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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивная обработка иерархических списков

Студент гр. 8304 _____ Алтухов А.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с иерархическими списками, способами их организации, общими базовыми функциями их обработки.

Задание.

Вариант 1.

Подсчитать общий вес заданного бинарного коромысла, т. е. суммарный вес его гирек.

Описание алгоритма.

Для определения веса коромысла используется следующий алгоритм: если голова обрабатываемого сегмента списка атом, а хвоста не существует, что значит, что атом последний и является весом. Значение веса прибавляется к аккумулирующей переменной. Если же верно обратное, то рекурсивно вызывается та же обрабатывающая функция, но уже для головы и хвоста текущего сегмента списка, разумеется, если эти элементы существуют и являются списками.

Основные функции и структуры.

1. class **IerList**

Содержит следующие приватные поля:

bool tag — значение true соответствует тому, что этот элемент является атомом, иначе — списком.

int atom — числовые данные.

IerList *head — указатель на первый элемент списка.

IerList *tail — указатель на список, содержащий все, кроме первого элемента, указанного в поле head.

Содержит следующие приватные методы:

void readData(IerList*& listToMerge, std::string str, int& iter) — читает символ с индексом iter, запускает собирание атома если считано число. В противном случае чтение продолжается функцией readSequence.

void readSequence(IerList*& listToMerge, std::string str, int& iter) — чтение, установление связей между частями списка. Эти части рекурсивно собираются в списке, указанном первым аргументом.

void **printSymbol**() const — печать отдельных атомов.

void **writeSequence**() const — рекурсивная печать списка.

Содержит следующие публичные методы:

IerList() — конструктор инициализирует поля head и tail значением nullptr, поле tag значением false

~**IerList**() — деконструктор рекурсивно запускает сам себя для головы и хвоста текущего списка.

bool **isNull**() const — проверяет, являются ли поля head и tail нулевыми указателями.

IerList* **getHead**() const — возвращает указатель на голову текущего списка.

IerList* **getTail**() const — возвращает указатель на хвост текущего списка.

IerList* **getTrueTail**() const — возвращает указатель на голову хвоста текущего списка.

int **getAtom**() const — возвращает числовое значение, хранящееся в этом узле.

bool **isAtom**() const — проверяет, является ли узел атомом.

void **makeAtom**(const int x) — записывает в поле tag значение true, в поле atom значение x.

void **merge**(IerList* head, IerList* tail) — записывает в поле tag значение false, в поле head значение head, в поле tail значение tail.

void readList(IerList*& listForMerge, std::string str) — функция-запускатор readData.

void **print**() const — функция-запускатор printSymbol.

bool **isBalanceBeam**() const — функция, информирующая пользователя, является ли введенный иерархический список бинарным коромыслом. Основывается на данных, полученных от goRoundBalanceBeam.

int **goRoundBalanceBeam**(bool isInShoulder) const — исследует, является ли переданный список бинарным коромыслом.

Графическое представление иерархического списка представлено на рис. 1.

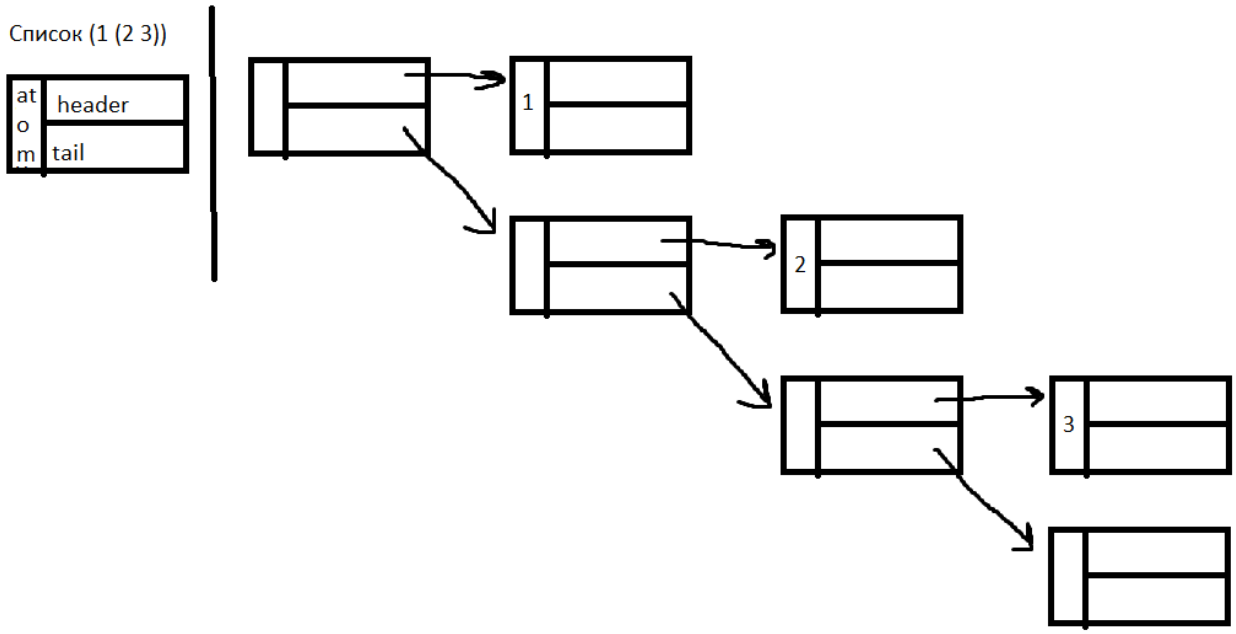

Рисунок 1 – Иерархический список

2. bool isListInString(std::string str) — определяет, являются ли входные данные корректными.

3. void **findWeight**(IerList* list, int& weight) — определяет вес бинарного коромысла. Алгоритм работы подробно описан в разделе «Описание алгоритма».

Тестирование.

Некорректные данные:

№	Исходные данные	Ответ программы	Ожидаемый ответ
1	()	Статус коромысла: нет плеча.	Сообщение об ошибке
2	(1()	Не хватает	Сообщение об ошибке

		закрывающей скобки. Некорректный ввод.	
3	((1 2)(1))	Статус коромысла: не указана масса.	Сообщение об ошибке

Корректные данные:

№	Исходные данные	Ответ программы	Ожидаемый ответ
4	((1 2)(3 4))	6	6
5	((1 ((1 2) (1 2))) (1 2))	6	6
6	((1 ((1 2) (1((1 2) (1 2))))) (1 ((1 2) (1 2))))	10	10
7	((1 2)(1 ((1 4)(1 4))))	10	10

Выводы.

В ходе выполнения лабораторной работы была исследована такая структура данных как иерархический список.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

Файл list.h

```
#ifndef IER_LIST
#define IER_LIST

class  IerList;

class  IerList{

private:
	bool tag; // true: atom, false: pair
	int  atom;
	IerList  *head;
	IerList  *tail;

	void readData( IerList*& listToMerge, std::string str, int& iter);
	void readSequence(IerList*& listToMerge, std::string str, int&
iter);
	void printSymbol() const;
	void writeSequence() const;

public:

	IerList();
	~IerList();

	bool isNull() const;

	IerList* getHead() const;
	IerList* getTail() const;
	IerList* getTrueTail() const;

	int getAtom() const;
	bool isAtom() const;
	void makeAtom(const int x);

	void merge(IerList* head, IerList* tail);

	void readList(IerList*& listForMerge, std::string str);
	void print() const;

	bool isBalanceBeam() const;
	int goRoundBalanceBeam(bool isInShoulder) const;

};

#endif
```

Файл list.cpp

```
#include <iostream>
#include <string>
#include "list.h"

void IerList::readData( IerList*& listToMerge, std::string str, int&
iter){

	//смотрит символ с индексом iter, запускает собирание атома если
считано число
	//в противном случае чтение продолжается функцией readSequence

	int prev = 0;
	char lastSymbol = str[iter];

	std::size_t pos = 0;
	if ( lastSymbol == ')' ){
		std::cerr << " ! List.Error 1 " << "\n";
		exit(1);
	}
	else if ( lastSymbol != '(' ) {

		if (isdigit(lastSymbol)){

			int tempIter = iter+1;
			int i = 0;

			while (isdigit(str[tempIter+i]))
				++i;

			prev = std::stoi(str.substr(iter, i+1), &pos, 10);
			iter += i;
			this->makeAtom(prev);
		}
	}
	else this->readSequence(listToMerge, str, iter);
}

void IerList::readSequence(IerList*& listToMerge, std::string str, int&
iter){//чтение, установление связей между частями списка

	IerList* list1,* list2;

	if (!(str[++iter])) {
		std::cerr << " ! List.Error 2 " << "\n";
		exit(1);
	}
	else {

		while  ( str[iter]==' ' )
```

```
                ++iter;
            if ( str[iter] == ')' ) {}//nothing
            else {
                list1 = new IerList;
                list2 = new IerList;
                list1->readData(list1, str, iter);
                list2->readSequence(list2, str, iter);
                listToMerge->merge(list1, list2);
            }
        }
}

void IerList::printSymbol() const {//печать отдельных атомов

        if ((this->isNull())) {
            std::cout << " () ";
            std::cout << "empty\n";
        }
        else if (this->isAtom())
            std::cout << ' ' << this->atom;
        else  {
            std::cout << " (" ;
            this->writeSequence();
            std::cout << " )";
        }
}

void IerList::writeSequence() const { //рекурсивная печать списка

        if (!(this->isNull())) {
            this->getHead()->printSymbol();
            this->getTail()->writeSequence();
        }
}

IerList::IerList():tag(false), head(nullptr), tail(nullptr) {}

IerList::~IerList(){
        if (!(this->isNull()) && !(this->isAtom()) ){
            delete this->getHead();
            delete this->getTail();
        }
}


bool IerList::isNull() const {

        if (!(this->tag) && (this->head == nullptr) && (this->tail ==
nullptr))
            return true;
        return false;
```

```
}


IerList* IerList::getHead() const {

	if (!this->isNull())
		if (!this->isAtom())
			return this->head;
		else {
			std::cerr << "Error: Head(atom) \n";
			exit(1);
		}
	else {
		return nullptr;
	}
}

IerList* IerList::getTail() const {

	if (!this->isNull())
		if (!this->isAtom())
			return this->tail;
		else {
			std::cerr << "Error: Tail(atom) \n";
			exit(1);
		}
	else {
		return nullptr;
	}
}

IerList* IerList::getTrueTail() const { //обычный tail всегда является
списком, а эта функция достает его первый элемент

	if (!this->isNull())
		if (!this->isAtom())
			if (!this->getTail()->isNull())
				return this->getTail()->getHead();
			else return nullptr;
		else {
			std::cerr << "Error: True Tail(atom) \n";
			exit(1);
		}
	else {
		return nullptr;
	}
}

int IerList::getAtom() const {
	if (this->isNull() || !(this->isAtom())) {
		std::cout << "Нет атома!\n";
```

```
                return 0;
        }
        else
                return this->atom;
}

bool IerList::isAtom() const {

        if (this->isNull())
                return false;
        else
                return (this->tag);
}



void IerList::merge(IerList* head, IerList* tail){

        if (tail->isAtom()) {
                std::cerr << "Error: cons(*, atom) \n";
                exit(1);
        }
        else {
                this->tag = false;
                this->head = head;
                this->tail = tail;
        }
}

void IerList::makeAtom(const int x){

        this->tag = true;
        this->atom = x;

}

void IerList::readList(IerList*& listToMerge, std::string str) {
//функция-запускатор чтения списка
        char x = '\0';
        int iter = 0;
        do{
                if (x!='\0')
                        ++iter;
                x = str[iter];

        }
        while (x==' ');
        this->readData(listToMerge, str, iter);
}

void IerList::print() const {//функция-запускатор печатания списка
```

```
	this->printSymbol();
	std::cout<<"\n";
}

bool IerList::isBalanceBeam() const {
	int res = this->isNull() ? 1 : this->goRoundBalanceBeam(false);
	std::string status = "";
	switch (res){
		case 0:
			status = "Все хорошо";
			break;
		case 1:
			status = "Нет плеча";
			break;
		case 2:
			status = "Не указана длина";
			break;
		case 3:
			status = "Лишние значения";
			break;
		case 4:
			status = "Не указана масса";
			break;
	}
	std::cout << "Статус коромысла: " << status << "\n";
	return res>0? false : true;
}

int IerList::goRoundBalanceBeam(bool isInShoulder) const {

	if (!(this->isNull()) ){

		if (!(isInShoulder)){ //если не в плече, то смотрим, есть
ли два вложенных списка

			if ((this->getHead() != nullptr) && !(this-
>getHead()->isNull()) && !(this->getHead()->isAtom())) {}//okey
			else return 1;//no shoulder

			if ((this->getTail() != nullptr) && !(this-
>getTail()->isNull()) && !(this->getTail()->isAtom())) {}//okey
			else return 1;//no shoulder

			int res = this->getHead()->goRoundBalanceBeam(true);
//запуск анализа левого плеча
			if (!res)
				return this->getTrueTail()-
>goRoundBalanceBeam(true); //запуск анализа правого плеча
			else return res;
		}
```

```
		else { //если внутри плеча то проверка наличия длины и массы/коромысла

			if ((this->getHead() != nullptr) && !(this->getHead()->isNull()) &&(this->getHead()->isAtom()) ) {}//okey
			else return 2;//no length

			if ((this->getTail() != nullptr) && !(this->getTail()->isNull()) && (this->getTail() != nullptr)){
				IerList* nextList = this->getTail();
				IerList* nextHead = this->getTrueTail();
				bool timeToRet = false;
				if (nextHead->isAtom() && (nextList->getTail() == nullptr || nextList->getTail()->isNull())){//length and mass
					timeToRet = true;
				}
				if (!(nextHead->isAtom()) && (nextList->getTail() == nullptr || nextList->getTail()->isNull())){//length and balance beam
					return nextHead->goRoundBalanceBeam(false); //запуск анализа наличия плечей вложенного коромысла
				}
				else if (!timeToRet){
					return 3; //more then two values
				}
				if (timeToRet)
					return 0;
			}
			else
				return 4; //less then two values
		}
	}
	return 1;
}
```

Файл main.cpp

```
#include <iostream>
#include <string>
#include "list.h"
#include <stack>

void findWeight(IerList* list, int& weight){

	std::cout << "Смотрю в список:";
	list->print();
	if ((!(list->isNull())) && (list->getTail()!=nullptr) && ((list->getTrueTail() == nullptr)) && list->getHead()->isAtom() ){
		weight += list->getHead()->getAtom();
```

```
                std::cout << "Текущий вес: " << weight << "\n";
        }
        if ( (list->getHead() != nullptr) && (!(list->getHead()-
>isAtom()))  ){
            findWeight(list->getHead(), weight);
        }
        if ( (list->getTail() != nullptr) && (!(list->getTail()-
>isAtom())) ){
                findWeight(list->getTail(), weight);
        }

}

bool isListInString(std::string str){

        std::stack <char> inputStack;

        for (unsigned int i=0; i<str.length(); i++){

                if (str[i] == '(') {
                        inputStack.push(str[i]);
                }
                else if (str[i] == ')') {
                        if (inputStack.empty()){
                                std::cout << "Не хватает открывающей
скобки\n";
                                return false;
                        }
                        inputStack.pop();
                }
                else if (str[i] == ' ' || isdigit(str[i])) {}//skip

                else{
                        std::cout << "Использование запрещенных символов\n";
                        return false;
                }
        }
        if (inputStack.empty()){
                return true;
        }
        std::cout << "Не хватает закрывающей скобки\n";
        return false;
}

int main(){

        std::string input;
        std::getline(std::cin, input );

        bool check = isListInString(input);
        if (!check){
```

```
                std::cout << "Некорректный ввод\n";
                return 0;
        }


    IerList* list = new IerList;
    list->readList(list, input);

    check = list->isBalanceBeam();
        if (!check){
                delete list;
                return 0;
        }

        int weight = 0;
        findWeight(list, weight);
        std::cout << "Вес: " << weight <<"\n";

        delete list;
    return 0;
}
```