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 _____ Александров Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2018

Цель работы.

Познакомиться с иерархическими списками и использованием их в практических задачах на языке программирования C++.

Постановка 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.

Вариант 1. Подсчитать общий вес заданного бинарного коромысла, то есть суммарный вес его гирек.

Реализация задачи.

Для решения поставленной задачи в работе были использованы 3 класса: Main, BinKor, Action.

В классе Main определяются функции для считывания данных:

- void consoleRead() - из консоли;
- void fileRead() - из файла.

Пользователю предлагается либо ввести данные вручную, либо указать текстовый файл, в котором они находятся.

В классе BinKor определяются иерархический список для хранения бинарного коромысла и функции взаимодействия с ним:

- lisp head(const lisp s) возвращает указатель на голову списка;
- lisp tail(const lisp s) возвращает указатель на хвост списка;
- lisp cons(const lisp h, const lisp t) создает новый элемент списка из головы и хвоста полученных на вход;
- lisp makeAtom(const base x) принимает на вход число и создает элемент списка – атомом;
- bool isAtom(const lisp s) проверяет, является ли элемент атомом;

- bool isNull(const lisp s) проверяет список на отсутствие в нем элементов;
- void destroy(lisp s) удаляет элементы списка;
- void readLisp(lisp &y, string &str) считывает символы из входящей строки, пока не встретит символ;
- void readSExp(lisp &y, string &str, int &iterator) выводит ошибку если символ полученный на вход является закрывающей скобкой или минусом, иначе вызывает readSeq для создания списка одного уровня, если символ является открывающей скобкой, вызывает makeAtom для создания атома, если символ не является круглой скобкой;
- void readSeq(lisp &y, string &str, int &iterator) выводит ошибку если входящая строка пуста. Создает пустой список, если следующий символ в потоке, не являющийся пробелом – закрывающая скобка. В остальных случаях считывает голову функцией readSExp, а хвост функцией readSeq;
- int getWeight() возвращает суммарный вес гирек бинарного коромысла;
- void getWeighHelper(const lisp x) подает голову коромысла в getWeight();
- const string &getOutputString() выводит результатирующую строку.

Тестирование программы.

Программа собрана и проверена в операционных системах Xubuntu 18.04 с использованием компилятора g++ и Windows с использованием MinGW. В других ОС и компиляторах тестирование не проводилось. Тесты находятся в приложении А.

Вывод.

В ходе лабораторной работы были получены основные навыки программирования иерархических списков на языке C++, изучены приемы хранения бинарного коромысла в иерархическом списке. Результатом стала программа, рассчитывающая суммарный вес гирек в коромысле.

ПРИЛОЖЕНИЕ А
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Таблица 1 – Тестирование программы

Input	Output
((8((2 6)(4 3)))(1((5 5)(1 2))))	Weight = 6 Result list ((8((2 6)(4 3)))
((2((4 8)(7 1)))(1 3))	Weight = 9 Result list ((2((4 8)(7 1)))
((2 4) (22 123))(2913129((9321	Weight = 127 Result list ((2 4) (22 123))
((2 4) (31 32))	Weight = 36 Result list ((2 4) (31 32))
((2 4) (-31 32))	This list has not only numbers or brackets

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "main.h"
#include "binkor.h"

using namespace std;

void Main::fileRead() {
    string fileName, str;
    cout << "What`s the file name?" << endl;
    cin >> fileName;
    cout << "--------------" << endl;
    cout << "Reading from " << fileName << endl;
    cout << "--------------" << endl;

    ifstream inFile;
    inFile.open(fileName);
    if (!inFile) {
        cout << "Cannot find this file" << endl;
        cout << endl;
        return;
    }
    while (!inFile.eof()) {
        getline(inFile, str);
        BinKor::lisp lisp;
        try {
            binKor.readLisp(lisp, str);
        } catch (invalid_argument e) {
            cout << endl;
            cout << "This list has an element less then zero" << endl;
            cout << endl;
            return;
        } catch (string s) {
            cout << endl;
            cout << s << endl;
            cout << endl;
            return;
        }
        cout << endl;
        int weight = binKor.getWeight(lisp);
        cout << "Weight = " << weight << endl;
        if (weight != 0) {
            cout << "Result list " << endl;
            cout << binKor.getOutputString();
```

```
            cout << endl;
        }
        cout << endl;
        binKor.destroy(lisp);
    }
    inFile.close();
}

void Main::consoleRead() {
    string str;
    cout << "Enter string:" << endl;
    getline(cin, str); // remove '\n'
    getline(cin, str);
    BinKor::lisp lisp;

    try {
        binKor.readLisp(lisp, str);
    } catch (invalid_argument e) {
        cout << endl;
        cout << "This list has an element less then zero" << endl;
        cout << endl;
        return;
    } catch (string s) {
        cout << endl;
        cout << s << endl;
        cout << endl;
        return;
    }
    cout << endl;
    int weight = binKor.getWeight(lisp);
    cout << "Weight = " << weight << endl;
    if (weight != 0) {
        cout << "Result list " << endl;
        cout << binKor.getOutputString();
        cout << endl;
    }
    cout << endl;
    binKor.destroy(lisp);
}

void Main::menu() {
    cout << "1. Enter numbers from the txt" << endl;
    cout << "2. Enter numbers from the console" << endl;
    cout << "0. Exit" << endl;
}

int main() {
    Main main;

    while (true) {
        main.menu();
```

```
        cin >> main.choice;
        switch (main.choice) {
            case 1:
                main.fileRead();
                break;
            case 2:
                main.consoleRead();
                break;
            case 0:
                exit(1);
        }
    }
}
```

main.h

```
#pragma once

#include "binkor.h"

class Main {
private:
    BinKor binKor;
public:
    Main() {}
    unsigned int choice;

    void consoleRead();

    void fileRead();

    void menu();
};
```

binkor.cpp

```
#include <iostream>
#include <cctype>
#include <string>
#include "binkor.h"

using namespace std;

BinKor::s_expr *BinKor::head(const lisp s) {
    if (s != nullptr) {
        return s->node.pair.hd;
    } else {
        return nullptr;
    }
}
```

```
BinKor::s_expr *BinKor::tail(const lisp s) {
    if (s != nullptr) {
        return s->node.pair.tl;
    } else {
        return nullptr;
    }
}

bool BinKor::isAtom(const lisp s) {
    if (s == nullptr)
        return false;
    else
        return (s->tag);
}

bool BinKor::isNull(const lisp s) {
    return s == nullptr;
}

BinKor::lisp BinKor::cons(const lisp h, const lisp t) {
    lisp p;
    p = new s_expr;
    p->tag = false;
    p->node.pair.hd = h;
    p->node.pair.tl = t;
    return p;
}

BinKor::lisp BinKor::makeAtom(const base x) {
    lisp s;
    s = new s_expr;
    s->tag = true;
    s->node.weight = x;
    return s;
}

void BinKor::readLisp(lisp &y, string &str) {
    int i = 0;
    do {
        if (str[i] == '(') outputString += '(';
        i++;
    } while (str[i] == ' ' && i < str.length());
    if (str[i])
        readSExp(y, str, i);
}

void BinKor::readSExp(lisp &y, string &str, int &iterator) {
    if (str[iterator] == ')') {
        throw string("Error: the initial brace is closing");
    } else if (str[iterator] == '-') {
        throw invalid_argument("Element less then zero");
```

```
    } else if (str[iterator] != '(') {
        if (!isdigit(str[iterator])) {
            throw string("This list has not only numbers or brackets");
        }
        string num;
        int atomWeight;
        if (isdigit(str[iterator + 1])) {
            while (isdigit(str[iterator])) {
                num.push_back(str[iterator]);
                iterator++;
            }
        }
        if (num.empty() && str[iterator + 1] == ')') {
            atomWeight = str[iterator] - '0';
            iterator++;
            outputString += to_string(atomWeight);
            y = makeAtom(atomWeight);
        } else if (str[iterator + 1] == ')') {
            atomWeight = atoi(num.c_str());
            outputString += to_string(atomWeight);
            y = makeAtom(atomWeight);
        } else {
            if (num.empty()) {
                outputString += to_string(str[iterator] - '0');
            } else {
                atomWeight = atoi(num.c_str());
                outputString += to_string(atomWeight);
            }
            // if the number = length, an atom will be -1
            y = makeAtom(-1);
        }
    } else {
        if (str[iterator] == '(') outputString += ('(');
        readSeq(y, str, iterator);
    }
}

void BinKor::readSeq(lisp &y, string &str, int &iterator) {
    char ch;
    lisp p1, p2;
    if (str[iterator] == ')' && iterator < str.length()) {
        outputString += ')';
    } else if (isspace(str[iterator])) {
        outputString += ' ';
    }
    if (!str[++iterator]) {
        throw string("Error: there is no closing bracket");
    } else {
        // skip whitespaces
        while (isspace(str[iterator]) && iterator < str.length()) {
            iterator++;
```

```
            if (outputString[outputString.length() - 1] != ' ') {
                outputString += ' ';
            }
        }
        // skip (
        while (str[iterator] == '(' && iterator < str.length()) {
            iterator++;
            outputString += '(';
        }
        if (str[iterator] == ')') {
            while (str[iterator] == ')' && iterator < str.length()) {
                y = nullptr;
                iterator++;
                outputString += ')';
            }
        } else {
            readSExp(p1, str, iterator);
            readSeq(p2, str, iterator);
            y = cons(p1, p2);
        }
    }
}

int BinKor::getWeight(const lisp x) {
    if (isAtom(x)) {
//        resultString.push_back((char)x->node.weight);
//        cout << "X node weight " << x->node.weight << endl;
        if (x->node.weight != -1)
            count += x->node.weight;
    } else {
        getWeighHelper(x);
    }
    return count;
}

void BinKor::getWeighHelper(const lisp x) {
    if (!isNull(x)) {
        getWeight(head(x));
        getWeighHelper(tail(x));
    }
}

void BinKor::destroy(lisp s) {
    outputString.clear();
    count = 0;
    if (s != nullptr) {
        if (!isAtom(s)) {
            destroy(head(s));
            destroy(tail(s));
        }
        delete s;
```

```
    }
}

const string &BinKor::getOutputString() {
    return outputString;
}
```

binkor.h

```
#pragma once

#include <sstream>

using namespace std;

class BinKor {
public:
    typedef int base;
    struct s_expr;
    struct two_ptr {
        s_expr *hd;
        s_expr *tl;
    };
    struct s_expr {
        bool tag;
        union {
            base weight;
            char atom;
            two_ptr pair;
        } node;
    };
    typedef s_expr *lisp;

    lisp head(const lisp s);

    lisp tail(const lisp s);

    lisp cons(const lisp h, const lisp t);

    lisp makeAtom(const base x);

    bool isAtom(const lisp s);

    bool isNull(const lisp s);

    void destroy(lisp s);

    void readLisp(lisp &y, string &str);

    void readSExp(lisp &y, string &str, int &iterator);
```

```
    void readSeq(lisp &y, string &str, int &iterator);

    int getWeight(const lisp x);

    void getWeighHelper(const lisp x);

    const string &getOutputString();
private:
    string outputString;
    int count = 0;
};
```

Makefile

```
CXX=g++
RM=rm -f
LDFLAGS=-g -Wall

SRCS=main.cpp binkor.cpp
OBJS=$(subst .cpp,.o,$(SRCS))

all: main

main: $(OBJS)
	$(CXX) $(LDFLAGS) -o main $(OBJS)

main.o: main.cpp main.h

binkor.o: binkor.cpp binkor.h

clean:
	$(RM) $(OBJS)

distclean: clean
	$(RM) main
```