0;
binTree* bin_ = nullptr;
public:
~node();
void setWeight(int new_);
void setBinTree(binTree* bin_);
int getWeight();
binTree* getBinTree();

};

```

Название файла node.cpp:

```

#include "node.h"

void node::setWeight(int new_) {
this->weight = new_;
}

void node::setBinTree(binTree* new_) {
this->bin_ = new_;
}

int node::getWeight() {
return this->weight;
}

binTree* node::getBinTree() {
return this->bin_;
}

node::~~node() {
delete this->bin_;
}

```