

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

**ОТЧЕТ**  
**по лабораторной работе №2 (вариант №2)**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: «Иерархические списки»**

Студент гр. 7381

\_\_\_\_\_

Адамов Я.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2018

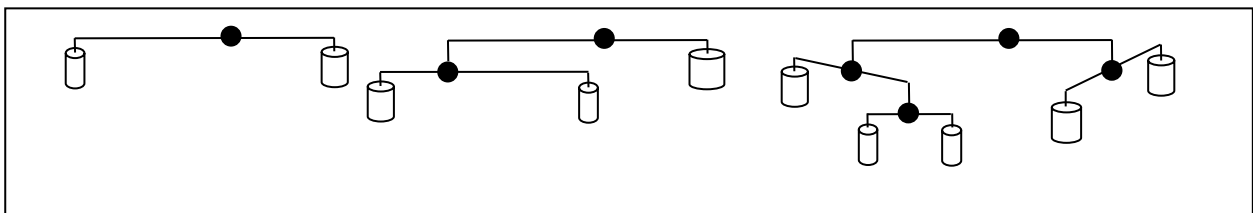
## Цель работы.

Ознакомиться с иерархическими списками и научиться применять их на практике.

## 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.

Возможный способ представления бинарного коромысла:



В соответствии с данным 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, бинарное коромысло есть специального вида иерархический список из натуральных чисел.

## Задание.

Вариант №2.

Подсчитать число всех гирек заданного бинарного коромысла `bk`. Для этого ввести рекурсивную функцию:

**unsigned int** numbers (**const** БинКор `bk`).

## Ход работы.

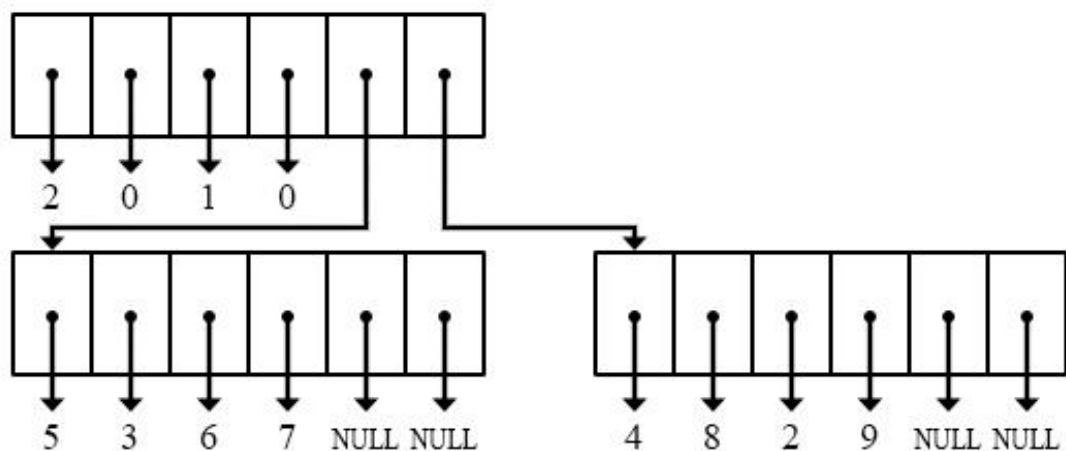
Программа написана на языке C.

Исходный файл: `main.c`

В начале работы происходит ввод данных: строка, которая является бинарным коромыслом (пример: `((5 ((142 67) (6 7))) (1 2))`), после чего происходит обработка строки и создание бинарного коромысла, для чего вызывается функция `createBinCor()`. Если введённые данные некорректны, то программа выводит соответствующую ошибку и завершает работу. В случае ввода корректных данных происходит дальнейший вызов функции `numbers()`, которая выводит на экран общее количество гирек в бинарном коромысле.

Схематическое изображение бинарного коромысла:

`((2 ((5 3) (6 7))) (1 ((4 8) (2 9))))`



Описание алгоритма функции `numbers()`: функция принимает указатель на бинарное коромысло. Количество гирек в бинарном коромысле записывается в переменную *result*, значение которой возвращает функция. Если с левого плеча свисает гиря, то значение *result* увеличивается на единицу, если свисает ещё одно бинарное коромысло, то функция `numbers()` вызывается рекурсивно и в неё передаётся указатель на это коромысло, возвращённый результат прибавляется к *result*. То же самое повторяется с правым плечом.

Для демонстрации работы было написано несколько тестов, а также скрипт `perform_tests.sh`, который запускает все эти тесты.

### Описание функций и структур.

#### 1) *struct BinCor*

Структура бинарного дерева.

Поля структуры:

- **unsigned int length1**: длина левого плеча.
- **unsigned int length1**: длина правого плеча.
- **unsigned int weight1**: вес левой гири.
- **unsigned int weight2**: вес правой гири.
- **struct BinCor \* cor\_1**: указатель на бинарное коромысло, исходящее из левого плеча.
- **struct BinCor \* cor\_2**: указатель на бинарное коромысло, исходящее из правого плеча.

#### 2) *int createBinCor(char \* str, BinCor \*\* binCor)*

Функция принимает указатель на строку, в которую записано бинарное коромысло и указатель на указатель на структуру `BinCor` (бинарное коромысло). Функция устанавливает начальный и конечный индексы на первый и последний символы

полученной строки, после чего передаёт эти данные функции `readBinCorElement()`, которая обрабатывает строку и создаёт бинарное коромысло.

Параметры:

- **str**: указатель на строку с бинарным коромыслом.
- **binCor**: указатель на указатель на структуру данных `BinCor`, содержащий адрес, куда будет заноситься результат обработки строки `str`.

Возвращаемое значение: 0 – если данные в строке `str` корректны, в ином случае – 1.

3) *int readBinCorElement(char \* str, int index\_1, int index\_2, BinCor \*\* element)*

Функция принимает указатель на строку, в которую записано бинарное коромысло, начальный и конечный индекс на первый и последний символы строки `str`, между которыми записано обрабатываемое на данном шаге бинарное коромысло и указатель на указатель на структуру `BinCor`(бинарное коромысло). Функция посимвольно обрабатывает два плеча бинарного коромысла, внося все данные по адресу, содержащемуся в указателе, на который указывает `element`. Если груз плеча является ещё одним бинарным коромыслом, то происходит рекурсивный вызов функции. При возникновении в какой-то момент ошибки вызывается функция `errorMessage()`, выводящей на экран сообщение об ошибке, после чего функция завершает работу.

Параметры:

- **str**: указатель на строку с бинарным коромыслом.
- **index\_1**: индекс, с которого надо начать обрабатывать строку `str`.
- **index\_2**: индекс, которым надо закончить обрабатывать строку `str`.
- **binCor**: указатель на указатель на структуру данных `BinCor`, содержащий адрес, куда будет заноситься результат обработки строки `str`.

Возвращаемое значение: 0 – если данные в строке `str` корректны, в ином случае – 1.

4) *unsigned int numbers(const BinCor binCor, int deep\_of\_recursion)*

Функция принимает структуру данных BinCor (бинарное коромысло) и целочисленное значение, указывающее на глубину рекурсии. Функция считает количество гирек в бинарном коромысле binCor.

Параметры:

- **binCor**: бинарное коромысло, в котором считается количество гирек.
- **deep\_of\_recursion**: глубина рекурсии (необходимо для вывода работы алгоритма на экран).

Возвращаемое значение: количество гирек в бинарном коромысле binCor.

### **Тестирование программы.**

Было создано несколько тестов для проверки работы программы. Помимо тестов, демонстрирующих работу алгоритма, были написаны тесты, содержащие некорректные данные, для демонстрации вывода сообщений об ошибках введенных данных (см. Приложение А).

### **Вывод.**

В ходе выполнения работы была изучена новая структура данных: иерархические списки. Также закреплены навыки работы с рекурсивными функциями.

## Приложение А. Тестирование.

Демонстрация работы:

input:

```
((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 ((2 2) (4 6))))))
```

output:

Программа выводит общее число гирек в указанном бинарном коромысле.

Бинарное коромысло записывается в виде:

(ПЛЕЧО ПЛЕЧО)

Плечо имеет следующий вид:

(ДЛИНА ГРУЗ)

В качестве груза может выступать ещё одно бинарное коромысло или груз (число).

Введите бинарное коромысло (не больше 500 символов):

```
((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 ((2 2) (4 6))))))
```

Введены корректные данные.

Ход работы алгоритма:

левое плечо: коромысло:

левое плечо: гиря (+1).

правое плечо: коромысло:

левое плечо: гиря (+1).

правое плечо: гиря (+1).

правое плечо: коромысло:

левое плечо: гиря (+1).

правое плечо: коромысло:

левое плечо: гиря (+1).

правое плечо: гиря (+1).

Общее количество гирек: 6.

| Входные данные  | Выходные данные   |
|---|---|
| ((5 3) (1 2))   | Общее количество гирек: 2.  |
| ((2 ((13 5) (8 1))) (4 5))  | Общее количество гирек: 3.  |
| ((5 ((142 67) (6 7))) (1 ((99 66) (1 1))))                        | Общее количество гирек: 4.  |
| ((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 19))))            | Общее количество гирек: 5.  |
| ((5 2) (5 ((2 19) (14 6))))                                       | Общее количество гирек: 3.  |
| ((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 ((2 2) (4 6)))))) | Общее количество гирек: 6.  |
| ((1 1)(1))  | Ошибка! Вы ввели некорректные данные:<br>Символ №7 - '('.<br>Ожидался пробел.           |
| ((11adsa 5) (4 3))  | Ошибка! Вы ввели некорректные данные:<br>Символ №5 - 'a'.<br>Ожидался пробел.           |
| ds((1 6) (4 3))   | Ошибка! Вы ввели некорректные данные:<br>Символ №1 - 'd'.<br>Ожидался символ - '('.     |
| ((1 6) (4 3)))))))))  | Ошибка! Вы ввели некорректные данные:<br>После символа №12 присутствуют лишние символы. |
| ((1 6) (4 3)dasda)  | Ошибка! Вы ввели некорректные данные:<br>После символа №12 присутствуют лишние символы. |
| ((dsa1 6) (4 3))  | Ошибка! Вы ввели некорректные данные:<br>Символ №3 - 'd'.                               |



## Приложение Б. Код программы.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define N 501

// Бинарное коромысло
typedef struct BinCor{
    unsigned int length1;
    unsigned int length2;
    unsigned int weight1;
    unsigned int weight2;
    struct BinCor * cor_1;
    struct BinCor * cor_2;
} BinCor;

// вывод сообщений об ошибках
void errorMessage(int error_number, char * str, int index){
    printf("\nОшибка! Вы ввели некорректные данные:\n");
    switch (error_number){
        case 1:
            printf("Символ №%d - '%c'.\n", index+1, str[index]);
            printf("Ожидался символ - '('.\n");
            break;
        case 2:
            printf("Символ №%d - '%c'.\n", index+1, str[index]);
            printf("Ожидался символ - ')'.\n");
            break;
        case 3:
            printf("Символ №%d - '%c'.\n", index+1, str[index]);
            printf("Ожидалось значение от 1 до 9.\n");
```

```

        break;
    case 4:
        printf("Символ №%d - '%c'.\n", index+1, str[index]);
        printf("Ожидался пробел.\n");
        break;
    case 5:
        printf("Символ №%d - '%c'.\n", index+1, str[index]);
        printf("Отсутствует закрывающая скобка.\n");
        break;
    case 6:
        printf("Символ №%d - '%c'.\n", index+1, str[index]);
        printf("Ожидалось значение от 1 до 9 или '('\n");
        break;
    case 7:
        printf("После символа №%d присутствуют лишние символы.\n", index+1);
        break;
    }
}

```

```

BinCor* initBinCorElement(){ // Инициализация элемента списка
    BinCor * element = (BinCor*)malloc(sizeof(BinCor));
    element->length1 = 0;
    element->length2 = 0;
    element->weight1 = 0;
    element->weight2 = 0;
    element->cor_1 = NULL;
    element->cor_2 = NULL;
    return element;
}

```

```

// считывание и создание бинарного коромысла
// функция возвращает 1, если возникла ошибка
int readBinCorElement(char * str, int index_1, int index_2, BinCor ** element){
    *element = initBinCorElement();

    if (str[index_1++] != '('){
        errorMessage(1, str, index_1 - 1);
        return 1;
    }
}

```

```

if (str[index_2--] != ')'){
    errorMessage(2, str, index_2 + 1);
    return 1;
}

// считывание левого плеча
if (str[index_1++] != '('){ // открывающая скобка левого плеча
    errorMessage(1, str, index_1 - 1);
    return 1;
}
// первое число(длина)
if (!isdigit(str[index_1]) || str[index_1] == '0'){
    errorMessage(3, str, index_1);
    return 1;
}
while(1){ // считывание числа
    if (isdigit(str[index_1])){
        (*element)->length1 = (*element)->length1 * 10 + str[index_1] - '0';
        index_1++;
    } else {
        break;
    }
}
if (str[index_1++] != ' '){ // пробел после первого числа
    errorMessage(4, str, index_1 - 1);
    return 1;
}
if (isdigit(str[index_1]) && str[index_1] != 0){ // случай, если гирька
    while(1){
        if (isdigit(str[index_1])){
            (*element)->weight1 = (*element)->weight1 * 10 + str[index_1] - '0';
            index_1++;
        } else {
            break;
        }
    }
}
} else if (str[index_1] == '('){ // случай, если ещё одно коромысло
    int bracket_count = 0;
    int index;
    // поиск закрывающей скобки
    for (index = index_1; index < index_2; index++){
        if (str[index]=='(')
            bracket_count++;
        if (str[index]==')')
            bracket_count--;
    }
}

```

```

        if (bracket_count == 0){
            if ( readBinCorElement(str, index_1, index, &((*element)->cor_1)) ){
                return 1;
            }
            index_1 = index + 1;
            break;
        }
    }
    if (bracket_count != 0){
        errorMessage(5, str, index_1);
        return 1;
    }
} else {
    errorMessage(6, str, index_1);
    return 1;
}
if (str[index_1++] != ')'){ // закрывающая скобка левого плеча
    errorMessage(2, str, index_1 - 1);
    return 1;
}

if (str[index_1++] != ' '){ // пробел между плечами
    errorMessage(4, str, index_1 - 1);
    return 1;
}

// считывание правого плеча
if (str[index_1++] != '('){ // открывающая скобка правого плеча
    errorMessage(1, str, index_1 - 1);
    return 1;
}
// первое число(длина)
if (!isdigit(str[index_1]) || str[index_1] == 0){
    errorMessage(3, str, index_1);
    return 1;
}
while(1){ // считывание числа
    if (isdigit(str[index_1])){
        (*element)->length2 = (*element)->length2 * 10 + str[index_1] - '0';
        index_1++;
    } else {
        break;
    }
}

```

```

}
if (str[index_1++] != ' '){ // пробел после первого числа
    errorMessage(4, str, index_1 - 1);
    return 1;
}
if (isdigit(str[index_1]) && str[index_1] != '0'){ // случай, если гирька
    while(1){
        if (isdigit(str[index_1])){
            (*element)->weight2 = (*element)->weight2 * 10 + str[index_1] - '0';
            index_1++;
        } else {
            break;
        }
    }
} else if (str[index_1] == '('){ // случай, если ещё одно коромысло
    int bracket_count = 0;
    int index;
    // поиск закрывающей скобки
    for (index = index_1; index < index_2; index++){
        if (str[index]=='(')
            bracket_count++;
        if (str[index]==')')
            bracket_count--;
        if (bracket_count == 0){
            if ( readBinCorElement(str, index_1, index, &((*element)->cor_2)) ){
                return 1;
            }
            index_1 = index + 1;
            break;
        }
    }
    if (bracket_count != 0){
        errorMessage(5, str, index_1);
        return 1;
    }
} else {
    errorMessage(6, str, index_1);
    return 1;
}
if (str[index_1] != ')'){ // закрывающая скобка левого плеча
    errorMessage(2, str, index_1);
    return 1;
}

// проверка на лишние символы

```

```

    if (index_2 != index_1){
        errorMessage(7, str, index_1);
        return 1;
    }

    return 0;
}

```

```

// создание бинарного коромысла
// функция возвращает 1, если возникла ошибка
int createBinCor(char * str, BinCor ** binCor){
    int index_1 = 0;
    int index_2 = strlen(str) - 2;

    return readBinCorElement(str, index_1, index_2, binCor);
}

```

```

// Возвращаемое значение равно количеству всех
// гирек в заданном бинарном коромысле.
unsigned int numbers(const BinCor binCor, int deep_of_recursion){
    int result = 0;

    for (int i = 0; i < deep_of_recursion; i++)
        printf("    ");
    printf("левое плечо: ");
    if (binCor.cor_1 == NULL){
        printf("гиря (+1).\n");
        result++;
    }
    else {
        printf("коромысло:\n");
        result += numbers(*(binCor.cor_1), deep_of_recursion+1);
    }

    for (int i = 0; i < deep_of_recursion; i++)
        printf("    ");
    printf("правое плечо: ");
    if (binCor.cor_2 == NULL){
        printf("гиря (+1).\n");
    }
}

```

```

        result++;
    }
    else{
        printf("коромысло:\n");
        result += numbers(*(binCor.cor_2), deep_of_recursion+1);
    }

    return result;
}

```

```

// очистка памяти
void free_memory(BinCor * binCor){
    if (binCor != NULL){
        free(binCor->cor_1);
        free(binCor->cor_2);
    }
    free(binCor);
}

```

```

int main(void)
{
    char str[N]; // строка для ввода
    BinCor * binCor = NULL; // бинарное коромысло

    // начало считывания и обработки данных
    printf("\nПрограмма выводит общее число гирек в указанном бинарном коромысле.\n");
    printf("\nБинарное коромысло записывается в виде:\n");
    printf("(ПЛЕЧО ПЛЕЧО)\n");
    printf("Плечо имеет следующий вид:\n");
    printf("(ДЛИНА ГРУЗ)\n");
    printf("В качестве груза может выступать ещё одно бинарное коромысло или груз\n(число).\n");
    printf("\nВведите бинарное коромысло (не больше 500 символов): ");
    fgets(str, N, stdin);

    // обработка данных и проверка на ошибки

```

```

if (createBinCor(str, &binCor)){
    printf("Программа завершила работу.\n\n");
    free_memory(binCor);
    return 0;
}
printf("\nВведены корректные данные.\n\n");
// конец считывания и обработки данных

// вывод результата
printf("Ход работы алгоритма:\n\n");
printf("\nОбщее количество гирек: %u.\n\n", numbers(*binCor, 1));

free_memory(binCor);

return 0;
}

```