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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Рандомизированные пирамиды поиска

Студент гр. 8381 _____ Сергеев А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными характеристиками и особенностями рандомизированных пирамид поиска, изучить особенности их реализации на языке программирования C++. Разработать программу, демонстрирующую принцип работы пирамид.

Задание.

- По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру данных определённого типа – БДП.

- Реализовать сцепление двух пирамид.

Основные теоретические положения.

Рандомизированная пирамида поиска — это бинарное дерево, в узлах которого хранятся пары (x,y), где x — это ключ, а y — это приоритет. Также оно является двоичным деревом поиска по x и пирамидой по y. Предполагая, что все x и все y являются различными, получаем, что если некоторый элемент дерева содержит (x0,y0), то у всех элементов в левом поддереве $x<x0$, у всех элементов в правом поддереве $x>x0$, а также и в левом, и в правом поддереве имеем: $y<y0$.

При каждой вставке в пирамиду очередного элемента ключ выбирается случайно из чисел, принадлежащих определённому отрезку.

Выполнение работы.

Написание работы производилось на базе операционной системы *Ubuntu*, в среде *CLion*, а также с использованием библиотек *qt* и среды *QTCreator*.

Для выполнения поставленной задачи был создан класс *rand_tree*, содержащий в себе методы работы с бинарным деревом поиска, также был создан класс *tree_node*, в котором была реализована вставка узлов в пирамиду при помощи двух методов: *split* и *merge*.

Split позволяет разрезать дерево по ключу *k*, так, что в одной половине все ключи будут меньше *k*, а в другой — больше. С помощью метода merge можно

слить два дерева в одно (но только при условии, что все ключи первого поддерева будут меньше ключей правого).

Класс *lab5* содержит в себе все методы для выполнения задания: построение пирамиды (в быстром и пошаговом режиме, с данными, введёнными в строку и в файл), слияние двух пирамид (в быстром и пошаговом режиме). Реализован пошаговый режим через простейший стэк (класс *stack*) на базе списка.

Для изображения пирамиды на экране используется класс *myglwidget*, наследующий от класса *qopenglwidget*, и содержащий два режима отрисовки дерева — с узлами и без. Режим отрисовки выбирается в зависимости от глубины дерева. Для наглядности узел, в котором было сделано последнее изменение подсвечивается красным (или, когда работа с пирамидой закончена, красным подсвечивается корень).

Дополнительно реализован ещё один, тестовый режим работы с пирамидой, проводящий построение 10000 раз и выводящий на экран среднюю и идеальную высоту.

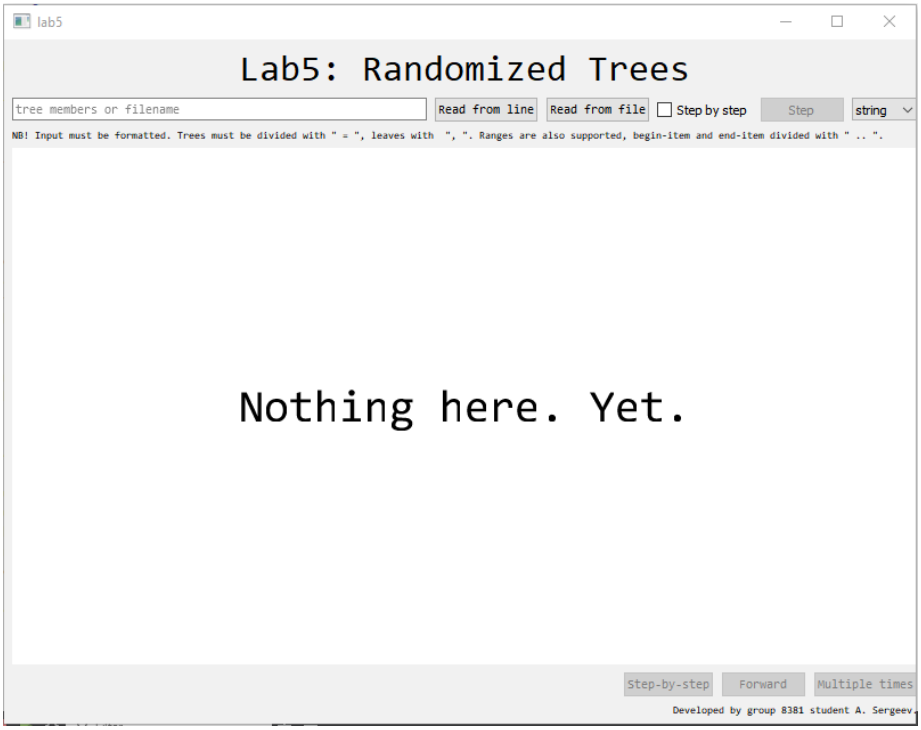
Исходный код программы см. в Приложении А.

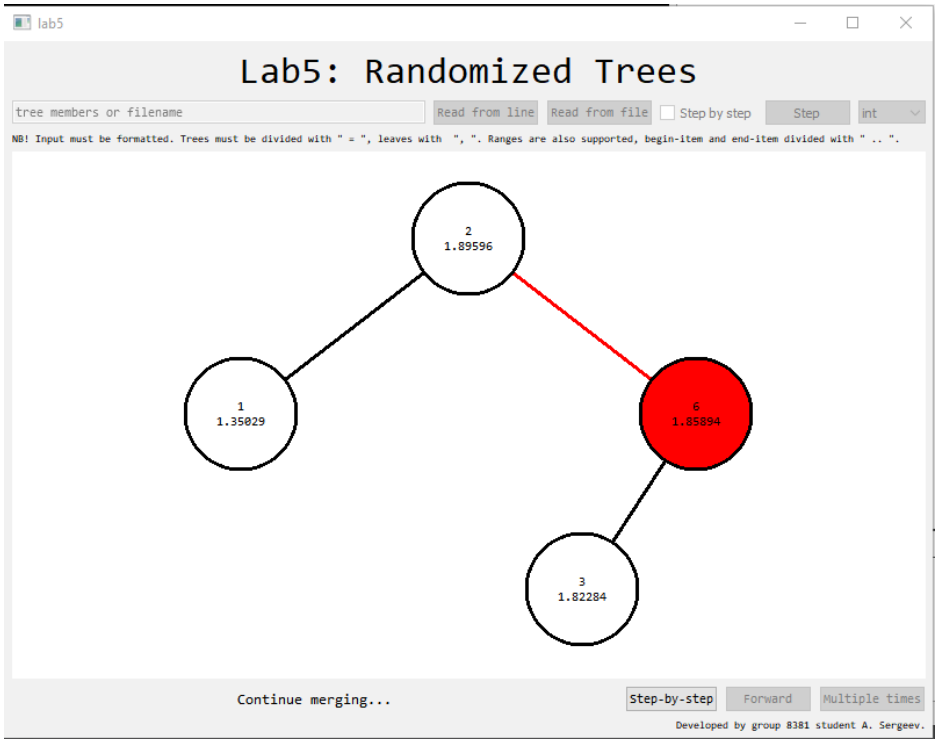
Оценка эффективности алгоритма.

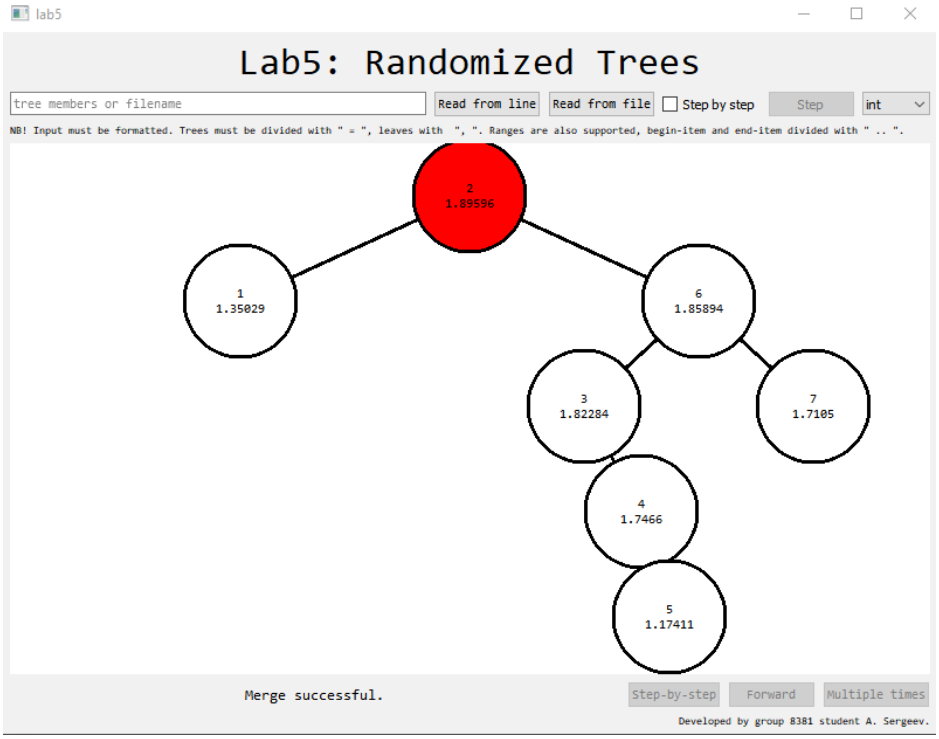
Все методы работы с пирамидой реализованы через вставку узлов в пирамиду. Вставка узлов состоит из двух методов merge и одного split. Как split, так и merge теоретически требуют O(n) операций, где n – высота дерева, так как в них рекурсивно вызывается O(1) операций для каждого дерева меньшей высоты. Операция слияния двух пирамид (ключи в которых не сортированы) требует рекурсивного обхода второй пирамиды, что, теоретически требует O(k) операций, где k – количество элементов во второй пирамиде.

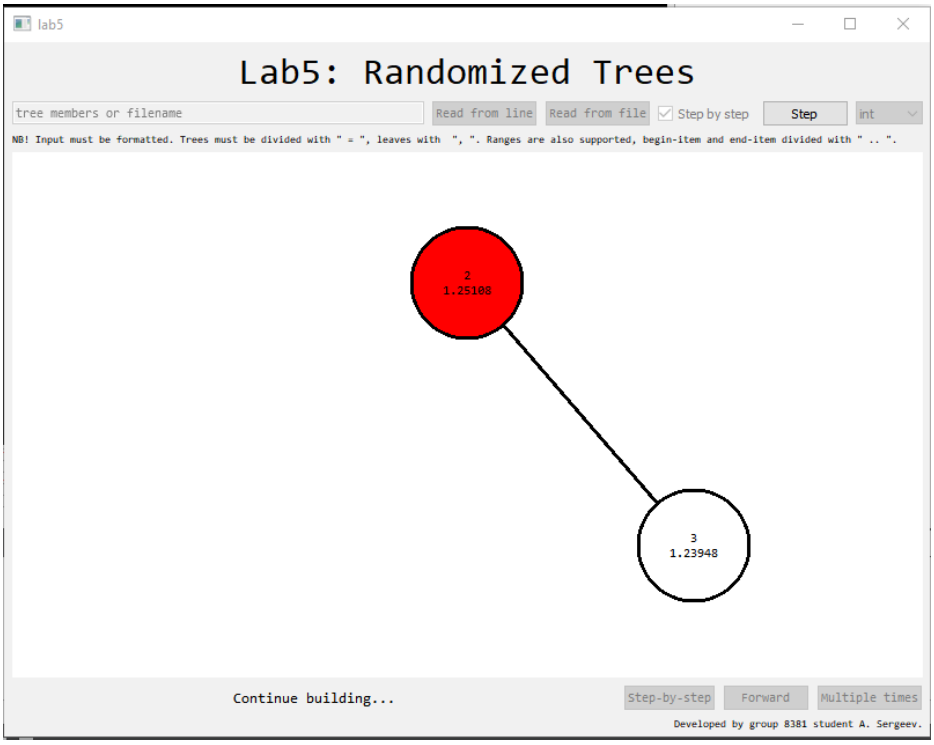
Тестирование программы.

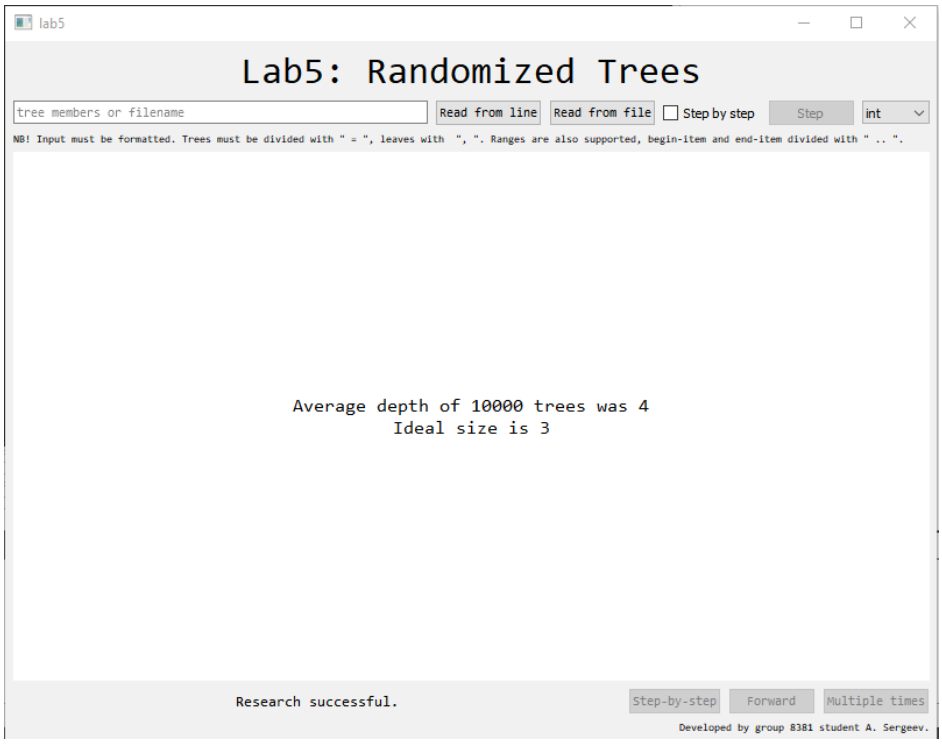
Ниже представлены снимки экрана работающей в режиме *gui* программы, а также результаты трёх различных тестов.




lab5
Lab5: Randomized Trees
tree members or filename
Read from line
Read from file
Step by step
Step
int
NB! Input must be formatted. Trees must be divided with " = ", leaves with ", ". Ranges are also supported, begin-item and end-item divided with " .. ".
2
1.89596
1
1.35029
6
1.85894
3
1.82284
Continue merging...
Step-by-step
Forward
Multiple times
Developed by group 8381 student A. Sergeev.


lab5
Lab5: Randomized Trees
tree members or filename
Read from line
Read from file
Step by step
Step
int
NB! Input must be formatted. Trees must be divided with " = ", leaves with ", ". Ranges are also supported, begin-item and end-item divided with " .. ".
2
1.89596
1
1.35029
6
1.85894
3
1.82284
7
1.7105
4
1.7466
5
1.17411
Merge successful.
Step-by-step
Forward
Multiple times
Developed by group 8381 student A. Sergeev.





Входные данные: «1 .. 3 = 4 .. 7», тип: *int*.

Выводы.

В ходе выполнения лабораторной работы была изучена такая структура данных как пирамида, а также методы её обработки. Была реализована программа на C++, демонстрирующая принцип работы рандомизированных пирамид поиска.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include "mainwindow.h"

#include <QApplication>
#include <QtQuick>
#include <sstream>
#include <iostream>

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

Файл lab5.h:

```
#ifndef LAB5_H
#define LAB5_H

#include "rand_tree.h"
#include "myglwidget.h"
#include <iostream>
#include <fstream>
#include <sstream>

class lab5 {
public:
     static int launch(MyGLWidget* mglw, std::string* input, bool
ff, int tp, bool step);

    template<typename T>
    static int launch_steps(MyGLWidget* mglw) {
        if (!tree1_stck<T>->isEmpty()) {
            auto val = tree1_stck<T>->pop();
            tree1<T>->add(val);

            mglw->set_tree(tree1<T>, val);
            mglw->show(&output_file);
            return 1;
        } else if (!tree2_stck<T>->isEmpty()) {
            auto val = tree2_stck<T>->pop();
            tree2<T>->add(val);
```

```
                mglw->set_tree(tree2<T>, val);
                mglw->show(&output_file);
                return 1;
            } else {
                free(tree1_stck<T>);
                tree1_stck<T> = nullptr;
                free(tree2_stck<T>);
                tree2_stck<T> = nullptr;

                        mglw->set_tree(tree2<T>,  tree2<T>->get_root()-
>get_state());
                mglw->show(&output_file);
                return 0;
            }
        }

        static int type;

        template<typename T>
        static rand_tree<T>* tree1;

        template<typename T>
        static rand_tree<T>* tree2;



        template<typename T>
        static int step(MyGLWidget* mglw) {
            if (stck<T> == nullptr) {
                stck<T> = new stack<tree_node<T>*>();
                stck<T>->push(tree2<T>->get_root());

                        mglw->set_tree(tree1<T>,  tree1<T>->get_root()-
>get_state());
                mglw->show(&output_file);
                return 1;

            } else if (!stck<T>->isEmpty()) {
                auto node = stck<T>->pop();

                tree1<T>->join(node, stck<T>);
                tree1<T>->get_root()->reset_weight();

                mglw->set_tree(tree1<T>, node->get_state());
                mglw->show(&output_file);

                free(node);
                return 1;
```

```
            } else {
                free(stck<T>);
                stck<T> = nullptr;

                        mglw->set_tree(tree1<T>,  tree1<T>->get_root()-
>get_state());
                mglw->show(&output_file);
                return 0;
            }
        }

        template<typename T>
        static int rush(MyGLWidget* mglw) {
            tree1<T>->join(tree2<T>->get_root(), nullptr);
                    mglw->set_tree(tree1<T>,  tree1<T>->get_root()-
>get_state());
            mglw->show(&output_file);
            return 0;
        }

        template<typename T>
        static int mult(MyGLWidget* mglw) {
            rand_tree<T>* new_tree = nullptr;
            long long int mid_size = 0;
            long long int theo_size = 0;

            for (int i = 0; i < 10000; i++) {
                auto tree_string = new std::string(input_str);
                new_tree = new rand_tree<T>(tree_string, nullptr);
                        if (theo_size == 0) theo_size = new_tree-
>theory_depth();
                mid_size += new_tree->max_depth();
                free(new_tree);
                free(tree_string);
                new_tree = nullptr;
            }

            std::stringstream ss;
              ss << "Average depth of 10000 trees was " << mid_size /
10000 << "\nIdeal size is " << theo_size;
            auto msg = new std::string(ss.str());
            mglw->declare(msg);

            return 0;
        }


    private:
        static const std::string input_file;
```

```
        static const std::string output_file;

        static std::string input_str;

        template<typename T>
        static stack<tree_node<T>*>* stck;

        template<typename T>
        static stack<T>* tree1_stck;
        template<typename T>
        static stack<T>* tree2_stck;
    };

    template<typename T>
    stack<tree_node<T>*>* lab5::stck = nullptr;

    template<typename T>
    stack<T>* lab5::tree1_stck = nullptr;
    template<typename T>
    stack<T>* lab5::tree2_stck = nullptr;



    template<typename T>
    rand_tree<T>* lab5::tree1 = nullptr;

    template<typename T>
    rand_tree<T>* lab5::tree2 = nullptr;



    #endif // LAB5_H
```

Файл lab5.cpp:

```
    #include "lab5.h"


    const           std::string           lab5::input_file           =
"C:/Users/miles/Documents/lab5/in.txt";
    const           std::string           lab5::output_file          =
"C:/Users/miles/Documents/lab5/out.png";

    const std::string tree_divider = " = ";

    int lab5::type = 0;

    std::string lab5::input_str = "";
```

```
     int lab5::launch(MyGLWidget* mglw, std::string* input, bool ff, int
tp, bool step) {
          mglw->prepare_drawing();

          std::string tree_string;
          if (ff) {
              std::ifstream is;
              if (input->empty()) {
                  is = std::ifstream(input_file);
              } else {
                  is = std::ifstream(*input);
              }

              if (is && !is.fail()) {
                  getline(is, tree_string);
              } else {
                    auto msg = new std::string("File can not be opened :
(");
                  mglw->declare(msg);
                  return 1;
              }
          } else {
              if (input->empty()) {
                  auto msg = new std::string("The input was empty.");
                  mglw->declare(msg);
                  return 1;
              } else {
                  tree_string = *input;
              }
          }

          unsigned long long div = tree_string.find(tree_divider);
          if (div == std::string::npos) {
                auto msg = new std::string("String for only one tree was
provided.");
              mglw->declare(msg);
              return 1;
          }

          std::string first = tree_string.substr(0, div);
                    std::string   second   =   tree_string.substr(div   +
tree_divider.length(), tree_string.length() - 1);
          if (first.empty() || second.empty()) {
               auto msg = new std::string("The input for one of the trees
was empty.");
              mglw->declare(msg);
              return 1;
          }
```

```
          input_str = tree_string.replace(div, tree_divider.length(), ",
");

          type = tp;

          try {
              switch (type) {
                  case 0: {
                      if (step) {
                                              tree1_stck<std::string>  =  new
stack<std::string>();
                                              tree2_stck<std::string>  =  new
stack<std::string>();
                      }
                                                 tree1<std::string>   =   new
rand_tree<std::string>(&first, tree1_stck<std::string>);
                                                 tree2<std::string>   =   new
rand_tree<std::string>(&second, tree2_stck<std::string>);
                      break;
                  }
                  case 1: {
                      if (step) {
                          tree1_stck<char> = new stack<char>();
                          tree2_stck<char> = new stack<char>();
                      }
                               tree1<char>  =  new  rand_tree<char>(&first,
tree1_stck<char>);
                              tree2<char>  =  new  rand_tree<char>(&second,
tree2_stck<char>);
                      break;
                  }
                  case 2: {
                      if (step) {
                          tree1_stck<int> = new stack<int>();
                          tree2_stck<int> = new stack<int>();
                      }
                                tree1<int>  =  new  rand_tree<int>(&first,
tree1_stck<int>);
                               tree2<int>  =  new  rand_tree<int>(&second,
tree2_stck<int>);
                      break;
                  }
                  case 3: {
                      if (step) {
                          tree1_stck<double> = new stack<double>();
                          tree2_stck<double> = new stack<double>();
                      }
                            tree1<double>  =  new  rand_tree<double>(&first,
tree1_stck<double>);
```

```
                        tree2<double> = new rand_tree<double>(&second,
tree2_stck<double>);
                    break;
                }
            }

        } catch (std::runtime_error re) {
            auto msg = new std::string(re.what());
            mglw->declare(msg);
            return 1;
        }

        std::string* msg;
        if (step) {
            msg = new std::string("Both strings successfully loaded. We
are ready 2 build.");
        } else {
             msg = new std::string("Both trees successfully loaded. We
are ready 2 go.");
        }

        mglw->declare(msg);
        return ((step) ? (2) : (0));
    }
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
  Q_OBJECT

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

private:
  Ui::MainWindow *ui;

  void enable_input(bool val);
  void step_input(bool val);
```

```
private slots:
  void read_from_file();
  void read_from_line();
  void step();
  void step_forward();
  void rush_forward();
  void mult_forward();
};

#endif // MAINWINDOW_H
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "lab5.h"

MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent),
ui(new Ui::MainWindow) {
  ui->setupUi(this);
     QStringList available_types = {"string", "char", "int",
"double"};

  ui->type_selector->addItems(available_types);

        connect(ui->line_button, SIGNAL(clicked()), this,
SLOT(read_from_line()));
        connect(ui->file_button, SIGNAL(clicked()), this,
SLOT(read_from_file()));
  connect(ui->step_button, SIGNAL(clicked()), this, SLOT(step()));

    connect(ui->straightforward_button, SIGNAL(clicked()), this,
SLOT(rush_forward()));
       connect(ui->proceed_button, SIGNAL(clicked()), this,
SLOT(step_forward()));
      connect(ui->multiple_button, SIGNAL(clicked()), this,
SLOT(mult_forward()));
}

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


void MainWindow::enable_input(bool val) {
    ui->straightforward_button->setEnabled(!val);
    ui->proceed_button->setEnabled(!val);
    ui->multiple_button->setEnabled(!val);
```

```
        ui->line_button->setEnabled(val);
        ui->file_button->setEnabled(val);
        ui->type_selector->setEnabled(val);
        ui->line_input->setEnabled(val);
        ui->step_box->setEnabled(val);
    }

    void MainWindow::step_input(bool val) {
        ui->straightforward_button->setEnabled(!val);
        ui->proceed_button->setEnabled(!val);
        ui->multiple_button->setEnabled(!val);

        ui->step_button->setEnabled(val);
    }


    void MainWindow::read_from_file() {
        std::string input = ui->line_input->text().toStdString();
           int result = lab5::launch(ui->canvas, &input, true, ui-
>type_selector->currentIndex(), ui->step_box->isChecked());

        if (result == 0) {
            enable_input(false);
            ui->answer->setText("Both trees successfully loaded.");
        } else if (result == 2) {
            enable_input(false);
            step_input(true);
            ui->answer->setText("Continue building...");
        } else {
            ui->answer->setText("Error occures, input again.");
        }
    }

    void MainWindow::read_from_line() {
        std::string input = ui->line_input->text().toStdString();
           int result = lab5::launch(ui->canvas, &input, false, ui-
>type_selector->currentIndex(), ui->step_box->isChecked());

        if (result == 0) {
            enable_input(false);
            ui->answer->setText("Both trees successfully loaded.");
        } else if (result == 2) {
            enable_input(false);
            step_input(true);
            ui->answer->setText("Continue building...");

        } else {
```

```
        ui->answer->setText("Error occures, input again.");
    }
}


void MainWindow::step() {
    int result = 1;

    switch (lab5::type) {
        case 0:
            result = lab5::launch_steps<std::string>(ui->canvas);
            break;
        case 1:
            result = lab5::launch_steps<char>(ui->canvas);
            break;
        case 2:
            result = lab5::launch_steps<int>(ui->canvas);
            break;
        case 3:
            result = lab5::launch_steps<double>(ui->canvas);
            break;
    }

    if (result == 0) {
        step_input(false);
        ui->answer->setText("Both trees successfully loaded.");
    } else {
        ui->answer->setText("Continue building...");
    }
}


void MainWindow::step_forward() {
    ui->straightforward_button->setEnabled(false);
    ui->multiple_button->setEnabled(false);

    int result = 1;

    switch (lab5::type) {
        case 0:
            result = lab5::step<std::string>(ui->canvas);
            break;
        case 1:
            result = lab5::step<char>(ui->canvas);
            break;
        case 2:
            result = lab5::step<int>(ui->canvas);
            break;
        case 3:
```

```
            result = lab5::step<double>(ui->canvas);
            break;
    }

    if (result == 0) {
        enable_input(true);
        ui->answer->setText("Merge successful.");
    } else {
        ui->answer->setText("Continue merging...");
    }
}

void MainWindow::rush_forward() {
    int result = 1;

    switch (lab5::type) {
        case 0:
            result = lab5::rush<std::string>(ui->canvas);
            break;
        case 1:
            result = lab5::rush<char>(ui->canvas);
            break;
        case 2:
            result = lab5::rush<int>(ui->canvas);
            break;
        case 3:
            result = lab5::rush<double>(ui->canvas);
            break;
    }

    if (result == 0) {
        enable_input(true);
        ui->answer->setText("Merge successful.");
    } else {
        ui->answer->setText("Error occures, merge again.");
    }
}

void MainWindow::mult_forward() {
    int result = 1;

    switch (lab5::type) {
        case 0:
            result = lab5::mult<std::string>(ui->canvas);
            break;
        case 1:
            result = lab5::mult<char>(ui->canvas);
            break;
        case 2:
```

```
            result = lab5::mult<int>(ui->canvas);
            break;
        case 3:
            result = lab5::mult<double>(ui->canvas);
            break;
    }

    if (result == 0) {
        enable_input(true);
        ui->answer->setText("Research successful.");
    } else {
        ui->answer->setText("Error occures, research again.");
    }
}
```

Файл 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>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="toolTipDuration">
   <number>5</number>
  </property>
  <widget class="QWidget" name="central_widget">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <widget class="QWidget" name="verticalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>0</y>
      <width>801</width>
      <height>601</height>
     </rect>
    </property>
    <property name="sizePolicy">
     <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
```

```
  <horstretch>0</horstretch>
  <verstretch>0</verstretch>
 </sizepolicy>
</property>
<layout class="QVBoxLayout" name="container">
 <property name="leftMargin">
  <number>7</number>
 </property>
 <property name="topMargin">
  <number>7</number>
 </property>
 <property name="rightMargin">
  <number>7</number>
 </property>
 <property name="bottomMargin">
  <number>7</number>
 </property>
 <item>
  <widget class="QLabel" name="name">
   <property name="sizePolicy">
    <sizepolicy hsizetype="Preferred" vsizetype="Minimum">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <property name="font">
    <font>
     <family>Consolas</family>
     <pointsize>24</pointsize>
    </font>
   </property>
   <property name="text">
    <string>Lab5: Randomized Trees</string>
   </property>
   <property name="alignment">
    <set>Qt::AlignCenter</set>
   </property>
  </widget>
 </item>
 <item>
  <layout class="QHBoxLayout" name="input_container">
   <item>
    <widget class="QLineEdit" name="line_input">
     <property name="font">
      <font>
       <family>Consolas</family>
      </font>
     </property>
     <property name="placeholderText">
      <string>tree members or filename</string>
```

```
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="line_button">
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Read from line</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="file_button">
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Read from file</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QCheckBox" name="step_box">
    <property name="text">
     <string>Step by step</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="step_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="text">
     <string>Step</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QComboBox" name="type_selector"/>
  </item>
 </layout>
</item>
<item>
```

```
            <widget class="QLabel" name="label">
             <property name="font">
              <font>
               <family>Consolas</family>
               <pointsize>7</pointsize>
              </font>
             </property>
             <property name="text">
              <string>NB! Input must be formatted. Trees must be divided with &amp;quot; = &amp;quot;, leaves with  &amp;quot;, &amp;quot;. Ranges are also supported,  begin-item  and  end-item  divided  with  &amp;quot;  .. &amp;quot;.</string>
             </property>
            </widget>
           </item>
           <item>
            <widget class="MyGLWidget" name="canvas">
             <property name="sizePolicy">
              <sizepolicy hsizetype="Preferred" vsizetype="Expanding">
               <horstretch>0</horstretch>
               <verstretch>0</verstretch>
              </sizepolicy>
             </property>
            </widget>
           </item>
           <item>
            <layout class="QHBoxLayout" name="output_container">
             <item>
              <widget class="QLabel" name="answer">
               <property name="sizePolicy">
                <sizepolicy hsizetype="Expanding" vsizetype="Preferred">
                 <horstretch>0</horstretch>
                 <verstretch>0</verstretch>
                </sizepolicy>
               </property>
               <property name="font">
                <font>
                 <family>Consolas</family>
                 <pointsize>10</pointsize>
                </font>
               </property>
               <property name="alignment">
                <set>Qt::AlignCenter</set>
               </property>
              </widget>
             </item>
             <item>
              <widget class="QPushButton" name="proceed_button">
               <property name="enabled">
                <bool>false</bool>
```

```
    </property>
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Step-by-step</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="straightforward_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Forward</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="multiple_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Multiple times</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <widget class="QLabel" name="subscription">
  <property name="font">
   <font>
    <family>Consolas</family>
    <pointsize>7</pointsize>
   </font>
```

```
            </property>
            <property name="text">
                        <string>Developed by group 8381 student A.
Sergeev.</string>
            </property>
            <property name="alignment">
              <set>Qt::AlignRight|Qt::AlignTrailing|Qt::AlignVCenter</set
>
            </property>
           </widget>
          </item>
         </layout>
        </widget>
       </widget>
      </widget>
      <customwidgets>
       <customwidget>
        <class>MyGLWidget</class>
        <extends>QOpenGLWidget</extends>
        <header location="global">myglwidget.h</header>
       </customwidget>
      </customwidgets>
      <resources/>
      <connections/>
     </ui>
```

Файл myglwidget.cpp:

```
#include "myglwidget.h"

MyGLWidget::MyGLWidget(QWidget *parent) : QOpenGLWidget(parent) {}

MyGLWidget::~MyGLWidget() {}


void MyGLWidget::paintGL() {
    if (map != nullptr) {
        QPainter painter(this);
        painter.drawPixmap(this->rect(), *map, map->rect());
    } else {
        QPainter painter(this);
        painter.fillRect(this->rect(), QBrush(Qt::white));
        painter.setPen(Qt::black);
        painter.setFont(QFont("Consolas", 30));
           painter.drawText(rect(), Qt::AlignCenter, "Nothing here.
Yet.");
    }
}
```

```
void MyGLWidget::show(const std::string* outfile) {
    this->update();

    QFile file(QString::fromStdString(*outfile));
    file.open(QIODevice::WriteOnly);
    map->save(&file, "PNG");
}
```

Файл myglwidget.h:

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QOpenGLWidget>
#include <QPainter>
#include <QFile>
#include <iostream>
#include <sstream>
#include "rand_tree.h"

const int BONES_LIMIT = 6;


class MyGLWidget : public QOpenGLWidget {
public:
    QPixmap* map = nullptr;


    MyGLWidget(QWidget *parent);
    ~MyGLWidget() override;


    template<typename T>
    void set_tree(rand_tree<T>* source, T outline) {
        pntr = new QPainter(map);
        pntr->fillRect(map->rect(), QBrush(Qt::white));
        pntr->setBrush(Qt::white);
        pntr->setFont(QFont("Consolas", 8));

        boolean drawing_bones = source->max_depth() >= BONES_LIMIT;
        if (!drawing_bones) pntr->setPen(QPen(Qt::black, 3));

        unsigned long depth = source->max_depth();
              stepY = static_cast<unsigned int>(map->height()) /
static_cast<unsigned int>(depth);

              paint_node(source->get_root()->get_left(), false, map-
>width() / 2, stepY / 2, map->width() / 4, outline);
```

```
                paint_node(source->get_root()->get_right(), true, map-
>width() / 2, stepY / 2, map->width() / 4, outline);
            if (!drawing_bones) {
                std::stringstream knot;
                    knot << source->get_root()->get_state() << "\n" <<
source->get_root()->get_index();

                if (source->get_root()->get_state() == outline) {
                    pntr->setBrush(Qt::red);
                     pntr->drawEllipse(QPointF(map->width() / 2, stepY /
2), 48, 48);
                    pntr->setBrush(Qt::white);
                } else {
                     pntr->drawEllipse(QPointF(map->width() / 2, stepY /
2), 48, 48);
                }
                 pntr->drawText(QRect(map->width() / 2 - 24, stepY / 2 -
24, 48, 48), Qt::AlignCenter, QString::fromStdString(knot.str()));

                   populate_node(source->get_root()->get_left(), false,
map->width() / 2, stepY / 2, map->width() / 4, outline);
                   populate_node(source->get_root()->get_right(), true,
map->width() / 2, stepY / 2, map->width() / 4, outline);
            }

            stepY = 0;
            pntr->end();
            free(pntr);
        }

        void declare(std::string* msg) {
            pntr = new QPainter(map);
            pntr->fillRect(map->rect(), QBrush(Qt::white));
            pntr->setPen(Qt::black);
            pntr->setFont(QFont("Consolas", 12));
                            pntr->drawText(rect(),   Qt::AlignCenter,
QString::fromStdString(*msg));
            pntr->end();
            free(pntr);

            this->update();
        }

        void prepare_drawing() {
            free(this->map);
            this->map = new QPixmap(this->width(), this->height());
        }

        void show(const std::string* outfile);
```

```
    protected:
        void paintGL() override;

    private:
        unsigned int stepY = 0;
        QPainter* pntr;

        template<typename T>
           void paint_node(tree_node<T>* node, boolean is_right, int
parent_x, int parent_y, int half_stepX, T outline) {
            if (node == nullptr) return;

              int child_x = parent_x + ((is_right) ? (half_stepX) : (-
half_stepX));
            int child_y = parent_y + stepY;

            if (node->get_state() == outline) {
                pntr->setPen(QPen(Qt::red, pntr->pen().width()));
                pntr->drawLine(parent_x, parent_y, child_x, child_y);
                pntr->setPen(QPen(Qt::black, pntr->pen().width()));
            } else {
                pntr->drawLine(parent_x, parent_y, child_x, child_y);
            }



                 paint_node(node->get_left(), false, child_x, child_y,
half_stepX / 2, outline);
                 paint_node(node->get_right(), true, child_x, child_y,
half_stepX / 2, outline);
        }

        template<typename T>
          void populate_node(tree_node<T>* node, boolean is_right, int
parent_x, int parent_y, int half_stepX, T outline) {
            if (node == nullptr) return;

              int child_x = parent_x + ((is_right) ? (half_stepX) : (-
half_stepX));
            int child_y = parent_y + stepY;

            std::stringstream knot;
            knot << node->get_state() << "\n" << node->get_index();

            if (node->get_state() == outline) {
                pntr->setBrush(Qt::red);
                pntr->drawEllipse(QPointF(child_x, child_y), 48, 48);
                pntr->setBrush(Qt::white);
```

```
            } else {
                pntr->drawEllipse(QPointF(child_x, child_y), 48, 48);
            }
             pntr->drawText(QRect(child_x - 24, child_y - 24, 48, 48),
Qt::AlignCenter, QString::fromStdString(knot.str()));

            if (node->get_left() != nullptr) {
                        populate_node(node->get_left(), false, child_x,
child_y, half_stepX / 2, outline);
            }
            if (node->get_right() != nullptr) {
                        populate_node(node->get_right(), true, child_x,
child_y, half_stepX / 2, outline);
            }
        }
    };

    #endif // MYGLWIDGET_H
```

Файл rand_tree.h:

```
#ifndef CURR_RAND_TREE_H
#define CURR_RAND_TREE_H

#include <cmath>
#include <experimental/type_traits>
#include <stdexcept>
#include <sstream>
#include "tree_node.h"


template<typename C>
class rand_tree {
private:
    tree_node<C>* root = nullptr;

public:
    unsigned long max_depth();
    unsigned long theory_depth();
    unsigned long get_weight();

    tree_node<C>* get_root();
    tree_node<C>** get_bi_root();

    void join(tree_node<C>* node, stack<tree_node<C>*>* stck);
    void add(C value);

    explicit rand_tree();
```

```
        explicit rand_tree(std::string* tree_string, stack<C>* stck);
    };


    template<typename C>
    using     less_than_t     =     decltype(std::declval<C>()     <
std::declval<C>());

    template<typename C>
    using     more_than_t     =     decltype(std::declval<C>()     >
std::declval<C>());

    template<typename C>
    constexpr              bool              comparable              =
std::experimental::is_detected<less_than_t,         C>::value         &&
std::experimental::is_detected<more_than_t, C>::value;


    template<typename C>
    unsigned long rand_tree<C>::max_depth() {
        return root->depth(root);
    }

    template<typename C>
    unsigned long rand_tree<C>::theory_depth() {
        unsigned long weight = get_weight();
        unsigned long grade = 0;
        unsigned int power = 0;
        while (grade < weight) {
            power++;
            grade = static_cast<unsigned long>(pow(2, power) - 1);
        }
        return power;
    }

    template<typename C>
    unsigned long rand_tree<C>::get_weight() {
        return root->get_weight();
    }


    template<typename C>
    tree_node<C>* rand_tree<C>::get_root() {
        return root;
    }

    template<typename C>
    tree_node<C>** rand_tree<C>::get_bi_root() {
```

```
        return &root;
    }


    template<typename C>
    void  rand_tree<C>::join(tree_node<C>*  node,  stack<tree_node<C>*>*
stck) {
        if (root == nullptr) {
            root = node;
        } else {
            root = tree_node<C>::insert(root, node->trim());

            if (stck == nullptr) {
                if (node->get_left() != nullptr) join(node->get_left(),
stck);
                        if  (node->get_right()  !=  nullptr)  join(node-
>get_right(), stck);
            } else {
                    if (node->get_right() != nullptr) stck->push(node-
>get_right());
                     if (node->get_left() != nullptr) stck->push(node-
>get_left());
            }
        }
    }

    template<typename C>
    void rand_tree<C>::add(C value) {
        auto node = new tree_node<C>(value);
        if (root == nullptr) {
            root = node;
        } else {
            if (!tree_node<C>::is(root, value)) {
                root = tree_node<C>::insert(root, node);
                root->reset_weight();
            }
        }
    }


    template<typename C>
    rand_tree<C>::rand_tree() {
        if constexpr(!comparable<C>){
            throw std::runtime_error("Tree nodes can not be compared by
the key!");
        }
    }

    template<typename C>
```

```
     rand_tree<C>::rand_tree(std::string* tree_string, stack<C>* stck) {
         if constexpr(!comparable<C>){
             throw std::runtime_error("Tree nodes can not be compared by
the key!");
         }

         try {
             std:: string delimiter = ", ";
             std::string range = " .. ";
             size_t pos = 0;
             size_t erase_pos = 0;
             std::string token;

             do {
                 pos = tree_string->find(delimiter);
                  erase_pos = ((pos == std::string::npos) ? tree_string-
>length() : pos);
                 token = tree_string->substr(0, pos);

                 unsigned long tos = token.find(range);
                 unsigned long npos = std::string::npos;
                 if (tos != npos) {
                          if (std::is_same<C, std::string>::value) throw
std::runtime_error("Ranges are not available for strings yet.");

                     C first, second;
                     std::stringstream first_part(token.substr(0, tos));
                        std::stringstream second_part(token.substr(tos +
range.length(), token.length() - 1));
                     first_part >> first;
                     second_part >> second;

                     for (C i = first; i <= second; i += 1) {
                         if (stck == nullptr) {
                             this->add(i);
                         } else {
                             stck->push(i);
                         }
                     }
                 } else {
                     C number;
                     std::stringstream num(token);
                     std::string str = num.str();
                     num >> number;
                     if (stck == nullptr) {
                         this->add(number);
                     } else {
                         stck->push(number);
                     }
```

```
                }

                tree_string->erase(0, erase_pos + delimiter.length());
            } while (tree_string->length() != 0);
        } catch (std::runtime_error re) {
            throw std::runtime_error(re.what());
        } catch (...) {
             throw std::runtime_error("Tree string contains error. Tree
can not be built.");
        }
    }


    #endif //CURR_RAND_TREE_H
```

Файл stack.h:

```
    #ifndef STACK_H
    #define STACK_H


    #include <cstdlib>
    #include <stdexcept>

    template <typename T>
    class stack {
    private:
        class stack_element {
        private:
            T value;
            stack_element* next;
            stack_element* previous;

        public:
            stack_element(T& value, stack_element *previous);

            virtual ~stack_element();

            T& getValue();

            void setNext(stack_element *next);

            stack_element *getNext();

            stack_element *getPrevious();
        };

        int size;
```

```
        stack_element* first;
        stack_element* last;

    public:
        stack();
        ~stack();

        void push(T element);
        T pop();

        bool isEmpty();
        const int* getStackSize() const;
    };



    template<typename T>
    stack<T>::stack_element::stack_element(T&                    value,
stack::stack_element* previous) {
        this->value = value;
        this->previous = previous;
        this->next = nullptr;
    }

    template<typename T>
    stack<T>::stack_element::~stack_element() {
        free(this->next);
        free(this->previous);
    }

    template<typename T>
    T& stack<T>::stack_element::getValue() {
        return value;
    }

    template<typename T>
    void stack<T>::stack_element::setNext(stack::stack_element *next) {
        stack_element::next = next;
    }

    template <typename T>
    typename                                  stack<T>::stack_element
*stack<T>::stack_element::getNext() {
        return this->next;
    }

    template <typename T>
```

```
    typename                                    stack<T>::stack_element
*stack<T>::stack_element::getPrevious() {
        return this->previous;
    }



    template<typename T>
    const int* stack<T>::getStackSize() const {
        return &size;
    }

    template<typename T>
    stack<T>::stack() {
        this->size = 0;
        this->first = nullptr;
        this->last = nullptr;
    }

    template<typename T>
    stack<T>::~stack() {
        free(this->first);
        free(this->last);
    }

    template<typename T>
    void stack<T>::push(T element) {
        stack_element* SE = new stack_element(element, last);

        if (size == 0) {
            first = SE;
        } else {
            last->setNext(SE);
        }

        size++;

        last = SE;
    }

    template<typename T>
    T stack<T>::pop() {
        if (size == 0)
            throw std::runtime_error("Stack is empty!");

        stack_element *decapitation = last;
        T value = decapitation->getValue();
```

```
    last = decapitation->getPrevious();
    if (size > 1) {
        last->setNext(nullptr);
    } else {
        first = nullptr;
    }
    free(decapitation);

    size--;

    return value;
}

template<typename T>
bool stack<T>::isEmpty() {
    return size == 0;
}

#endif // STACK_H
```

Файл tree_node.h:

```
#ifndef CURR_TREE_NODE_H
#define CURR_TREE_NODE_H

#include "stack.h"
#include <cstdlib>
#include <random>


template<typename C>
class tree_node {
private:
    C comparable;
    double index;
    unsigned long weight = 1;

    tree_node* left = nullptr;
    tree_node* right = nullptr;

public:
    unsigned long get_weight();
    void reset_weight();

    unsigned long depth(tree_node<C>* node);

    tree_node* get_right() {
        return right;
```

```
        }

        tree_node* get_left() {
            return left;
        }

        C get_state() {
            return comparable;
        }

        double get_index() {
            return index;
        }

        tree_node<C>* trim();

        static bool is(tree_node<C>* node, C elem);

         static void split(tree_node<C>* root, C comp, tree_node<C>**
left, tree_node<C>** right);
         static tree_node<C>* merge( tree_node<C>* first, tree_node<C>*
second);

         static tree_node<C>* insert(tree_node<C>* first, tree_node<C>*
second);

        explicit tree_node(C base);
        tree_node();
    };


    template<typename C>
    unsigned long tree_node<C>::get_weight() {
        return weight;
    }

    template<typename C>
    void tree_node<C>::reset_weight() {
        unsigned long right_weight = 0;
        unsigned long left_weight = 0;
        if (right != nullptr) {
            right->reset_weight();
            right_weight = right->weight;
        }
        if (left != nullptr) {
            left->reset_weight();
            left_weight = left->weight;
        }
```

```
        weight = right_weight + left_weight + 1;
    }


    template<typename C>
    unsigned long tree_node<C>::depth(tree_node<C> *node) {
        if (node == nullptr) {
            return 0;
        } else {
            unsigned long left_depth = depth(node->left);
            unsigned long right_depth = depth(node->right);

            if (left_depth > right_depth) {
                return (left_depth + 1);
            } else {
                return (right_depth + 1);
            }
        }
    }


    template<typename C>
    tree_node<C>* tree_node<C>::trim() {
        auto node = new tree_node<C>();
        *node = *this;
        node->left = node->right = nullptr;

        return node;
    }


    template<typename C>
    bool tree_node<C>::is(tree_node<C>* node, C elem) {
        if (!node) return false;
        if (elem == node->get_state())
            return true;
        if (elem < node->get_state())
            return is(node->get_left(), elem);
        else
            return is(node->get_right(), elem);
    }


    template<typename C>
    void tree_node<C>::split(tree_node<C> *root, C comp, tree_node<C>
**left, tree_node<C> **right) {
        if (root == nullptr) {
            *left = nullptr;
            *right = nullptr;
```

```
            return;
        } else if (comp > root->get_state()) {
            split(root->right, comp, left, right);
            root->right = *left;

            *left = root;
            return;
        } else {
            split(root->left, comp, left, right);
            root->left = *right;

            *right = root;
            return;
        }
    }

    template<typename C>
    tree_node<C> *tree_node<C>::merge(tree_node<C> *first, tree_node<C>
*second) {
        if (second == nullptr) return first;
        if (first == nullptr) return second;

        if (first->index > second->index) {
            first->right = merge(first->right, second);
            return first;
        } else {
            second->left = merge(first, second->left);
            return second;
        }
    }


    template<typename C>
    tree_node<C>       *tree_node<C>::insert(tree_node<C>*       first,
tree_node<C>* second) {
        if (first == nullptr) {
            return second;
        }
        if (second == nullptr) {
            return first;
        }

        auto left_tree = new tree_node();
        auto right_tree = new tree_node();

        split(first, second->comparable, &left_tree, &right_tree);
        left_tree = merge(left_tree, second);
        return merge(left_tree, right_tree);
    }
```

```
template<typename C>
tree_node<C>::tree_node(C base) {
    this->comparable = base;
    this->left = this->right = nullptr;
    this->weight = 1;

    this->index = ((double) rand() / (RAND_MAX)) + 1;
}

template<typename C>
tree_node<C>::tree_node() {
    this->left = this->right = nullptr;
    this->weight = 1;
}


#endif //CURR_TREE_NODE_H
```

Файл lab5.pro:

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets quick

CONFIG += c++17

# 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 \
    myglwidget.cpp \
```

```
    lab5.cpp

HEADERS += \
    lab5.h \
    mainwindow.h \
    myglwidget.h \
    stack.h \
    tree_node.h \
    rand_tree.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
```