

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

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Иерархические списки.

Студент гр. 7383

Бергалиев М.А.

Преподаватель

Размочева Н.В.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

Цель работы.....	3
Реализация задачи.....	4
Тестирование.....	7
Вывод.....	8
Приложение А. Тестовые случаи.....	9
Приложение Б. Исходный код программы.....	10

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с иерархическими списками и использованием их в практических задачах на языке программирования C++.

Формулировка задачи: 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. Подсчитать число всех гирек заданного бинарного коромысла.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В функции `main` выводится приглашение выбрать способ ввода входных данных либо выйти из программы. В случае выбора файла, программа предлагает ввести имя файла. Создается объект типа `std::istream`, из которого считывается строка. В случае ввода информации с консоли, то она считывается из `std::cin`. После считывания данных вызывается конструктор бинарного коромысла, которому передается считанная строка. Вызывается метод `numbers`, который считает количество гирек, после чего. Далее выводится полученное число на экран и процесс вновь повторяется с приглашения для выбора способа ввода данных. Если в ходе процесса были введены некорректные данные, то выводится сообщение о неправильных данных и процесс продолжается сначала.

Переменные, используемые в функции `main`:

- `command` — строка, содержащая номер команды, отвечающий за выбор способа ввода данных или выхода из программы.
- `file` — буффер потока файла, участвующий в создании объекта `fin` типа `istream`, передаваемого в функцию `input_parser`.
- `filename` — имя файла, из которого берутся входные данные.
- `input` — считанная строка исходных данных.
- `rocker` — бинарное коромысло, созданное из полученной строки.

Класс `Cargo` представляет собой интерфейс классов груза. Он не содержит полей и имеет два виртуальных метода без реализаций:

- `numbers` — возвращает количество гирек в грузе.
- `to_str` — возвращает строковое представление груза.

Класс `Shoulder` представляет из себя плечо бинарного коромысла. Имеет два дружественных метода из класс `Bin_rocker`: `numbers` и `to_str`. Он содержит два поля:

- `length` — длина плеча бинарного коромысла.
- `cargo` — указатель на объект груза.

Содержит конструктор, принимающий итератор строки. Сначала проверяется, присутствует ли скобка на месте, куда указывает итератор. Если ее нет, то выбрасывается исключение, поскольку строка некорректна. Далее считывается число. Если его нет, то выбрасывается соответствующее исключение. После считывается произвольное число пробелов. Если далее идет скобка, то вызывается конструктор класса `Bin_rocker` от итератора и указатель на объект записывается в поле `cargo`. Иначе в `cargo` записывается указатель на объект класса `Weight`, конструктор которого вызывается от итератора. После проверяется, идет ли дальше скобка. Если ее нет, то выбрасывается исключение.

Класс `Weight` определяется в классе `Shoulder`, наследуется от класса `Cargo` и представляет из себя гирю. Содержит единственное поле `weight`, которое является весом гири. Имеет следующие методы:

- `Weight` — конструктор, принимающий итератор строки. Считывает число и записывает его в поле `weight`. Если числа нет, то выбрасывается исключение.
- `Numbers` — возвращает количество гирь, то есть всегда возвращает 1.
- `to_str` — возвращает строковое представление гири, то есть число, записанное в строку.

Класс `Bin_rocker` наследуется от класса `Cargo` и представляет из себя бинарное коромысло. Содержит два поля, `left` и `right`, указывающие на объекты класса `Shoulder`. Содержит следующие методы:

- `Bin_rocker` — конструктор, принимающий итератор строки. Считывает скобку, после чего вызывает сначала конструктор для поля `left`, считывает произвольное число пробелов, а далее вызывает конструктор для `right` и считывает скобку. Если одна из скобок отсутствует, то выбрасывается исключение.
- `numbers` — возвращает количество гирек в бинарном коромысле, то

есть сумму гирек в грузе левого плеча и в грузе правого плеча.

- `to_str` — возвращает строковое представление бинарного коромысла. В процессе вызывает метод `to_str` для грузов левого и правого плеч.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе G++ с ключом -std=c++14 в OS Linux Ubuntu 16.04 LTS.

В ходе тестирования ошибки не были найдены.

Некорректный случай представлен в табл. 1, в котором пропущено число. В данном случае строка не является бинарным коромыслом, потому найти число гирек невозможно.

Таблица 1 — Некорректный случай с пропущенным двоеточием.

Входные данные	Результат
((1 asdasd) (2 5))	Missing number: ((1 asdasd) (2 5)) ^

Корректные тестовые случаи представлены в приложении А.

4. ВЫВОД

В ходе работы были получены навыки работы с иерархическими списками. Поскольку структура иерархического списка определена рекурсивно, рекурсивный подход является простым и удобным способом поиска решения. Количество гирек больше на 1 количества бинарных коромысел в данной структуре.

ПРИЛОЖЕНИЕ А.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 2 — Корректные тестовые случаи

Входные данные	Результат
((1 ((1 2) (2 4)))) (2 ((3 5) (6 7)))	Short form: ((1 ((1 2) (2 4))) (2 ((3 5) (6 7)))) Number of weights: 4
((1 ((1 2) (3 4))) (5 6))	Short form: ((1 ((1 2) (3 4))) (5 6)) Number of weights: 3
((24 ((25 82) (45 ((32 95) (53 85))))) (24 ((45 ((23 69) (34 45)))) (26 46))))	Short form: ((24 ((25 82) (45 ((32 95) (53 85)))) (24 ((45 ((23 69) (34 45))) (26 46)))) Number of weights: 6
((32 53) (91 ((43 ((53 ((58 32) (23 25))) (24 54))) (42 ((34 ((67 ((48 45) (87 74))) (48 34))) (12 15))))))	Short form: ((32 53) (91 ((43 ((53 ((58 32) (23 25))) (24 54))) (42 ((34 ((67 ((48 45) (87 74))) (48 34))) (12 15)))))) Number of weights: 8

ПРИЛОЖЕНИЕ Б.

ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile:

```
all: main.o bin_rocker.o
    g++ main.o bin_rocker.o -o bin_rocker
main.o: main.cpp bin_rocker.hpp
    g++ main.cpp -std=c++14 -c
bin_rocker.o: bin_rocker.cpp bin_rocker.hpp
    g++ bin_rocker.cpp -std=c++14 -c
```

main.cpp:

```
#include <iostream>
#include <fstream>
#include "bin_rocker.hpp"

int main(){
    std::string command;
    std::filebuf file;
    std::string filename;
    std::string input;
    while(true){
        std::cout << "Enter 0 to read input from consol or
1 to read from file or 2 to exit: ";
        getline(std::cin, command);
        try{
            if(std::stoi(command) == 2)
                break;
        }
        catch(std::exception &e){
            std::cout << "Invalid command, try again" <<
std::endl;
            continue;
        }
        switch(std::stoi(command)){
            case 0:{
                getline(std::cin, input);
                break;
            }
        }
    }
}
```

```

    }
    case 1:{
        std::cout << "Enter file name: ";
        getline(std::cin, filename);
        if(file.open(filename, std::ios::in)){
            std::istream fin(&file);
            getline(fin, input);
            file.close();
        }
        else{
            std::cout << "Incorrect filename" <<
std::endl;
            file.close();
            continue;
        }
        break;
    }
    default:{
        std::cout << "Incorrect command, try again" <<
std::endl;
        continue;
    }
}
    std::string::iterator it = input.begin();
    try{
        Bin_rocker rocker(it);
        std::cout << "Short form:" << std::endl;
        std::cout << rocker.to_str() << std::endl;
        std::cout << "Number of weights: " <<
rocker.numbers() << std::endl;
    }
    catch(std::exception &e){
        std::cout << e.what() << ":" << std::endl;
        std::cout << input << std::endl;
        std::cout <<
std::string(distance(input.begin(), it), ' ') << '^' <<
std::endl;
    }
}

```

```

    }
    return 0;
}
bin_rocker.hpp:
#pragma once
class Cargo{
    public:
        virtual unsigned int numbers() const = 0;
        virtual std::string to_str() const = 0;
        Cargo(){}
        virtual ~Cargo(){}
};

```

```

class Bin_rocker : public Cargo{
public:
    Bin_rocker(std::string::iterator&);
    ~Bin_rocker();
    unsigned int numbers() const;
    std::string to_str() const;
private:
    class Shoulder;
    Shoulder* left;
    Shoulder* right;
};

```

```

bin_rocker.cpp:
#include <string>
#include <cstdlib>
#include "bin_rocker.hpp"
#include <iostream>
#include <stdexcept>

```

```

class Bin_rocker::Shoulder {
    class Weight : public Cargo {
    public:
        Weight(std::string::iterator &it) {
            char* num = &(*it);

```

```

        size_t count = 0;
        while(isdigit(*it)) {
            ++count;
            ++it;
        }
        if(count == 0)
            throw std::invalid_argument("Missing
number");
        weight = atoi(num);
    }
    unsigned int numbers() const {
        return 1;
    }
    std::string to_str() const {
        return std::to_string(weight);
    }
private:
    unsigned int weight;
};

friend unsigned int Bin_rocker::numbers() const;
friend std::string Bin_rocker::to_str() const;

public:
    Shoulder(std::string::iterator &it) {
        if(*it != '(')
            throw std::invalid_argument("Missing
opening bracket");
        ++it;
        while(isspace(*it))
            ++it;
        size_t count = 0;
        char* num = &(*it);
        while(isdigit(*it)) {
            ++count;
            ++it;
        }
        if(count == 0)

```

```

        throw std::invalid_argument("Missing
number");
        lenght = atoi(num);
        while(isspace(*it))
            ++it;
        if(*it == '(')
            cargo = new Bin_rocker(it);
        else
            cargo = new Weight(it);
        while(isspace(*it))
            ++it;
        if(*it != ')')
            throw std::invalid_argument("Missing
closing bracket");
        ++it;
    }
    ~Shoulder(){
        delete cargo;
    }
private:
    int lenght;
    Cargo* cargo;
};

Bin_rocker::Bin_rocker(std::string::iterator &it) {
    while(isspace(*it))
        ++it;
    if(*it != '(')
        throw std::invalid_argument("Missing opening
bracket");
    ++it;
    while(isspace(*it))
        ++it;
    left = new Shoulder(it);
    while(isspace(*it))
        ++it;
    right = new Shoulder(it);
    while(isspace(*it))

```

```

        ++it;
        if(*it != ')')
            throw std::invalid_argument("Missing closing
bracket");
        ++it;
    }

    Bin_rocker::~Bin_rocker(){
        delete left;
        delete right;
    }

    unsigned int Bin_rocker::numbers() const {
        return left->cargo->numbers() + right->cargo-
>numbers();
    }

    std::string Bin_rocker::to_str() const {
        std::string result = "(";
        result += std::to_string(left->lenght);
        result.push_back(' ');
        result += left->cargo->to_str();
        result.push_back(')');
        result.push_back(' ');
        result.push_back('(');
        result += std::to_string(right->lenght);
        result.push_back(' ');
        result += right->cargo->to_str();
        result.push_back(')');
        result.push_back(')');
        return result;
    }

```