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: Сцепляемые очереди (упорядоченные линейные списки) на основе рандомизированных пирамид поиска

Студент гр. 8381 _____ Сергеев А.Д.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Сергеев А.Д.

Группа 8381

Тема работы: Сцепляемые очереди (упорядоченные линейные списки) на основе рандомизированных пирамид поиска.

Исходные данные: Необходимо написать программу для демонстрации структуры данных сцепляемые очереди на основе рандомизированных пирамид поиска и некоторых базовых действий с ней (сцепление и расщепление).

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

Предполагаемый объем пояснительной записки:
Не менее 60 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент		Сергеев А.Д.
Преподаватель		Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения курсовой работы была разработана программа с GUI, позволяющая демонстрировать устройство и принципы работы сцепляемых очередей на основе рандомизированных пирамид поиска. Программа обладает следующей функциональностью: построение сцепляемых очередей по входным данным из файла и из GUI по шагам и моментально, сцепление очередей по шагам и моментально, ращепление очередей по шагам и моментально; сохранение и загрузка состояния программы, откат действий в некоторых пределах; вывод информации о рандомизированных пирамидах поиска и о функциональности программы, вывод логов работы; демонстрация средней высоты дерева списка для данного набора узлов; генерация простых входных данных.

SUMMARY

During the course work, a program with a GUI was developed that allows you to demonstrate the device and the principles of operation of concatenated queues based on randomized search pyramids. The program has the following functionality: building concatenated queues by input from a file and from the GUI in steps and instantly, concatenating queues in steps and instantly, splitting queues in steps and instantly; saving and loading the program state, rollback actions within certain limits; displaying information about randomized search pyramids and program functionality, displaying logs of work; Demonstration of the average height of the list tree for a given set of nodes; Simple input generation.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Демонстрация основных функций сцепляемых очередей (упорядоченных линейных списков) на основе рандомизированных пирамид поиска.

Основные задачи

Построение сцепляемых очередей, сцепление и расщепление; выполнение данных действий как моментально, так и по шагам, возможность отката состояний списков.

Методы решения

Разработка программы велась на базе операционной системы Windows 10 в среде разработки QtCreator. Для создания графической оболочки использовался редактор интерфейса в QtCreator и система сигналов-слотов Qt. Для реализации сцепляемых очередей были созданы классы treap и treap_node. Для контроля выполнения, пошагового выполнения и отката операций был создан класс core. Для графического отображения результатов были созданы классы MyGLWidget и outlog. Для случайной генерации входных данных был создан класс generator.

1. ЗАДАНИЕ

Необходимо провести демонстрацию работы сцепляемых очередей и некоторых операций с ними.

Демонстрация должна содержать:

1. Построение сцепляемых очередей, из файла и из строки, моментально и по шагам.
2. Сцепление сцепляемых очередей, моментально и по шагам.
3. Расщепление сцепляемых очередей, моментально и по шагам.
4. Запись истории работы программы в лог.
5. Откат произведённых действий.
6. Сохранение и загрузку состояния программы в файл.
7. Подробное описание всех действий с пояснениями.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание интерфейса пользователя

Интерфейс программы разделен на четыре части: панель шагов и информации, панель ввода данных для создания сцепляемых очередей, панель вывода данных для демонстрации текущего состояния сцепляемых очередей и панель действий. Основные виджеты панели шагов и информации и их назначение представлены в табл. 1.

Таблица 1 – Основные виджеты панели шагов и информации

Класс объекта	Название виджета	Назначение
QPushButton	about_treap_button	Кнопка вывода краткой информации о сцепляемых очередях
QPushButton	about_app_button	Кнопка вывода краткой информации о функциях программы
QPushButton	log_button	Кнопка вывода лога на экран
QPushButton	save_button	Кнопка загрузки состояния программы из внешнего файла
QPushButton	load_button	Кнопка загрузки состояния программы из внешнего файла
QPushButton	reset_button	Кнопка сброса стека состояний программы и очистки выходных файлов
QPushButton	back_button	Кнопка шага назад
QPushButton	step_button	Кнопка шага вперёд

Основные виджеты панели ввода данных для создания сцепляемых очередей и их назначение представлены в табл. 2.

Таблица 2 – Основные виджеты панели ввода данных для создания сцепляемых очередей

Класс объекта	Название виджета	Назначение
QPushButton	generator_button	Кнопка автоматической генерации простых входных данных в строку line_input
QLineEdit	line_input	Поле ввода входных данных для генерации сцепляемых очередей или пути к файлу, содержащему входные данные
QPushButton	line_button	Кнопка генерации сцепляемых очередей по входным данным из поля line_input
QPushButton	file_button	Кнопка генерации сцепляемых очередей по входным данным, полученным из файла, на который указывает путь в поле line_input
QCheckBox	step_box	Флаг включения и отключения функции исполнения по шагам
QComboBox	type_selector	Селектор типа входных данных, для которых будут построены сцепляемые очереди

Основные виджеты панели вывода данных для демонстрации текущего состояния сцепляемых очередей и их назначение представлены в табл. 3.

Таблица 3 – Основные виджеты панели вывода данных для демонстрации текущего состояния сцепляемых очередей

Класс объекта	Название виджета	Назначение
MyGLWidget	canvas1	Холст для демонстрации состояния первой сцепляемой очереди
MyGLWidget	canvas2	Холст для демонстрации состояния второй сцепляемой очереди

Основные виджеты панели действий и их назначение представлены в табл. 4.

Таблица 4 - Основные виджеты панели действий

Класс объекта	Название виджета	Назначение
QPushButton	merge_button	Кнопка сцепления сцепляемых очередей
QSpinBox	split_box_pos	Селектор позиции расщепления первой сцепляемой очереди
QPushButton	split_button	Кнопка расщепления первой сцепляемой очереди
QPushButton	multiple_button	Кнопка демонстрации средней высоты дерева сцепляемой очереди

Также к интерфейсу программы относятся окно вывода лога и два диалоговых окна, описывающие структуру сцепляемых очередей и функции программы.

2.2. Описание основных классов для сцепляемых очередей

Для реализации сцепляемых очередей были созданы шаблонные классы treap_node и treap. Класс treap_node содержит шаблонное поле comparable для хранения элементов любого типа, поле для хранения числа с плавающей точкой — ключей двоичной кучи и поле для хранения веса узла. Также класс содержит указатели на двух потомков узла и методы работы со сцепляемым списком. Основные методы класса treap_node представлены в табл. 5.

Таблица 5 – Основные методы класса Pair

Метод	Назначение
unsigned long get_weight();	Возвращает вес узла — количество его потомков + 1
void reset_weight();	Пересчитывает вес для всех потомков узла

Окончание таблицы 5

static unsigned long depth(treap_node<C>* node);	Рассчитывает максимальную глубину поддерева, корнем которого является узел node
treap_node* get_right();	Возвращает указатель на правого потомка
treap_node* get_left();	Возвращает указатель на левого потомка
C get_state();	Возвращает элемент очереди, хранящийся в этом узле
double get_index();	Возвращает ключ этого узла
std::string* to_string(int pos, int* curr);	Возвращает строковую запись этого узла и его потомков в порядке очереди (обход ЛЦП) начиная с номера pos
treap_node<C>* trim();	Возвращает копию этого узла без потомков
static void split(treap_node<C>* root, int x, treap_node<C>** left, treap_node<C>** right);	Расщепляет поддерево этого узла по элементу, находящемуся на месте x при обходе ЛЦП, записывая левое поддерево при расщеплении в узел left, а правое в узел right
static treap_node<C>* merge(treap_node<C>* first, treap_node<C>* second);	Сцепляет поддеревья узлов first и second в узел first
static treap_node<C>* insert(treap_node<C>* first, treap_node<C>* second);	Вставляет узел second в поддерево узла first в конец очереди
static treap_node<C>* remove(treap_node<C>* root, int x);	Удаляет узел под номером x из поддерева узла root

Класс treap является обёрткой для класса treap_node. Он содержит указатель на корень дерева сцепляемой очереди и позволяет обращаться к его методам в удобном для пользователя виде и выполнять необходимые проверки. Основные методы класса приведены в табл. 6.

Таблица 6 – Основные методы класса treap

Метод	Назначение
unsigned long max_depth();	Возвращает максимальную высоту дерева очереди
unsigned long theory_depth();	Рассчитывает и возвращает идеальную высоту дерева очереди исходя из количества узлов
unsigned long get_weight();	Возвращает количество элементов очереди
treap_node<C>* get_root();	Возвращает корень дерева очереди
void add(C value);	Добавляет элемент со значением C в конец очереди
void add(treap_node<C>* value);	Добавляет элемент value в конец очереди
void remove(int index);	Удаляет элемент под номером index из очереди
void merge(treap<C>* another);	Сцепляет эту очередь с очередью another
bool split(int pos, treap<C>* stat, treap<C>* another);	Расщепляет эту очередь по позиции x, записывая правую часть в очередь stat, а левую в очередь another; возвращает true, если это удалось сделать и false в обратном случае
std::string* to_string(int pos);	Возвращает строковую запись элементов очереди, начиная с номера pos

2.3. Описание алгоритма сцепления очередей

На вход подаются две очереди, требуется объединить их в одну. Исходя из того, что каждая очередь является кучей по приоритетам (index), необходимо вычислить, какой из корней двух деревьев очередей будет являться корнем объединённой очереди. Если это корень дерева первой очереди, то дерево второй очереди становится его правым поддеревом и далее рекурсивно выполняется функция слияния его правого и левого поддеревьев. Если это корень дерева второй очереди, то дерево первой очереди становится его левом

поддеревом и далее рекурсивно выполняется функция слияния его левого и правого поддеревьев.

2.3. Описание алгоритма расщепления очередей

На вход подаётся очередь и индекс, требуется разделить её на две так, чтобы индекс элементов лежал в первой новой очереди, а все остальные — во второй новой очереди. Ключей, как в декартовом дереве, в сцепляемой очереди нет, их место займёт количество элементов в очереди. Необходимо вычислить, в какую из новых очередей попадёт корень дерева исходной очереди. Если вес его левого поддерева + 1 меньше или равен индексу, то корень будет в первой новой очереди, и дальше необходимо рекурсивно разрезать правое поддерево корня дерева исходной очереди, но с индексом, равным индекс — вес левого поддерева корня дерева исходной очереди — 1, так как корень и левое поддерево уже попали в первую новую очередь. Если вес левого поддерева корня дерева исходной очереди + 1 больше индекса, то корень будет во второй новой очереди, и дальше необходимо рекурсивно разрезать правое поддерево корня дерева исходной очереди с исходным индексом.

2.4. Описание класса core и демонстрации функций очередей

Для функций, отвечающих за демонстрацию вставки элементов в очередь, сцепления и расщепления очередей, был создан класс core. Основные поля и их краткие описанияя приведены в табл. 7.

Таблица 7 – Основные поля класса core

Тип поля	Название поля	Описание поля
int	step_action	Поле, хранящее состояние программы на данный момент: 0 — программа готова к считыванию данных 1 — программа готова к обработке данных

Продолжение таблицы 7

		10 — программа готова к пошаговому считыванию очередей 11 — программа считывает первую очередь по шагам 12 — программа считывает вторую очередь по шагам 20 — программа готова к пошаговому сцеплению очередей 21 — программа сцепляет очереди по шагам 30 — программа готова к пошаговому расщеплению очередей 31 — программа расщепляет очереди по шагам
int	type	Поле, хранящее тип данных, хранящихся в очередях: 0 — int 1 — char 2 — double 3 — std::string
int	split_maximum	Поле, хранящее актуальное количество элементов в первом дереве, это количество — максимальное для расщепления
treap<T>*	tree1	Указатель на первую очередь
treap<T>*	tree2	Указатель на вторую очередь
outlog*	log	Указатель на объект класса outlog, производящего сбор записей в лог
std::string	input_file	Путь к файлу по умолчанию с входными данными
std::string	output_file1	Путь к файлу, изображающему первую

		10 — программа готова к пошаговому считыванию очередей 11 — программа считывает первую очередь по шагам 12 — программа считывает вторую очередь по шагам 20 — программа готова к пошаговому сцеплению очередей 21 — программа сцепляет очереди по шагам 30 — программа готова к пошаговому расщеплению очередей 31 — программа расщепляет очереди по шагам
		очередь
std::string	output_file2	Путь к файлу, изображающему вторую очередь

Окончание таблицы 7

std::string	stack_file	Путь к файлу стека состояний программы по умолчанию
int	stack_size	Размер доступного стека состояний
std::istream*	in_stream	Указатель на поток входных данных

По умолчанию целочисленное возвращаемое значение всех методов класса, если оно есть, имеет следующее значение:

1 — метод завершился с ошибкой.

0 — метод завершился успешно.

-1 — метод завершился, но пошаговый процесс не завершён.

Класс имеет методы, демонстрирующие функции работы с очередями мгновенно и по шагам. Методы, выполняющиеся по шагам, не полностью повторяют принципы работы моментально выполняющихся методов. Это

связано с рекурсивной природой функций сцепления и расщепления сцепляемых очередей. Все эти методы класса core приведены в табл. 8.

Таблица 8 – Обрабатывающие очереди методы класса core

Метод	Назначение
static int build_step(MyGLWidget* mglw1, MyGLWidget* mglw2);	Метод, строящий по шагам обе или одну очереди, обрабатывая данные из входного потока in_stream и добавляя новые считанные узлы в очереди, результат для первой очереди выводятся на холст mglw1, для второй — на mglw2
static int build_rush(MyGLWidget* mglw1, MyGLWidget* mglw2, int mode, bool from_val);	Метод, моментально строящий обе или одну очереди, обрабатывая данные из входного потока in_stream и добавляя новые считанные узлы в очереди, результат для первой очереди выводятся на холст mglw1, для второй — на mglw2 Флаг from_val передаётся в том случае, если данные во входном потоке представляют из себя строковые представления узлов очереди

Окончание таблицы 8

	Значение mode означает: 1 — строятся и отображаются обе очереди 2 — строится только первая очередь, отображаются обе 3 — строятся обе очереди, ни одна из них не обображается
static int merge_step(MyGLWidget* mglw1, MyGLWidget* mglw2);	Метод, сцепляющий по шагам очереди, в первом прохождении записывает во входной поток строковое отображение второй очереди, в последующие добавляющий к первой очереди узлы из входного потока
static int merge_rush(MyGLWidget* mglw1, MyGLWidget*	Метод, моментально сцепляющий очереди

	Значение mode означает: 1 — строятся и отображаются обе очереди 2 — строится только первая очередь, отображаются обе 3 — строятся обе очереди, ни одна из них не обображается
mglw2);	
static int split_step(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2);	Метод, расщепляющий по шагам очереди, в первом прохождении записывает во входной поток строковое отображение первой очереди, начиная с позиции pos, в последующие добавляющий ко второй очереди узлы из входного потока, в последнем удаляющий узлы, начиная с позиции pos, из первой очереди
static int split_rush(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2);	Метод, моментально расщепляющий очереди
static int mult(MyGLWidget* mglw);	Метод, демонстрирующий среднюю высоту дерева первой очереди, основываясь на её построении 100 раз, в сравнении с идеальной высотой её дерева

Класс также имеет методы чтения входных данных из строки, файла или стека состояний и записи состояния программы в стек состояний. Эти методы класса core приведены в табл. 9.

Таблица 9 – Считывающие и записывающие данные методы класса core

Метод	Назначение
static int launch(MyGLWidget* mglw, std::string* input, bool ff, int tp);	Метод, считывающий данные во входной поток Флаг ff передаётся в том случае, если считать необходимо из файла, заданного строкой input Значение tp обозначает тип входных данных (см. описание поля type) На холст mglw выводится сообщение об

	ошибке или неудаче
static int push(QPushButton* load_button, std::string* filename);	Записывает текущее состояние программы в стек состояний, в файл, на которой указывает путь filename Состояние включает в себя информацию о текущем типе данных, о построенных очередях, о входном потоке, а также о состоянии программы (см. описание поля step_action) После записи увеличивает размер стека и, если он больше нуля, включает кнопку load_button
static int pop(MyGLWidget* mglw1, MyGLWidget* mglw2, QPushButton* load_button, std::string* filename);	Считывает состояние программы из стека состояний, из файла, на которой указывает путь filename (см. описание метода push) После чтения уменьшает размер стека и, если он меньше нуля, отключает кнопку load_button
static void clear_stack(QPushButton* back_button);	Очищает стек состояний программы по умолчанию
static T get_value(std::string* str);	Считывает и возвращает значение узла из строки str, используя преобразование к нужному типу данных при помощи std::stringstream
static treap_node<T>* get_node(std::string* str);	Считывает и возвращает узел из строки str, используя преобразование к нужному типу данных при помощи std::stringstream
static std::string* read_tree(std::istream* stream, int* delim_num);	Считывает и возвращает очередное значение из входного потока istream, до первого разделителя (разделители также хранятся в классе core), и записывает номер разделителя в delim_num

3. ТЕСТИРОВАНИЕ

3.1. Вид программы

Программа представляет собой окно с графическим интерфейсом шириной 1000 пикселей и высотой 600 пикселей. Вид программы после запуска представлен на рис. 1.

Рисунок 1 – Вид программы после запуска

После создания очередей на холсты помещается их изображение, а в строку под ними — сообщения об ошибках. Вид программы после создания очередей представлен на рис. 2.

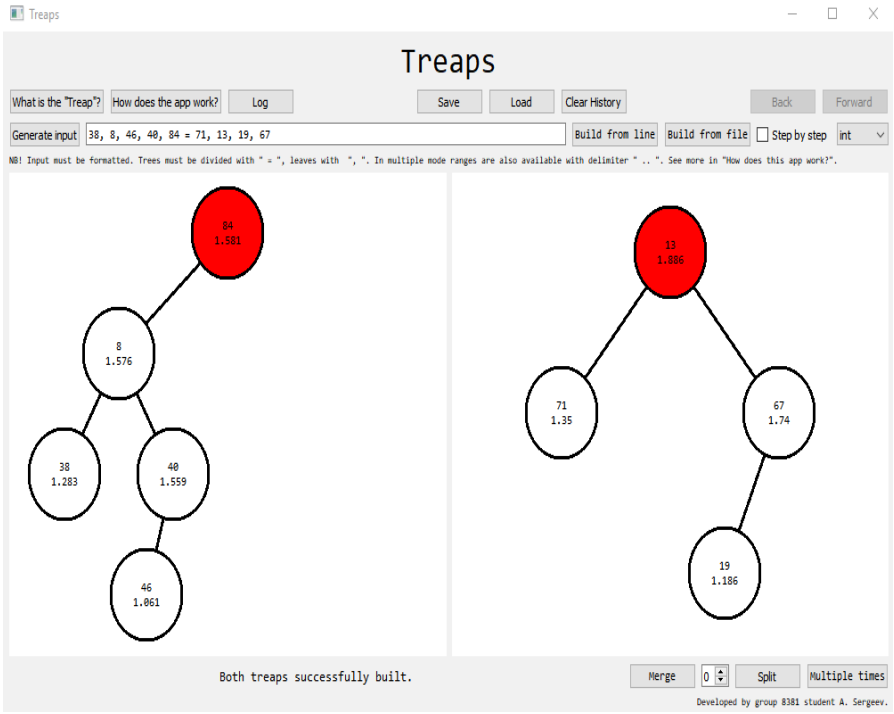

Рисунок 2 – Вид программы после создания очередей

3.2. Тестирование сцепления очередей

Было протестировано сцепление очередей для трёх различных случаев. Результаты представлены на рисунках с 3 по 5.

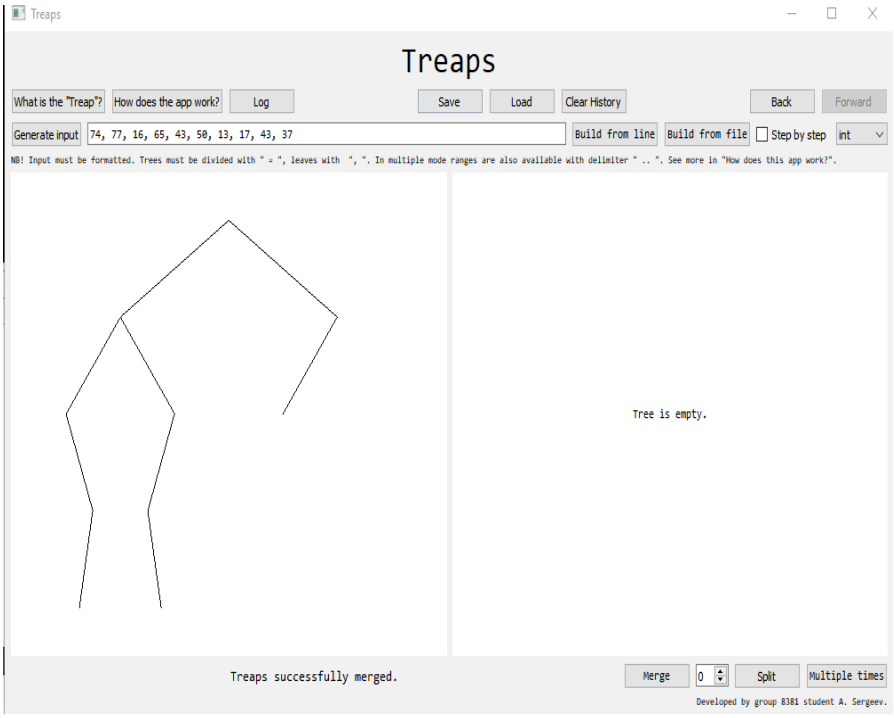

Рисунок 3 — Вторая очередь пуста

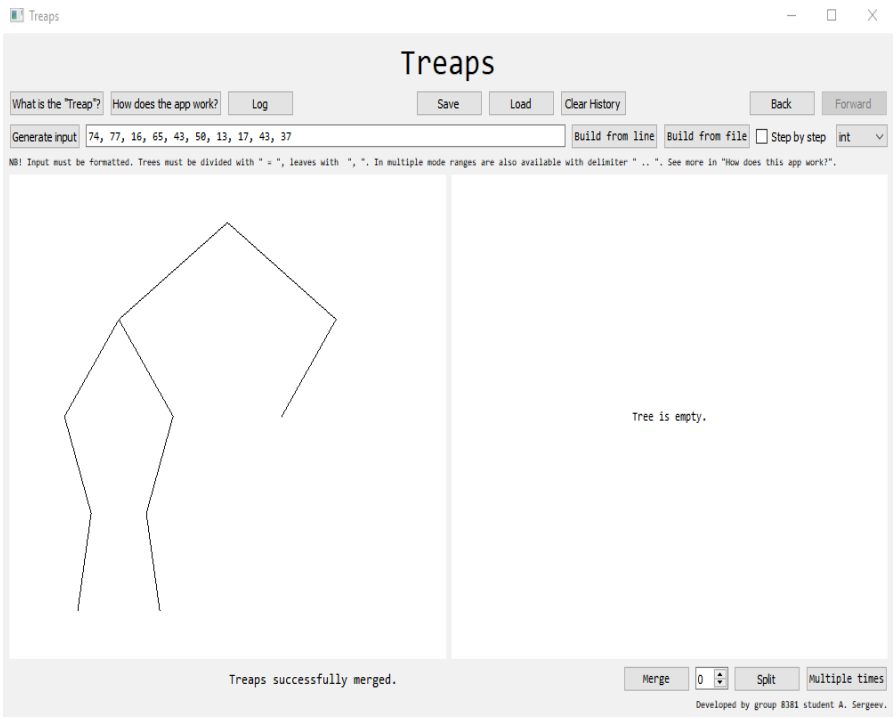

Рисунок 4 — Первая очередь пуста

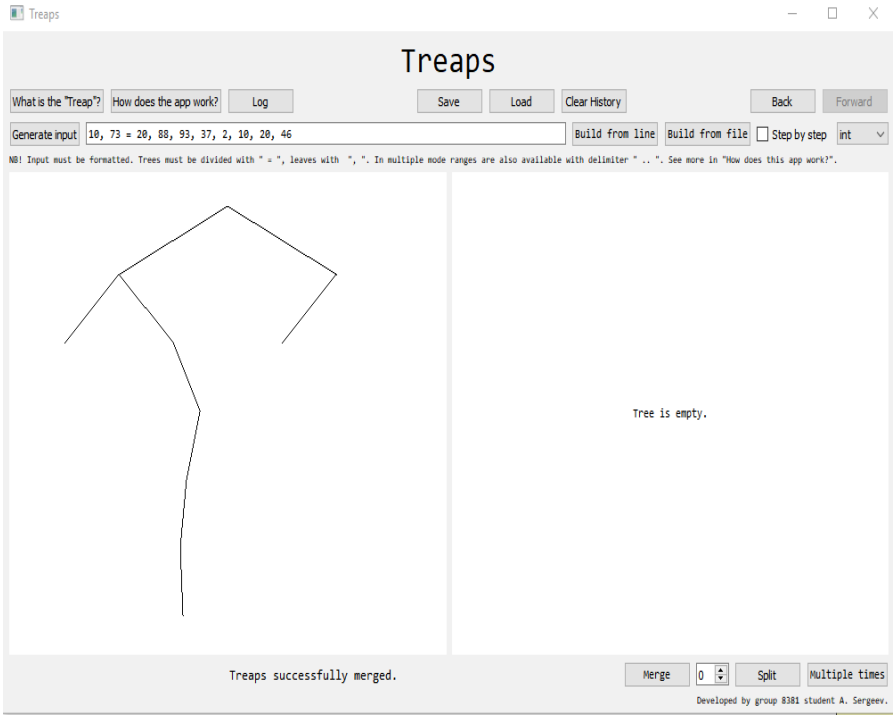

Рисунок 5 — Обе очереди не пусты

3.3. Тестирование расщепления очередей

Было протестировано расщепление очередей для трёх различных случаев. Результаты представлены на рисунках с 6 по 8.

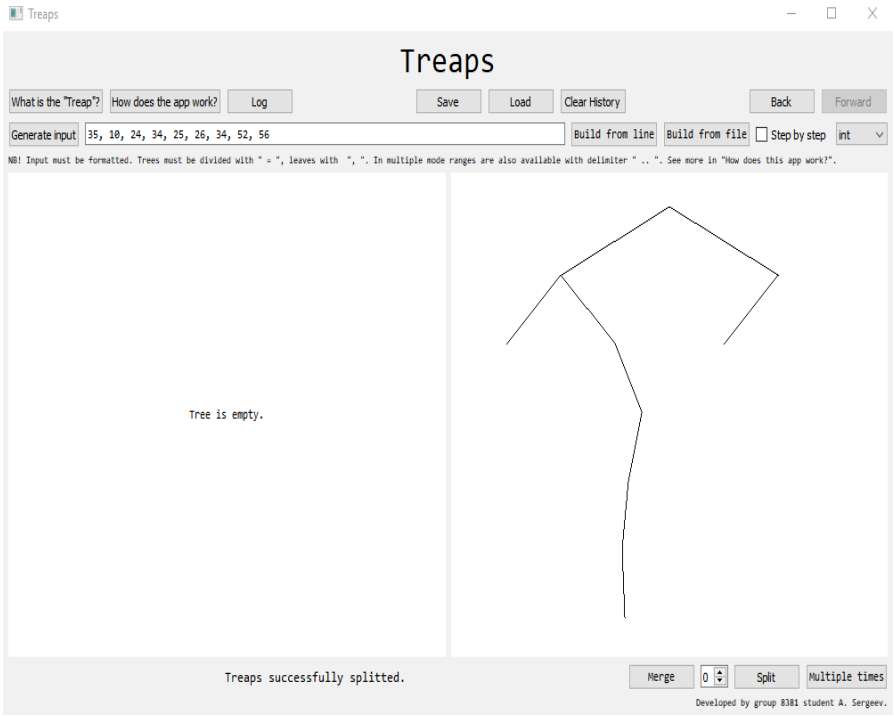

Рисунок 6 — Вторая очередь пуста

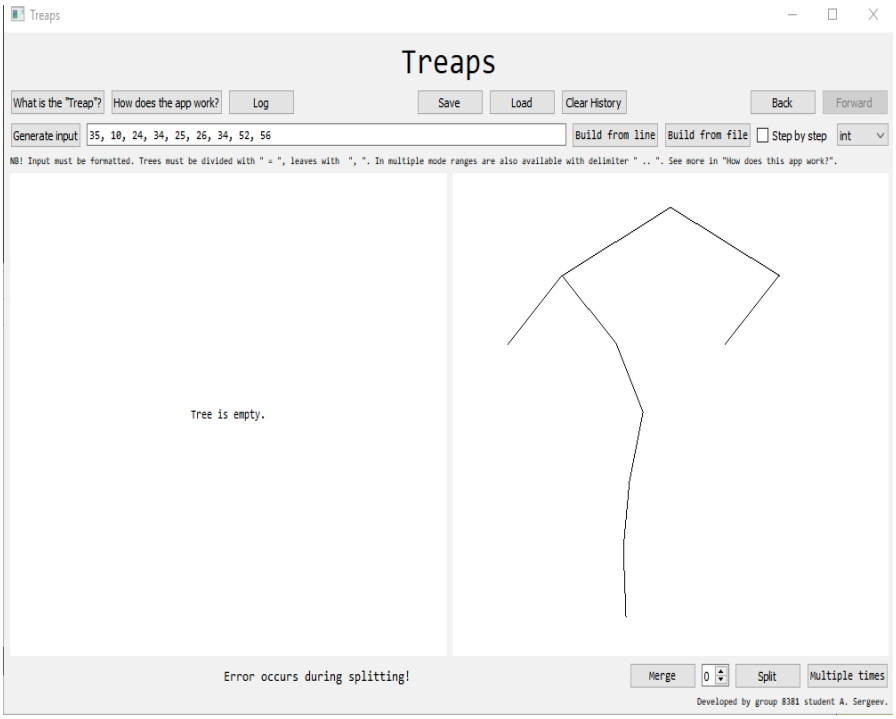

Рисунок 7 — Первая очередь пуста (ошибка возникла т. к. расщепляемая очередь пуста)

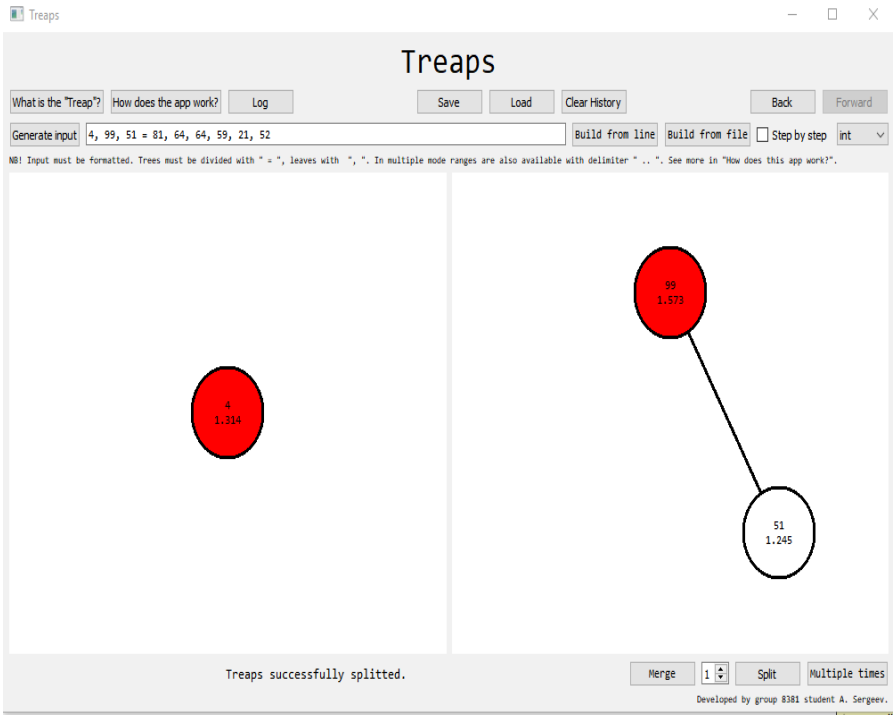

Рисунок 8 — Обе очереди не пусты

ЗАКЛЮЧЕНИЕ

построение сцепляемых очередей по входным данным из файла и из GUI по шагам и моментально, сцепление очередей по шагам и моментально, ращепление очередей по шагам и моментально; сохранение и загрузка состояния программы, откат действий в некоторых пределах; вывод информации о рандомизированных пирамидах поиска и о функциональности программы, вывод логов работы; демонстрация средней высоты дерева списка для данного набора узлов; генерация простых входных данных. С помощью программы была проведена демонстрация работы основных функций сцепляемых очередей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Описание сцепляемых очередей // habr.com/en/post/101818/, обращение 19.12.2019
2. Описание декартовых деревьев „ https://neerc.ifmo.ru/wiki/index.php?title=%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE, обращение 19.12.2019
3. Перевод и дополнение документации QT // CrossPlatform.RU. URL: http://doc.crossplatform.ru/
4. Qt Documentation // Qt. URL: https://doc.qt.io/qt-5/index.html

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

```
#include <QApplication>
#include <QScreen>
#include <sstream>
#include <iostream>
#include <QDir>
#include "outlog.h"
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);

    MainWindow w;
    QRect screen_geometry = QGuiApplication::screens().first()->geometry();
    int x = (screen_geometry.width() - w.geometry().width() - outlog::width) /
2;
    int y = (screen_geometry.height() - w.geometry().height()) / 2;
    w.move(x, y);
    w.setWindowTitle("Treaps");
    w.show();
    return a.exec();
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. TREAP.H

```
#ifndef TREAP_H
#define TREAP_H

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


template<typename C>
class treap {
private:
    treap_node<C>* root = nullptr;

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

    treap_node<C>* get_root();

    void add(C value);
    void add(treap_node<C>* value);
    void remove(int index);

    void merge(treap<C>* another);
    bool split(int pos, treap<C>* stat, treap<C>* another);

    std::string* to_string(int pos);

    explicit treap();
};


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 treap<C>::max_depth() {
    return treap_node<C>::depth(root);
}

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


template<typename C>
treap_node<C>* treap<C>::get_root() {
    return root;
}


template<typename C>
void treap<C>::add(C value) {
    treap_node<C>* node = new treap_node<C>(value);
    return this->add(node);
}

template<typename C>
void treap<C>::add(treap_node<C>* node) {
    if (root == nullptr) {
        root = node;
    } else {
        root = treap_node<C>::insert(root, node);
        if (root != nullptr) root->reset_weight();
    }
}

template<typename C>
```

```
void treap<C>::remove(int index) {
    root = treap_node<C>::remove(root, index);

    if (root != nullptr) root->reset_weight();
}


template<typename C>
void treap<C>::merge(treap<C>* another) {
    if (root == nullptr) {
        root = another->root;
        return;
    }
    if ((another == nullptr) || (another->root == nullptr)) return;

    root = treap_node<C>::merge(root, another->get_root());

    if (root != nullptr) root->reset_weight();
}

template<typename C>
bool treap<C>::split(int pos, treap<C>* stat, treap<C>* another) {
    if ((root == nullptr) || (static_cast<unsigned>(pos) > root->get_weight()))
return false;

    treap_node<C>* first_root = new treap_node<C>;
    treap_node<C>* second_root = new treap_node<C>;

    treap_node<C>::split(root, pos, &first_root, &second_root);
    if (first_root != nullptr) first_root->reset_weight();
    if (second_root != nullptr) second_root->reset_weight();

    stat->root = first_root;
    another->root = second_root;

    return true;
}


template<typename C>
std::string* treap<C>::to_string(int pos) {
    std::string* res = new std::string();
    if (root == nullptr) {
        res->append("(0;0.0)");
        return res;
    }
    int dest = 0;
    res->append(*(root->to_string(pos, &dest)));
```

```
    *res = res->substr(0, res->length() - 2);

    return res;
}


template<typename C>
treap<C>::treap() {
    generator::ground();

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


#endif //TREAP_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. CORE.H

```
#ifndef CORE_H
#define CORE_H

#include <fstream>
#include <sstream>
#include <QPushButton>
#include "treap.h"
#include "myglwidget.h"
#include "outlog.h"


class core {
public:
    static const std::string PATH;

      // functions return: 0 - successful; 1 - error; -1 - forward execution required

    // 0 - nothing done, 1 - trees built
     // 10 - prepare to build trees step by step, 11 - first tree is buildind step by step, 12 - second tree is building step by step,
     // 20 - prepare to merge trees step by step, 21 - trees are merging step by step,
    // 30 - prepare to split trees step by step, 31 - trees are splitting step by step
    static int step_action;


     static int launch(MyGLWidget* mglw, std::string* input, bool ff, int tp);


    template<typename T>
    static int build_step(MyGLWidget* mglw1, MyGLWidget* mglw2);
```

```
        // mode 1 - building 2 trees showing both, mode 2 - building 1 tree showing both, mode 3 - building one tree showing none (mdlw1 -> mglw2)
        template<typename T>
        static int build_rush(MyGLWidget* mglw1, MyGLWidget* mglw2, int mode, bool from_val);


        template<typename T>
        static int merge_step(MyGLWidget* mglw1, MyGLWidget* mglw2);

        template<typename T>
        static int merge_rush(MyGLWidget* mglw1, MyGLWidget* mglw2);


        template<typename T>
        static int split_step(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2);

        template<typename T>
        static int split_rush(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2);


        template<typename T>
        static int mult(MyGLWidget* mglw);


        static int push(QPushButton* load_button, std::string* filename);

          static int pop(MyGLWidget* mglw1, MyGLWidget* mglw2, QPushButton* load_button, std::string* filename);

        static void clear_stack(QPushButton* back_button);


        static int type;

        static int split_maximum;
```

```
    template<typename T>
    static treap<T>* tree1;

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

    static outlog* log;

private:
    static const std::string input_file;
    static const std::string output_file1;
    static const std::string output_file2;

    static const std::string stack_file;

    static int stack_size;


    static std::istream* in_stream;


    template<typename T>
    static T get_value(std::string* str);
    template<typename T>
    static treap_node<T>* get_node(std::string* str);


    static const std::string subitem_delimiter;
    static const std::string item_delimiter;  // number 2
    static const std::string range_delimiter; // number 4
    static const std::string tree_delimiter; // number 3
    static const std::string end_delimiter;  // number 1

    static std::string* read_tree(std::istream* stream, int* delim_num);
};
```

```
template<typename T>
treap<T>* core::tree1 = nullptr;

template<typename T>
treap<T>* core::tree2 = nullptr;


template<typename T>
int core::build_rush(MyGLWidget* mglw1, MyGLWidget* mglw2, int mode, bool
from_val) {
            log->notify(new std::string("Quick building treap in mode " +
std::to_string(mode) + " from value " + std::to_string(from_val) + "..."));

          if (mode < 3) {
              mglw1->prepare_drawing();
              mglw2->prepare_drawing();
          }

          if (mode != 2) {
              free(tree1<T>);
              tree1<T> = new treap<T>();
          }
          if (mode < 2) {
              free(tree2<T>);
              tree2<T> = new treap<T>();
          }

          int prev_delim = 0;
          T prev;
          T processing;

          std::string* msg;

          int errno1 = -1, errno2 = -1;
          try {
              int delim_pos = 0;
```

```
            do {
                if (from_val) {
                                tree1<T>->add(get_node<T>(read_tree(in_stream,
&delim_pos)));
                } else {
                              processing = get_value<T>(read_tree(in_stream,
&delim_pos));
                    tree1<T>->add(processing);
                }

                if (mode == 3) {
                                if ((prev_delim == 4) && (!(std::is_same<T,
std::string>::value))) {
                        log->notify(new std::string("Processing range..."));
                        for (T i = prev; i < processing; i += 1) {
                            tree1<T>->add(i);
                        }
                    }

                    prev_delim = delim_pos;
                    prev = processing;
                }
              } while ((delim_pos != 1) && (delim_pos != 3) && (delim_pos !=
0));
            if ((delim_pos == 0) && (mode < 2)) errno2 = 0;

        } catch (std::runtime_error re) {
            msg = new std::string(re.what());
            log->notify(msg);
            if (mode < 3) mglw1->declare(msg);
            return 1;
        } catch (int e) {
            if (mode == 2) errno2 = e;
            else errno1 = e;
        }

        if ((mode < 2) && (errno2 == -1)) {
```

```
            try {
                int delim_pos = 0;
                do {
                    if (from_val) {
                                tree2<T>->add(get_node<T>(read_tree(in_stream,
&delim_pos)));
                    } else {
                               tree2<T>->add(get_value<T>(read_tree(in_stream,
&delim_pos)));
                    }
                } while ((delim_pos != 1) && (delim_pos != 0));

            } catch (std::runtime_error re) {
                msg = new std::string(re.what());
                log->notify(msg);
                mglw2->declare(msg);
                return 1;
            } catch (int e) {
                errno2 = e;
            }
        }

        if (mode < 3) {
            if (errno1 == -1) {
                mglw1->set_tree(tree1<T>, tree1<T>->get_root()->get_index());
                mglw1->show(&output_file1);
            } else {
                mglw1->declare(new std::string("Tree is empty."));
            }
            if (errno2 == -1) {
                mglw2->set_tree(tree2<T>, tree2<T>->get_root()->get_index());
                mglw2->show(&output_file2);
            } else {
                mglw2->declare(new std::string("Tree is empty."));
            }
        }
```

```
            if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
          else  split_maximum = 0;

          return 0;
      }

      template<typename T>
      int core::merge_rush(MyGLWidget* mglw1, MyGLWidget* mglw2) {
          log->notify(new std::string("Quick merging treap..."));

          mglw1->prepare_drawing();
          mglw2->prepare_drawing();

          tree1<T>->merge(tree2<T>);
          tree2<T> = new treap<T>();

          if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr)) {
              mglw1->set_tree(tree1<T>, tree1<T>->get_root()->get_index());
              mglw1->show(&output_file1);
          } else {
              mglw1->declare(new std::string("Tree is empty."));
          }

          if ((tree2<T> != nullptr) && (tree2<T>->get_root() != nullptr)) {
              mglw2->set_tree(tree2<T>, tree2<T>->get_root()->get_index());
              mglw2->show(&output_file2);
          } else {
              mglw2->declare(new std::string("Tree is empty."));
          }

            if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
          else  split_maximum = 0;

          return 0;
      }
```

```
template<typename T>
int core::split_rush(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2) {
    log->notify(new std::string("Quick splitting treap..."));

    mglw1->prepare_drawing();
    mglw2->prepare_drawing();

    bool is = tree1<T>->split(pos, tree1<T>, tree2<T>);


    if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr)) {
        mglw1->set_tree(tree1<T>, tree1<T>->get_root()->get_index());
        mglw1->show(&output_file1);
    } else {
        mglw1->declare(new std::string("Tree is empty."));
    }

    if ((tree2<T> != nullptr) && (tree2<T>->get_root() != nullptr)) {
        mglw2->set_tree(tree2<T>, tree2<T>->get_root()->get_index());
        mglw2->show(&output_file2);
    } else {
        mglw2->declare(new std::string("Tree is empty."));
    }

      if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
    else  split_maximum = 0;

    if (!is) {
        return 1;
    } else return 0;
}


template<typename T>
int core::build_step(MyGLWidget* mglw1, MyGLWidget* mglw2) {
```

```
        log->notify(new std::string("Building treap step-by-step..."));

        mglw1->prepare_drawing();
        mglw2->prepare_drawing();

        int result = -1;
        std::string* msg;

        if (step_action == 10) {
            tree1<T> = new treap<T>();
            tree2<T> = new treap<T>();

            msg = new std::string("Tree is empty.");
            mglw1->declare(msg);
            mglw2->declare(msg);

            step_action = 11;
            return result;
        }

        treap_node<T>* processing;

        if (step_action == 11) {
            try {
                int delim_pos = 0;

                                                        processing    =    new
treap_node<T>(get_value<T>(read_tree(in_stream, &delim_pos)));
                tree1<T>->add(processing);

                if ((delim_pos == 0) || (delim_pos == 1)) {
                    step_action = 1;
                    mglw1->set_tree(tree1<T>, processing->get_index());
                    mglw1->show(&output_file1);
                    mglw2->declare(new std::string("Tree is empty."));
                    return 0;
                } else if (delim_pos == 3) step_action = 12;
```

```
            } catch (std::runtime_error re) {
                msg = new std::string(re.what());
                log->notify(msg);
                mglw1->declare(msg);
                return 1;
            }

            mglw1->set_tree(tree1<T>, processing->get_index());
            mglw1->show(&output_file1);
            msg = new std::string("Tree is empty.");
            mglw2->declare(msg);

        } else if (step_action == 12) {
            try {
                int delim_pos = 0;

                                                    processing    =    new
treap_node<T>(get_value<T>(read_tree(in_stream, &delim_pos)));
                tree2<T>->add(processing);

                if ((delim_pos == 0) || (delim_pos == 1)) {
                    step_action = 1;
                    result = 0;
                }

            } catch (std::runtime_error re) {
                msg = new std::string(re.what());
                log->notify(msg);
                mglw1->declare(msg);
                return 1;
            }

            mglw1->set_tree(tree1<T>, tree1<T>->get_root()->get_index());
            mglw1->show(&output_file1);
            mglw2->set_tree(tree2<T>, processing->get_index());
            mglw2->show(&output_file2);
```

```
        }

          if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
        else  split_maximum = 0;

        return result;
    }

    template<typename T>
    int core::merge_step(MyGLWidget* mglw1, MyGLWidget* mglw2) {
        log->notify(new std::string("Merging treap step-by-step..."));

        mglw1->prepare_drawing();
        mglw2->prepare_drawing();

        int result = -1;

        if (step_action == 20) {
            std::string tree(*(tree2<T>->to_string(0)));
            std::stringstream* ss = new std::stringstream(tree);
            in_stream = ss;

            step_action = 21;
            return result;
        }

        treap_node<T>* processing;

        if (step_action == 21) {
            try {
                int delim_pos = 0;

                processing = get_node<T>(read_tree(in_stream, &delim_pos));
                tree1<T>->add(processing);

                if ((delim_pos == 0) || (delim_pos == 1)) {
```

```
                    tree2<T> = new treap<T>;
                    step_action = 1;
                    result = 0;
                }

            } catch (std::runtime_error re) {
                auto msg = new std::string(re.what());
                log->notify(msg);
                mglw1->declare(msg);
                return 1;
            } catch (...) {
                step_action = 1;
                tree2<T> = new treap<T>;
                result = 0;
            }

        }

        if (step_action == 1) {
            mglw2->declare(new std::string("Tree is empty."));
            if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr)) {
                mglw1->set_tree(tree1<T>, tree1<T>->get_root()->get_index());
                mglw1->show(&output_file1);
            } else {
                mglw1->declare(new std::string("Tree is empty."));
            }
        } else {
            mglw2->set_tree(tree2<T>, processing->get_index());
            mglw2->show(&output_file2);
            mglw1->set_tree(tree1<T>, processing->get_index());
            mglw1->show(&output_file1);
        }

          if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
        else  split_maximum = 0;
```

```
    return result;
}

template<typename T>
int core::split_step(int pos, MyGLWidget* mglw1, MyGLWidget* mglw2) {
    log->notify(new std::string("Splitting treap step-by-step..."));

    mglw1->prepare_drawing();
    mglw2->prepare_drawing();

    int result = -1;

    if (step_action == 30) {
        if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr)) {
            tree2<T> = new treap<T>;

            std::string tree(*(tree1<T>->to_string(pos)));
            std::stringstream* ss = new std::stringstream(tree);
            in_stream = ss;

            step_action = 31;
        } else {
            step_action = 1;
            result = 1;
        }
        return result;
    }

    treap_node<T>* processing;

    if (step_action == 31) {
        try {
            int delim_pos = 0;

            processing = get_node<T>(read_tree(in_stream, &delim_pos));
            tree2<T>->add(processing);
```

```
                if ((delim_pos == 0) || (delim_pos == 1)) {
                    long long init_size = tree1<T>->get_weight();
                     for (long long i = init_size; i > (init_size - tree2<T>-
>get_weight()); i--) {
                        tree1<T>->remove(i);
                    }

                    step_action = 1;
                    result = 0;
                }

            } catch (std::runtime_error re) {
                auto msg = new std::string(re.what());
                log->notify(msg);
                mglw1->declare(msg);
                return 1;
            } catch (...) {
                log->notify(new std::string("Error during splitting!"));
                result = 1;
            }

        }

        if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr)) {
            mglw1->set_tree(tree1<T>, processing->get_index());
            mglw1->show(&output_file1);
        } else {
            mglw1->declare(new std::string("Tree is empty."));
        }

        if ((tree2<T> != nullptr) && (tree2<T>->get_root() != nullptr)) {
            mglw2->set_tree(tree2<T>, processing->get_index());
            mglw2->show(&output_file2);
        } else {
            mglw2->declare(new std::string("Tree is empty."));
        }
```

```
            if ((tree1<T> != nullptr) && (tree1<T>->get_root() != nullptr))
split_maximum = tree1<T>->get_root()->get_weight();
          else  split_maximum = 0;

          return result;
      }


      template<typename T>
      int core::mult(MyGLWidget* mglw) {
                    log->notify(new    std::string("Multiple    treap    building
researching..."));

          mglw->prepare_drawing();

          long long int mid_size = 0;
          long long int theo_size = 0;

          for (int i = 0; i < 100; i++) {
              long long first_pos = in_stream->tellg();
              int result = build_rush<T>(nullptr, nullptr, 3, false);

              in_stream->clear();

              long long length = in_stream->tellg() - first_pos;
              for (long long i = 0; i < length; i++) in_stream->unget();
              if (result != 0) return 1;

              if (theo_size == 0) theo_size = tree1<T>->theory_depth();
              mid_size += tree1<T>->max_depth();
              free(tree1<T>);
              tree1<T> = nullptr;

                        log->notify(new  std::string("Building  tree  nuber  "  +
std::to_string(i) + "..."));
          }
```

```
            std::stringstream ss;
            ss << "Average depth of 100 trees was " << mid_size / 100 << "\nIdeal
size is " << theo_size;
            auto msg = new std::string(ss.str());
            log->notify(msg);
            mglw->declare(msg);

            return 0;
        }


        template<typename T>
        T core::get_value(std::string* str) {
             log->notify(new std::string("Reading value for new tree node: " +
*str + "..."));

            std::stringstream ss;
            T value;

            ss << *str;
            if (!(ss >> value)) {
                throw std::runtime_error("Node value can not be read!");
            }

            return value;
        }

        template<typename T>
        treap_node<T>* core::get_node(std::string* str) {
              log->notify(new std::string("Reading new tree node: " + *str +
"..."));

            unsigned long long pos = str->find(';');
            std::stringstream ss1, ss2;

            T value;
            double index = 0;
```

```
    if (str->empty()) {
        throw 1;
    }

    ss1 << (*str).substr(1, pos);
    ss2 << (*str).substr(pos + 1, (str->length() - 2 - pos));
    if (!(ss1 >> value)) {
        throw std::runtime_error("Node value can not be read!");
    }
    if (!(ss2 >> index)) {
        throw std::runtime_error("Node index can not be read!");
    } else if (index == 0.0) {
        throw 0;
    }

    return new treap_node<T>(value, index);
}


#endif // CORE_H
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. CORE.CPP

```
#include "core.h"


outlog* core::log = nullptr;

const std::string core::PATH = "C:/Users/miles/Documents/lab5/fls/";

int core::step_action = 0;

int core::split_maximum = 0;

const std::string core::input_file = PATH + "in.txt";
const std::string core::output_file1 = PATH + "out1.png";
const std::string core::output_file2 = PATH + "out2.png";
const std::string core::stack_file = PATH + "stack.svx";

int core::stack_size = 0;

const std::string core::subitem_delimiter = ";";  // number 2;
const std::string core::item_delimiter = ", ";  // number 2;
const std::string core::range_delimiter = " .. "; // number 4;
const std::string core::tree_delimiter = " = "; // number 3;
const std::string core::end_delimiter = "\n";  // number 1;

std::istream* core::in_stream = nullptr;

int core::type = 0;


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

    log->notify(new std::string("Reading source data from " + *input + ", which
is file (" + std::to_string(ff) +") of type " + std::to_string(type) + "..."));

    std::string* msg;
    std::string tree_string;
    if (ff) {
        std::ifstream* is = new std::ifstream();
        if (input->empty()) {
             log->notify(new std::string("Opening default file, located at " +
input_file));
            is->open(input_file);
        } else {
            log->notify(new std::string("Opening file, located at " + *input));
```

```
            is->open(*input);
        }

        if (*is && !is->fail()) {
            in_stream = is;
            log->notify(new std::string("File opened!"));
        } else {
            msg = new std::string("File can not be opened :(");
            log->notify(msg);
            mglw->declare(msg);
            return 1;
        }

    } else {
        if (input->empty()) {
            msg = new std::string("The input was empty.");
            log->notify(msg);
            mglw->declare(msg);
            return 1;
        } else {
            log->notify(new std::string("Opening string from input line."));
            std::stringstream* is = new std::stringstream();
            *is << *input;
            in_stream = is;
        }
    }

    type = tp;

    msg = new std::string("Data source loaded. We are ready 2 build.");
    log->notify(msg);
    mglw->declare(msg);
    return 0;
}


std::string* core::read_tree(std::istream* stream, int* delim_num) {
    log->notify(new std::string("Reading data from input stream."));

    std::string* result = new std::string();
    std::string buffer = std::string(3, ' ');
    *delim_num = 0;

    unsigned long long pos = 0;
    long long read = 0;
    while ((*delim_num == 0) && (*delim_num != 1)) {
        bool quit_flag = false;
```

```
         if (!(stream->read(&buffer[0], static_cast<signed>(buffer.length()))))
quit_flag = true;
        read = stream->gcount();
        result->append(buffer);

        pos = result->find(item_delimiter);
        if (pos != std::string::npos) {
            *delim_num = 2;
            continue;
        }

        pos = result->find(range_delimiter);
        if (pos != std::string::npos) {
            *delim_num = 4;
            continue;
        }

        pos = result->find(tree_delimiter);
        if (pos != std::string::npos) {
            *delim_num = 3;
            continue;
        }

        pos = result->find(end_delimiter);
        if (pos != std::string::npos) {
            *delim_num = 1;
            continue;
        }

        if (quit_flag) break;
    }

    if (pos != std::string::npos) {
        stream->clear();
            for (unsigned long long i = 0; i < (result->length() - pos) -
(buffer.length() - static_cast<unsigned>(read)); i++) {
            stream->unget();
        }
        stream->ignore(*delim_num);

    } else {
                          pos   =   result->length()   -   (buffer.length()   -
static_cast<unsigned>(read));
    }
    *result = result->substr(0, pos);

    return result;
}
```

```
int core::push(QPushButton* back_button, std::string* filename) {
    std::string name;
    if (filename != nullptr) name = *filename;
    else name = stack_file;

    log->notify(new std::string("Pushing state to file at " + name + "..."));

    std::stringstream stack_data;
    std::ifstream file_input(stack_file);

    stack_data << type << '\n';
    stack_data << step_action << '\n';
    switch (core::type) {
        case 0:
                if ((tree1<int> == nullptr) || (tree2<int> == nullptr)) throw
std::runtime_error("Trees have not been initialized.");
                stack_data << *(tree1<int>->to_string(0)) << tree_delimiter <<
*(tree2<int>->to_string(0)) << '\n';
            break;
        case 1:
               if ((tree1<char> == nullptr) || (tree2<char> == nullptr)) throw
std::runtime_error("Trees have not been initialized.");
               stack_data << *(tree1<char>->to_string(0)) << tree_delimiter <<
*(tree2<char>->to_string(0)) << '\n';
            break;
        case 2:
            if ((tree1<double> == nullptr) || (tree2<double> == nullptr)) throw
std::runtime_error("Trees have not been initialized.");
              stack_data << *(tree1<double>->to_string(0)) << tree_delimiter <<
*(tree2<double>->to_string(0)) << '\n';
            break;
        case 3:
                if ((tree1<std::string> == nullptr) || (tree2<std::string> ==
nullptr)) throw std::runtime_error("Trees have not been initialized.");
             stack_data << *(tree1<std::string>->to_string(0)) << tree_delimiter
<< *(tree2<std::string>->to_string(0)) << '\n';
            break;
    }

    long long before_pos = in_stream->tellg();
    if (in_stream->peek() != EOF) stack_data << in_stream->rdbuf();

    in_stream->clear();

    long long after_pos = in_stream->tellg() - before_pos;
    for (long long i = 0; i < after_pos; i++) in_stream->unget();
```

```
    stack_data << '\n';

     if ((file_input.peek() != EOF) && (filename == nullptr)) stack_data <<
file_input.rdbuf();

    std::ofstream file_output(name);
    file_output << stack_data.rdbuf();
    file_output.flush();
    file_output.close();

    if (filename == nullptr) {
        stack_size++;
        back_button->setEnabled(true);
    }

    return 0;
}

int core::pop(MyGLWidget* mglw1, MyGLWidget* mglw2, QPushButton* back_button,
std::string* filename) {
    std::string name;
    if (filename != nullptr) name = *filename;
    else name = stack_file;

    log->notify(new std::string("Getting state from file at " + name + "..."));

    std::stringstream ss1, ss2;
    std::string* support = new std::string();
    std::ifstream file_input(name);

    std::getline(file_input, *support);
    ss1 << *support;
    ss1 >> type;

    std::getline(file_input, *support);
    ss2 << *support;
    ss2 >> step_action;

    std::getline(file_input, *support);
    std::string double_buff;
    std::getline(file_input, double_buff);
    support->append("\n");
    support->append(double_buff);

    std::istringstream* os = new std::istringstream(support->c_str());
    in_stream = os;
```

```
    int result = 0;
    switch (core::type) {
        case 0:
            result = build_rush<int>(mglw1, mglw2, 1, true);
            break;
        case 1:
            result = build_rush<char>(mglw1, mglw2, 1, true);
            break;
        case 2:
            result = build_rush<double>(mglw1, mglw2, 1, true);
            break;
        case 3:
            result = build_rush<std::string>(mglw1, mglw2, 1, true);
            break;
    }
    free(support);

    if (filename == nullptr) stack_size--;
    else stack_size = 0;
    if (stack_size <= 0) back_button->setEnabled(false);

    std::stringstream buffer;
    buffer << file_input.rdbuf();

    std::ofstream file_output(stack_file);
    file_output << buffer.rdbuf();
    file_output.flush();
    file_output.close();
    file_input.close();

    return result;
}

void core::clear_stack(QPushButton* load_button) {
    log->notify(new std::string("Clearing all files."));

    in_stream = new std::istringstream();
    load_button->setEnabled(false);
    stack_size = 0;

    std::ofstream stack_output(stack_file);
    stack_output.close();

    std::ofstream output1(output_file1);
    output1.close();

    std::ofstream output2(output_file2);
    output2.close();
```

```
}
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QFileDialog>
#include <QMessageBox>
#include "ui_mainwindow.h"
#include "generator.h"
#include "core.h"



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 configure_input(int new_step_action);

     void read(bool from_file);

private slots:
     void generate();
     void read_from_file();
     void read_from_line();
```

```
        void save();
        void load();
        void clear();

        void back();
        void step();

        void log();
        void about_treap();
        void about_app();

        void merge();
        void split();
        void mult();
};

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

```
#include "mainwindow.h"


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

    ui->type_selector->addItems(available_types);

    connect(ui->generator_button, SIGNAL(clicked()), this, SLOT(generate()));
    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->back_button, SIGNAL(clicked()), this, SLOT(back()));

    connect(ui->save_button, SIGNAL(clicked()), this, SLOT(save()));
    connect(ui->load_button, SIGNAL(clicked()), this, SLOT(load()));
    connect(ui->reset_button, SIGNAL(clicked()), this, SLOT(clear()));

    connect(ui->log_button, SIGNAL(clicked()), this, SLOT(log()));
                connect(ui->about_treap_button,    SIGNAL(clicked()),    this,
SLOT(about_treap()));
    connect(ui->about_app_button, SIGNAL(clicked()), this, SLOT(about_app()));

    connect(ui->merge_button, SIGNAL(clicked()), this, SLOT(merge()));
    connect(ui->split_button, SIGNAL(clicked()), this, SLOT(split()));
    connect(ui->multiple_button, SIGNAL(clicked()), this, SLOT(mult()));

     std::string log_path(core::PATH);
    core::log = new outlog(this, &log_path);

    core::log->notify(new std::string("Application started..."));

    ui->canvas1->setHidden(true);
    ui->split_pos_box->setMinimum(0);
    generator::ground();
}

MainWindow::~MainWindow() {
    core::log->notify(new std::string("Application terminated..."));
    core::log->notify(new std::string("Have a good day, Mr. User!"));
    delete ui;
}
```

```
void MainWindow::configure_input(int new_step_action) {
    switch (new_step_action) {
        case 0:
            ui->merge_button->setEnabled(false);
            ui->split_button->setEnabled(false);
            ui->split_pos_box->setEnabled(false);
            ui->multiple_button->setEnabled(true);

            ui->generator_button->setEnabled(true);
            ui->line_button->setEnabled(true);
            ui->file_button->setEnabled(true);
            ui->type_selector->setEnabled(true);
            ui->line_input->setEnabled(true);
            ui->step_box->setEnabled(true);

            ui->step_button->setEnabled(false);

            core::log->notify(new std::string("Input configured: Initial."));
            break;
        case 1:
            ui->merge_button->setEnabled(true);
            ui->split_button->setEnabled(true);
            ui->split_pos_box->setEnabled(true);
            ui->split_pos_box->setMaximum(core::split_maximum);
            ui->multiple_button->setEnabled(true);

            ui->generator_button->setEnabled(true);
            ui->line_button->setEnabled(true);
            ui->file_button->setEnabled(true);
            ui->type_selector->setEnabled(true);
            ui->line_input->setEnabled(true);
            ui->step_box->setEnabled(true);

            ui->step_button->setEnabled(false);

                core::log->notify(new std::string("Input configured: Ready to
act."));
            break;
        case 10:
        case 11:
        case 12:
        case 20:
        case 21:
        case 30:
        case 31:
            ui->merge_button->setEnabled(false);
```

```
            ui->split_button->setEnabled(false);
            ui->split_pos_box->setEnabled(false);
            ui->multiple_button->setEnabled(false);

            ui->generator_button->setEnabled(false);
            ui->line_button->setEnabled(false);
            ui->file_button->setEnabled(false);
            ui->type_selector->setEnabled(false);
            ui->line_input->setEnabled(false);
            ui->step_box->setEnabled(false);
            ui->step_box->setChecked(true);

            ui->step_button->setEnabled(true);

                 core::log->notify(new std::string("Input configured: Steps in
progress."));
            break;
    }

    core::step_action = new_step_action;
}


void MainWindow::generate() {
    std::string* gen = generator::generate();
    core::log->notify(new std::string("New input line generated: " + *gen));
    ui->line_input->setText(QString::fromStdString(*gen));
}

void MainWindow::read_from_file() {
    core::log->notify(new std::string("Reading from file requested."));
    read(true);
}

void MainWindow::read_from_line() {
    core::log->notify(new std::string("Reading from line requested."));
    read(false);
}

void MainWindow::read(bool from_file) {
    configure_input(0);
    ui->canvas1->setHidden(true);
    ui->canvas1->prepare_drawing();
    ui->canvas2->prepare_drawing();

    std::string input = ui->line_input->text().toStdString();
          int result = core::launch(ui->canvas2, &input, from_file, ui-
>type_selector->currentIndex());
```

```
    std::string* msg;
    if (result == 0) {
        msg = new std::string("Data sources successfully loaded!");
        core::log->notify(msg);

        ui->canvas1->setHidden(false);
        ui->canvas1->prepare_drawing();
        ui->canvas2->prepare_drawing();

        if (!ui->step_box->isChecked()) {
            msg = new std::string("Quick treap building requested.");
            core::log->notify(msg);

            int res = 0;

            switch (core::type) {
                case 0:
                       res = core::build_rush<int>(ui->canvas1, ui->canvas2, 1,
false);
                    break;
                case 1:
                      res = core::build_rush<char>(ui->canvas1, ui->canvas2, 1,
false);
                    break;
                case 2:
                     res = core::build_rush<double>(ui->canvas1, ui->canvas2, 1,
false);
                    break;
                case 3:
                           res = core::build_rush<std::string>(ui->canvas1, ui-
>canvas2, 1, false);
                    break;
            }

            if (res == 0) {
                msg = new std::string("Both treaps successfully built.");
                core::log->notify(msg);
                ui->answer->setText(QString::fromStdString(*msg));
                configure_input(1);
                clear();
            } else {
                    msg = new std::string("Error occurs during building, check
input format again!");
                core::log->notify(msg);
                ui->answer->setText(QString::fromStdString(*msg));
            }
        } else {
```

```
            msg = new std::string("Step-by-step treap building requested.");
            core::log->notify(msg);
            configure_input(10);
            step();
        }

    } else {
         msg = new std::string("Error occurs during data source loading. Check
the input again!");
        core::log->notify(msg);
        ui->answer->setText(QString::fromStdString(*msg));
    }
}

void MainWindow::back() {
    core::log->notify(new std::string("Step back requested."));

    int result = core::pop(ui->canvas1, ui->canvas2, ui->back_button, nullptr);

    if (result == 0) {
                    core::log->notify(new   std::string("Step   back   performed
successfully."));
    } else {
        core::log->notify(new std::string("Step back failed!"));
    }

    configure_input(core::step_action);
}

void MainWindow::step() {
    core::log->notify(new std::string("Step forward requested."));

    if (core::step_action % 10 != 0) core::push(ui->back_button, nullptr);

    int result = 0;
    switch (core::type) {
        case 0:
                if (core::step_action < 20) result = core::build_step<int>(ui-
>canvas1, ui->canvas2);
              else if ((core::step_action >= 20) && (core::step_action < 30))
result = core::merge_step<int>(ui->canvas1, ui->canvas2);
             else result = core::split_step<int>(ui->split_pos_box->value(), ui-
>canvas1, ui->canvas2);
            break;
        case 1:
               if (core::step_action < 20) result = core::build_step<char>(ui-
>canvas1, ui->canvas2);
```

```
            else if ((core::step_action >= 20) && (core::step_action < 30))
result = core::merge_step<char>(ui->canvas1, ui->canvas2);
            else result = core::split_step<char>(ui->split_pos_box->value(),
ui->canvas1, ui->canvas2);
            break;
        case 2:
             if (core::step_action < 20) result = core::build_step<double>(ui-
>canvas1, ui->canvas2);
            else if ((core::step_action >= 20) && (core::step_action < 30))
result = core::merge_step<double>(ui->canvas1, ui->canvas2);
            else result = core::split_step<double>(ui->split_pos_box->value(),
ui->canvas1, ui->canvas2);
            break;
        case 3:
                                if (core::step_action < 20) result =
core::build_step<std::string>(ui->canvas1, ui->canvas2);
            else if ((core::step_action >= 20) && (core::step_action < 30))
result = core::merge_step<std::string>(ui->canvas1, ui->canvas2);
              else result = core::split_step<std::string>(ui->split_pos_box-
>value(), ui->canvas1, ui->canvas2);
            break;
    }

    std::string* msg = nullptr;
    if (result == 0) {
        msg = new std::string("Steps forward finished successfully.");
    } else if (result == 1) {
        msg = new std::string("Step forward failed!");
        configure_input(1);
    } else {
        msg = new std::string("Step forward performed.");
    }
    core::log->notify(msg);
    ui->answer->setText(QString::fromStdString(*msg));

    configure_input(core::step_action);
}


void MainWindow::save() {
    core::log->notify(new std::string("Save state requested."));

      QString file = QFileDialog::getSaveFileName(this, tr("Open Directory"),
QString::fromStdString(core::PATH + "saving.sv"), tr("State save files
(*.sv)"));
    std::string fstring(file.toStdString());

    if (fstring != "") {
```

```
        try {
            int result = core::push(ui->load_button, &fstring);
                if (result == 0) core::log->notify(new std::string("Save state
performed."));
                 else core::log->notify(new std::string("Save state ended with
error!"));
        } catch (std::runtime_error re) {
            core::log->notify(new std::string("Save state can not be performed:
" + std::string(re.what())));
        }
    } else core::log->notify(new std::string("Save state cancelled."));
}

void MainWindow::load() {
    ui->canvas1->setHidden(false);
    ui->canvas1->prepare_drawing();
    ui->canvas2->prepare_drawing();

    core::log->notify(new std::string("Load state requested."));

      QString file = QFileDialog::getOpenFileName(this, tr("Open Directory"),
QString::fromStdString(core::PATH), tr("State save files (*.sv)"));
    std::string fstring(file.toStdString());

    if (fstring != "") {
            int result = core::pop(ui->canvas1, ui->canvas2, ui->load_button,
&fstring);
        if (result == 0) {
            configure_input(core::step_action);
            ui->answer->setText(QString::fromStdString("Loded state from file "
+ fstring));
        }
              else core::log->notify(new std::string("Load state ended with
error!"));
    } else core::log->notify(new std::string("Load state cancelled."));
}

void MainWindow::clear() {
    core::log->notify(new std::string("Clear history requested."));

    core::clear_stack(ui->back_button);
}


void MainWindow::log() {
    core::log->notify(new std::string("Log window requested."));

    core::log->show();
```

```
}

void MainWindow::about_treap() {
     core::log->notify(new std::string("It was decided to learn something new
about treaps."));

    QMessageBox* msgbox = new QMessageBox(this);
    msgbox->setWindowTitle("About: treaps");
     msgbox->setText(QString::fromStdString("Treap is a combination of \"binary
search tree\" and \"binary heap\". It contains values in binary tree order and
keys in binary heap order.\n") +
                     QString::fromStdString("In this particular app a list based
on treap is shown. A search operation can be performed for O(log2N) iterations
on this kind of list."));
    msgbox->open();
}

void MainWindow::about_app() {
    core::log->notify(new std::string("It was decided to read user manual."));

    QMessageBox* msgbox = new QMessageBox(this);
    msgbox->setWindowTitle("About: treaps");
     msgbox->setText(QString::fromStdString("This program shows how list based
on treap works. A few words about its functionality:\n") +
                      QString::fromStdString("1. Input: Input string (in line or
in file) must be formatted. Elements should be separated by \", \", trees (up
to two) with \" = \". ") +
                      QString::fromStdString("There can be one or two trees, but
the first should be present in any case.\n") +
                       QString::fromStdString("2. Steps: Every tree function in
the programm may be performed by steps. All of them (except build) differ from
their fast analogues ") +
                       QString::fromStdString("due to their recursive nature. Do
not be misled, this was made for demonstrative purposes only!\n") +
                     QString::fromStdString("3. Build: Nothing interesting about
this function - it builds a treap from given array values.\n") +
                     QString::fromStdString("4. Generate input: Generates from 8
up to 13 integers, divided in two treaps or not. Use for quick and simple input
source.\n") +
                      QString::fromStdString("5. Merge: Merges second tree into
first one.\n") +
                       QString::fromStdString("6. Split: Splits first tree into
two from the given position. Given number means number of elements left in the
first tree.\n") +
                       QString::fromStdString("7. Step/Back: While step is only
available during \"step mode\", back also keeps all operations performed with
tree. Be careful, mode is also kept!\n") +
```

```
                    QString::fromStdString("8. Save/Load: You can save or load
a programm state to/from separate file. Operations stack is NOT included!\n") +
                     QString::fromStdString("9. Multiple times: This function
goal is to demonstrate the depth of the treap for some big (comparably)
numbers. ") +
                    QString::fromStdString("WARNING: this can cause performance
freezes. It runs NOT in separate thread. 2000 - is big enough for
demonstration. I warned you!"));
    msgbox->open();
}


void MainWindow::merge() {
    std::string* msg = new std::string("Treap merge requested.");
    core::log->notify(msg);

    try {
        core::push(ui->back_button, nullptr);
    } catch (std::runtime_error re) {
            core::log->notify(new std::string("Can not push state - trees not
initialized."));
    }

    if (!ui->step_box->isChecked()) {
        msg = new std::string("Quick treap merge requested.");
        core::log->notify(msg);

        int res = 0;

        switch (core::type) {
            case 0:
                res = core::merge_rush<int>(ui->canvas1, ui->canvas2);
                break;
            case 1:
                res = core::merge_rush<char>(ui->canvas1, ui->canvas2);
                break;
            case 2:
                res = core::merge_rush<double>(ui->canvas1, ui->canvas2);
                break;
            case 3:
                res = core::merge_rush<std::string>(ui->canvas1, ui->canvas2);
                break;
        }

        if (res == 0) {
            msg = new std::string("Treaps successfully merged.");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
```

```
            configure_input(1);
        } else {
            msg = new std::string("Error occurs during merging!");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
        }
    } else {
        msg = new std::string("Step-by-step treap merging requested.");
        core::log->notify(msg);
        configure_input(20);
        step();
    }
}

void MainWindow::split() {
    std::string* msg = new std::string("Treap split requested.");
    core::log->notify(msg);

    try {
        core::push(ui->back_button, nullptr);
    } catch (std::runtime_error re) {
            core::log->notify(new std::string("Can not push state - trees not
initialized."));
    }

    if (!ui->step_box->isChecked()) {
        msg = new std::string("Quick treap split requested.");
        core::log->notify(msg);

        int res = 0;

        switch (core::type) {
            case 0:
                    res = core::split_rush<int>(ui->split_pos_box->value(), ui-
>canvas1, ui->canvas2);
                break;
            case 1:
                   res = core::split_rush<char>(ui->split_pos_box->value(), ui-
>canvas1, ui->canvas2);
                break;
            case 2:
                 res = core::split_rush<double>(ui->split_pos_box->value(), ui-
>canvas1, ui->canvas2);
                break;
            case 3:
                res = core::split_rush<std::string>(ui->split_pos_box->value(),
ui->canvas1, ui->canvas2);
                break;
```

```
        }

        if (res == 0) {
            msg = new std::string("Treaps successfully splitted.");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
            configure_input(1);
        } else {
            msg = new std::string("Error occurs during splitting!");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
        }
    } else {
        msg = new std::string("Step-by-step treap splitting requested.");
        core::log->notify(msg);
        configure_input(30);
        step();
    }
}

void MainWindow::mult() {
      std::string* msg = new std::string("Multiple treap building research
requested.");
    core::log->notify(msg);

    configure_input(0);
    ui->canvas1->setHidden(true);
    ui->canvas1->prepare_drawing();
    ui->canvas2->prepare_drawing();

    std::string input = ui->line_input->text().toStdString();
     int result = core::launch(ui->canvas2, &input, false, ui->type_selector-
>currentIndex());

    if (result == 0) {
        msg = new std::string("Data sources successfully loaded!");
        core::log->notify(msg);

        log();

        switch (core::type) {
            case 0:
                result = core::mult<int>(ui->canvas2);
                break;
            case 1:
                result = core::mult<char>(ui->canvas2);
                break;
            case 2:
```

```
                result = core::mult<double>(ui->canvas2);
                break;
            case 3:
                result = core::mult<std::string>(ui->canvas2);
                break;
        }

        if (result == 0) {
            msg = new std::string("Research successful.");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
        } else {
            msg = new std::string("Error occuresduring research!");
            core::log->notify(msg);
            ui->answer->setText(QString::fromStdString(*msg));
        }

        clear();
    } else {
        msg = new std::string("Error occures, input again (in \"research\" mode
you can only input from line)!");
        core::log->notify(msg);
        ui->answer->setText(QString::fromStdString(*msg));
    }
}
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ. 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>1000</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>1001</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>Treaps</string>
  </property>
  <property name="alignment">
   <set>Qt::AlignCenter</set>
  </property>
 </widget>
</item>
<item>
 <layout class="QHBoxLayout" name="coordinator">
  <item>
   <widget class="QPushButton" name="about_treap_button">
    <property name="text">
     <string>What is the &amp;quot;Treap&amp;quot;?</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="about_app_button">
    <property name="text">
     <string>How does the app work?</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="log_button">
    <property name="text">
     <string>Log</string>
    </property>
```

```
 </widget>
</item>
<item>
 <spacer name="first_spacer">
  <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>
 <widget class="QPushButton" name="save_button">
  <property name="text">
   <string>Save</string>
  </property>
 </widget>
</item>
<item>
 <widget class="QPushButton" name="load_button">
  <property name="text">
   <string>Load</string>
  </property>
 </widget>
</item>
<item>
 <widget class="QPushButton" name="reset_button">
  <property name="text">
   <string>Clear History</string>
  </property>
 </widget>
</item>
<item>
 <spacer name="second_spacer">
  <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>
   <widget class="QPushButton" name="back_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="text">
     <string>Back</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="step_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="text">
     <string>Forward</string>
    </property>
   </widget>
  </item>
 </layout>
</item>
<item>
 <layout class="QHBoxLayout" name="input_container">
  <item>
   <widget class="QPushButton" name="generator_button">
    <property name="text">
     <string>Generate input</string>
    </property>
   </widget>
  </item>
  <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>Build 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>Build 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="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;. In multiple mode ranges are also available with delimiter &amp;quot; .. &amp;quot;. See more in &amp;quot;How does this app work?&amp;quot;.</string>
      </property>
     </widget>
    </item>
```

```
<item>
 <layout class="QHBoxLayout" name="canvases">
  <item>
   <widget class="MyGLWidget" name="canvas1">
    <property name="sizePolicy">
     <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
   </widget>
  </item>
  <item>
   <widget class="MyGLWidget" name="canvas2">
    <property name="sizePolicy">
     <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
   </widget>
  </item>
 </layout>
</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="merge_button">
    <property name="enabled">
```

```
     <bool>false</bool>
    </property>
    <property name="font">
     <font>
      <family>Consolas</family>
     </font>
    </property>
    <property name="text">
     <string>Merge</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QSpinBox" name="split_pos_box">
    <property name="enabled">
     <bool>false</bool>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="split_button">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="text">
     <string>Split</string>
    </property>
   </widget>
  </item>
  <item>
   <widget class="QPushButton" name="multiple_button">
    <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>
```

ПРИЛОЖЕНИЕ З

ИСХОДНЫЙ КОД ПРОГРАММЫ. MYGLWIDGET.H

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QOpenGLWidget>
#include <QPainter>
#include <QFile>
#include <sstream>
#include <iomanip>
#include "treap.h"


const int BONES_LIMIT = 5;
const int CIRCLE_SIZE = 40;


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


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


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

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

        int depth = static_cast<int>(source->max_depth());
        stepY = static_cast<int>(map->height() / 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" <<
std::setprecision(4) << source->get_root()->get_index();
```

```
            if (abs(source->get_root()->get_index() - outline) < 0.0001) {
                pntr->setBrush(Qt::red);
                      pntr->drawEllipse(QPointF(map->width() / 2, stepY / 2),
CIRCLE_SIZE, CIRCLE_SIZE);
                pntr->setBrush(Qt::white);
            } else {
                      pntr->drawEllipse(QPointF(map->width() / 2, stepY / 2),
CIRCLE_SIZE, CIRCLE_SIZE);
            }
             pntr->drawText(QRect(map->width() / 2 - CIRCLE_SIZE / 2, stepY / 2
-    CIRCLE_SIZE    /    2,    CIRCLE_SIZE,    CIRCLE_SIZE),    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", 8));
        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());

        pntr = new QPainter(map);
        pntr->fillRect(map->rect(), QBrush(Qt::white));
        pntr->end();
        free(pntr);
    }

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

```
protected:
    void paintGL() override;

private:
    int stepY = 0;
    QPainter* pntr;

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

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

        if (abs(node->get_index() - outline) < 0.0001) {
            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(treap_node<T>* node, bool is_right, int parent_x, int
parent_y, int half_stepX, double 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" << std::setprecision(4) << node-
>get_index();

        if (abs(node->get_index() - outline) < 0.0001) {
            pntr->setBrush(Qt::red);
                    pntr->drawEllipse(QPointF(child_x, child_y), CIRCLE_SIZE,
CIRCLE_SIZE);
            pntr->setBrush(Qt::white);
```

```
        } else {
                    pntr->drawEllipse(QPointF(child_x,  child_y),  CIRCLE_SIZE,
CIRCLE_SIZE);
        }
        pntr->drawText(QRect(child_x - CIRCLE_SIZE / 2, child_y - CIRCLE_SIZE /
2,            CIRCLE_SIZE,            CIRCLE_SIZE),            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
```

ПРИЛОЖЕНИЕ И

ИСХОДНЫЙ КОД ПРОГРАММЫ. 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.");
        painter.end();
    }
}

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

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

ПРИЛОЖЕНИЕ К

ИСХОДНЫЙ КОД ПРОГРАММЫ. OUTLOG.H

```
#ifndef OUTLOG_H
#define OUTLOG_H

#include <QWidget>
#include <QCloseEvent>
#include <QScrollArea>
#include <QHBoxLayout>
#include <QLabel>
#include <fstream>
#include <sstream>


class outlog : public QWidget {
    Q_OBJECT

public:
    outlog(QWidget * parent, std::string* path_to_dir);
    ~outlog() override;

    void notify(std::string* message);

    static int width;

private:
    static std::string log_file;

    void showEvent(QShowEvent *event) override;
    void closeEvent(QCloseEvent *event) override;

    QWidget* parent;
    QLabel* logs;

};

#endif // OUTLOG_H
```

ПРИЛОЖЕНИЕ Л

ИСХОДНЫЙ КОД ПРОГРАММЫ. OUTLOG.CPP

```
#include "outlog.h"


int outlog::width = 500;

std::string outlog::log_file = "";


outlog::outlog(QWidget* parent, std::string* path_to_dir) : QWidget() {
    this->parent = parent;

    QScrollArea *scroll = new QScrollArea(this);
    scroll->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOn);
    scroll->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOn);

    logs = new QLabel(this);
    logs->setContentsMargins(7, 7, 7, 7);
    logs->setAlignment(Qt::AlignTop | Qt::AlignLeft);
    scroll->setWidget(logs);
    scroll->setWidgetResizable(true);

    QHBoxLayout *dialog_layout = new QHBoxLayout(this);
    dialog_layout->addWidget(scroll);

    setLayout(dialog_layout);

    log_file = *path_to_dir + "log.lg";

    std::ofstream fstr(log_file);
    fstr.flush();
    fstr.close();
}

outlog::~outlog() {}


void outlog::notify(std::string* message) {
    std::ofstream outfile(log_file, std::ios_base::app);
    outfile << *message;
    outfile << '\n';
    outfile.flush();
    outfile.close();

    if (this->isVisible()) {
        QString text = logs->text();
```

```
        text.append(QString::fromStdString(*message));
        text.append("\n");
        logs->setText(text);
    }
}

void outlog::showEvent(QShowEvent *event) {
    std::stringstream buffer;

    std::ifstream infile(log_file);
    buffer << infile.rdbuf();
    infile.close();

    QString text = logs->text();
    text.append(QString::fromStdString(buffer.str()));
    logs->setText(text);

      setGeometry(parent->geometry().right(),  parent->geometry().top(),  width,
parent->geometry().height());
    this->setWindowTitle("Logs");
    event->accept();
}

void outlog::closeEvent (QCloseEvent *event) {
    QString text;
    logs->setText(text);
    event->accept();
}
```

ПРИЛОЖЕНИЕ М

ИСХОДНЫЙ КОД ПРОГРАММЫ. GENERATOR.H

```
#ifndef GENERATOR_H
#define GENERATOR_H

#include <cstdlib>
#include <string>
#include <time.h>
#include <sstream>


class generator {
public:
    static std::string* generate();
    static void ground();

private:
    static int length;
};

#endif // GENERATOR_H
```

ПРИЛОЖЕНИЕ Н

ИСХОДНЫЙ КОД ПРОГРАММЫ. GENERATOR.CPP

```
#include "generator.h"


int generator::length = 12;

void generator::ground() {
    long long ltime = time(nullptr);
    unsigned int stime = static_cast<unsigned int>(ltime/2);
    srand(stime);
}

std::string* generator::generate() {
    std::stringstream result;
    double randomite = 0;
    bool two_trees = (static_cast<double>(rand()) / RAND_MAX) > 0.5;

    while (randomite < length) {
        double ran = static_cast<double>(rand());

        randomite += (ran / RAND_MAX) + 1;
        int comp = static_cast<int>(ran) % 100;

        result << comp;
        if (two_trees && (randomite > (static_cast<int>(rand()) % length))) {
            result << " = ";
            two_trees = false;
        }
        else result << ", ";
    }

    result << static_cast<int>(rand()) % 100;
    return new std::string(result.str());
}
```

ПРИЛОЖЕНИЕ О

ИСХОДНЫЙ КОД ПРОГРАММЫ. LR5.PRO

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

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 += \
    core.cpp \
    generator.cpp \
    main.cpp \
    mainwindow.cpp \
    myglwidget.cpp \
    outlog.cpp

HEADERS += \
    core.h \
    generator.h \
    mainwindow.h \
    myglwidget.h \
    outlog.h \
    treap.h \
    treap_node.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
```