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. 7383 _____ Васильев А.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы

Ознакомиться с иерархическими списками, изучить применение иерархических списков 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 представлен на рис. 1.

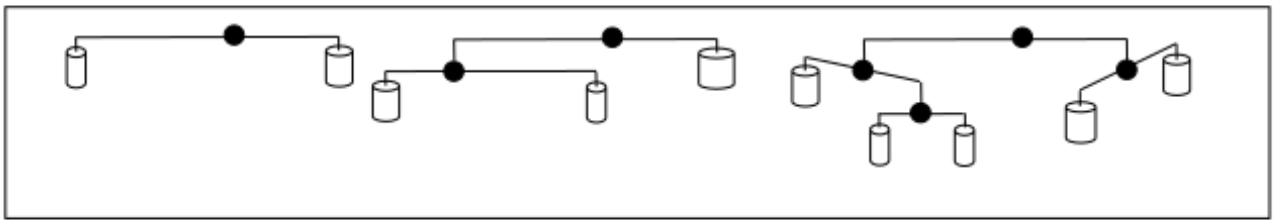


Рисунок 1 — Способ представления бинарного коромыслаВ соответствии с данным определением бинарного коромысла представим бинарное коромысло (БинКор) списком из двух элементов:

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 — это специального вида иерархический список из натуральных чисел.

Постановка задачи

Подсчитать общую длину всех плеч заданного бинарного коромысла bk. Для этого ввести рекурсивную функцию

```
short length(const БинКор bk).
```

Выполнение работы

Программа написана на языке программирования C.

В начале работы происходит ввод данных. Выбор способа ввода данных предоставляется пользователю. Вводится строка, которая является бинарным коромыслом, после чего происходит обработка строки и создание бинарного коромысла, для чего вызывается функция `createBinCor()`. Если введенные данные некорректны, то программа выводит соответствующую ошибку и завершает работу. В случае ввода корректных данных происходит дальнейший вызов функции `length()`, которая выводит на экран общую длину всех плеч бинарного коромысла.

Текст программы представлен в приложении Б.

Описание функций

1. `int createBinCor(char *str, BinCor **binCor)`

Функция принимает указатель на строку, в которую записано бинарное коромысло и указатель на структуру `BinCor` (бинарное коромысло). Функция устанавливает начальный и конечный индексы на первый и последний символы полученной строки, после чего передает эти данные функции `readBinCorElement()`, которая обрабатывает строку и создает 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.

2. `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, содержащемся в указателе, на который указывает `element.` Если груз плеча является еще одним бинарным коромыслом, то происходит рекурсивный вызов функции. При возникновении в какой-то момент ошибки вызывается функция `errorMassage()`, которая выводит на экран сообщение об ошибке, после чего функция завершает свою работу.

Параметры:

`str:` указатель на строку с бинарным коромыслом;

`index_1:` индекс, с которого надо начать обработку строки `str`;

`index_2:` индекс, на котором нужно закончить обработку строки `str`;

`binCor:` указатель на указатель на структуру данных `BinCor`, содержащий адрес, куда будет заносится результат обработки строки `str`.

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

3. `short length(const BinCor binCor, int deep_of_recursion)`

Функция принимает структуру данных `BinCor` и целочисленное значение, указывающее на глубину рекурсии. Функция считает длину всех плеч в бинарном коромысле `binCor`.

Параметры:

`binCor:` бинарное коромысло, в котором считается длина плеч;

`deep_of_recursion:` глубина рекурсии (необходимо для вывода работы алгоритма на экран).

Возвращаемое значение: длина всех плеч в бинарном коромысле.

Тестирование программы

Было создано несколько тестов для проверки работы программы. Помимо тестов, демонстрирующих работу алгоритма, были написаны тесты, содержащие некорректные входные данные, для демонстрации вывода сообщений об ошибках введенных данных. Тестовые случаи представлены в приложении А.

Выводы

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

ПРИЛОЖЕНИЕ А

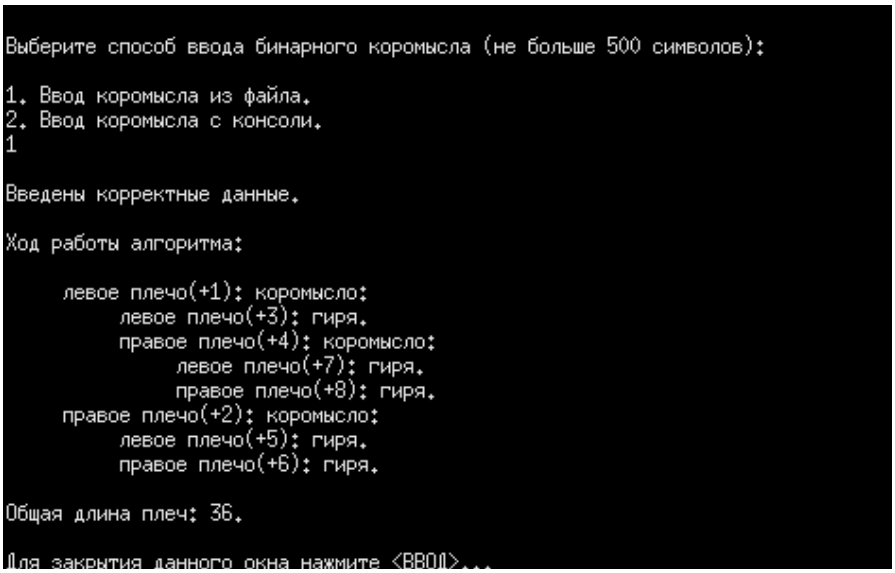
Тестирование

Демонстрация работы:

Входные данные:

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

Результат работы программы:



Входные данные	Выходные данные
((5 3) (1 2))	Общая длина плеч: 6
((2 ((13 5) (8 1))) (4 5))	Общая длина плеч: 27
((5 ((142 67) (6 7))) (1 ((99 66) (1 1))))	Общая длина плеч: 254
((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 19))))	Общая длина плеч: 73
((5 2) (5 ((2 19) (14 6))))	Общая длина плеч: 26
((13 ((11 2) (2 ((1 1) (7 3))))) (19 ((3 2) (17 ((2 2) (4 6))))))	Общая длина плеч: 79
((1 1)(1))	Ошибка! Вы ввели некорректные данные: Символ №7 — '('. Ожидался пробел.
((11adsa 5) (4 3))	Ошибка! Вы ввели некорректные данные: символ №5 — 'a'. Ожидался пробел.

ds((1 6) (4 3))	Ошибка! Вы ввели некорректные данные: символ №1 — 'd'. Ожидался символ - '('.
((1 6) (4 3))))))))))	Ошибка! Вы ввели некорректные данные: после символа №12 присутствуют лишние символы.
((1 6) (4 3)dasda)	Ошибка! Вы ввели некорректные данные: после символа №12 присутствуют лишние символы.
((dsa1 6) (4 3))	Ошибка! Вы ввели некорректные данные: Символ №3 - 'd'. Ожидалось значение от 1 до 9.

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 length_left;
    unsigned int length_right;
    unsigned int weight_left;
    unsigned int weight_right;
    struct BinCor * cor_1;
    struct BinCor * cor_2;
} BinCor;

// вывод сообщений об ошибках
void errorMassage(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->length_left = 0;
    element->length_right = 0;
    element->weight_left = 0;
    element->weight_right = 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++] != '(')
    {
        errorMassage(1, str, index_1 - 1);
        return 1;
    }
    if (str[index_2--] != ')')
    {
        errorMassage(2, str, index_2 + 1);
        return 1;
    }

    // считывание левого плеча
    if (str[index_1++] != '(')// открывающая скобка левого плеча
    {
        errorMassage(1, str, index_1 - 1);
        return 1;
    }
    // первое число(длина)
    if (!isdigit(str[index_1]) || str[index_1] == '0')
    {
        errorMassage(3, str, index_1);
        return 1;
    }

    while(1)// считывание числа
    {
        if (isdigit(str[index_1]))
        {
            (*element)->length_left = (*element)->length_left * 10 +
str[index_1] - '0';
            index_1++;
        }
        else
            break;
    }

    if (str[index_1++] != ' ')// пробел после первого числа
    {
        errorMassage(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)->weight_left = (*element)->weight_left * 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)
        {
            errorMassage(5, str, index_1);
            return 1;
        }
    }
        else
        {
        errorMassage(6, str, index_1);
        return 1;
        }

    if (str[index_1++] != ')')// закрывающая скобка левого плеча
    {
        errorMassage(2, str, index_1 - 1);
        return 1;
    }

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

    // считывание правого плеча
    if (str[index_1++] != '(')// открывающая скобка прового плеча
    {
        errorMassage(1, str, index_1 - 1);
```

```
        return 1;
    }

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

    if (str[index_1++] != ' ')// пробел после первого числа
    {
        errorMassage(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)->weight_right = (*element)->weight_right * 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)
        {
            errorMassage(5, str, index_1);
            return 1;
        }
    }
        else
        {
        errorMassage(6, str, index_1);
        return 1;
        }
    if (str[index_1] != ')')// закрывающая скобка левого плеча
    {
        errorMassage(2, str, index_1);
        return 1;
    }

    // проверка на лишние символы
    if (index_2 != index_1)
    {
        errorMassage(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 длине всех плеч
//в заданном бинарном коромысле
short length(const BinCor binCor, int deep_of_recursion)
{
    int result = 0;

    for (int i = 0; i < deep_of_recursion; i++)
        printf("     ");
    printf("левое плечо(+%d): ", binCor.length_left);
    result+=binCor.length_left;

    if (binCor.cor_1 == NULL)
    {
        printf("гиря.\n");
    }
    else
    {
        printf("коромысло: \n");
        result += length(*(binCor.cor_1), deep_of_recursion+1);
    }
```

```
    for (int i = 0; i < deep_of_recursion; i++)
        printf("     ");
    printf("правое плечо(+%d): ", binCor.length_right);
    result+=binCor.length_right;

    if (binCor.cor_2 == NULL)
    {
        printf("гиря.\n");
    }
    else
    {
        printf("коромысло:\n");
        result += length(*(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()
{
    char str[N]; // строка для ввода
    BinCor * binCor = NULL; // бинарное коромысло
    int in=0;

    // 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 (число).\n");
    printf("\nВыберите способ ввода бинарного коромысла (не больше 500
символов): \n");
    printf("\n1. Ввод коромысла из файла.");
    printf("\n2. Ввод коромысла с консоли.\n");
    scanf ("%d", &in);

    if (in == 1)
    {
        FILE* file = fopen ("input.txt", "r");
        if (file == NULL)
            perror("Ошибка открытия файла");
        fgets(str, N, file);
        fclose(file);
    }
    if (in == 2)
    {
```

```
        printf("\nВведите бинарное коромысло (не более 500 символов):");
        fgets(str, 2, stdin);
        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("\nОбщая длина плеч: %u.\n\n", length(*binCor, 1));

    free_memory(binCor);

    return 0;
}
```