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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студент гр. 9303 _____ Дюков В. А.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Создание программы для сравнения бинарных деревьев.

Задание.

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

- подобны ли они (два бинарных дерева подобны, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);
- равны ли они (два бинарных дерева равны, если они подобны и их соответствующие элементы равны);
- 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);
- симметричны ли они (два бинарных дерева симметричны, если они зеркально подобны и их соответствующие элементы равны).

Основные теоретические положения.

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка.

Описать такую структуру можно следующим образом:

```
struct point {

      int data;         //информационное поле
      point *left;      //адрес левого поддерева
      point *right;     //адрес правого поддерева
};
```

Начальный узел называется корнем дерева. Узел, не имеющий поддеревьев, называется листом. Исходящие узлы называются предками,

входящие — потомками. Высота дерева определяется количеством уровней, на которых располагаются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется деревом поиска. Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу, 2 двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле. Такой поиск гораздо эффективнее поиска по списку, поскольку время поиска определяется высотой дерева, а она пропорциональна двоичному логарифму количества узлов.

В идеально сбалансированном дереве количество узлов справа и слева отличается не более чем на единицу.

Линейный список можно представить как вырожденное бинарное дерево, в котором каждый узел имеет не более одной ссылки. Для списка среднее время поиска равно половине длины списка.

Деревья и списки являются рекурсивными структурами, т. к. каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

- либо пустой структурой;
- либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

Выполнение работы.

Для выполнения работы были реализованы следущие функции:

- *bool isEqual(Tree*, Tree*)* – рекурсивная функция, принимает на вход ссылки на два бинарных дерева. Рекурсивно проверяет полное равенство каждого узла деревьева. Если все узлы равны, возвращает *true*.

- *bool isMirrorEqual(Tree*, Tree*)* – рекурсивная функция, принимает на вход ссылки на два бинарных дерева. Рекурсивно проверяет равенство каждого узла деревьева, но сравнивает противоположные ветви каждого узла. Если все узлы равны, возвращает *true*.
- *bool isSuch(Tree*, Tree*)* – рекурсивная функция, принимает на вход ссылки на два бинарных дерева. Рекурсивно проверяет равенство каждого узла деревьева, кроме информации, хранимую узлами. Если все узлы подобны, возвращает *true*.
- *bool isMirrorSuch(Tree*, Tree*)* – рекурсивная функция, принимает на вход ссылки на два бинарных дерева. Рекурсивно проверяет равенство каждого узла деревьева, кроме информации, хранимую узлами, и сравнивает противоположные ветви каждого узла. Если все узлы подобны, возвращает *true*.

Также были перегружены операторы ввода (>>) и вывода (<<) для класса дерева (Tree). Оператор ввода рекурсивно считывает каждый узел соответственного дерева / под-узла.

Тестирование

№ т.	Входные данные	Результат	Вывод
1	(a) (a)	+ Деревья равны + Деревья зеркально равны + Деревья подобны + Деревья зеркально подобны	Программа работает верно
2	(a) (b)	+ Деревья подобны + Деревья зеркально подобны	Программа работает верно
3	(a) (a (a))	У деревьев нет совпадений	Программа работает верно
4	(a (b)) (a N (s))	+ Деревья зеркально подобны	Программа работает верно

5	(a (a (b)) (c)) (a (a (b)) (c))	+ Деревья равны + Деревья подобны	Программа работает верно
6	(a (a (b)) (c)) (a (c) (a (b)))	+ Деревья зеркально равны + Деревья зеркально подобны	Программа работает верно

Выводы

Были изучены теоретические материалы по бинарным деревьям, были разработаны алгоритмы по выводу и обработке бинарных деревьев.

ПРИЛОЖЕНИЕ А

header.h :

```
#pragma once

#include <iostream>


class Tree {

public:
  char info;
  Tree* left;
  Tree* right;

  Tree();
  ~Tree();

  friend std::istream& operator>> (std::istream&, Tree**);
  friend std::ostream& operator<< (std::ostream&, Tree*);
};


class TwoTree {

private:
  Tree* first;
  Tree* second;

public:
  TwoTree();
  TwoTree(Tree*, Tree*);

  bool isEqual(Tree*, Tree*);
  bool isEqual();
  bool isMirrorEqual(Tree*, Tree*);
  bool isMirrorEqual();
  bool isSuch(Tree*, Tree*);
  bool isSuch();
  bool isMirrorSuch(Tree*, Tree*);
  bool isMirrorSuch();

  void printTrees(std::ostream&);
};
```

Tree.cpp :

```
#include "Header.h"

Tree::Tree() {

  info = 0;
  left = nullptr;
  right = nullptr;
}

Tree::~Tree() {

  if (this->left) delete this->left;
  if (this->right) delete this->right;
}

std::istream& operator>>(std::istream& in, Tree** tree) {

  char symb;
  short brackets = 0;
  Tree* stack[100];
  stack[0] = nullptr;
  bool r = false;

  do {

        in >> symb;
        if (symb == '(') {

              stack[brackets] = new Tree;
              in >> stack[brackets]->info;
              if (brackets > 0) {

                    if (stack[brackets - 1]->right) break;
                    if ((stack[brackets - 1]->left && !stack[brackets - 1]-
>right) || r) stack[brackets - 1]->right = stack[brackets];
                    if (!stack[brackets - 1]->left && !r) stack[brackets -
1]->left = stack[brackets];
              }
              brackets++;
              r = false;
        }
        else if (symb == ')') brackets--;
        else if (symb == 'N') r = true;
        else break;

  } while (brackets > 0);
```

```
  if (brackets == 0 && stack[0]) *tree = stack[0];
  else if (stack[0]) delete stack[0];
  return in;
}

std::ostream& operator<<(std::ostream& out, Tree* tree) {

  if (tree) {

        out << '(' << tree->info;
        if (tree->left) out << ' ' << tree->left;
        if (tree->right) {

              if (!tree->left) out << " N";
              out << ' ' << tree->right;
        }
        out << ')';
  }
  return out;
}
```

TwoTree.cpp :

```
#include "Header.h"


TwoTree::TwoTree() : first(nullptr), second(nullptr) {}

TwoTree::TwoTree(Tree* f, Tree* s) : first(f), second(s) {}

bool TwoTree::isEqual(Tree* f, Tree* s) {

  if (f && s) {

        if (f->info == s->info) return (isEqual(f->left, s->left) &&
isEqual(f->right, s->right));
        else return false;
  }
  if (f || s) return false;
  return true;
}

bool TwoTree::isEqual() {

  Tree* f = this->first;
  Tree* s = this->second;
```

```
  if (f && s) {

        if (f->info == s->info) return (isEqual(f->left, s->left) &&
isEqual(f->right, s->right));
        else return false;
  }
  if (f || s) return false;
  return true;
}

bool TwoTree::isMirrorEqual(Tree* f, Tree* s) {

  if (f && s) {

        if (f->info == s->info) return (isMirrorEqual(f->left, s->right) &&
isMirrorEqual(f->right, s->left));
        else return false;
  }
  if (f || s) return false;
  return true;
}

bool TwoTree::isMirrorEqual() {

  Tree* f = this->first;
  Tree* s = this->second;

  if (f && s) {

        if (f->info == s->info) return (isMirrorEqual(f->left, s->right) &&
isMirrorEqual(f->right, s->left));
        else return false;
  }
  if (f || s) return false;
  return true;
}

bool TwoTree::isSuch(Tree* f, Tree* s) {

  if (f && s) return (isSuch(f->left, s->left) && isSuch(f->right, s-
>right));
  if (f || s) return false;
  return true;
}

bool TwoTree::isSuch() {

  Tree* f = this->first;
```

```
  Tree* s = this->second;

  if (f && s) return (isSuch(f->left, s->left) && isSuch(f->right, s-
>right));
  if (f || s) return false;
  return true;
}

bool TwoTree::isMirrorSuch(Tree* f, Tree* s) {

  if (f && s) return (isSuch(f->left, s->right) && isSuch(f->right, s-
>left));
  if (f || s) return false;
  return true;
}

bool TwoTree::isMirrorSuch() {

  Tree* f = this->first;
  Tree* s = this->second;

  if (f && s) return (isSuch(f->left, s->right) && isSuch(f->right, s-
>left));
  if (f || s) return false;
  return true;
}

void TwoTree::printTrees(std::ostream& out) {

  out << first << '\n' << second << '\n';
}
```

main.cpp :

```
#include "Header.h"
#include <windows.h>
#include <fstream>
#include <conio.h>

void sys_print() {

  system("cls");
  std::cout << "Программа по обработке бинарных деревьев\nСинтаксис: (a N (b
...)) , где N - пустая ветвь.\n\n";
}

int button_get(std::string* buttons, int height, int width, std::string title
= "") {
```

```
  int _w = 0, _h = 0;

  std::cout << title;

  while (1) {

        for (int i = 0; i < height; i++) {
              std::cout << '\r';
              for (int j = 0; j < width; j++) {

                    if (i == _h && j == _w) std::cout << "\t<" << buttons[i *
width + j] << ">";
                    else std::cout << "\t " << buttons[i * width + j] << " ";
              }
        }

        unsigned char ch = _getch();
        if (ch == 224) ch = _getch();
        switch (ch) {
        case 72:
              _h--;
              break;
        case 80:
              _h++;
              break;
        case 75:
              _w--;
              break;
        case 77:
              _w++;
              break;
        case 13:
              return _h * width + _w + 1;
        case 27:
              return 0;
        }

        _w = abs(_w) % width;
        _h = abs(_h) % height;
  }
}

int main() {

  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);

  bool looping = true;
```

```
  while (looping) {

        sys_print();
        std::string str1[] = { "из файла", "из консоли" };
        int way = button_get(str1, 1, 2, "Каким образом вы хотите вводить
данные?\n");
        if (way == 0) return 0;
        std::cout << "\n";

        std::ifstream file;
        TwoTree trees;

        if (way == 1) {


              while (!file.is_open()) {

                    std::cout << "\nВведите имя файла:\n\n";
                    std::string name;
                    std::cin >> name;
                    file.open(name);
                    std::cout << "\nОшибка ввода имени файла.\n\n";
                    std::string str2[] = { "да", "нет" };
                    int check = button_get(str2, 1, 2, "Попробовать ввести
имя ещё раз?\n");
                    if (check == 2) break;
                    if (check == 0) return 0;
                    sys_print();
              }
        }

        while (file.is_open() || way == 2) {

              Tree* f = nullptr, * s = nullptr;

              if (way == 1) {

                    file >> &f >> &s;
                    trees = TwoTree(f, s);
              }
              if (way == 2) {

                    std::cout << "\nВведите первое бинарное дерево\n";
                    std::cin >> &f;
                    std::cout << "\nВведите второе бинарное дерево\n";
                    std::cin >> &s;
                    trees = TwoTree(f, s);
              }
```

```
            sys_print();
            std::cout << "Вы ввели:\n\n" << f << '\n' << s << '\n';
            std::cout << "\nРезультаты:\n\n";

            bool nothing = true;
            if (trees.isEqual()) { std::cout << "+ Деревья равны\n";
nothing = false; }
            if (trees.isMirrorEqual()) { std::cout << "+ Деревья зеркально
равны\n"; nothing = false; }
            if (trees.isSuch()) { std::cout << "+ Деревья подобны\n";
nothing = false; }
            if (trees.isMirrorSuch()) { std::cout << "+ Деревья зеркально
подобны\n"; nothing = false; }
            if (nothing) std::cout << "У деревьев нет совпадений.\n";

            if (f) delete f;
            if (s) delete s;

            if (way == 2) {

                  std::cout << "\n";
                  std::string str2[] = { "продолжить" };
                  if (!button_get(str2, 1, 1)) return 0;
                  else break;
            }
            if (way == 1) {

                  std::cout << "\n";
                  std::string str2[] = { "да", "нет" };
                  int fl = button_get(str2, 1, 2, "Продолжить вывод из
файла?\n");
                  if (fl == 0) return 0;
                  if (fl == 1) continue;
                  if (fl == 2) break;
            }
      }
      if (file.is_open()) file.close();
  }

  return 0;
}
```