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: Иерархические списки

Студент гр. 9381 _____ Щеглов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

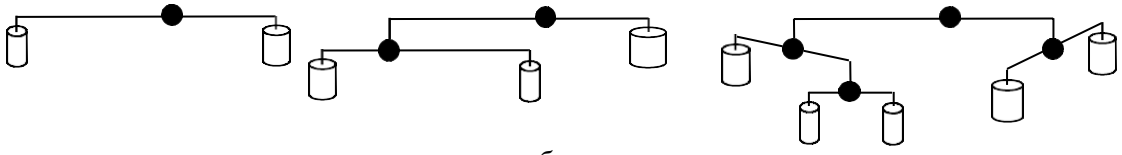
Цель работы.

Познакомиться с иерархическими списками и с их рекурсивной обработкой, основываясь на одном из примеров таких иерархических списков – бинарном коромысле.

Задание.

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.

Можно (но не обязательно) представлять себе *бинарное коромысло*, чем-то похожим на конструкции, изображенные на рисунке.



а *б* *в*

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

БинКор ::= (Плечо Плечо),

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

Плечо ::= (Длина Груз),

где Длина есть натуральное число, а Груз представляется вариантами

Груз ::= Гирька | БинКор,

где в свою очередь Гирька есть натуральное число. Таким образом, БинКор есть специального вида иерархический список из натуральных чисел. Например, следующие списки представляют бинарные коромысла, изображенные выше на рисунке (вместо натуральных чисел здесь для общности и удобства восприятия представлены их обозначения с учетом места появления в списке):

а) $((l_1\ m_1)\ (l_2\ m_2))$;
б) $((l_1\ ((l_{11}\ m_{11})\ (l_{12}\ m_{12})))\ (l_2\ m_2))$;
в) $((l_1\ ((l_{11}\ m_{11})\ (l_{12}((l_{121}\ m_{121})\ (l_{122}\ m_{122}))))))\ (l_2\ ((l_{21}\ m_{21})\ (l_{22}\ m_{22}))))$.

Для работы с бинарными коромыслами в таком представлении следует использовать базовые функции для работы с иерархическими списками.

5) Написать рекурсивную функцию или процедуру, формирующую линейный список номеров всех вхождений одного бинарного коромысла в другое.

Выполнение работы.

Для реализации программы были созданы 2 класса элементов:

- Линейного списка, содержащего номер этого элемента в бинарном коромысле и указателя на следующий элемент этого списка.
- бинарного коромысла со строковыми полями левой и правой части элемента, целочисленной переменной type, в которой будет содержаться тип конкретного рассматриваемого элемента и ссылками на левое и правое плечо коромысла, тем самым определяя иерархический список. Помимо этого, имеется поле указателя на линейный список. Также, реализован конструктор, присваивающий ссылкам на последующие элементы изначально нулевые указатели и рекурсивный метод, выполняющий непосредственно проход по введенной строке и обрабатывающий иерархический список.

Также, создана глобальная целочисленная переменная curTmpNum, которая будет определять номер элемента по прохождении по бинкору.

В главной функции, считав данные из файла, проверяем длину введенной строки, если она меньше или равна двум, то и смысла нет ее отправлять на исследование, выводим ошибку ввода. Выделив память под голову бинкора и под начало линейного списка (впоследствии он станет концом) запускаем рекурсивный метод бинкора, передавая ему введенную строку, изначальную глубину и начало линейного списка. Данный метод возвращает 0 в случае ошибки.

Функция isNumber(string):

Принимает на вход строку и проверяет, является ли строка числом, проходя по символам в строке и сравнивая ее код с кодами от 49(вкл) до 58(вкл) - цифрами

Рекурсивный метод divide(string, int):

Метод получает на вход исследуемую строку(str), глубину рекурсии в виде целого числа(depth) и указатель на элемент списка от которого должен создаваться новый элемент списка.

Получая строку, первым делом функция проверяет, ограничена ли строка скобочками, потому что иные строки на вход быть переданы не могут (выводится ошибка ввода). Далее обрезав скобки и получив выражение, которое необходимо исследовать, запускаем поиск левой части (рассматривая количество скобочек открытых и закрытых если левая часть не число, поиск пробела в ином случае).

Если найдена только левая часть выдаем ошибку ввода. Далее выделив память под левую и правую ссылку, и записав левую и правую части выражения в соответствующие поля, выводим промежуточный результат.

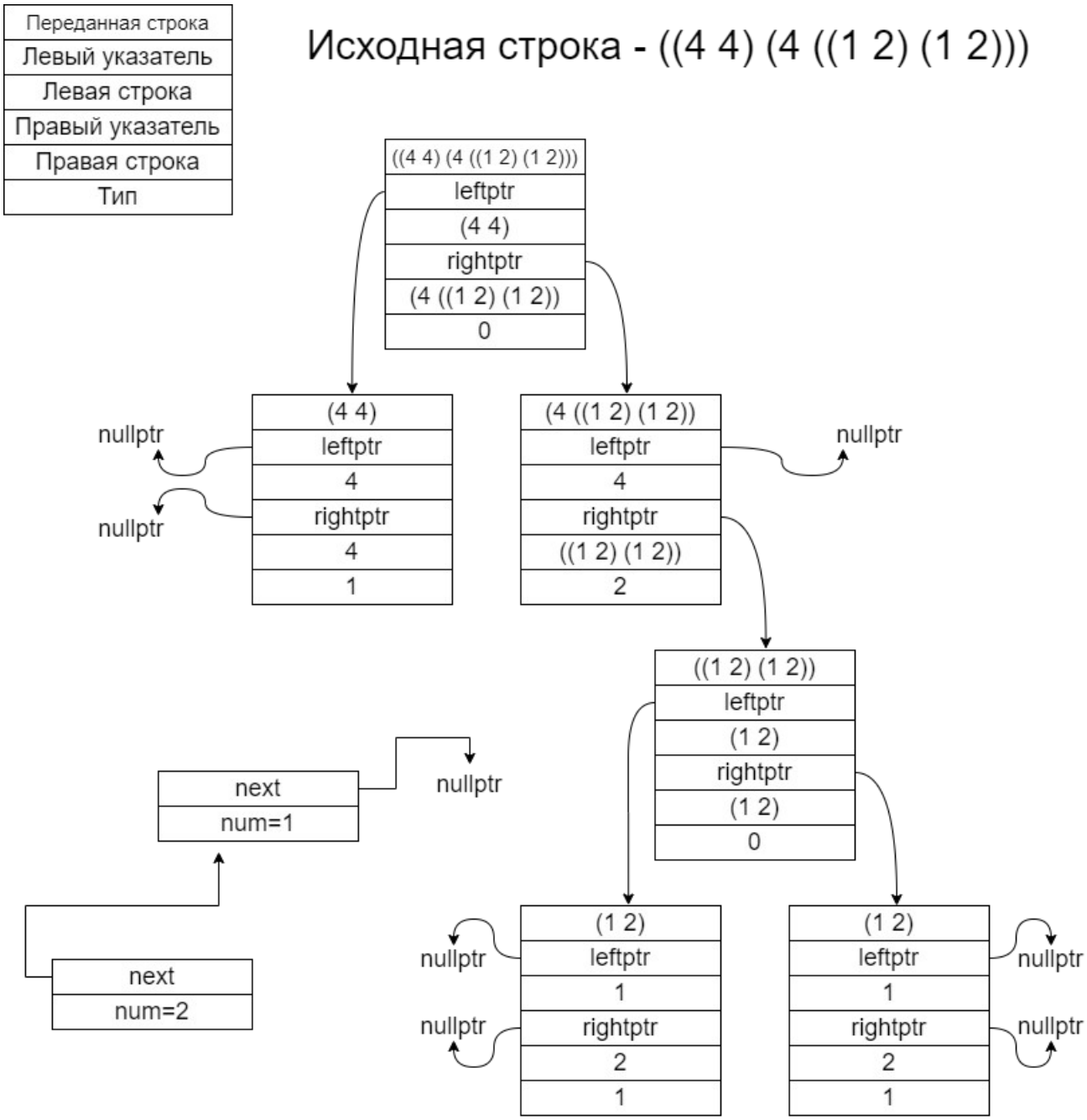
Далее исследуем левую и правую часть на принадлежность к числам.

- Если справа и слева числа – это плечо с весом. Здесь в поле элемента списка оставляем предыдущий элемент списка бинкора
- Если только слева число – это плечо с бинкором. Поле списка – аналогично плечу с весом
- Если слева и справа не числа – это бинкор, для него создаем новый элемент в списке с помощью статического метода, последующие элементы этого списка – верхние бинкоры. Выводим данный бинкор методом вывода.

Тем самым выполним задачу. Однако для последующий бинкоров также необходимо вывести списки их номеров, поэтому при условии, что выражение не является числом (isNumber) запускаем с каждым новым выражением заново рекурсию. Тем самым просто «пройдясь» по списку.

Разработанный программный код см. в приложении А.

Пример работы программы(и линейный список и бинкор) на строке ((4 4) (4 ((1 2) (1 2))):



Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	((4 ((3 ((3 2) (2 ((3 2) (2 3))))) (2 ((3 2) (2 3))))) (6 ((3 2) (2 3))))	Input: ((4 ((3 ((3 2) (2 ((3 2) (2 3))))) (2 ((3 2) (2 3))))) (6 ((3 2) (2 3)))) Output: L: (4 ((3 ((3 2) (2 ((3 2) (2 3))))) (2 ((3 2) (2 3))))) R: (6 ((3 2) (2 3))) For current binkor:1

		L: 4 R: ((3 ((3 2) (2 ((3 2) (2 3))))) (2 ((3 2) (2 3)))) L: (3 ((3 2) (2 ((3 2) (2 3))))) R: (2 ((3 2) (2 3))) For current binkor:2 1 L: 3 R: ((3 2) (2 ((3 2) (2 3)))) L: (3 2) R: (2 ((3 2) (2 3))) For current binkor:3 2 1 L: 3 R: 2 L: 2 R: ((3 2) (2 3)) L: (3 2) R: (2 3) For current binkor:4 3 2 1 L: 3 R: 2 L: 2 R: 3 L: 2 R: ((3 2) (2 3)) L: (3 2) R: (2 3) For current binkor:5 2 1 L: 3 R: 2 L: 2 R: 3 L: 6 R: ((3 2) (2 3)) L: (3 2) R: (2 3) For current binkor:6 1 L: 3 R: 2 L: 2 R: 3
2.	((4 4) ((1 2) (1 2)))	Input: ((4 4) ((1 2) (1 2))) Output: L: (4 4) R: ((1 2) (1 2)) For current binkor:1 L: 4 R: 4 L: (1 2) R: (1 2) For current binkor:2 1 L: 1 R: 2 L: 1 R: 2
3.	()()()	()()() Input:

		()()() Output: Wrong Input!
4.		Input: Output: Wrong Input!

Выводы.

Были изучены методы работы с иерархическими списками, рекурсивным обходом их. Реализована программа вывода линейных списков у бинарных коромыслов с их номерами и номерами предшествующих бинкоров.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>

int curTmpNum=0;

using namespace std;
class LineList{
public:
    LineList(){
        next= nullptr;
        num = 0;
    }
    static LineList* addNew(LineList* newone);
    void printList();
private:
    LineList* next;
    int num;
};
class Bin{
public:
    Bin(){
        Leftptr = nullptr;
        rightptr= nullptr;
    }
    int divide(string str, int depth,LineList* l);
private:
    int type;
    LineList* ll;
    string left;//left res
    string right;//right res
    Bin* leftptr;//left pointer
    Bin* rightptr;//right pointer
};

int main() {
    //Input string
    string str;
    cout << "Cin: 1\nFile: 2\n";
    string choose;

    getline(cin,choose);
    if(choose=="1")
        getline(cin,str);
    if(choose=="2"){
        ifstream fp("C:\\Labs\\Lab2\\Lab2Input.txt");
        getline(fp, str);
    }
    cout << "Input:\n" << str << "\n";
    cout << "Output:\n";
    if(str.size()<=2){
        cout << "Wrong Input!";
        return 0;
    }
    Bin* head = new Bin;
    int depth = 0;
    LineList* t = new LineList;
    //Check result
    switch(head->divide(str,depth,t)) {
```

```
        case 0: cout << "Wrong Input!"; break;
    }
    return 0;
}

bool isNumber(string s){
    for (int a = 0; a < s.length(); a++){
        if ((s[a] < 48) || (s[a] > 57))  return false;
    }
    return true;
}

int Bin::divide(string str, int depth,LineList* cur) {
    if(!(str[0]=='('&&str[str.size()-1]==')'))//Check if string is not between
"(" and ")"
        return 0;
    string tmp = str.substr(1,str.size()-2);//delete "(" ")"
    int i=0;
    int tmp9=0, tmp10=0;
    if(tmp[0]=='(')
        do{
            if(tmp[i]=='(')
                tmp9++;
            if(tmp[i]==')')
                tmp10++;
            i++;
        }while(tmp9!=tmp10);//Find left string
    else {
        while (tmp[i] != ' ') {//Find left string if not (...)
            i++;
            if (i == tmp.size())
                return 0;
        }
    }
    if (i == tmp.size())
        return 0;
    this->leftptr = new Bin;
    this->rightptr = new Bin;
    this->left=tmp.substr(0,i);//make left string
    this->right=tmp.substr(i+1,tmp.size()-i);//make right string
    for(int i=0;i<depth;i++)
        cout << " ";
    cout << "L: " << this->left << " R: " << this->right << "\n";//test output
    depth+=1;
    if(!isNumber(this->left)&&isNumber(this->right))
        return 0;
    if(!isNumber(this->left)&&!isNumber(this->right)) {
        type = 0;//Binkor
        this->ll=LineList::addNew(cur);
        this->ll->printList();
    }
    if(isNumber(this->left)&&isNumber(this->right)) {
        type = 1;//Shoulder with weight
        this->ll=cur;
    }
    if(isNumber(this->left)&&!isNumber(this->right)) {
        type = 2;//Shoulder with binkor
        this->ll=cur;
    }
    if(!isNumber(this->left)) {
        if(this->leftptr->divide(this->left, depth,this->ll)==0) {//Divide left
string if it is not a number
            return 0;
        }
    }
    if(!isNumber(this->right)) {
```

```
        if(this->rightptr->divide(this->right, depth,this->ll)==0) {//Divide
right string if it is not a number
            return 0;
        }
    }

    return 1;
}
LineList* LineList::addNew(LineList* newone) {
    LineList* tmp=new LineList;
    tmp->next=newone;
    tmp->num=curTmpNum+1;
    curTmpNum++;
    return tmp;
}

void LineList::printList() {
    LineList*  tmp;
    tmp=this;
    cout << "For current binkor:";
    do{
        cout << tmp->num << " ";
        tmp=tmp->next;
    }while(tmp->next!= nullptr);
    cout << "\n";
}
```