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. 7381 Минуллин М. А.

Преподаватель Фирсов М. А.

Санкт-Петербург

2018

Задание:

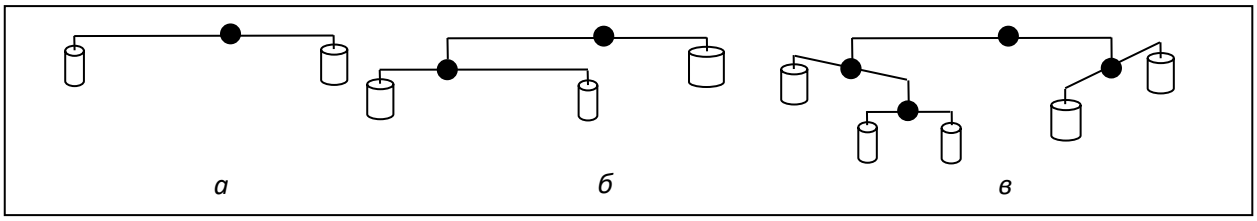
3. Подсчитать общую длину всех плеч заданного бинарного коромысла bk. Для этого ввести рекурсивную функцию

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

Пояснение задания:

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.



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.

Необходимо написать программу, на вход которой подаётся строка, а на выходе получается информация о корректности входной строки и, если строка корректна, вывод информации о бинарном коромысле.

Описание алгоритма:

Линейно просматривается входная строка. При разборе каждой из грамматик используется разная проверка. Для коромысла делается проверка наличия открывающейся скобки, возможности получить из дальнейшей последовательности два плеча и наличие закрывающейся скобки после них. Для плеча делается проверка наличия открывающейся скобки, возможности считать число и разветвление алгоритма, в зависимости от условия:

если встречена открывающаяся скобка, то дальше считывается новое коромысло, иначе считывается гирька, после чего проверяется наличие закрывающейся скобки. Если на каких-либо этапах возникает ошибка, то дерево считается не валидным, в консоль выводятся сообщения о возникших ошибках.

Описание функций:

`void print_tabs(size_t tabs_count);`

- функция используется для создания отступа в консоли символами табуляции.

`size_t tabs_count` – количество знаков табуляции, которые необходимо вывести.

Возвращаемое значение: функция ничего не возвращает.

`size_t read_number(const std::string& str, size_t& index);`

- функция используется для чтения числа из строки.

`const std::string& str` - строка, из которой считывается число.

`size_t& index` - индекс, откуда следует начать считывание.

Возвращаемое значение: число, считанное из строки.

`void error_handler(const std::string& str, size_t index, int error_code, size_t depth);`

- функция-обработчик ошибок, возникших при анализе входных данных. Используется для вывода сообщений об ошибках.

`const std::string& str` - анализируемая строка.

`size_t index` - индекс строки, в котором возникла ошибка.

`int error_code` - код ошибки.

`size_t depth` – размер отступа для вывода сообщений.

Возвращаемое значение: функция ничего не возвращает.

`enum side_t { LEFT, RIGHT };`

- перечисление для передачи информации о том, какое коромысло считывается (левое или правое)

`class Rocker`

- класс, хранящий информацию о бинарном коромысле.

`size_t length1` - длина левого плеча.

`size_t length2` - длина правого плеча.

`Rocker *left` - указатель на левое бинарное коромысло.

`Rocker *right` - указатель на правое бинарное коромысло.

`size_t weight1` - масса левой гирьки (если отсутствует левое бинарное коромысло).

`size_t weight2` - масса правой гирьки (если отсутствует правое бинарное коромысло).

`bool is_valid` - флаг, обозначающий корректно ли построено дерево.

`Rocker();` - базовый конструктор класса, устанавливает все поля в ноль.

`Rocker(const std::string& str);` - конструктор, строящий дерево по заданной входной строке.

`const std::string& str` - входная строка.

`bool validity();` - функция для определения корректности построения коромысла.

`size_t length();` - функция для подсчёта суммы длин всех плеч коромысла.

`size_t weight();` - функция для подсчёта суммы масс всех гирек.

`size_t dumbbells();` - функция для подсчёта количества гирек.

`bool balanced();` - функция для определения сбалансированности коромысла.

`void print_info();` - функция, выводящая информацию о коромысле.

`~Rocker();` - деструктор коромысла.

`bool rocker(const std::string& str, size_t& index, size_t depth);`
- функция для разбора грамматики БинКор.

`const std::string& str` - просматриваемая строка.

`size_t& index` - индекс текущего просматриваемого символа строки.

`size_t depth` - глубина рекурсии при разборе коромысла.

Возвращаемое значение: логический тип, успешно ли разобрана лексема.

`bool arm(const std::string& str, size_t& index, side_t side, size_t depth);`
- функция для разбора грамматики Плечо.

`const std::string& str` - просматриваемая строка.

`size_t& index` - индекс текущего просматриваемого символа строки.

`size_t depth` - глубина рекурсии при разборе коромысла.

`side_t side` - анализируемое плечо коромысла (левое или правое).

Возвращаемое значение: логический тип, успешно ли разобрана лексема.

`bool cargo(const std::string& str, size_t& index, side_t side, size_t depth);`
- функция для разбора грамматики Груз.

`const std::string& str` - просматриваемая строка.

`size_t& index` - индекс текущего просматриваемого символа строки.

`size_t depth` - глубина рекурсии при разборе коромысла.

`side_t side` - анализируемое плечо коромысла (левое или правое).

Возвращаемое значение: логический тип, успешно ли разобрана лексема.

Тестирование:

Для проверки работы программы был использован скрипт с прошлой лабораторной работы. Были написаны тесты с корректными и некорректными входными данными. В таблице ниже приведены результаты тестирования. Для первого теста представлен полный вывод, для остальных только с основной информацией.

№	Входные данные	Выходные данные
1	((1 1)(1 1))	correct test: "test2.txt" input: ((1 1)(1 1)) call .rocker [index: 0 , symbol: (] call .arm [index: 1 , symbol: (] call .cargo [index: 4 , symbol: 1] llac .cargo llac .arm call .arm [index: 6 , symbol: (] call .cargo [index: 9 , symbol: 1] llac .cargo llac .arm llac .rocker 2 : summary length of arms 2 : summary weight of cargoes 2 : count of dumbbells yes : balancing status result: success
2	((2 8)(4 4))	6 : summary length of arms 12 : summary weight of cargoes 2 : count of dumbbells yes : balancing status result: success
3	(()())	result: failure (error_code 3, 4)
4	((2 8)(4))	result: failure (error_code 4)
5	((abc)(def))	result: failure (error_code 1, 2, 3, 4)
6	((-1 1)(1 0))	result: failure (error_code 1, 2, 3, 4)
7	((1 1)(1 1))((1 1)(1 1))	result: failure (error_code 5);

Вывод:

В процессе выполнения лабораторной работы были поли получены знания и навыки по ООП, деревьям, иерархическим спискам, рекурсивным функциям, bash-скриптам и автоматизации тестирования. Работа была написана на C++.

ПРИЛОЖЕНИЯ

Приложение А. Код main.cpp

```
#include "rocker.hpp"

#include <iostream>

int main() {
    std::string str;
    std::getline(std::cin, str);

    std::cout << "input: " << str << std::endl;
    Rocker rocker(str);

    if (rocker.validity())
        rocker.print_info();

    std::cout << "result: ";
    if (rocker.validity())
        std::cout << "success";
    else
        std::cout << "failure";
    std::cout << std::endl << std::endl;

    return 0;
}
```

Приложение Б. Код rocker.hpp

```
#ifndef __ROCKER_HPP__
#define __ROCKER_HPP__

#include <string>

enum side_t { LEFT, RIGHT };
```

```
  void print_tabs(size_t tabs_count);


  size_t read_number(const std::string& str, size_t& index);


  void error_handler(const std::string& str, size_t index, int error_code,
size_t depth);


  class Rocker {
  private:
      size_t length1;
      size_t length2;
      size_t weight1;
      size_t weight2;
      Rocker *left;
      Rocker *right;
      bool is_valid;


      bool rocker(const std::string& str, size_t& index, size_t depth);
      bool arm(const std::string& str, size_t& index, side_t side, size_t
depth);
      bool cargo(const std::string& str, size_t& index, side_t side, size_t
depth);
  public:
      Rocker();
      Rocker(const std::string& str);


      bool validity();
      void print_info();


      size_t length();
      size_t weight();
      size_t dumbbells();
```

```
    bool balanced();


    ~Rocker();
};


#endif
```

Приложение В. Код rocker.cpp

```
#include "rocker.hpp"

#include <iostream>

void print_tabs(size_t tabs_count) {
    for (size_t i = 0; i < tabs_count; ++i)
        std::cout << '\t';
}

size_t read_number(const std::string& str, size_t& index) {
    size_t result = 0;
    while (isdigit(str[index])) {
        result *= 10;
        result += str[index] - '0';
        ++index;
    }
    return result;
}

void error_handler(int error_code, size_t index, const std::string& str,
size_t depth) {
    if (error_code == 0)
        return;
    print_tabs(depth);
    std::cout << "ERROR CODE " << error_code << " at index " << index << "
(symbol '" << str[index] << "'): ";
    switch (error_code) {
    case 1:
        std::cout << "'(' expected";
        break;
    case 2:
        std::cout << "')' expected";
        break;
    case 3:
        std::cout << "['0'-'9'] expected";
        break;
    case 4:
        std::cout << "'(' or ['0'-'9'] expected";
        break;
    case 5:
```

```
            std::cout << "excess symbols";
            break;
        }
        std::cout << std::endl;
    }
    bool Rocker::rocker(const std::string& str, size_t& index, size_t depth) {
        print_tabs(depth);
        std::cout << "call .rocker [index: " << index << " , symbol: " <<
str[index] << "]" << std::endl;

        bool result = true;

        if (str[index] == '(') {
            ++index;
            // Считывается левое плечо
            if (!arm(str, index, LEFT, depth + 1))
                result = false;
            // Считывается правое плечо
            if (!arm(str, index, RIGHT, depth + 1))
                result = false;

            if (str[index] == ')')
                ++index;
            else {
                // Ожидалась закрывающаяся скобка
                error_handler(2, index, str, depth);
                result = false;
            }
        }
        else {
            // Ожидалась открывающаяся скобка
            error_handler(1, index, str, depth);
            result = false;
        }

        print_tabs(depth);
        std::cout << "llac .rocker" << std::endl;

        return result;
    }

    bool Rocker::arm(const std::string& str, size_t& index, side_t side, size_t
depth) {
        print_tabs(depth);
        std::cout << "call .arm [index: " << index << " , symbol: " << str[index]
<< "]" << std::endl;

        bool result = true;

        if (str[index] == '(')
            ++index;
```

```
        else {

            error_handler(1, index, str, depth);
            result = false;
        }
        if (!isdigit(str[index])) {
            // Ожидалась цифра
            error_handler(3, index, str, depth);
            result = false;
        }
        size_t length = read_number(str, index);

        if (side == LEFT)
            length1 = length;
        if (side == RIGHT)
            length2 = length;

        if (str[index] == ' ')
            ++index;
        if (!cargo(str, index, side, depth + 1))
            result = false;

        if (str[index] == ')')
            ++index;
        else {
            error_handler(2, index, str, depth);
            result = false;
        }

        print_tabs(depth);
        std::cout << "llac .arm" << std::endl;

        return result;
    }

    bool Rocker::cargo(const std::string& str, size_t& index, side_t side, size_t
depth) {
        print_tabs(depth);
        std::cout << "call .cargo [index: " << index << " , symbol: " <<
str[index] << "]" << std::endl;

        bool result = true;

        if (isdigit(str[index])) {
            size_t weight = read_number(str, index);
            if (side == LEFT)
                weight1 = weight;
            if (side == RIGHT)
                weight2 = weight;
        }
        else if (str[index] == '(') {
```

```
        if (side == LEFT) {
            left = new Rocker;
            if (!(left->rocker(str, index, depth + 1)))
                result = false;
        }
        if (side == RIGHT) {
            right = new Rocker;
            if (!right->rocker(str, index, depth + 1))
                result = false;
        }
    }
    else {
        // Ожидалась открывающаяся скобка или цифра
        error_handler(4, index, str, depth);
        result = false;
    }

    print_tabs(depth);
    std::cout << "llac .cargo" << std::endl;

    return result;
}

Rocker::Rocker() {
    length1 = 0;
    length2 = 0;
    weight1 = 0;
    weight2 = 0;
    left = right = NULL;
    is_valid = false;
}

Rocker::Rocker(const std::string& str) {
    length1 = 0;
    length2 = 0;
    weight1 = 0;
    weight2 = 0;
    left = right = NULL;
    is_valid = false;

    size_t index = 0;
    if (rocker(str, index, 0)) {
        is_valid = true;
        if (index != str.length()) {
            is_valid = false;
            // Лишние символы в строке
            error_handler(5, index, str, 0);
        }
    }
}
```

```
  bool Rocker::validity() {
      return is_valid;
  }

  void Rocker::print_info() {
      std::cout <<    length() << "\t : summary length of arms" << std::endl;
      std::cout <<    weight() << "\t : summary weight of cargoes" << std::endl;
      std::cout << dumbbells() << "\t : count of dumbbells" << std::endl;
      std::cout << (balanced() ? "yes" : "no") << "\t : balancing status" <<
std::endl;
  }

  size_t Rocker::length() {
      size_t result = length1 + length2;

      if (left)
          result += left->length();
      if (right)
          result += right->length();

      return result;
  }

  size_t Rocker::weight() {
      size_t result = weight1 + weight2;

      if (left)
          result += left->weight();
      if (right)
          result += right->weight();

      return result;
  }

  size_t Rocker::dumbbells() {
      size_t result = 0;

      if (left)
          result += left->dumbbells();
      else
          ++result;

      if (right)
          result += right->dumbbells();
      else
          ++result;

      return result;
  }

  bool Rocker::balanced() {
```

```
    // Вращающий момент левого поддерева
    size_t torque1 = (length1 * (left ? left->weight() : weight1));
    // Вращающий момент правого поддерева
    size_t torque2 = (length2 * (right ? right->weight() : weight2));
    return (torque1 == torque2);
}

Rocker::~Rocker() {
    if (left)
        delete left;
    if (right)
        delete right;
}
```

Приложение Г. Код Makefile

```
CODE    = ./source/

OBJ     = main.o rocker.o

EXE     = binary_rocker

CXX     = g++

CFLAGS  = -Wall -Wextra -c


all: $(OBJ)

    $(CXX) $(OBJ) -o $(EXE)


main.o: $(CODE)main.cpp

    $(CXX) $(CFLAGS) $(CODE)main.cpp


rocker.o: $(CODE)rocker.hpp $(CODE)rocker.cpp

    $(CXX) $(CFLAGS) $(CODE)rocker.cpp


clean:

    rm $(OBJ)
```

Приложение Д. Код checker.sh

```
#!/bin/bash

make

if [ -f "result.txt" ]; then

    rm result.txt
```

```
fi
touch result.txt
for i in $(ls tests/correct); do
    echo "running test: \"tests/correct/$i\" ";
    sleep 0.13s;
    echo "correct test: \"$i\"" >>result.txt;
    ./binary_rocker <tests/correct/$i >>result.txt;
done;
for i in $(ls tests/incorrect); do
    echo "running test: \"tests/incorrect/$i\" ";
    sleep 0.13s;
    echo "incorrect test: \"$i\"" >>result.txt;
    ./binary_rocker <tests/incorrect/$i >>result.txt;
done;
make clean
```