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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студент гр. 7383 _____ Васильев А.И.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы.

Познакомиться со структурой бинарного дерева и его реализацией на языке программирования C++. Освоить методы работы с бинарными деревьями.

Задание.

Заданы два бинарных дерева b1 и b2 типа BT с произвольным типом элементов. Проверить:

а) подобны ли они (два бинарных дерева подобны, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);

б) равны ли они (два бинарных дерева равны, если они подобны и их соответствующие элементы равны);

в) зеркально подобны ли они (два бинарных дерева зеркально подобны, если они оба пусты либо они оба непусты и для каждого из них левое поддерево одного подобно правому поддереву другого);

г) симметричны ли они (два бинарных дерева симметричны, если они зеркально подобны и их соответствующие элементы равны).

Пояснение к заданию.

Нужно написать программу, принимающую на вход два скобочных представления бинарных деревьев (в сокращенной форме), сравнить их и сделать вывод об их подобности, равенстве, зеркальности и симметричности. Правила сокращенной формы скобочного представления деревьев: (<корень> <непустное БД> <пустое БД>) = (<корень> <непустое БД>); (<корень> <пустое БД> <пустое БД>) = (<корень>); (<пустое БД>) = (#)

Описание алгоритма.

1. Алгоритм создания бинарного деревья.

Происходит посимвольное считывание из строки со скобочной записью бинарного дерева. Первая скобка в строке игнорируется. При встрече символа, он записывается в `root` текущего бинарного дерева. При встрече в записи открывающейся скобки происходит один из двух вариантов: если `flag == true`, то создается левое поддерево, если `flag == false`, то создается правое поддерево. При встрече закрывающейся скобки флаг меняется на `false` и текущим деревом становится родитель текущего дерева. При встрече в записи символа # флаг так же меняется на `false`, но текущее дерево не меняется. Дерево строится с помощью рекурсивной функции, до тех пор, пока не будет прочитана вся строка.

2. Алгоритм сравнения деревьев.

На подобие: сравнение производится рекурсивным способом. На каждой итерации рекурсии проверяется пустота текущего первого и второго дерева, если они пусты одновременно, то эти деревья подобные, если одно из них не пустое, а другое пустое — такие деревья не подобны. Если оба дерева не пустые, то рекурсивно вызывается алгоритм для левых сыновей, затем для правых сыновей.

На равенство: аналогично сравнению на подобие, но, кроме проверок, описанных выше, производится проверка значений в корнях первого и второго дерева. Если они в какой-либо момент оказываются не равны, значит деревья не равны.

На зеркальное подобие: сравнение начинается с первого элемента. Если первое дерево имеет левого (правого) ребенка, а второе дерево — правого (левого), то рекурсивно вызывается алгоритм для левого (правого) ребенка первого и правого (левого) ребенка второго. Иначе деревья не являются зеркальными.

На симметричность: аналогично сравнению на зеркальное подобие, но так же проверяется равенство корней соответствующих деревьев.

Описание функций и структур данных.

Код программы приведен в приложении А.

1. `class BinTree` — в данном классе реализовано бинарное дерево и функции работы с ним.

Содержит поля:

`root` — хранит значение элемента (корня);

`left` — хранит адрес левого ребенка;

`right` — хранит адрес правого ребенка;

Функции, для работы с бинарным деревом, реализованные в классе:

`BASE get_vl()` - возвращает значение корня дерева;

`void set_vl(BASE vl)` — записывает значение `vl` в корень дерева;

`BinTree *get_left()` - возвращает адрес левого ребенка;

`void set_left()` - выделяет память под левого ребенка;

`BinTree *get_right()` - возвращает адрес правого ребенка;

`void set_right()` - выделяет память под правого ребенка;

2. Функция `bool is_open_bracket (char ch)`

Функция принимает на вход символьную переменную, возвращает `true`, если это `'('`, иначе возвращает `false`.

3. Функция `bool is_close_bracket (char ch)`

Функция принимает на вход символьную переменную, возвращает `true`, если это `')'`, иначе возвращает `false`.

4. Функция `bool is_value (char ch)`

Функция принимает на вход символьную переменную, возвращает `true`, если это не скобка и не решетка, иначе возвращает `false`.

5. Функции `bool is_latt (char ch)`

Функция принимает на вход символьную переменную, возвращает `true`, если это `'#'`, иначе возвращает `false`.

6. Функция `bool create_binTree(istream& , BinTree<BASE> *, unsigned i=0, bool flag = false)`

Рекурсивная функция принимает на вход поток `istream` со строкой, в которой написано скобочное представление бинарного дерева, указатель на объект класса `BinTree`, так же `i` — счетчик глубины рекурсии, изначально инициализирован нулем, и флаг. Алгоритм построения бинарного дерево по скобочному представлению описан в пункте 1 раздела «Описание алгоритмов».

7. `void print_binTree(BinTree<BASE>* , unsigned l=0)`

Рекурсивная функция вывода бинарного дерева на консоль. Принимает в качестве аргументов указатель на бинарное дерево, которое требуется вывести и `l` — показывает глубину рекурсии (необходима для корректного вывода на консоль).

8.Функции `bool similar(BinTree<BASE>*, BinTree<BASE>* );`
`bool equal(BinTree<BASE>* , BinTree<BASE>* );`
`bool speculary_similar(BinTree<BASE>* , BinTree<BASE>* );`
`bool symmetry(BinTree<BASE>* , BinTree<BASE>* );`

Функции, проверяющие для двух бинарных деревьев подобие, равенство, зеркальное подобие и симметричность соответственно. Алгоритмы, использованные для проверки вышеперечисленных свойств, приведены в пункте 2 раздела «Описание алгоритмов».

9. Функция `bool analyzer(string bt, int length)`

Функция — синтаксический анализатор. Проверяет правильность скобочных представлений, поступивших на вход программе. В случае ошибки в представлении выводит на экран соответствующее сообщение и возвращает `true`. Если в строке ошибки не обнаружены — возвращает `false`.

10. Головная функция `main()`

Функция создает 2 бинарных дерева, затем считывает данные из

двух файлов со скобочным представлением бинарного дерева, с помощью функции `bool analyzer()` проверяет их на возможные ошибки, в случае обнаруженной ошибки во входных данных, завершает работу программы. Далее с помощью функции `create_binTree()` бинарные деревья записываются в структуру данных и выводятся на консоль функцией `print_binTree()`. Далее вызываются функции, проверяющие свойства бинарных деревьев, и, в зависимости от возвращаемого ими значения, функция выводит сообщения об этих свойствах.

Тестирование.

По результатам тестирования можно судить о том, что поставленная задача была выполнена. Результаты некоторых из тестов приведены в табл.1.

Таблица 1 — входные и выходные данные

Входные данные	Выходные данные	Верный результат?
(q(w)(e)) (q(w)(e))	Подобны Равны Зеркально подобны Не симметричны	Да
(q(e)(w)) (q(w)(e))	Подобны Не равны Зеркально подобны Симметричны	Да
(a(b(c)(d#(e)))) (a#(b(d(e))(c)))	Не подобны Не равны Зеркально подобны Симметричны	Да
(a(b(c)(d#(e)))) (a#(b(c(e))(d))))	Не подобны Не равны Зеркально подобны Не симметричны	Да
(#) (#)	Подобны Равны Зеркально подобны Симметричны	Да

Окончание таблицы 1

(a(b(d)(e))(c(g)(z)))) (a(b(d)(e))(c(g)(z)))	Подобны Равны Зеркально подобны Не симметричны	Да
(a(b(d)(e))(c(g)(z))) (a(c(z)(g))(b(e)(d)))	Подобны Не равны Зеркально подобны Симметричны	Да
(a#(b#(c#(d)))) (a#(b#(c#(d))))	Подобны Равны Не зеркально подобны Не симметричны	Да
(a(b(c(d)))) (d#(c#(b#(a))))	Не подобны Не равны Зеркально подобны Не симметричны	Да
(a(b#(c(d#(e))))) (a#(b(c#(d(e)))))	Не подобны Не равны Зеркально подобны Симметричны	Да
(a(b(d))(c#(e))) (a(c(e))(b#(z)))	Подобны Не равны Зеркально подобны Не симметричны	Да
(a(e)(g)) (a(e)(g#(z#(i))))	Не подобны Не равны Не зеркально подобны Не симметричны	Да
(qqqq)(W)(E) (a)	Ошибка в символе № 3. Ошибка в воде бинарного дерева № 1.	Да
(a(b) (a(b))	Количество открытых и закрытых скобок не равны. Ошибка в воде бинарного дерева № 1.	Да
(a(b)(c(d)(2))) (q)	Ошибка в символе №12. после (должна следовать буква. Ошибка в воде бинарного дерева № 1.	Да

Выводы.

В ходе работы была изучена и закреплена тема бинарных деревьев, изучено представление бинарного дерева в динамической памяти, а также работа с ним.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Файл main.cpp:

```
#include <iostream>
#include <istream>
#include <fstream>
#include <string>
#include <sstream>
#include <bintr.h>
#include <func.h>

using namespace std;

int main()
{
    BinTree<char> *bTree1 = new BinTree<char>;
    BinTree<char> *bTree2 = new BinTree<char>;

    stringbuf str_buf;
    istream is_str(&str_buf);

    ifstream file1 ("bt1.txt");
    string bt1;
    getline (file1, bt1);
    file1.close();

    if (analyzer(bt1, bt1.length()))
    {
        cerr << endl << "Ошибка в вводе бинарного дерева № 1. Программа
аварийно завершила свою работу." << endl;
        return 0;
    }

    str_buf.str(bt1);
    create_binTree(is_str, bTree1);
    is_str.clear();

    cout << "Первое бинарное дерево:" << endl << endl;

    print_binTree(bTree1);

    ifstream file2 ("bt2.txt");
    string bt2;
    getline (file2, bt2);
    file2.close();

    if (analyzer(bt2, bt2.length()))
    {
        cerr << endl << "Ошибка в вводе бинарного дерева № 2. Программа
аварийно завершила свою работу." << endl;
        return 0;
    }

    str_buf.str(bt2);
    create_binTree(is_str, bTree2);
```

```
    is_str.clear();

    cout << endl << "Второе бинарное дерево:" << endl << endl;
    print_binTree(bTree2);

    cout << endl << endl;

    if (similar(bTree1, bTree2))             //подобие
        cout << "Деревья подобны." << endl;
    else
        cout << "Деревья не подобны." << endl;

    if (equal(bTree1, bTree2))                  //равенство
        cout << "Деревья равны." << endl;
    else
        cout << "Деревья не равны." << endl;

    if (speculary_similar(bTree1, bTree2))     //зеркально подобны
        cout << "Деревья зеркально подобны." << endl;
    else
        cout << "Деревья не зеркально подобны." << endl;

    if (symmetry(bTree1, bTree2))     //симметричны
        cout << "Деревья симметричны." << endl;
    else
        cout << "Деревья не симметричны." << endl;

    return 0;
}
```

Файл bintr.h:

```
#ifndef BINTR_H
#define BINTR_H

template<typename BASE>

class BinTree
{
private:
    BASE root;
    BinTree *left;
    BinTree *right;

public:
    BinTree()
    {
        root = 0;
        left = nullptr;
        right = nullptr;
    }

    BASE get_vl()
    { return root;}

    void set_vl(BASE vl)
    {
        root = vl;
    }
```

```
    BinTree *get_left()
    { return left;}

    void set_left()
    { left = new BinTree;}

    BinTree *get_right()
    { return right;}

    void set_right()
    { right = new BinTree;}

    ~BinTree()
    {
        if(left)
            delete left;
        if(right)
            delete right;
    }
};

#endif // BINTR_H
```

Файл func.h:

```
#ifndef FUNC_H
#define FUNC_H

#include <iostream>
#include <istream>
#include <fstream>
#include <string>
#include <sstream>
#include <bintr.h>

using namespace std;

bool is_open_bracket (char ch)
{ return ch=='(';}

bool is_close_bracket (char ch)
{ return ch==')';}

bool is_value (char ch)
{ return ch!='(' && ch!=')' && ch!='#';}

bool is_latt (char ch)
{ return ch=='#';}

template<typename BASE>
bool create_binTree(istream& is_str, BinTree<BASE> *bTree, unsigned i=0,
bool flag = false)
{
    char ch;

    if (is_str>>ch)
    {
        if (i==0)
            flag = create_binTree(is_str, bTree, ++i, true);
```

```
        else if (is_value(ch))
        {
           bTree->set_vl(ch);
        }
        else if (is_open_bracket(ch) && flag == true)
        {
           bTree->set_left();
           flag = create_binTree(is_str, bTree->get_left(), ++i, true);
        }
        else if (is_close_bracket(ch))
        {
            return false;

        }
        else if (is_open_bracket(ch) && flag == false)
        {
            bTree->set_right();
            flag = create_binTree(is_str, bTree->get_right(), ++i, true);
        }
        else if (is_latt(ch))
        {
            flag = create_binTree(is_str, bTree, ++i, false);
            return flag;
        }
        flag = create_binTree(is_str, bTree, ++i, flag);
    }

}

template<typename BASE>
void print_binTree(BinTree<BASE>* bTree, unsigned l=0)
{
   if(!bTree)
   {
       for(unsigned int i=0; i<l; ++i)
          cout << '\t';
       cout << '#' << endl;
       return;
   }
   print_binTree(bTree->get_right(), l+1);
   for(unsigned int i=0; i<l; i++)
       cout << '\t';
   cout << bTree->get_vl() << endl;
   print_binTree(bTree->get_left(), l+1);
}

template<typename BASE>
bool similar(BinTree<BASE>* bTree1, BinTree<BASE>* bTree2)    //проверка
подобия
{
    if (bTree1 == nullptr && bTree2 ==nullptr)
    {
        return true;
    }
    else
    if (bTree1 == nullptr || bTree2 == nullptr)
    {
        return false;
    }
```

```
    return similar(bTree1->get_left(), bTree2->get_left()) &&
           similar(bTree1->get_right(), bTree2->get_right());
}

template<typename BASE>
bool equal(BinTree<BASE>* bTree1, BinTree<BASE>* bTree2)      //проверка
на равенство
{
    if (bTree1 == nullptr && bTree2 ==nullptr)
    {
        return true;
    }
    else
    if (bTree1 == nullptr || bTree2 == nullptr)
    {
        return false;
    }
    else
    if (bTree1->get_vl() != bTree2->get_vl())
    {
        return false;
    }
    return equal(bTree1->get_left(), bTree2->get_left()) &&
            equal(bTree1->get_right(), bTree2->get_right());
}

template<typename BASE>
bool speculary_similar(BinTree<BASE>* bTree1, BinTree<BASE>* bTree2)
//зеркально подобны
{
    if (bTree1 == nullptr && bTree2 ==nullptr)
    {
        return true;
    }
    else
    if (bTree1 == nullptr || bTree2 == nullptr)
    {
        return false;
    }

    return speculary_similar(bTree1->get_left(), bTree2->get_right()) &&
            speculary_similar(bTree1->get_right(), bTree2->get_left());
}

template<typename BASE>
bool symmetry(BinTree<BASE>* bTree1, BinTree<BASE>* bTree2)
//проверка на симметричность
{
    if (bTree1 == nullptr && bTree2 ==nullptr)
    {
        return true;
    }
    else
    if (bTree1 == nullptr || bTree2 == nullptr)
    {
        return false;
    }
    else
    if (bTree1->get_vl() != bTree2->get_vl())
    {
        return false;
```

```
    }
    return symmetry(bTree1->get_left(), bTree2->get_right()) &&
            symmetry(bTree1->get_right(), bTree2->get_left());
}

bool analyzer(string bt, int length)
{
    int open_brackets = 0;
    int close_brackets = 0;

    if (length == 3 && bt[0] == '(' && bt[1] == '#' && bt[2] == ')')
        return false;

    if (bt[0] != '(')
    {
        cerr << "Ошибка в символе № " << 1 << ". Первым символом всегда
должна быть '('.";
        return true;
    }
    else
        open_brackets++;

    if (bt[length-1] != ')')
    {
        cerr << "Ошибка в символе № " << length << ". Последним всегда
должен быть символ ')'.";
        return true;
    }
    else
        close_brackets++;

    for (int i = 1; i < length-1; i++)
    {
        if (bt[i] == '(' && (bt[i+1] < 'a' || bt[i+1] > 'z'))
        {
            cerr << "Ошибка в символе № " << i+2 << ". После '(' должна
следовать буква.";
            return true;
        }
        else  if (bt[i] == '(')
            open_brackets++;

        if (bt[i] > 'a' && bt[i] < 'z' && (bt[i+1] != '(' && bt[i+1] !=
'#' && bt[i+1] != ')'))
        {
            cerr << "Ошибка в символе № " << i+2 << ". После буквы должен
следовать один из вариантов: '(', '#', ')'.";
            return true;
        }

        if (bt[i] == '#' && bt[i+1] != '(')
        {
            cerr << "Ошибка в символе № " << i+2 << ". После '#' ожидается
ввод '('.";
            return true;
        }

        if (bt[i] == ')' && (bt[i+1] != '(' && bt[i+1] != ')'))
        {
```

```
            cerr << "Ошибка в символе № " << i+2 << ". После ')' ожидается
ввод '(' или ')'.";
            return true;
        }
        else
        if (bt[i] == ')')
            close_brackets++;
    }

    if (open_brackets != close_brackets)
    {
       cerr << "Количество открытых и закрытых скобок не равны";
       return true;
    }
    else
        return false;
}

#endif // FUNC_H
```