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: Рекурсия

Студент гр. 7383 _____ Власов Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Содержание

Цель работы

Цель работы: познакомиться с иерархическими списками, бинарными деревьями и бинарными коромыслами, получить навык их представления на языке программирования C++.

Формулировка задачи: Вариант 4. Говорят, что бинарное коромысло сбалансировано, если момент вращения, действующий на его левое плечо, равен моменту вращения, действующему на правое плечо (то есть длина левого стержня, умноженная на вес груза, свисающего с него, равна соответствующему произведению для правой стороны), и если все подкоромысла, свисающие с его плеч, также сбалансированы.

Написать рекурсивную функцию или процедуру, которая проверяет заданное коромысло на сбалансированность (выдает значение true, если коромысло сбалансировано, и false в противном случае).

Реализация задачи

Для решения поставленной задачи было принято по требованию преподавателя создать класс LIST для хранения бинарного коромысла в виде иерархического списка.

```
class LIST{
private:
    bool flag; //true if element else atom
    union CONTENT{
     struct ATOM{
          LIST* elem;
          LIST* next;
     } atom;
     struct ELEM{
          bool balanced;
          int lever_length;
          int mass;
          LIST* down;
     } elem;
    } content;
public:
    LIST(stringstream& s, int side = 1); // 0 - elem, 1 - left, 2
- right
    int lever (stringstream &s);
    int Balanced();
};
```

В программе предусмотрен ввод данных из файла, либо вручную. За бинарное коромысло было принято считать следующую конструкцию:

$$((l_1\ m_1)\ (l_2\ m_2)), \qquad (1)$$

где l_1 и l_2 – длины плеч (натуральные числа), а m_1 и m_2 – массы, представленные натуральными числами, либо конструкцией (1).

Конструктор класса и метод int lever (stringstream &s) последовательно обрабатывают строку заполняя содержимое структур.

Метод int Balanced() рекурсивно вызывает себя сначала во вложенных бинарных коромыслах, а после сравнивает значение на левом и правом плечах коромысла. Если бинарное коромысло не сбалансированно, метод возвращает значение -1, иначе суммарный вес текущего коромысла.

Исходный код программы представлен в приложении Б.

Тестирование

1. Процесс тестирования

Программа собрана в операционной системе Ubuntu 18.04.1 LTS bionic компилятором g++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0. В других ОС и компиляторах тестирование не проводилось.

2. Результаты тестирования

Тестовые случаи представлены в Приложении А.

По результатам тестирования было выявлено, что программа возвращает корректный результат на любых входных данных.

Вывод

В ходе выполнения данной работы были изучены иерархические списки, бинарные деревья и бинарные коромысла. Была написана создана структура иерархического списка для хранения бинарного коромысла и написана программа для проверки бинарного коромысла на сбалансированность.

ПРИЛОЖЕНИЕ А: ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод	Корректность
((1 1)(1 1))	Balanced	Верно
((1 2)(2 1))	Balanced	Верно
((1 1)(2 2))	Not balanced	Верно
((1 1)(1 ((1 1)(1 1))))	Not balanced	Верно
((1 2)(1 ((1 1)(1 1))))	Balanced	Верно
((1 1)(0 2))	Not balanced	Верно
((1 1)(1 ((1 1)(1 ((1 1)(1 1))))))	Not balanced	Верно
((1 1)(1 ((1 1)(1((1 1)(1((1 1)(1 1)))))))	Not balanced	Верно
((2 1)(1 ((1 1)(1 1))))	Balanced	Верно

ПРИЛОЖЕНИЕ Б: ИСХОДНЫЙ КОД

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cctype>
#include <string>
using namespace std;
class LIST;
struct ATOM{
    LIST* elem;
    LIST* next;
};
struct ELEM{
    bool balanced;
    int lever_length;
    int mass;
    LIST* down;
};
union CONTENT{
    ATOM atom;
    ELEM elem;
};
class LIST{
private:
    bool flag; //true if element else atom
    union CONTENT content;

public:
    LIST(stringstream& s, int side = 1); // 0 - elem, 1 - left, 2 -
right
    int lever (stringstream &s);
    int Balanced();
    ~LIST();
};
LIST::LIST(stringstream& s, int side)
{
    char ch;
    switch (side)
    {
        case 0:
            flag = true;
            content.elem.down = NULL;
            if (s.peek() == '(')
```

```
                s >> ch; // remove (
            lever(s);
            s >> ch; // remove )
            break;
        case 1:
            flag = false;
            content.atom.elem = new LIST(s, 0);
            content.atom.next = new LIST(s, 2);
            break;
        case 2:
            flag = false;
            content.atom.elem = new LIST(s, 0);
            content.atom.next = NULL;
            break;
    }
}
int LIST::lever(stringstream& s)
{
    char ch;
    if (s.peek() == '(')
        s >> ch; // remove '('
    s >> content.elem.lever_length;
    if (s.peek() == '(')
    {
        content.elem.down = new LIST(s);
    }
    else
    {
        s >> content.elem.mass;
    }
    s >> ch; // remove ')'
}

int LIST::Balanced()
{
    int m;
    if (flag == false)
    {
        if (content.atom.next)
        {
            return ((m = (content.atom.next)->Balanced()) ==
                        (content.atom.elem)->Balanced() &&
                        m != -1 ? 2*m : -1);
        }
```

```
        else
            return (content.atom.elem)->Balanced();
    }
    else
    {
        if (content.elem.down)
        {
            if ((m = (content.elem.down)->Balanced()) != -1)
                return m * content.elem.lever_length;
            else
                return -1;
        }
        else
            return content.elem.lever_length * content.elem.mass;
    }
}
LIST::~LIST()
{
    if (flag == false)
    {
        delete content.atom.elem;
        if (content.atom.next)
            delete content.atom.next;
    }
    else if (content.elem.down)
            delete content.elem.down;
}
void run(string str)
{
    stringstream s;
    string str1;
    if (str.size() == 0)
    {
        cout << "The string is empty!" << endl;
        return;
    }
    for (auto el : str)
    {
        if (el == '(')
        {
            if (isspace(str1.back()))
                str1.erase(str1.size()-1, 1);
            str1.push_back(el);
        }
```

```
        else if (el == ' ')
        {
            if (isdigit(str1.back()))
                str1.push_back(el);
        }
        else if (el == ')')
        {
            if (isspace(str1.back()))
                str1.erase(str1.size()-1, 1);
            str1.push_back(el);
        }
        else if (isdigit(el))
        {
            str1.push_back(el);
        }
        else
        {
            cout << "There are some unexpected characters in the
string: " << str << endl;
            return;
        }
    }
    for (int i = 0; i < str1.size(); i++)
    {
        switch (str1[i])
        {
            case '(':
                i++;
                if (str1[i] == '(')
                {
                    i++;
                    if (isdigit(str1[i]))
                        continue;
                }
                cout << "The string doesn't match to the format." <<
endl;
                return;
            case ')':
                i++;
                if (str1[i] == '(')
                {
                    i++;
                    if (isdigit(str1[i]))
                        continue;
```

```
                }
                else if (str[i] == ')')
                    continue;
                cout << "The string doesn't match to the format." <<
endl;
                return;
        }
    }
    cout << str1;
    s << str1;
    LIST list(s);
    cout << (list.Balanced() != -1 ? "Balanced\n" : "Not balanced\n");
}
int main()
{
    int n, c;
    string inp;
    int *el;
    string str, str1;
    while(true)
    {
        cout << "Press 1 to get input from a file\n" <<
                "Press 2 to enter binary beam by yourself\n" <<
                "Press 3 to exit." << endl;
        cin >> inp;
        if (!isdigit(inp[0]))
            continue;
        c = stoi(inp);
        inp.clear();
        switch (c)
        {
            case 1:
                break;
            case 2:
                getline(cin, str); // remove '\n'
                getline(cin, str);
                run(str);
                break;
            case 3:
                return 0;
            default:
                cout << "Something went wrong. try again!" << endl;
                continue;
        }
```

```
        if(c == 1)
        {
            cout << "Enter file name: ";
            cin >> str;
            ifstream f;
            f.open(str);
            if (!f)
            {
                cout << "Unable to open the file!" << endl;
                continue;
            }
            while(!f.eof())
            {
                getline(f, str);
                run(str);
            }
            f.close();
        }
    }
}
```