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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: АВЛ деревья вставка и удаление

Студент гр. 8381 _____ Перелыгин Д.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Перелыгин Д,С,

Группа 8381

Тема работы: АВЛ деревья

Исходные данные: необходимо выполнить визуализацию структуры данных – АВЛ дерево. Вставка и удаление элементов. Должна быть возможность графически отобразить изменения в дереве.

Содержание пояснительной записки:
«Содержание», «Введение», «Задание», «Описание программы», «Теоретические положения», «Демонстрация», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент		Перелыгин Д.С.
Преподаватель		Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения курсовой работы была разработана программа с GUI, позволяющая визуализировать алгоритм построения АВЛ-дерева, добавления и удаления элементов из него. Программа обладает следующим функционалом: в основном окне есть кнопка для выбора и открытия файла с начальными данными, кнопка для построения дерева из полученных из файла данных, а также две кнопки для добавления и удаления элемента введенного в специальное окно. Помимо этого, большую часть основного окна занимает интерфейс для визуализации АВЛ-дерева и текстовое окно для вывода файлов лога.

SUMMARY

During the course work, a program was developed with a GUI that allows you to visualize the algorithm for constructing the AVL tree, adding and removing elements from it. The program has the following functionality: in the main window there is a button for selecting and opening a file with initial data, a button for building a tree from the data received from the file, and two buttons for adding and removing an element entered in a special window. In addition, most of the main window is occupied by the interface for visualizing the AVL tree and a text window for displaying log files.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Реализация и демонстрация работы АВЛ-дерева, функций вставки и удаления.

Основные задачи

Реализация ввода из файла, визуализация таблицы, визуализация АВЛ дерева, визуализация алгоритмов вставки и удаления, с отображением элементов, подверженных повороту при балансировке дерева.

Методы решения

Разработка программы велась на базе операционной системы Windows 10 в интегрированной среде разработки QtCreator. Для работы с АВЛ деревьями были реализованы классы Head_AVL_Tree и Node_AVL_Tree, позволяющие работать с деревом и с его элементами с помощью описанных методов.

1. ЗАДАНИЕ

Необходимо продемонстрировать алгоритм построения АВЛ дерева, а также вставки и удаления элемента.

Демонстрация должна:

1. Быть подробной и понятной;
2. Выполняться отдельными шагами для наглядности отображая происходящие изменения;
3. Иметь возможность быть использованной в обучающих целях.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание интерфейса пользователя

После запуска программы пользователя встречает окно приложения, в котором предоставляется возможность загрузки файла с исходными данными, кнопки для построения АВЛ дерева, добавление и удаление элементов, введенных в специальное поле. На рис. 1 представлена демонстрация графического отображения построения для данных, принятых из строки ввода.

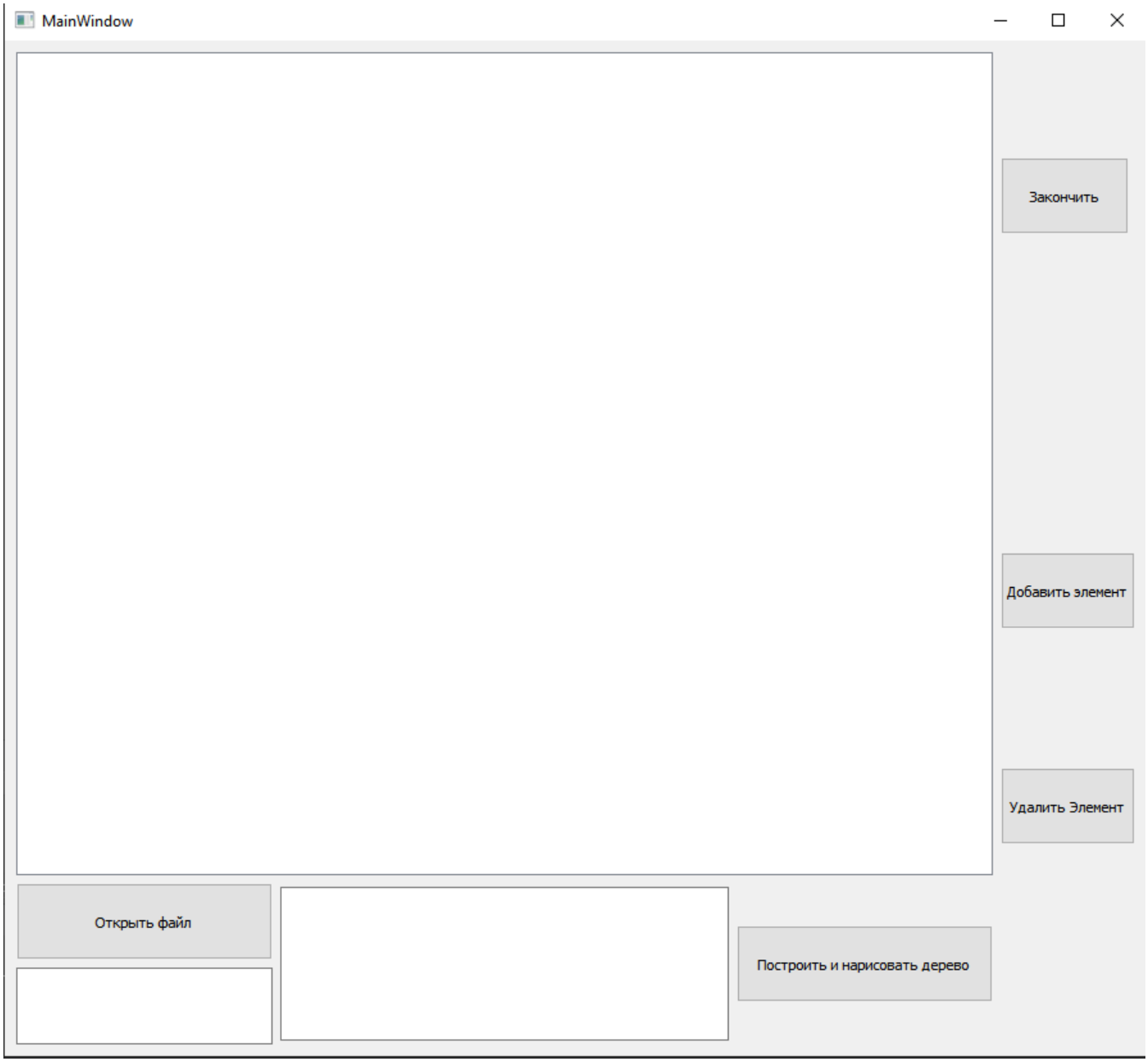

Рисунок 1 — Вид окна приложения

При нажатии кнопки «Открыть файл», открывается диалоговое окно с выбором файла для чтения.

Кнопка «Построить и нарисовать дерево» создает и рисует дерево из данных в выбранном файле.

Кнопки «Добавить элемент» и «Удалить элемент» считывают введенную в небольшое окошко под кнопкой «Открыть файл» информацию и выполняют соответствующее действие с ней.

Кнопка «Закончить» завершает выполнение программы.

Большую часть окна занимает виджет graphicsView, который служит для графического отображения состояния дерева.

Под ним находится виджет QtextEdit, в который выводятся логи работы программы.

2.2. Описание классов, методов и функций.

1. `Class Head_AVL_Tree` — реализована работа с АВЛ-деревом.
    1. Методы класса:
        1. `void insert(Type)` — вставка элемента в АВЛ-дерево.
        2. `void print_tree1()` — вывод АВЛ-дерева на экран
        3. `bool is_contain(Type)` — возвращает true, если заданный элементы содержится в АВЛ-дереве.
        4. `Head_AVL_Tree()` — конструктор класса. Инициализирует данные.
        5. `~Head_AVL_Tree()` — деструктор класса. Очищает память, выделенную под АВЛ-дерево.
        6. `void remove(Type)` — удаление заданного элемента.
    2. Данные класса:
        1. `class Node_AVL_Tree<Type>* head` — содержит указатель на АВЛ-дерево.

2. `class Node_AVL_Tree` – Реализована работа с элементами АВЛ-дерева.
    1. Методы класса:
        1. `Bool is_contain(Type)` — возвращает true, если заданный элементы содержится в АВЛ-дереве.
        2. `Int set_height()` — устанавливает высоту данного элемента АВЛ-Дерева.
        3. `int get_balance()` — Получает значение баланса для заданного элемента АВЛ-дерева.
        4. `void print_tree(int)` — выводит на экран АВЛ-дерево
        5. `class Node_AVL_Tree<Type>* insert(Type)` — вставка в АВЛ- дерево
        6. `class Node_AVL_Tree<Type>* left_rotate()` — левое вращение
        7. `class Node_AVL_Tree<Type>* right_rotate()` — правое вращение
        8. `class Node_AVL_Tree<Type>* make_balance()` — установка баланса путем вращения дерева вокруг элемента
        9. `Node_AVL_Tree()` — конструктор класса. Инициализирует данные.
        10. `~Node_AVL_Tree()` — деструктор класса. Очищает память выделенную под АВЛ-дерево.
        11. `Class Node_AVL_Tree<Type>* remove(Type)` — Функция удаления заданного элемента из дерева.
        12. `Class Node_AVL_Tree<Type>* remove_min()` — Функция удаления минимального элемента.
        13. `Class Node_AVL_Tree<Type>* find_min()` — Нахождение минимального элемента в дереве.
    2. Данные класса:

1. `int height` — определяет высоту дерева.
2. `int balance` — определяет баланс дерева.
3. `Type data` — данные дерева.
4. `Bool rotate` — флаг состояния, отражающий, участвовал ли узел в повороте.
5. `class Node_AVL_Tree<Type>* left` — левый сын
6. `class Node_AVL_Tree<Type>* right` — правый

3. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ РАБОТЫ

3.1 Общее описание работы АВЛ дерева

Описание алгоритма вставки:

Элементы дерева вставляют как в обычное упорядоченное дерево (справа от элемента – элементы большие, а слева от элемента элементы меньшие). Но есть отличие от обычного упорядоченного бинарного дерева. По возвращению из рекурсии дерево преобразуется таким образом чтобы максимальное различие высоты между двумя его любыми ветвями было не более чем 1 – это обеспечивает логарифмическую сложность. Эти преобразования происходят если у элемента дерева *balance* не лежит в пределах [-1;1]. Существуют 4 типа вращений для восстановления баланса:

1. Малое левое вращение.
2. Малое правое вращение.
3. Больше левое вращение.
4. Большое правое вращение

деревьев. Повороты БД изображены на рис. 2.

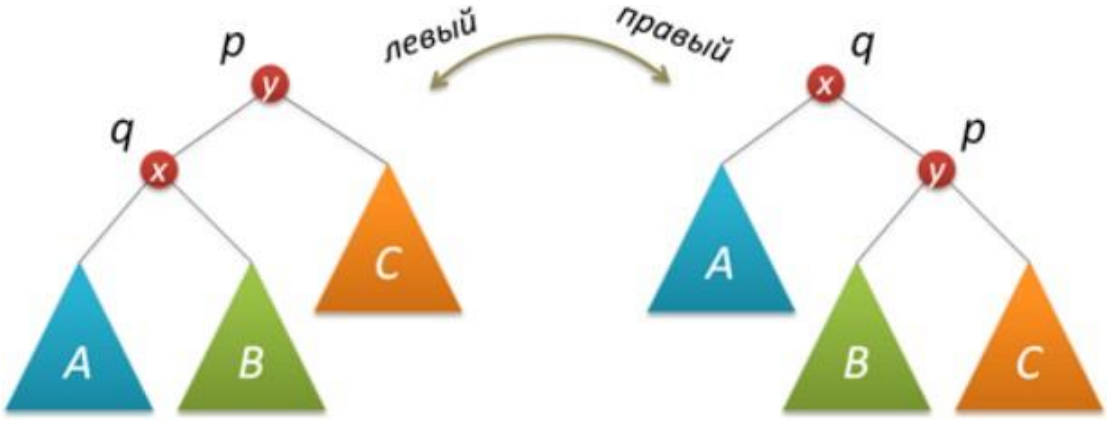

Рисунок 2 – повороты АВЛ дерева.

Описание алгоритма поиска элемента в АВЛ_дереве:

Поиск элемент в АВЛ_дереве происходит как в обычном упорядоченном дереве. Если узел больше заданного значения, переходим к правому сыну, если

меньше заданного значения переходим к левому сыну. Алгоритм заканчивает свою работу в 2-х случаях:

1. Если значения узла равно заданному значению.
2. Если был достигнут лист и не встречен элемент равный заданному.

Алгоритм удаления элемента из АВЛ_дерева:

Находим заданный для удаления элемент в дереве. Создаем новое дерево с корнем со значением минимального элемента в правом сыне исходного дерева. Левый сын нового – левый сын старого. Правый сын нового – старый правый сын без минимального элемента. По выходу из рекурсии обновляем высоту и балансировку деревьев и при необходимости корректируем высоту дерева.

3.2 Реализация методов поворота

В приведённом ниже фрагменте кода демонстрируется реализация операции правого поворота АВЛ-дерева (левое сделано по аналогии).

```
 template <class Type>
 class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::right_rotate(){
     std::cout << "right rotate around element: " << this->data << 
std::endl;
     Node_AVL_Tree<Type>* temp;
     temp = left;

     temp->rotate = true;
     if (temp->left!=nullptr)
     temp->left->rotate = true;
     if (temp->right!=nullptr)
     temp->right->rotate = true;

     left = temp->right;
     this->height = this->set_height();
     this->balance = this->set_balance();
     if(temp->left){
         temp->left->height = temp->left->set_height();
         temp->left->balance = temp->left->set_balance();
     }
     temp->right = this;
     temp->height = temp->set_height();
     temp->balance = temp->set_balance();
```

```
    return temp;
}
```

4. ДЕМОНСТРАЦИЯ

4.1 Окна интерфейса программы

На рис. 3 представлено стартовое меню программы.

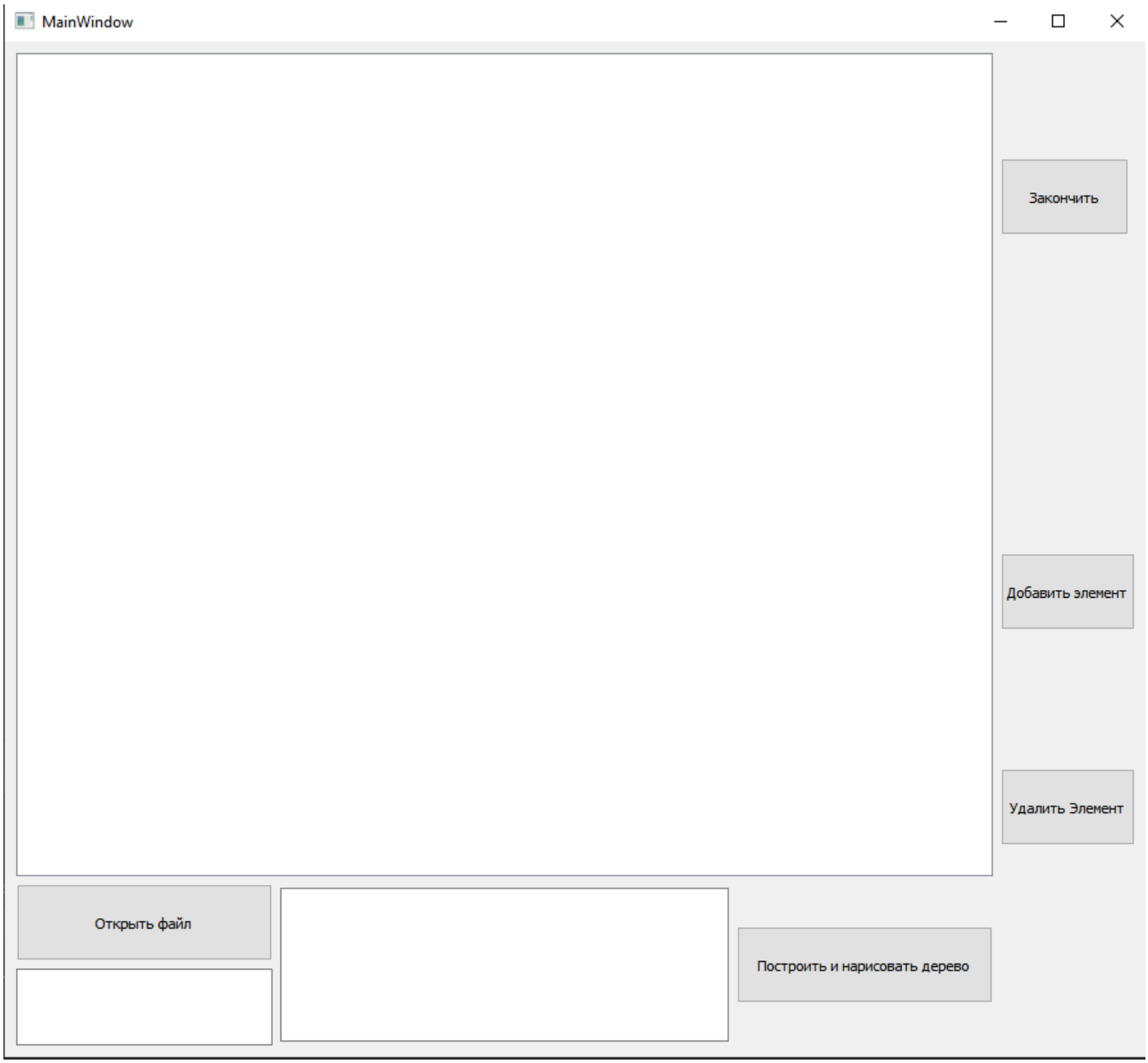

Рисунок 3 – стартовое меню программы

4.2 Результаты графической обработки АВЛ дерева

На рис. 4 представлен результат построение дерева.

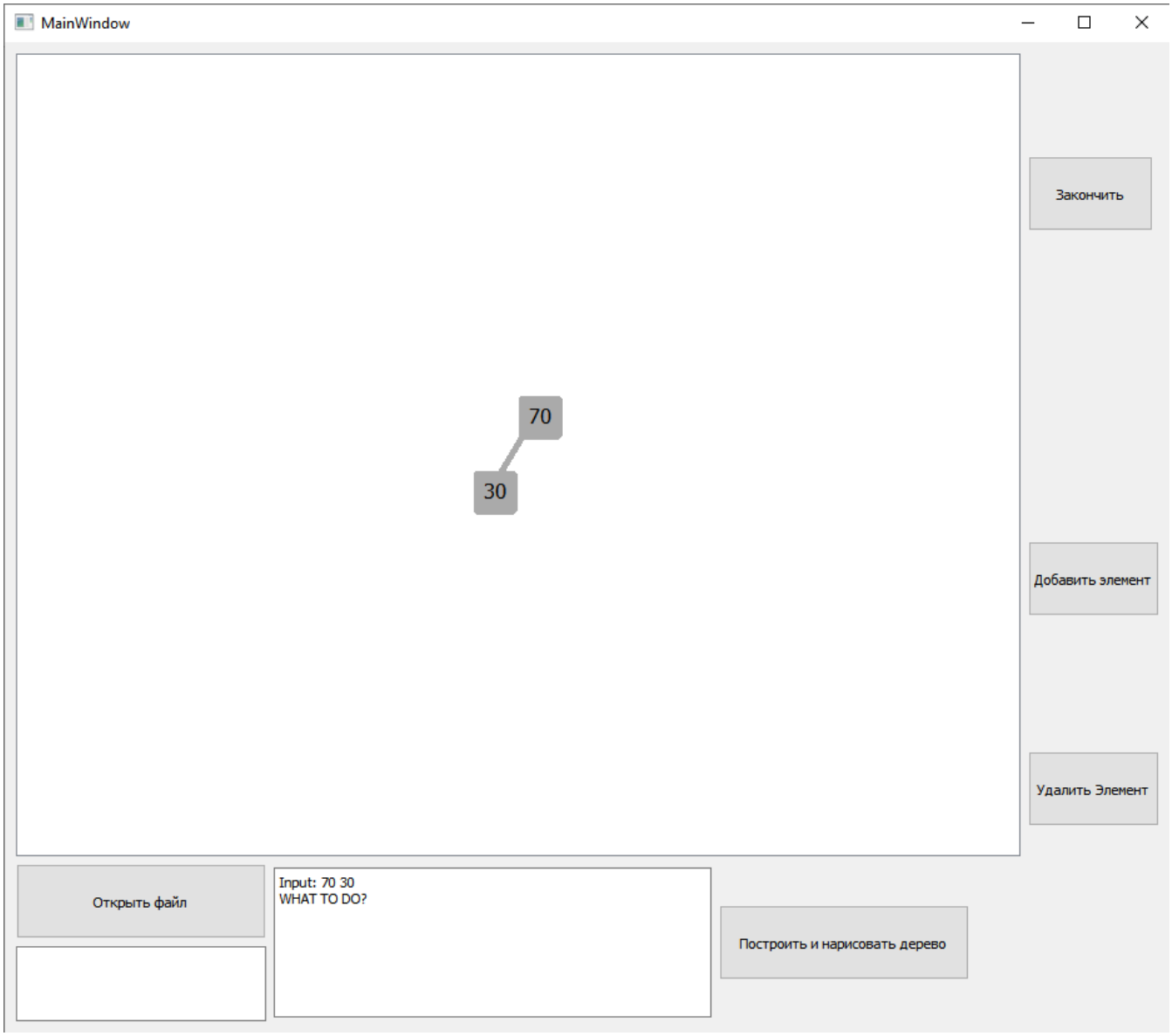

Рисунок 4 – полностью построенное АВЛ дерево для 2 значений

После вводится 20 в окно для ввода и нажимается кнопка «Добавить элемент». Выполняется поиск такого элемента в дереве, и в случае, если он отсутствует, программа добавляет его, балансируя дерево.

На рис. 5 представлен результат добавления нового элемента в таблицу.

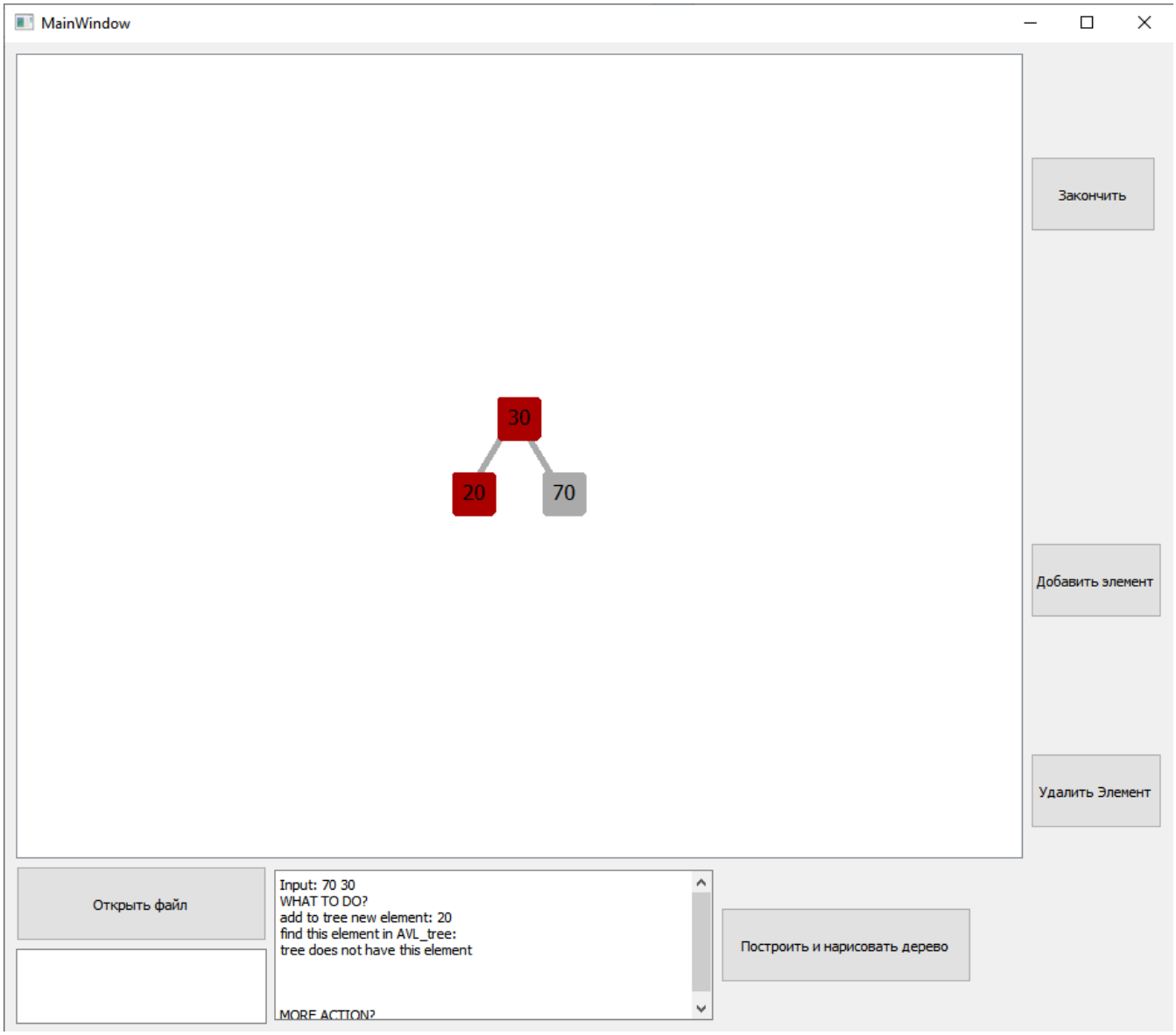

Рисунок 5 – результат добавления в дерево элемента 20. Красным цветом выделены элементы, подвергшиеся повороту.

После выполнения прошлого шага в окно ввода записывается 30, а затем нажимается кнопка «Удалить Элемент». Программа аналогично предыдущему шагу производит поиск элемента в дереве и в случае, если элемент содержится в дереве, удаляет его, после чего производит балансировку дерева.

На рис. 6 представлен результат добавленного элемента из таблицы.

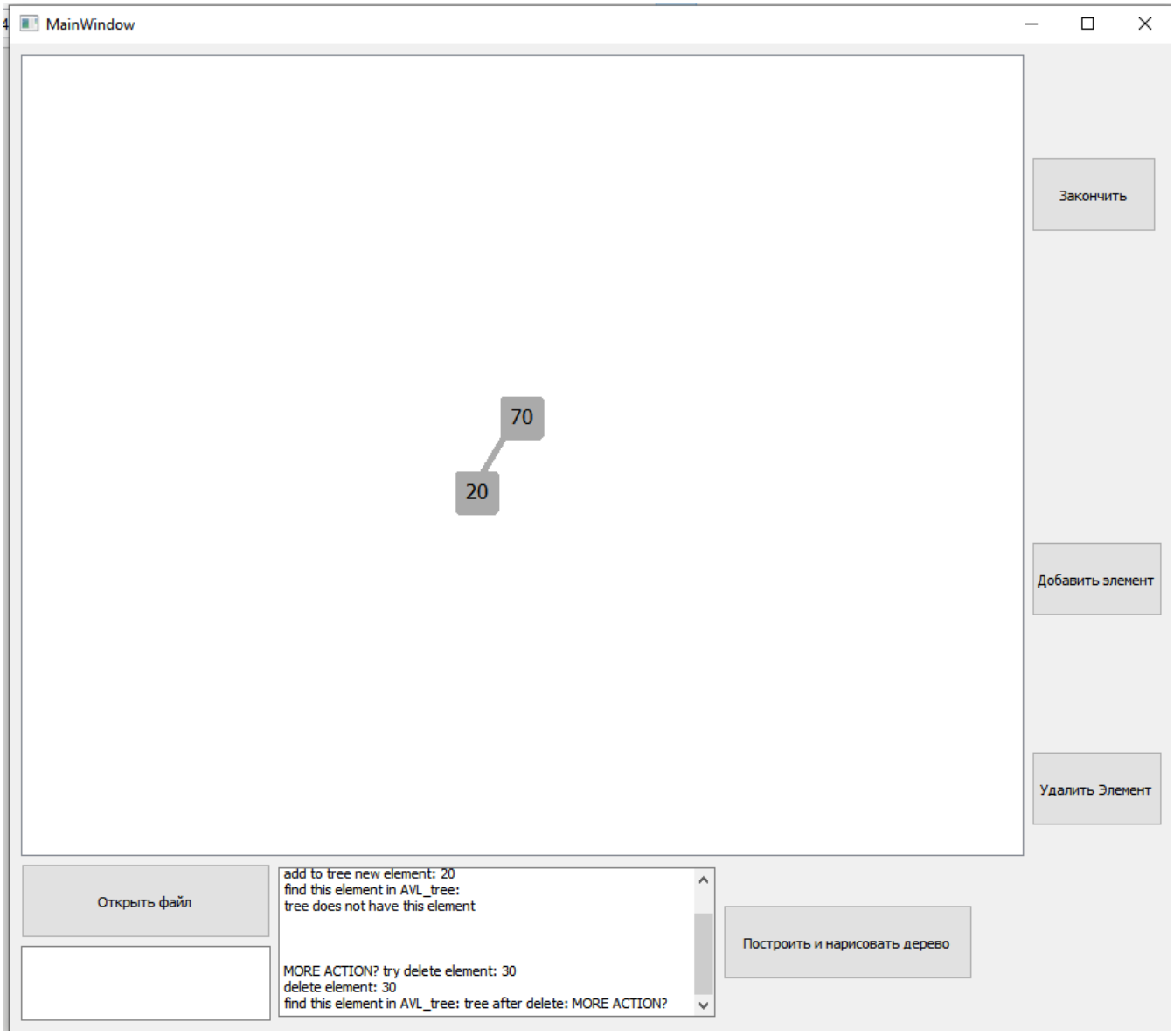

Рисунок 6 – результат удаления элемента 30

На рис. 7 представлен пример дерева для 15 элементов.

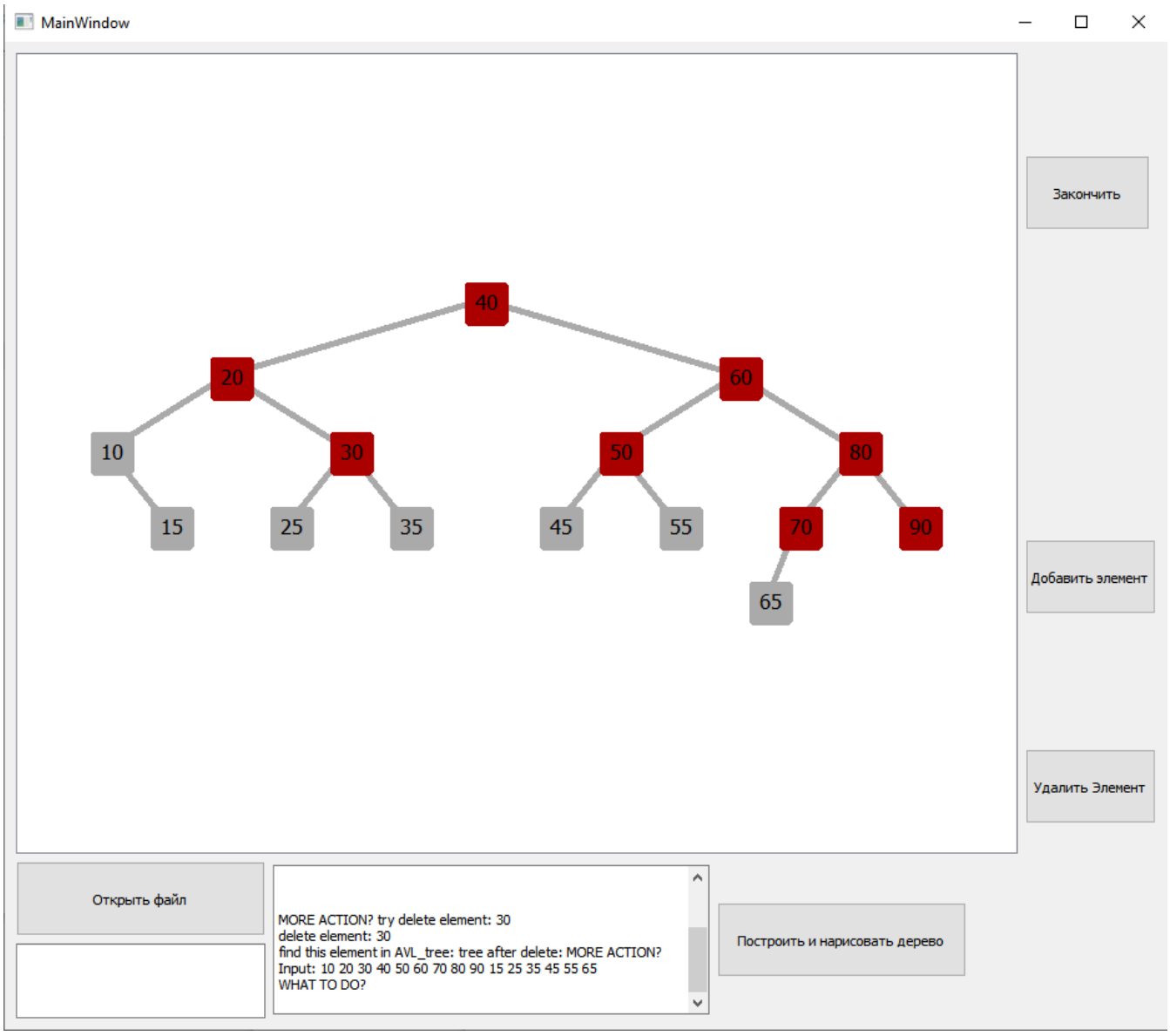

Рисунок 6 – дерево для 15 элементов.

Все данные были прочитаны с файла, как видно по окну логов. Сразу заметно, что большинство элементов при построении дерева подверглись балансировке.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, которая обладает следующей функциональностью: Построение АВЛ дерева, его балансировкой, визуализация процесса с вывод логов. Также были реализованы операции вставки и удаления, осуществление которых также возможно с визуализацией и выводом поясняющих сообщений. Программу можно использовать в обучающих целях, для наглядной демонстрации работы АВЛ дерева.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bjarne Stroustrup. A Tour of C++. M.: Addison-Wesley, 2018. 217 с.
2. Перевод и дополнение документации QT // CrossPlatform.RU. URL: http://doc.crossplatform.ru/
3. Qt Documentation // Qt. URL: https://doc.qt.io/qt-5/index.html (дата обращения: 20.12.2019).
4. Habrahabr. // Habr. URL: https://habr.com/ (дата обращения: 20.12.2019).
5. Вирт Н. Алгоритмы и структуры данных.
6. Кнут Д. Искусство программирования.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Tree.h

```
#ifndef AVL_TREE_H // for include only one time
 #define AVL_TREE_H
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <map>
#include <algorithm>
#include <memory>
#include <iostream>
#include <fstream>
#include <sstream>
#include <QMainWindow>
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>
#include <QFileDialog>
#include <QStandardPaths>
#include <QtGui>
#include <QLabel>
#include <QColorDialog>
#include <QInputDialog>
#include <QMainWindow>
#include <QPushButton>
#include <QMessageBox>
#include <QStringList>
#include <QTextBrowser>
#include <QTextEdit>
#include <stack>
```

```
 template <class Type>
 class Head_AVL_Tree;

 template <class Type>
 class Node_AVL_Tree{
 public:
```

```
    friend class Head_AVL_Tree<Type>;

    bool is_contain(Type, int);
    int set_height();
    int set_balance();
    void print_tree();
    void print_tree(int);

    class Node_AVL_Tree<Type>* insert(Type);
    class Node_AVL_Tree<Type>* remove(Type);
    class Node_AVL_Tree<Type>* remove_min();
    class Node_AVL_Tree<Type>* find_min();
    class Node_AVL_Tree<Type>* left_rotate();
    class Node_AVL_Tree<Type>* right_rotate();
    class Node_AVL_Tree<Type>* make_balance();
    class Node_AVL_Tree<Type>* get_r();
    class Node_AVL_Tree<Type>* get_l();
    Type get_d();
    void set_rot(bool);
    bool get_rot();
    int countDeep(Node_AVL_Tree<Type>*);
    Node_AVL_Tree();
    ~Node_AVL_Tree();

private:

    bool rotate = false;
    int height;
    int balance;
    Type data;
    class Node_AVL_Tree<Type>* left;
    class Node_AVL_Tree<Type>* right;
};

template <class Type>
class Head_AVL_Tree{
public:
    Head_AVL_Tree();
    ~Head_AVL_Tree();
    void insert(Type);
    void print_tree();
    bool is_contain(Type);
    void remove(Type);
    void print_tree1(QGraphicsScene *&);
private:
    class Node_AVL_Tree<Type>* head;
};


template <class Type>
class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::get_r(){
```

```
    return right;
}

template <class Type>
class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::get_l(){
    return left;
}

template <class Type>
Type Node_AVL_Tree<Type>::get_d(){
    return data;
}


template <class Type>
void Node_AVL_Tree<Type>::set_rot(bool rot){
    rotate = rot;
}

template <class Type>
bool Node_AVL_Tree<Type>::get_rot(){
    return rotate;
}



template <class Type>
Node_AVL_Tree<Type>::Node_AVL_Tree(){
    left = nullptr;
    right = nullptr;
}

template <class Type>
Node_AVL_Tree<Type>::~Node_AVL_Tree(){ // очищаем память под дерево
    if(left)
        delete left;
    if(right)
        delete right;
}

template <class Type>
bool  Node_AVL_Tree<Type>::is_contain(Type  desired,  int  depth){//проверка
есть ли элемент в дереве
    if(data == desired)
        return true;
    if(left && data > desired){
        std::cout << "find in left" << std::endl;
        if(left->is_contain(desired, depth+1))
            return true;
    }
    if(right && data < desired){
```

```
        std::cout << "find in right" << std::endl;
        return right->is_contain(desired, depth+1);
    }
    return false;
}



template <class Type>
int countDeep(Node_AVL_Tree<Type>* node)
{
    if (node == nullptr)
        return 0;
    int cl = countDeep(node->get_l());
    int cr = countDeep(node->get_r());
    return 1 + ((cl>cr)?cl:cr);
}



template <class Type>
QGraphicsScene    *graphic(Node_AVL_Tree<Type>*    head,    QGraphicsScene
*&scene)//Задаем параметры для рисования
{
    scene->clear();
    if (head == nullptr)
        return scene;
    QPen pen;
    pen.setWidth(5);
    QColor color;
    color.setRgb(170, 170, 170);
    pen.setColor(color);
    QBrush brush (color);
    color.setRgb(170, 0, 0);
    QBrush brush2 (color);
    QPen pen2 (color);
    pen2.setWidth(5);
    QFont font;
    font.setFamily("Heretica");

    int w_deep = static_cast<int>(pow(2, countDeep(head))+2);
    int h = 60;
    int w = 12;
    font.setPointSize(w);
    int width = (w*w_deep)/2;
    paint(scene, head, width/2, h, w, h, pen, pen2, brush, brush2, font,
w_deep);
    return scene;
}

template <class Type>
```

```
 void  Head_AVL_Tree<Type>::print_tree1(QGraphicsScene  *&scene){  //  рисуем
дерево
      graphic(head, scene);
 }



 template <class Type>//Непосредственно рисование
 int  paint(QGraphicsScene  *&scene,  Node_AVL_Tree<Type>*  node,  int  width,
int  height,  int  w,  int  h,  QPen  &pen,  QPen  &pen2,  QBrush  &brush,QBrush
&brush2, QFont &font, int depth)
 {

     if (node == nullptr)
         return 0;
     QString out;
     out = QString::number(node->get_d());
     QGraphicsTextItem *elem = new QGraphicsTextItem;
     if(out.size() == 3)
        elem->setPos(width - 7, height);
     else if (out.size() == 2)
         elem->setPos(width - 4, height);
     else
         elem->setPos(width, height);
     elem->setPlainText(out);
     elem->setFont(font);
     if (node->get_l() != nullptr)
         scene->addLine(width+w/2,      height+w,      width-(depth/2)*w+w/2,
height+h+w, pen);
     if (node->get_r() != nullptr)
         scene->addLine(width+w/2,      height+w,      width+(depth/2)*w+w/2,
height+h+w, pen);
     if (node->get_rot() == true)
     {
        scene->addRect(width-w/2, height, w*5/2, w*5/2, pen2, brush2);
        node->set_rot(false);
     }
     else
     scene->addRect(width-w/2, height, w*5/2, w*5/2, pen, brush);
     scene->addItem(elem);
     paint(scene,  node->get_l(),  width-(depth/2)*w,  height+h,  w,  h,  pen,
pen2, brush, brush2, font, depth/2);
     paint(scene,  node->get_r(),  width+(depth/2)*w,  height+h,  w,  h,  pen,
pen2, brush, brush2, font, depth/2);
     return 0;
 }
```

```
 template <class Type>
  class         Node_AVL_Tree<Type>*         Node_AVL_Tree<Type>::remove(Type
to_remove){//удаление элемента
      if(data == to_remove){
          if(!left && !right){
              delete this;
              return nullptr;
          }
          if(!right){
              class Node_AVL_Tree<Type>* temp = left;

              this->left = nullptr;
              delete this;
              return temp;
          }
          class Node_AVL_Tree<Type>* new_root;
          new_root = new Node_AVL_Tree<Type>;
          new_root->left=(right->find_min()->left);
          new_root->right=(right->find_min()->right);
          new_root->data=(right->find_min()->data);
          new_root->balance=(right->find_min()->balance);

          right = right->remove_min();
          new_root->left = left;
          new_root->right = right;
          new_root->height = set_height();
          new_root->balance = set_balance();
          return new_root->make_balance();
      }
      if(data < to_remove)
          right = right->remove(to_remove);
      if(data > to_remove)
          left = left->remove(to_remove);
      height = set_height();
      balance = set_balance();
      return make_balance();
 }

 template <class Type>
 class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::find_min(){
     return left?left->find_min():this;
 }

 template <class Type>
 class   Node_AVL_Tree<Type>*   Node_AVL_Tree<Type>::remove_min(){//удаление
минимального элемента
```

```
    if(!left){
        class Node_AVL_Tree<Type>* temp = right;
        this->right = nullptr;
        delete this;
        return temp;
    }
    left = left->remove_min();
    height = set_height();
    balance = set_balance();
    return make_balance();
}

template <class Type>
class  Node_AVL_Tree<Type>*  Node_AVL_Tree<Type>::insert(Type  value){  //
вставка
    if(value >= data){
        if(!right){
            right = new Node_AVL_Tree<Type>; // находим нужный узел для
вставки как в обычном упорядоченном дереве
            right->data = value;
            right->height = 1;
        }
        else
            right = right->insert(value);
    }
    if(value < data){
        if(!left){
            left = new Node_AVL_Tree<Type>;
            left->data = value;
            left->height = 1;
        }
        else
            left = left->insert(value);
        }
    height = set_height();
    balance = set_balance();
    return make_balance(); // балансировке по возврату из рекурсии
}

template <class Type>
class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::right_rotate(){
    std::cout  <<  "right  rotate  around  element:  "  <<  this->data  <<
std::endl;
    Node_AVL_Tree<Type>* temp;
    temp = left;

    temp->rotate = true;
    if (temp->left!=nullptr)
    temp->left->rotate = true;
    if (temp->right!=nullptr)
    temp->right->rotate = true;
```

```
    left = temp->right;
    this->height = this->set_height();
    this->balance = this->set_balance();
    if(temp->left){
        temp->left->height = temp->left->set_height();
        temp->left->balance = temp->left->set_balance();
    }
    temp->right = this;
    temp->height = temp->set_height();
    temp->balance = temp->set_balance();
    return temp;
}

template <class Type>
class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::left_rotate(){
    std::cout << "left rotate around element: " << this->data <<
std::endl;
    Node_AVL_Tree<Type>* temp;
    temp = right;

    temp->rotate = true;
    if (temp->left!=nullptr)
    temp->left->rotate = true;
    if (temp->right!=nullptr)
    temp->right->rotate = true;

    right = temp->left;
    this->height = this->set_height();
    this->balance = this->set_balance();
    if(temp->right){
        temp->right->height = temp->right->set_height();
        temp->right->balance = temp->right->set_balance();
    }
    temp->left = this;
    temp->height = temp->set_height();
    temp->balance = temp->set_balance();
    return temp;
}

template <class Type>
class Node_AVL_Tree<Type>* Node_AVL_Tree<Type>::make_balance(){//установка
баланса в дереве
    Node_AVL_Tree<Type>* temp;
    temp = this;
    if(balance == 2){
        if(right->balance == -1)
            temp->right = right->right_rotate();
        temp = left_rotate();
    }
    if(balance == -2){
```

```
        if(left->balance == 1)
            temp->left = left->left_rotate();
        temp = right_rotate();
    }
    return temp;
}

template <class Type>
int Node_AVL_Tree<Type>::set_height(){ // установка высоты
    if(!left && !right)
         return 1;
    if(!left)
        return (right->height + 1);
    if(!right)
        return (left->height + 1);
    if(left->height >= right->height)
        return (1 + left->height);
    if(left->height < right->height)
        return (1 + right->height);
    return 0;
}

template <class Type>
int Node_AVL_Tree<Type>::set_balance(){ // установка баланса в вершине
    if(!left && !right)
         return 0;
    if(!left)
        return right->height;
    if(!right)
        return (left->height * (-1));
    return (right->height - left->height);
}

template <class Type>
Head_AVL_Tree<Type>::Head_AVL_Tree(){
    head = nullptr;
}

template <class Type>
Head_AVL_Tree<Type>::~Head_AVL_Tree(){
    delete head;
}

template <class Type>
void Head_AVL_Tree<Type>::print_tree(){
    if(!head){
        std::cout << "tree is empty" << std::endl;
        return;
    }
    head->print_tree(0);
}
```

```
template <class Type>
void Head_AVL_Tree<Type>::insert(Type value){
    if(!head){
        Node_AVL_Tree<Type>* temp = new Node_AVL_Tree<Type>;
        temp->data = value;
        temp->height = 1;
        head = temp;
        return;
    }
    head = head->insert(value);
}

template <class Type>
bool Head_AVL_Tree<Type>::is_contain(Type desired){
    if(!head)
        return false;
    if(head->data == desired){
   std::cout << "this element is root" << std::endl;
        return true;
    }
    if(head->left && head->data > desired){
        std::cout << "find in left " << std::endl;
        return head->left->is_contain(desired, 1);
    }
    if(head->right && head->data < desired){
        std::cout << "find in right " << std::endl;
        return head->right->is_contain(desired, 1);
    }
    return false;
}

template <class Type>
void Head_AVL_Tree<Type>::remove(Type to_remove){
    head = head->remove(to_remove);
}




#endif
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <map>
#include <algorithm>
#include <memory>
#include <iostream>
#include <fstream>
#include <sstream>
#include <QMainWindow>
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>
#include <QFileDialog>
#include <QStandardPaths>
#include <QtGui>
#include <QLabel>
#include <QColorDialog>
#include <QInputDialog>
#include <QMainWindow>
#include <QPushButton>
#include <QMessageBox>
#include <QStringList>
#include <QTextBrowser>
#include <QTextEdit>
#include <stack>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();
```

```
    void on_pushButton_3_clicked();

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();

private:
    Ui::MainWindow *ui;
    QGraphicsScene *scene;
    //QTextEdit *text_sreen;
};
#endif // MAINWINDOW_H
```

main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

mainwindow.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <map>
#include <algorithm>
#include <memory>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstring>
#include <QMainWindow>
#include <QGraphicsItem>
#include <QGraphicsView>
#include <QGraphicsEffect>
#include <QFileDialog>
```

```
#include <QStandardPaths>
#include <QtGui>
#include <QLabel>
#include <QString>
#include <QColorDialog>
#include <QInputDialog>
#include <QMainWindow>
#include <QPushButton>
#include <QMessageBox>
#include <QStringList>
#include <QTextEdit>
#include <stack>
 #include <QTextEdit>
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "Tree.h"
static bool started = true;
static std::string Name;
static int action = 0;
static int del = 0;
typedef int Type;




MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),

    ui(new Ui::MainWindow)
{

    ui->setupUi(this);

    scene = new QGraphicsScene;
    ui->graphicsView->setScene(scene);
    ui->outTextEdit->setReadOnly(true);
    ui->inputTextEdit->setValidator(new        QRegExpValidator(QRegExp("[0-
9]{3}")));
}

MainWindow::~MainWindow()
{
    delete ui;
}



void MainWindow::on_pushButton_clicked()
{
```

```
    Head_AVL_Tree<Type> head;


        std::string main_str;
        std::ifstream file;


        file.open(Name);
        if (!file.is_open()) {
            ui->outTextEdit->insertPlainText("Error! File isn't open\n");
            return;
        }
        getline(file, main_str);
        ui->outTextEdit->insertPlainText("Input:             "             +
QString::fromStdString(main_str) +"\n");

        int count_num = 0;
        for(size_t i = 0; i < main_str.size(); i++){
            int temp = 0;
            bool is_num = false;
            for(size_t  j  =  i;  isdigit(main_str[j]);  j++,  i++){  //  create
tree from string
                temp = temp*10 + main_str[j]-'0';
                is_num = true;
            }
            if(is_num){
                count_num++;
                head.insert(temp);
            }

        }

      head.print_tree1(scene);

        if(!count_num){
            ui->outTextEdit->insertPlainText("no   one   element   to   tree.
Program will end\n\n");
            exit(0);
        }


        ui->outTextEdit->insertPlainText("WHAT TO DO? \n");
        for( ; ; )
        {
        QApplication::processEvents();
        if(!started)break;


        }
        started = true;
```

```
        while(1){
            switch (action) {



                case 0:
                    exit (0);
                case 1:
                    Type insert_tree_element = 0;
                    QString str= ui->inputTextEdit->text();
                    ui->inputTextEdit->clear();
                    insert_tree_element  = str.toInt();




                    if(del == 0)
                    {
                            ui->outTextEdit->insertPlainText("add  to  tree
new element: " + QString::number(insert_tree_element) + "\n");
                        if(insert_tree_element==0){
                            ui->outTextEdit->insertPlainText("input   error.
May be it`s not a number, very big number or more than 1 number\n");

                        }

                        else{
                            ui->outTextEdit->insertPlainText("find     this
element in AVL_tree: \n");
                        if(!head.is_contain(insert_tree_element)){
                            ui->outTextEdit->insertPlainText("tree  does  not
have this element\n");
                            head.insert(insert_tree_element);
                            ui->outTextEdit->insertPlainText("\n\n\n");
                            head.print_tree1(scene);
                        }
                        else
                            ui->outTextEdit->insertPlainText("element
already in tree\n");

                        }
                    }
                    else
                        if (del==1)
                    {
                        ui->outTextEdit->insertPlainText("try        delete
element: " + QString::number(insert_tree_element) +"\n");
                        if(insert_tree_element==0)
                        {
                            ui->outTextEdit->insertPlainText("input   error.
May be it`s not a number, very big number or more than 1 number\n");
```

```
                        }

                        else{
                            ui->outTextEdit->insertPlainText("delete
element: " + QString::number(insert_tree_element) +"\n");
                                                ui->outTextEdit-
>insertPlainText("find this element in AVL_tree: ");

if(!head.is_contain(insert_tree_element)){
                                                    ui->outTextEdit-
>insertPlainText("tree does not have this element");
                                                    head.print_tree1(scene);
                                                }
                                                else{

head.remove(insert_tree_element);
                                                    ui->outTextEdit-
>insertPlainText("tree after delete: ");
                                                    head.print_tree1(scene);
                                                    }

                         del = 0;

                    }
                         }
                    action = 0;
                    ui->outTextEdit->insertPlainText("MORE ACTION? ");

                    for( ; ; )
                    {
                    QApplication::processEvents();
                    if(!started)break;


                    }
                    started = true;



                    break;

}
}
}


void MainWindow::on_pushButton_2_clicked()
{
    QString FileName = QFileDialog::getOpenFileName(this, "OpenDialog",
    QDir::homePath(), "*.txt;; *.*");
```

```
    Name = FileName.toStdString();
}

void MainWindow::on_pushButton_3_clicked()
{
    started=false;
    action = 1;
}




void MainWindow::on_pushButton_4_clicked(){
    started=false;
    action = 0;
}

void MainWindow::on_pushButton_5_clicked()
{
    started=false;
    action = 1;
    del = 1;
}
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1218</width>
    <height>792</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QGridLayout" name="gridLayout_2">
    <item row="0" column="0">
     <widget class="QGraphicsView" name="graphicsView"/>
    </item>
```

```
<item row="0" column="1">
 <layout class="QVBoxLayout" name="verticalLayout">
  <item>
   <spacer name="verticalSpacer_3">
    <property name="orientation">
     <enum>Qt::Vertical</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>20</width>
      <height>98</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="pushButton_4">
    <property name="minimumSize">
     <size>
      <width>100</width>
      <height>60</height>
     </size>
    </property>
    <property name="maximumSize">
     <size>
      <width>100</width>
      <height>60</height>
     </size>
    </property>
    <property name="text">
     <string>Закончить</string>
    </property>
   </widget>
  </item>
  <item>
   <spacer name="verticalSpacer">
    <property name="orientation">
     <enum>Qt::Vertical</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>20</width>
      <height>258</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <widget class="QPushButton" name="pushButton_3">
    <property name="minimumSize">
     <size>
```

```
    <width>105</width>
    <height>60</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>105</width>
    <height>60</height>
   </size>
  </property>
  <property name="text">
   <string>Добавить элемент</string>
  </property>
 </widget>
</item>
<item>
 <spacer name="verticalSpacer_2">
  <property name="orientation">
   <enum>Qt::Vertical</enum>
  </property>
  <property name="sizeHint" stdset="0">
   <size>
    <width>20</width>
    <height>118</height>
   </size>
  </property>
 </spacer>
</item>
<item>
 <widget class="QPushButton" name="pushButton_5">
  <property name="minimumSize">
   <size>
    <width>105</width>
    <height>60</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>105</width>
    <height>16777215</height>
   </size>
  </property>
  <property name="text">
   <string>Удалить Элемент</string>
  </property>
 </widget>
</item>
<item>
 <spacer name="verticalSpacer_4">
  <property name="orientation">
   <enum>Qt::Vertical</enum>
```

```
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>20</width>
      <height>40</height>
     </size>
    </property>
   </spacer>
  </item>
 </layout>
</item>
<item row="2" column="0">
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <spacer name="horizontalSpacer">
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
     <size>
      <width>40</width>
      <height>20</height>
     </size>
    </property>
   </spacer>
  </item>
  <item>
   <layout class="QGridLayout" name="gridLayout">
    <item row="0" column="0">
     <widget class="QPushButton" name="pushButton_2">
      <property name="minimumSize">
       <size>
        <width>200</width>
        <height>60</height>
       </size>
      </property>
      <property name="maximumSize">
       <size>
        <width>200</width>
        <height>60</height>
       </size>
      </property>
      <property name="text">
       <string>Открыть файл</string>
      </property>
     </widget>
    </item>
    <item row="1" column="0">
     <widget class="QLineEdit" name="inputTextEdit">
      <property name="minimumSize">
       <size>
```

```
      <width>200</width>
      <height>60</height>
     </size>
    </property>
    <property name="maximumSize">
     <size>
      <width>200</width>
      <height>60</height>
     </size>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <widget class="QTextEdit" name="outTextEdit">
  <property name="minimumSize">
   <size>
    <width>350</width>
    <height>120</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>350</width>
    <height>120</height>
   </size>
  </property>
 </widget>
</item>
<item>
 <widget class="QPushButton" name="pushButton">
  <property name="minimumSize">
   <size>
    <width>200</width>
    <height>60</height>
   </size>
  </property>
  <property name="maximumSize">
   <size>
    <width>200</width>
    <height>60</height>
   </size>
  </property>
  <property name="text">
   <string>Построить и нарисовать дерево</string>
  </property>
 </widget>
</item>
<item>
 <spacer name="horizontalSpacer_3">
```

```
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
        <property name="sizeHint" stdset="0">
         <size>
          <width>201</width>
          <height>20</height>
         </size>
        </property>
       </spacer>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```

CW_pro

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain
version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000      # disables all the
APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    Tree.h \
```

```
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
     !isEmpty(target.path): INSTALLS += target
```