**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: **Хеш-таблицы с открытой адресацией**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Почаев Н.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

# ЗАДАНИЕ

# на курсовую работу

|  |  |  |
| --- | --- | --- |
| Студент Почаев Н.А. | | |
| Группа 8381 | | |
| Тема работы: Хеш-таблицы с цепочками | | |
| Исходные данные: необходимо выполнить визуализацию структуры данных – хеш-таблица с использованием цепочек, как метода разрешения коллизий. Должны быть реализована возможность пошагово построения и выполнения операций вставка и удаление. | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Задание», «Описание программы», «Теоретические положения», «Демонстрация», «Заключение», «Список использованных источников». | | |
| Предполагаемый объем пояснительной записки:  Не менее 60 страниц. | | |
| Дата выдачи задания: | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Почаев Н.А. |
| Преподаватель |  | Жангиров Т.Р. |

# Аннотация

В ходе выполнения курсовой работы была разработана программа с GUI, позволяющая визуализировать алгоритм построения хеш-таблицы с возможностью пошагового выполнения: доступны шаги как вперёд, так и назад. Также на построенном графическом представлении доступны для выполнения операции вставки и удаления через соответствующие элементы графического интерфейса. Созданная программа обладает следующей функциональностью: на начальном экране возможно открыть файл, содержащий данный для хеш-таблицы, в данной демонстрации для этого используются строки, а также выбрать режим построения: пошаговый или моментальный. В окне графического отображения пользователю доступны функции перехода вперёд и назад по ходу построения таблица и указанные выше операции изменения таблицы.

# Summary

During the course work, a program was developed with a GUI that allows you to visualize an algorithm for constructing a hash table with the possibility of step-by-step execution: steps forward and backward are available. Also, on the constructed graphical representation, operations for inserting and deleting through the corresponding elements of the graphical interface are available. The created program has the following functionality: on the initial screen, it is possible to open a file containing the data for the hash table, in this demonstration, lines are used for this, and you can also select the build mode: step-by-step or instant. In the graphical display window, the user can use the functions of moving forward and backward during the construction of the table and the above table change operations.

# Содержание

[ЗАДАНИЕ 2](#_Toc27694140)

[на курсовую работу 2](#_Toc27694141)

[Аннотация 3](#_Toc27694142)

[Summary 3](#_Toc27694143)

[Содержание 4](#_Toc27694144)

[ВВЕДЕНИЕ 6](#_Toc27694145)

[Цель работы 6](#_Toc27694146)

[Основные задачи 6](#_Toc27694147)

[Методы решения 6](#_Toc27694148)

[1. Задание 7](#_Toc27694149)

[2. Описание программы 8](#_Toc27694150)

[2.1. Описание интерфейса пользователя 8](#_Toc27694151)

[2.3. Описание основных полей и методов класса графического отображения 9](#_Toc27694152)

[2.4 Описание методов базового класса TableHandler 12](#_Toc27694153)

[3. Теоретические положения работы 14](#_Toc27694154)

[3.1 Общее описание работы хеш-таблицы с цепочками (хеширование) 14](#_Toc27694155)

[3.2 Реализация методов вставки и удаления элемента 16](#_Toc27694156)

[4. демонстрация 17](#_Toc27694157)

[4.1 Окна интерфейса программы 17](#_Toc27694158)

[4.2 Результаты графической обработки таблицы 17](#_Toc27694159)

[Заключение 20](#_Toc27694160)

[Список использованных источников 21](#_Toc27694161)

[Приложение а. 22](#_Toc27694162)

[Исходный код программы. main.c 22](#_Toc27694163)

[Приложение Б. 23](#_Toc27694164)

[Исходный код программы. drawing.h 23](#_Toc27694165)

[Приложение В. 26](#_Toc27694166)

[Исходный код программы. Hashtable.h 26](#_Toc27694167)

[Приложение Г. 36](#_Toc27694168)

[Исходный код программы. mainwindow.h 36](#_Toc27694169)

[Приложение Д. 37](#_Toc27694170)

[Исходный код программы. Tablerhandler.h 37](#_Toc27694171)

[Приложение Е. 38](#_Toc27694172)

[Исходный код программы. Drawing.cpp 38](#_Toc27694173)

[Приложение Ж. 41](#_Toc27694174)

[Исходный код программы. functionsfordrawing.cpp 41](#_Toc27694175)

[Приложение И. 47](#_Toc27694176)

[Исходный код программы. mainwindow.cpp 47](#_Toc27694177)

[Приложение К. 49](#_Toc27694178)

[Исходный код программы. setupui.cpp 49](#_Toc27694179)

[Приложение Л. 53](#_Toc27694180)

[Исходный код программы. setupui.cpp 53](#_Toc27694181)

[Приложение М. 55](#_Toc27694182)

[Исходный код программы. tablehandler.cpp 55](#_Toc27694183)

[Приложение Н. 57](#_Toc27694184)

[Исходный код программы. Allheaders.h 57](#_Toc27694185)

# ВВЕДЕНИЕ

## Цель работы

Реализация и демонстрация работы хеш-таблицы, использующей для разрешения коллизий метод цепочек.

## Основные задачи

Реализация ввода из файла, визуализация таблицы, а также алгоритмов вставки и удаления с возможностью пошагового выполнения начального построения таблицы.

## Методы решения

Разработка программы велась на базе операционной системы Linux Manjaro в интегрированной среде разработки QtCreator. Для создания графической оболочки использовалась сторонняя графическая библиотека GitHub/laserpants/qt-material-widgets по открытой лицензии. Реализация хеш-таблицы с методом разрешения коллизий путём использования цепочек была выполнена в контейнерном классе HashTable. Для выполнения сторонних операций, связанных с обработкой входных строк для отрисовки в таблице и осуществлении задержек в ходе ожидания нажатия пользователем кнопки, был реализован базовый класс TableHandler. Для осуществления отрисовок на QGraphicsScene и сопутствующих операциям перехода и преобразования изменений был создан класс окна – DrawingWindow.

# Задание

Необходимо продемонстрировать алгоритм построения хеш-таблицы, а также вставки и удаления элемента.

Демонстрация должна:

1. Быть подробной и понятной (в том числе сопровождаться пояснениями);
2. Иметь возможность выполнения в пошаговом режиме;
3. Иметь возможность быть использованной в обучающих целях.

# 2. Описание программы

## 2.1. Описание интерфейса пользователя

После запуска программы пользователя встречает небольшое окно – меню приложения, в котором предоставляется возможность загрузки файла с исходными данными, выбора режима выполнения (при его отсутствии, по умолчанию, осуществляется моментальный режим). Описанные графические объекты интерфейса представлены в табл 1.

Таблица 1 – Основные виджеты начального меню

|  |  |  |
| --- | --- | --- |
| Класс объекта | Название виджета | Назначение |
| QtMaterialFlatButton | fileOpenButton | Кнопка открытия и загрузки файла |
| QLabel | fileWayLabel | Поле вывода пути до выбранного файла |
| QtMaterialFlatButton | runButton | Кнопка запуска графического окна приложения |
| QtMaterialRadioButton | stepByStepMode | Кнопка активации пошагового начального построения |
| QtMaterialRadioButton | allInMomentMode | Кнопка активации моментального начального построения |

После нажатия кнопки запуска окна отрисовки графического представления хеш-таблицы, перед пользователем открывается полноэкранный интерфейс взаимодействия с таблицей. Объекты, помещённые в данный интерфейс описаны в табл. 2.

Таблица 2 - Основные виджеты окна графической отрисовки

|  |  |  |
| --- | --- | --- |
| Класс объекта | Название виджета | Назначение |
| QtMaterialFlatButton | nextStepButton | Кнопка перехода на предыдущий шаг начального построения хеш-таблицы |
| QtMaterialFlatButton | prevStepButton | Кнопка перехода на следующий шаг начального построения хеш-таблицы |
| QtMaterialFlatButton | addElButton | Кнопка добавления нового элемента в хеш-таблицы |
| QtMaterialFlatButton | delElButton | Кнопка удаления элемента из хеш-таблицы |
| QtMaterialFlatButton | showAllButton | Кнопка прерывания пошагового режима построения таблицы и её полный вывод |
| QtMaterialTextField | inputEl | Строка ввода элемента для вставки или удаления из хеш-таблицы |

## 2.3. Описание основных полей и методов класса графического отображения

В табл. 3 приведены основные поля класса DrawingWindow, отвечающие за отрисовку всех графических объектов.

Таблица 3 – основные поля класса DrawingWindow

|  |  |  |
| --- | --- | --- |
| Класс / тип поля | Название поля | Назначение |
| QPoint | startDrawPos | Начальная позиция отрисовки относительно начала координат |
| int | nodeHight  nodeWidth | Высота и ширина начальных ячеек хеш-таблица (индексация) |
| int | spaceBetNodes | Ширина вертикального пробела между узлами |
| int | rectHieght  rectWidth | Высота и ширина прямоугольника – ячейки хеш-таблицы |
| int | arrowLength | Длина стрелки, соединяющей узлы |
| int | arrowTriangleHight  arrowTriangleLength | Высота и длина наконечника стрелки |
| int | levelsCount | Количество графических уровней в хеш-таблице |
| std::vector<int> | levelsLength | Вектор длин каждого уровня хеш-таблицы |
| int | stepsCount | Счётчик текущих шагов во время начального построения хеш-таблицы |
| std::vector<std::string> | inpStrs | Вектор входных строк (для проверки и преобразования при отрисовке) |

Далее в табл. 4 представлены поля специального подкласса nodeRect, хранящего всю необходимую информацию для отрисовки одного прямоугольника, представляющего собой ячейку хеш-таблицы.

Таблица 4 – поля подкласса nodeRect

|  |  |  |
| --- | --- | --- |
| Класс / тип поля | Название поля | Назначение |
| QRect | geomPar | Графические координаты прямоугольника в заданных координатах графической сцены Qt |
| std::string | data | Строка – содержимая текущей ячейки в обработанном для вывода на экран виде |
| int | level | Уровень узла в таблица |
| int | posInLevel | Позиция узла на данном уровне |

Далее в табл. 5 представлены основные методы описываемого класса, за счёт которых происходит отрисовка всех графических объектов в программе.

Таблица 5 – основные методы класса DrawWindow

|  |  |  |
| --- | --- | --- |
| Возвращаемое значение | Сигнатура | Назначение |
| - | DrawingWindow(std::vector<std::string>, std::vector<std::string>&, bool) | Консруктор класса, вызываемый из основного окна MainWindow и осуществляющий начальную инициализацию всех полей и проверку переданных флагов на режим работы |
| void | drawNodesStartGrid(int) | Отрисовка начальной сетки с индексацией для хеш-таблицы |
| void | drawNode(std::string data, int level, int pos) | Отрисовка единичного экземпляра узла – прямоугольника на графической сцене с по его содержимому и координатам |
| void | makeNodesRects(lrstruct::HashTable<std::string>\*, std::vector<nodeRect>&) | Высчитывание для переданной таблицы массива структур nodeRect, описанного ранее, для дальнешего размещения узлов на сцене |
| void | drawBaseRects() | Мгновенная отрисовка полной таблице на основе данных, вычисленных благодаря предыдущему методу |
| void | drawRectsStepByStep() | Отрисовка каждого узла – ячейки хеш-таблицы в пошаговом режиме |
| void | drawRect() | Отрисовка единичного экземпляра узла – прямоугольника на графической сцене с заданными параметрами цвета |
| void | closeEvent() | Слот, отлавливающий событие закрытия экрана демонстрации для обратной активации окна меня |

## 2.4 Описание методов базового класса TableHandler

В базовом классе TableHandler описаны методы, не имеющие прямого отношения к контейнерному классу хеш-таблицы и её непосредственной отрисовке, но тем не менее выполняющие вспомогательную техническую роль. Их описание приведено в табл. 6.

Таблица 6 – Описание методов базового класса TableHandler

|  |  |  |
| --- | --- | --- |
| Возвращаемое значение | Сигнатура | Назначение |
| void | makeHashTable(std::vector<std::string>, lrstruct::HashTable<std::string>\*&) | По заданному вектору строк форирует хэш-таблицу |
| void | static void makeHashTableDump(const lrstruct::HashTable<std::string>\*) | Проиводит вывод форматированного лога вида таблицы в файл log.txt |
| void | separateNames(std::string&) | Для демонстрации работы хеш-таблицы используются имена и фамилии людей. Данный метод выполняет их разделение по строкам для форматированного вывода |
| std::string | connectNames(std::string) | Выполняет обратное соединение имени и фамилии в одну строку через пробельный символ |
| void | loopLatency() | Функция задержки пошагового выполнения отрисовки хеш-таблицы (смена состояния происходит по флагу stepLoopSwitcher) |
| void | disableLoop() | Выполняет прерывание текущей итерации выполнения задержки выполнения программы |

# 3. Теоретические положения работы

## 3.1 Общее описание работы хеш-таблицы с цепочками (хеширование)

Хеширование (англ. hashing) — класс методов поиска, идея которого состоит в вычислении хеш-кода, однозначно определяемого элементом с помощью хеш-функции, и использовании его, как основы для поиска (индексирование в памяти по хеш-коду выполняется за . В общем случае, однозначного соответствия между исходными данными и хеш-кодом нет в силу того, что количество значений хеш-функций меньше, чем вариантов исходных данных, поэтому существуют элементы, имеющие одинаковые хеш-коды — так называемые коллизии, но если два элемента имеют разный хеш-код, то они гарантированно различаются.

Разрешение коллизий с помощью цепочек. Каждая ячейка массива содержит указатель на начало списка всех элементов, хеш-код которых равен , либо указывает на их отсутствие. Коллизии приводят к тому, что появляются списки размером больше одного элемента.

В зависимости от того нужна ли нам уникальность значений операции вставки у нас будет работать за разное время. Если не важна, то мы используем список, время вставки в который будет в худшем случае равна . Иначе мы проверяем есть ли в списке данный элемент, а потом в случае его отсутствия мы его добавляем. В таком случае вставка элемента в худшем случае будет выполнена за .

Время работы поиска в наихудшем случае пропорционально длине списка, а если все ключей захешировались в одну и ту же ячейку (создав список длиной ) время поиска будет равно плюс время вычисления хеш-функции, что ничуть не лучше, чем использование связного списка для хранения всех элементов.

Удаления элемента может быть выполнено за , как и вставка, при использовании двусвязного списка.

Перехеширование. При добавлении в хеш-таблицу большого количества элементов могут возникнуть ухудшения в ее работе. Обработка любого вызова будет занимать больше времени из-за увеличения размеров цепочек при хешировании на списках или кластеризации при хешировании с открытой адресацией, также, при хешировании с открытой адресацией может произойти переполнение таблицы. Для избежание таких ситуаций используется выбор новой хеш-функции и (или) хеш-таблица большего размера. Этот процесс называется перехеширование (англ. rehashing).

При использовании хеширования цепочками, элементы с одинаковым результатом хеш-функции помещают в список. Так как операции добавления, поиска и удаления работают за, где - длина списка, то с некоторого момента выгодно увеличить размер хеш-таблицы, чтобы поддерживать амортизационную стоимость операции .

Рассмотрим следующий алгоритм перехеширование: когда в хеш-таблицу добавлено элементов, где - размер хеш-таблицы, создадим новую хеш-таблицу размера , и последовательно переместим в нее все элементы первой таблицы. При этом, сменим хеш-функцию так, чтобы она выдавала значения .

Найдем амортизационную стоимость добавления, после которого было сделано перехеширование, используя метод предоплаты. С момента последнего перехеширования было произведено не менее операций добавления, так как изначально в массиве находится элементов (или 0 в начале работы), а перехеширование происходит при наличии элементов.

Для проведения перехеширования необходимо произвести операций добавления, средняя стоимость которых составляет , потратить операций на проход хеш-таблицы, и столько же на удаление предыдущей таблицы. В итоге, если мы увеличим стоимость каждой операции добавления на 6, то есть на , операция перехеширования будет полностью предоплачена. Значит, амортизационная стоимость перехеширования при открытом типе хеш-таблицы равна .

## 3.2 Реализация методов вставки и удаления элемента

В приведённом ниже фрагменте кода демонстрируется реализация операции вставки нового элемента в хеш-таблицу по его значению.

std::pair<iterator,bool> insert(const\_reference key) {

if (!count(key)) {

insert\_unchecked(key);

return std::make\_pair(find(key), true);

}

return std::make\_pair(find(key), false);

}

В случае, если значение ключа, сгенерированного хеш-функцией, является уникальным для данной таблицы, то производится прямая вставка без проверок, иначе осуществляется нахождение пары для нового узла в списке по уровню данного значения.

Реализация метода удаления erase() строится на прохождении таблицы, учитывая списки сгенерированных ключей, для более быстрого поиска элемента, подлежащего удалению. Исходный код данного и других методов взаимодействия с хеш-таблицей представлен в Приложениях.

# 4. демонстрация

## 4.1 Окна интерфейса программы

На рис. 1 представлено стартовое меню программы.

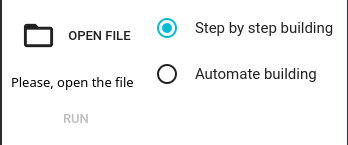


Рисунок 1 – стартовое меню программы

На рис. 2 представлено окно графического представления со стартовой сеткой для пошаговой отрисовки таблицы.



Рисунок 2 – начало пошаговой отрисовки таблицы

## 4.2 Результаты графической обработки таблицы

На рис. 3 представлен результат мгновенного построения таблицы.

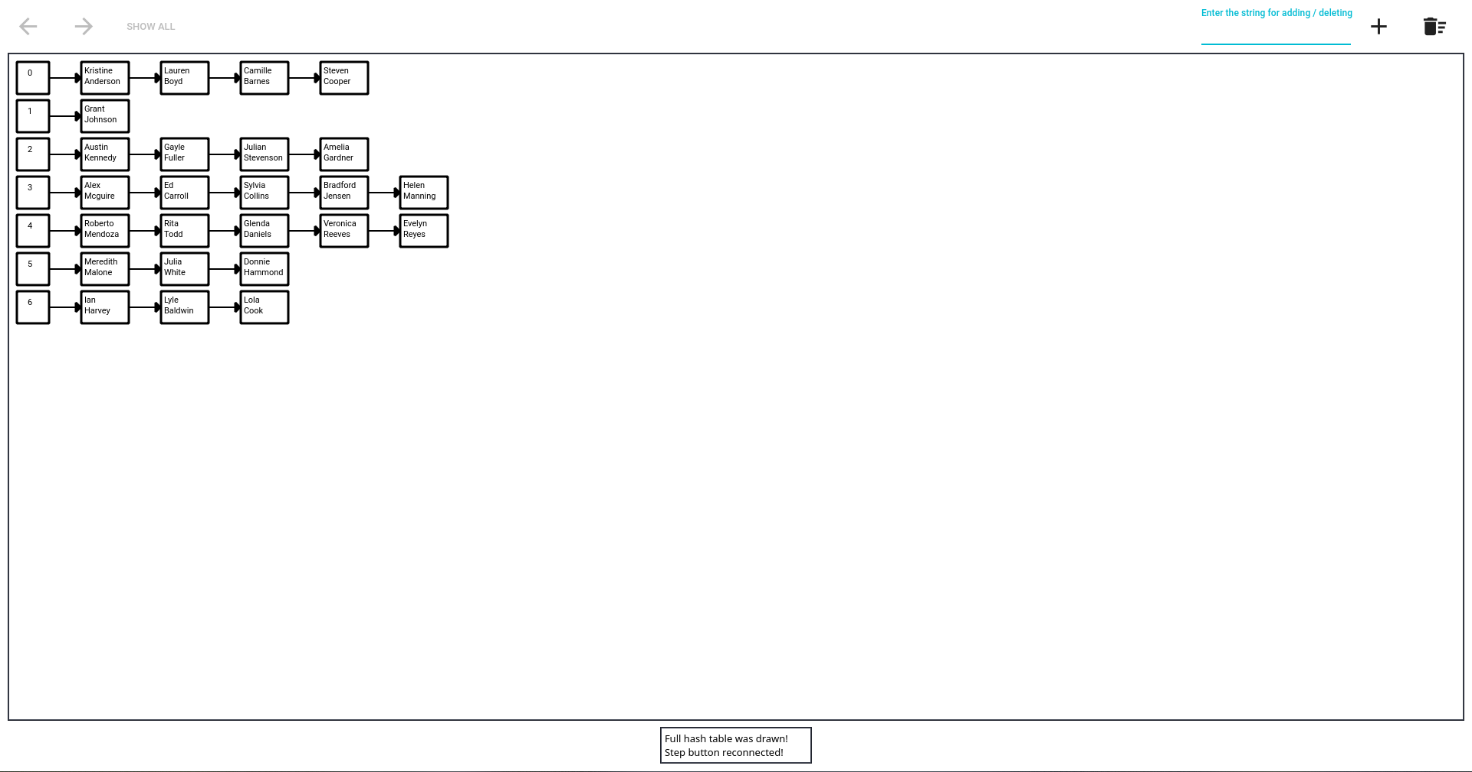


Рисунок 3 – полностью построенная хеш-таблица для 25 значений

На рис. 4 представлен результат добавления нового элемента в таблицу.

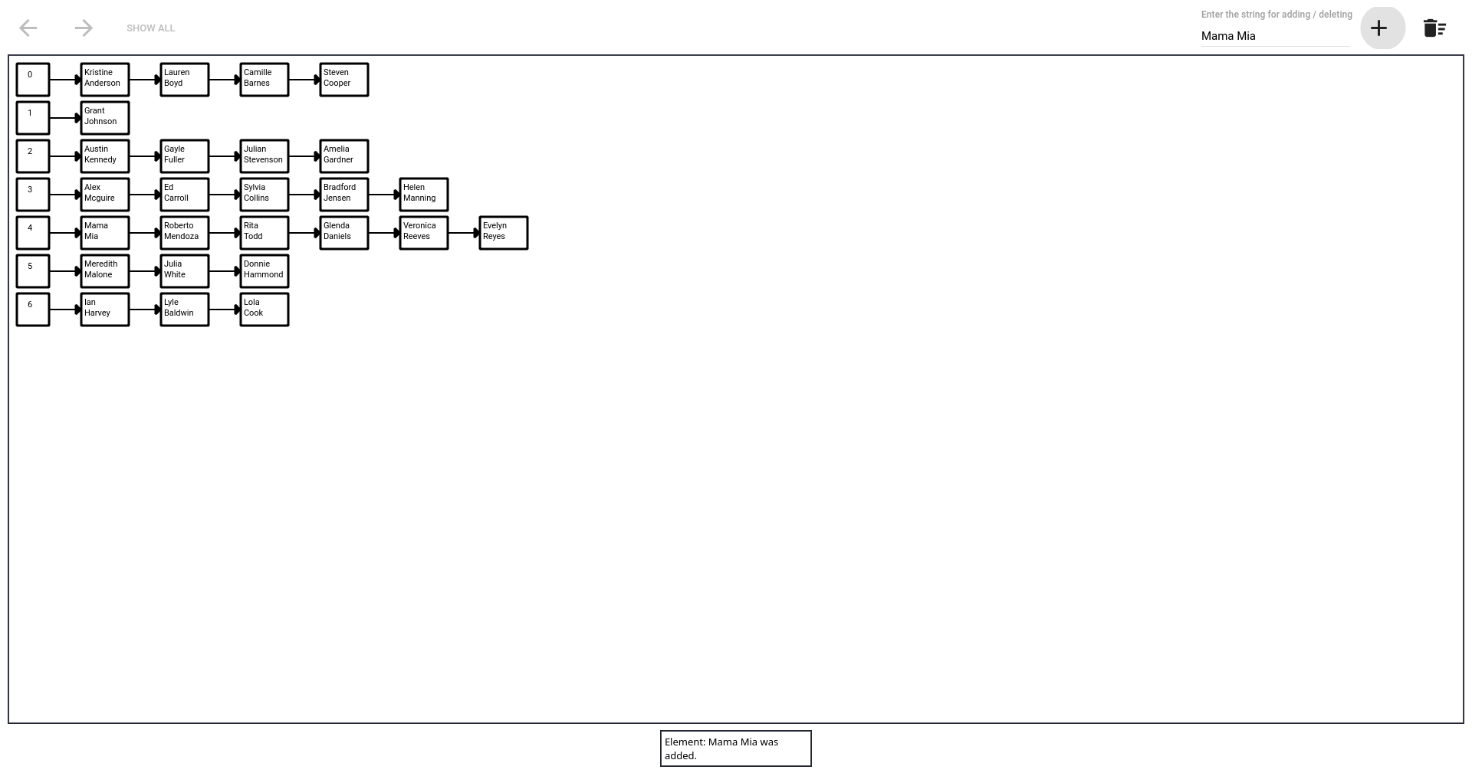


Рисунок 4 – результат добавления в таблицу элемента Mama Mia

На рис. 5 представлен результат добавленного элемента из таблицы.

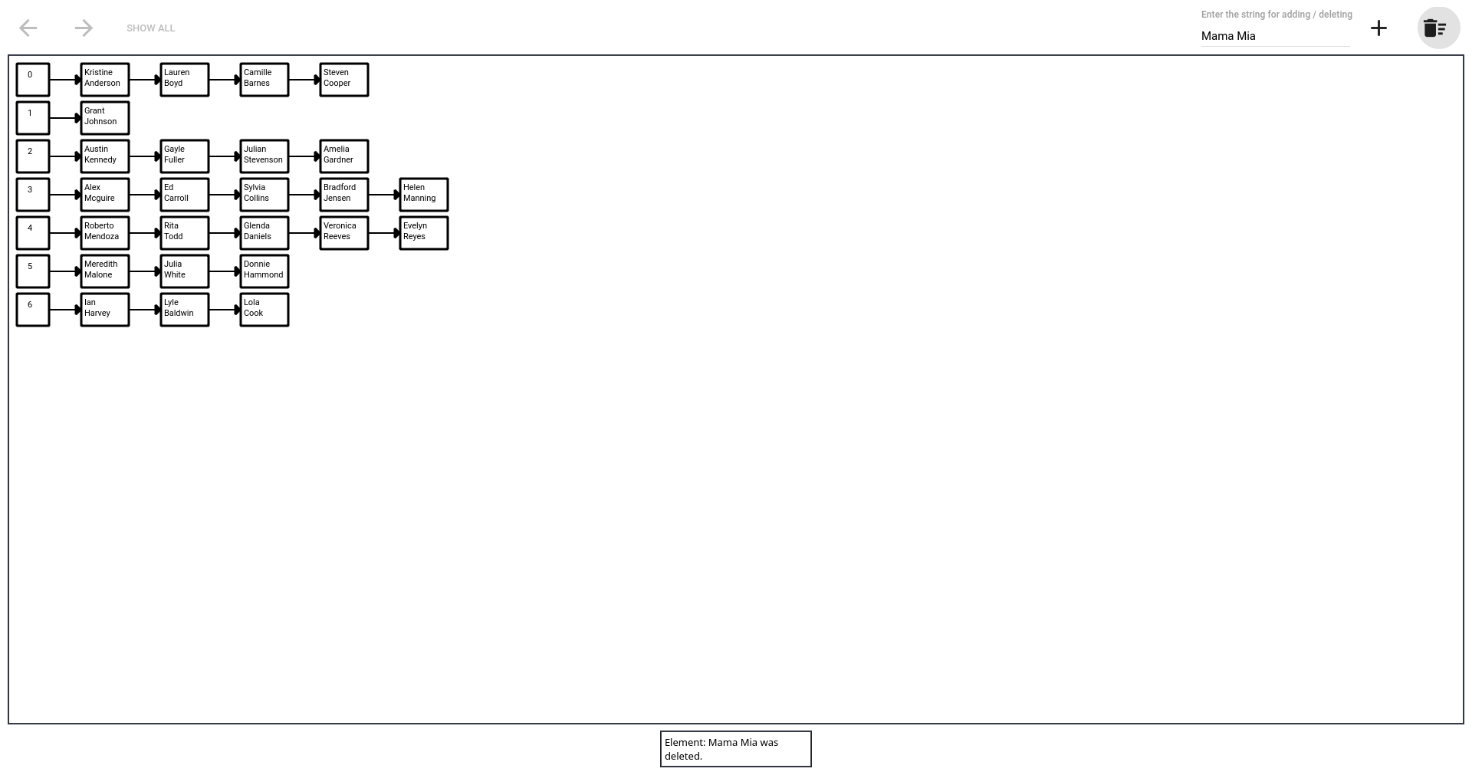


Рисунок 5 – результат удаления элемента Mama Mia

На рис. 6 представлен пример таблицы, сгенерированной для 1000 входных строк. В данном случае к графической сцене были добавлены полосы прокрутки для масштабирования.

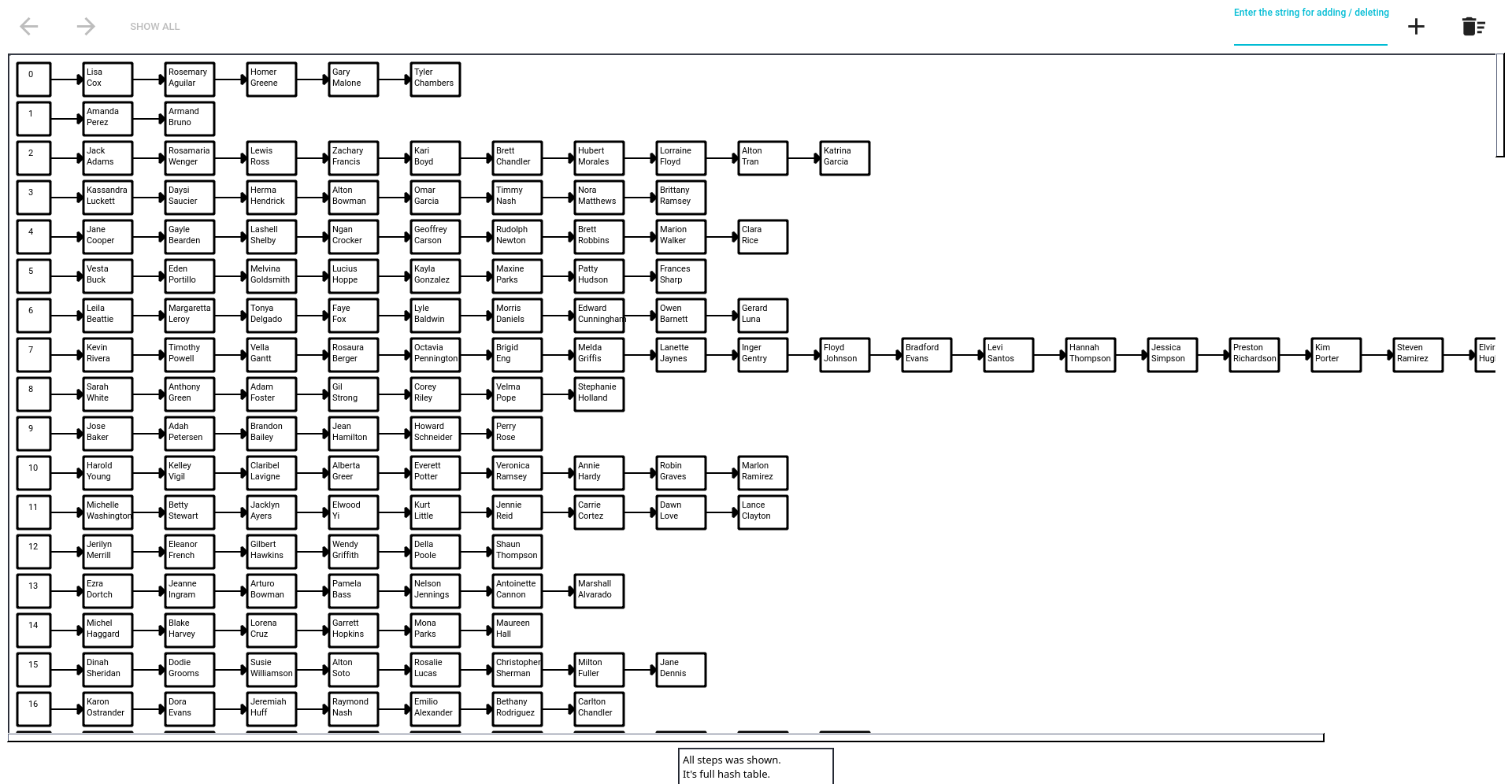


Рисунок 6 – таблица для 1000 элементов

# Заключение

В ходе выполнения курсовой работы была разработана программа, которая обладает следующей функциональностью: пошаговое построение хеш-таблицы по заданному набору данных, визуализация процесса с выводом пояснений. Также были реализованы операции вставки и удаления, осуществление которых также возможно с визуализацией и выводом поясняющих сообщений. Результативную программу возможно использовать в обещающих целях, для изучения структуры хеш-таблицы с цепочками.

# Список использованных источников

1. Bjarne Stroustrup. A Tour of C++. М.: Addison-Wesley, 2018. 217 с.
2. Макс Шлее. Qt5.10. Профессиональное программирование на C++. M.: BHV-СПб, 2018, 513 c.
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> (дата обращения: 13.12.2019).
5. Habrahabr. // Habr. URL: <https://habr.com/> (дата обращения: 16.12.2019).

# Приложение а.

# Исходный код программы. main.c

#include <QApplication>

#include <QDebug>

#include "mainwindow.h"

#include "allheaders.h"

#include "TableHandler.h"

int main(int argc, char \*argv[]) {

*//* *force* *initialization* *of* *library* *resources*

Q\_INIT\_RESOURCE(resources);

QApplication a(argc, argv);

MainWindow window;

window.show();

*return* a.exec();

}

# Приложение Б.

# Исходный код программы. drawing.h

#ifndef DRAWINGWINDOW\_H

#define DRAWINGWINDOW\_H

#include "allheaders.h"

#include "HashTable.h"

#include "TableHandler.h"

#include <QSpacerItem>

#include <qtmaterialslider.h>

#include <qtmaterialradiobutton.h>

#include <qtmaterialtoggle.h>

#include <qtmaterialtextfield.h>

#include <qtmaterialscrollbar.h>

#include <qtmaterialflatbutton.h>

#include <qtmaterialdialog.h>

#include <qtmaterialcheckbox.h>

#include "lib/qtmaterialstyle.h"

class DrawingWindow : public QWidget {

Q\_OBJECT

public:

DrawingWindow();

DrawingWindow(std::vector<std::string>, std::vector<std::string>&, bool);

private:

void setUpUI();

void setUpDrawingTools();

QGraphicsView \*mainGraphicsView;

// correct, cuz there's the only one scene + view

QGraphicsScene \*mainGraphicsScene;

QTextEdit \*mainTextOutput;

QtMaterialFlatButton \*nextStepButton;

QtMaterialFlatButton \*prevStepButton;

QtMaterialFlatButton \*addElButton;

QtMaterialFlatButton \*delElButton;

QtMaterialFlatButton \*showAllButton;

QtMaterialTextField \*inputEl;

const QPoint startDrawPos{10, 10};

// The Ultimate Question of Life,

// the Universe, and Everything, also Node Size

const int nodeHight{42};

const int nodeWidth{42};

const int spaceBetNodes{8};

const int rectWidth{62};

const int rectHieght{42};

const int arrowLength{42};

const int arrowTriangleLength{6};

const int arrowTriangleHight{6};

int levelsCount{0};

std::vector<int> levelsLength;

int stepsCount{0};

std::vector<std::string> inpStrs;

class nodeRect {

public:

QRect geomPar;

std::string data;

int level;

int posInLevel;

bool operator == (const nodeRect& rv) const {

return (geomPar == rv.geomPar && data == rv.data && level == rv.level && posInLevel == rv.posInLevel);

}

bool operator != (const nodeRect& rv) const {

return (geomPar != rv.geomPar || data != rv.data || level != rv.level || posInLevel != rv.posInLevel);

}

};

// current rects

std::vector<nodeRect> tableNodesRect;

// BACKUP FOR STEP BACK

std::vector<std::vector<nodeRect>> tableBackUp;

QPen pen;

QColor color;

QBrush brush;

QFont font;

void drawNodesStartGrid(int);

void drawNode(std::string data, int level, int pos);

void makeNodesRects(lrstruct::HashTable<std::string>\*, std::vector<nodeRect>&);

void drawBaseRects();

void drawRectsStepByStep();

void drawRect(nodeRect&, QPen, QBrush);

lrstruct::HashTable<std::string> \*workTable;

protected:

void closeEvent(QCloseEvent\*);

protected slots:

void reconnectForStepByStep();

void baseDrawStepBack();

void showAllSteps();

void reconectForAddDel();

void addEl();

void delEl();

signals:

void closeSignal();

};

#endif // DRAWINGWINDOW\_H

# Приложение В.

# Исходный код программы. Hashtable.h

#ifndef HASHTEBLE\_H

#define HASHTEBLE\_H

#include "allheaders.h"

namespace lrstruct {

template <typename Key>

class HashTable {

public:

/\* ----- CONTAINER FIELDS ----- \*/

/\*\*

\* There are two arguments from the readability perspective

\* to prefer using over typedef. First, using comes first when used.

\* Second, using feels quite similar to auto.

\* Additionally, using can easily be used for template aliases.

\*/

class Iterator;

// using is to call variable from class namespace without ::

using value\_type = Key;

// key\_type - the first template parameter (Key)

// In other words, it's just alias name

using key\_type = Key;

// declares a named variable as a reference,

// that is, an alias to an already-existing key

using reference = key\_type&;

// is the same as const Ty&

using const\_reference = const key\_type&;

using size\_type = size\_t;

using difference\_type = std::ptrdiff\_t;

using iterator = Iterator;

using const\_iterator = Iterator;

// works for anything that supports a less-than comparison

using key\_compare = std::less<key\_type>; // B+-Tree

// HASHING

using key\_equal = std::equal\_to<key\_type>;

/\*\*

\* Returns a value of type std::size\_t that represents

\* the hash value of the parameter.

\*/

using hasher = std::hash<key\_type>;

private:

size\_t hashKeySize = 7;

/\*\*

\* @brief The Node struct

\* for B+-Tree.

\*/

struct Node {

Key data;

Node \*next = nullptr;

Node \*head = nullptr;

~Node() {

delete head;

delete next;

}

};

Node \*table = nullptr;

size\_type maxSize{hashKeySize};

size\_type cSize{0}; // current Hash Table size

/\*\*

\* @brief insert\_unchecked

\* Insert an ellement without checking for collision

\* (for copy constructor mostly)

\* @param k

\*/

void insert\_unchecked(const\_reference k) {

Node \*help = new Node;

auto position = hash\_idx(k);

help->next = table[position].head;

help->data = k;

table[position].head = help;

++cSize;

if (maxSize \* 10 < cSize) rehash();

}

/\*\*

\* @brief hash\_idx

\* @param k

\* @return

\* load factor of current hash table

\*/

size\_type hash\_idx(const key\_type& k) const {

return hasher{}(k) % maxSize;

}

/\*\*

\* @brief rehash

\* The size of the array is increased (doubled) and all the values

\* are hashed again and stored in the new double sized array

\* to maintain a low load factor and low complexity

\*/

void rehash() {

std::vector<Key> dataBuff;

dataBuff.reserve(cSize);

for (const auto &k : \*this)

dataBuff.push\_back(k);

delete[] table;

maxSize \*= 2;

cSize = 0;

table = new Node[maxSize];

/\*

\* FIXME: for test with more then 71 element (rehashing happening)

\* it falls. Why??

for (const auto &fromBuff : dataBuff)

insert\_unchecked(fromBuff);

\*/

for (int i = 0; i < static\_cast<int>(dataBuff.size()); i++) {

insert\_unchecked(dataBuff[i]);

}

if(DEBUG) qDebug() << "table rehashed!" << endl;

}

public:

/\* ----- CONSTRUCTORS ----- \*/

HashTable() : table(new Node[hashKeySize]) {

if(DEBUG) qDebug() << "DEFAULT\_CONSTRUCTOR" << endl;

}

HashTable(std::initializer\_list<key\_type> ilist) : HashTable() {

insert(ilist);

}

/\*\*

\* Recursion call as foreach work

\*/

template<typename InputIt> HashTable(InputIt first, InputIt last) : table(new Node[hashKeySize]) {

insert(first, last);

if(DEBUG) qDebug() << "RANGE\_CONSTRUCTOR" << endl;

}

HashTable(const HashTable &other) : table(new Node[hashKeySize]) {

for (const auto &key : other)

insert\_unchecked(key);

if(DEBUG) qDebug() << "COPY\_CONSTRUCTOR" << endl;

}

~HashTable() {

delete[] table;

}

/\* ----- METHODS ----- \*/

size\_type size() {

return cSize;

}

std::vector<int> getChainsLength() {

std::vector<int> resChainsLength;

for (size\_type i{0}; i < maxSize; ++i) {

resChainsLength.push\_back(0);

for (Node\* a = table[i].head; a != nullptr; a = a->next) {

resChainsLength[i]++;

}

}

return resChainsLength;

}

size\_type getMaxSize() {

return maxSize;

}

bool empty() const {

if (cSize == 0)

return true;

return false;

}

size\_type count(const key\_type &key) const {

const auto row = hash\_idx(key); // hash the key to find apropriate row

for (Node \*node = table[row].head; node != nullptr; node = node->next)

if (key\_equal{}(node->data, key)) {

return 1;

}

return 0; // return 0 if key not found

}

iterator find(const key\_type& key) const {

const auto row = hash\_idx(key);

for (Node\* n = table[row].head; n != nullptr; n = n->next)

if (key\_equal{}(n->data, key)) {

return const\_iterator{ this, n, row, maxSize };

}

return end();

}

bool findEl(const key\_type& key) {

const auto row = hash\_idx(key);

for (Node\* n = table[row].head; n != nullptr; n = n->next) {

if (key\_equal{}(n->data, key)) {

return true;

}

}

return false;

}

void getPosOfEl(const key\_type &key, int &level, int &pos) {

const int row = hash\_idx(key);

pos = 0;

level = row;

for (Node\* n = table[row].head; n != nullptr; n = n->next) {

if (key\_equal{}(n->data, key)) {

return;

}

pos++;

}

}

void swap(HashTable& other) {

std::swap(maxSize, other.maxSize);

std::swap(table, other.table);

std::swap(cSize, other.cSize);

}

void insert(std::initializer\_list<key\_type> ilist) {

for (auto const &element : ilist) // iterate through the list

if (!count(element))

insert\_unchecked(element); // if element(key) already exists in table then skip

}

std::pair<iterator,bool> insert(const\_reference key) {

if (!count(key)) {

insert\_unchecked(key);

return std::make\_pair(find(key), true);

}

return std::make\_pair(find(key), false);

}

template<typename InputIt> void insert(InputIt first, InputIt last) {

for (auto it = first; it != last; ++it)

insert(\*it);

}

void clear() {

delete[] table;

cSize = 0;

maxSize = hashKeySize;

table = new Node[maxSize];

}

size\_type erase (const key\_type& key) {

if (count(key)) {

auto idx = hash\_idx(key);

Node \*current = table[idx].head;

Node \*previous = nullptr;

for ( ; current != nullptr; current = current->next) {

if (key\_equal{}(current->data, key)) {

if (current == table[idx].head) {

table[idx].head = current->next;

current->next = nullptr;

delete current;

--cSize;

return 1;

}

if (previous) {

previous->next = current->next;

current->next = nullptr;

delete current;

previous = nullptr;

delete previous;

--cSize;

return 1;

}

}

previous = current;

}

}

return 0;

}

/\* -------- ITERATORS --------\*/

const\_iterator begin() const {

for (size\_t i = 0; i < maxSize; i++)

if (table[i].head) {

return const\_iterator(this, table[i].head, i, maxSize);

}

return end();

}

const\_iterator end() const {

return const\_iterator(nullptr);

}

void dump(std::ostream& outStream = std::cerr) const {

for (size\_type i{0}; i < maxSize; ++i) {

if (table[i].head == nullptr) {

outStream << "[" << i << "]" << ": nullptr" << '\n';

continue;

}

outStream << "[" << i << "]" << ": ";

for (Node\* a = table[i].head; a != nullptr; a = a->next) {

outStream << a->data;

if (a->next != nullptr)

outStream << " -> ";

}

outStream << '\n';

}

}

void dump(std::string fileWay) const {

std::fstream fs;

for (size\_type i{0}; i < maxSize; ++i) {

fs.open (fileWay, std::fstream::in | std::fstream::out | std::fstream::app);

if (table[i].head == nullptr) {

fs << "[" << i << "]" << ": nullptr" << '\n';

continue;

}

fs << "[" << i << "]" << ": ";

for (Node\* a = table[i].head; a != nullptr; a = a->next) {

fs << a->data;

if (a->next != nullptr)

fs << " -> ";

fs.close();

fs.open (fileWay, std::fstream::in | std::fstream::out | std::fstream::app);

}

fs << '\n';

fs.close();

}

}

HashTable& operator = (const HashTable& other) {

clear();

for (const auto &key : other)

insert(key);

return \*this;

}

HashTable& operator = (std::initializer\_list<key\_type> ilist) {

clear();

insert(ilist);

return \*this;

}

friend bool operator == (const HashTable& lhs, const HashTable& rhs) {

if (lhs.cSize != rhs.cSize)

return false;

for (const auto &key : lhs) {

if (!rhs.count(key)) {

return false;

}

}

return true;

}

friend bool operator != (const HashTable& lhs, const HashTable& rhs) {

return !(lhs == rhs);

}

};

/\* -------- HASH TABLE ITERATORS --------\*/

template <typename Key>

class HashTable<Key>::Iterator {

private:

const HashTable<Key> \*ptr;

Node \*to;

size\_type itPos;

size\_type tblSz;

public:

using value\_type = Key;

using difference\_type = std::ptrdiff\_t;

using reference = const value\_type&;

using pointer = const value\_type\*;

using iterator\_category = std::forward\_iterator\_tag;

explicit Iterator(const HashTable \*ads = nullptr, Node \*tbl = nullptr,

size\_type idx = 0, size\_type sz = 0)

: ptr(ads), to(tbl), itPos(idx), tblSz(sz){

if(DEBUG) qDebug() << "ITERATOR\_CONSTRUCTOR" << endl;

}

reference operator\*() const {

return to->data;

}

Iterator& operator ++() {

while (itPos < tblSz) {

if (to->next) {

to = to->next; return \*this;

}

else ++itPos;

if (itPos == tblSz) {

to = nullptr; return \*this;

}

auto transit = ptr->table[itPos].head;

if (transit) {

to = transit;

return \*this;

}

}

return \*this;

}

Iterator operator ++ (int) {

Iterator tmp(\*this);

operator++();

return tmp;

}

friend bool operator == (const Iterator& lhs, const Iterator& rhs) {

return lhs.to == rhs.to;

}

friend bool operator != (const Iterator& lhs, const Iterator& rhs) {

return !(lhs.to == rhs.to);

}

};

template <typename Key, size\_t N> void swap(HashTable<Key>& lhs, HashTable<Key>& rhs) {

lhs.swap(rhs);

}

}

#endif // HASHTEBLE\_H

# Приложение Г.

# Исходный код программы. mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "allheaders.h"

#include "HashTable.h"

#include "drawingwindow.h"

#include <qtmaterialslider.h>

#include <qtmaterialradiobutton.h>

#include <qtmaterialtoggle.h>

#include <qtmaterialtextfield.h>

#include <qtmaterialscrollbar.h>

#include <qtmaterialflatbutton.h>

#include <qtmaterialdialog.h>

#include <qtmaterialcheckbox.h>

#include "lib/qtmaterialstyle.h"

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow();

public slots:

void onRunButtonClicked();

void onFileOpenButtonClicked();

void activateWindow();

private:

void setUpUI();

// WORK BUTTONS

QtMaterialFlatButton \*fileOpenButton;

QLabel \*fileWayLabel;

QtMaterialFlatButton \*runButton;

QtMaterialRadioButton \*stepByStepMode;

QtMaterialRadioButton \*allInMomentMode;

DrawingWindow \*drawingWindow;

std::vector<std::string> inputDataStr;

};

#endif // MAINWINDOW\_H

# Приложение Д.

# Исходный код программы. Tablerhandler.h

#ifndef TEST\_H

#define TEST\_H

#include "allheaders.h"

#include "HashTable.h"

class TableHandler : public QObject {

Q\_OBJECT

public:

static void makeHashTable(std::vector<std::string>, lrstruct::HashTable<std::string>\*&);

static void makeHashTableDump(const lrstruct::HashTable<std::string>\*);

static void separateNames(std::string&);

static std::string connectNames(std::string);

inline static bool stepLoopSwitcher;

static void loopLatency();

public slots:

static void disableLoop();

private:

static std::vector<std::string> convertStrToWords(std::string str);

};

#endif // TEST\_H

# Приложение Е.

# Исходный код программы. Drawing.cpp

#include "drawingwindow.h"

DrawingWindow::DrawingWindow() :

mainGraphicsView(new QGraphicsView),

mainGraphicsScene(new QGraphicsScene),

mainTextOutput(new QTextEdit),

showAllButton(new QtMaterialFlatButton),

inputEl(new QtMaterialTextField)

{

setUpUI();

}

DrawingWindow::DrawingWindow(std::vector<std::string> inputDatastr, std::vector<std::string> &inpStrs,\

bool startDrawMode = false) :

mainGraphicsView(new QGraphicsView),

mainGraphicsScene(new QGraphicsScene),

mainTextOutput(new QTextEdit),

nextStepButton(new QtMaterialFlatButton),

prevStepButton(new QtMaterialFlatButton),

addElButton(new QtMaterialFlatButton),

delElButton(new QtMaterialFlatButton),

showAllButton(new QtMaterialFlatButton),

inputEl(new QtMaterialTextField)

{

// setups

setUpUI();

setUpDrawingTools();

// functional work

TableHandler::makeHashTable(inputDatastr, workTable);

// FIXME: little trick to activate correct coords

mainGraphicsScene->addLine(0, 0, 1, 0);

levelsCount = static\_cast<int>(workTable->getMaxSize());

levelsLength = workTable->getChainsLength();

TableHandler::makeHashTableDump(workTable);

makeNodesRects(workTable, tableNodesRect);

this->inpStrs = inpStrs;

drawNodesStartGrid(levelsCount);

if(startDrawMode == false) {

drawBaseRects();

showAllButton->setDisabled(true);

reconectForAddDel();

}

else {

// default conditions

addElButton->setDisabled(true);

delElButton->setDisabled(true);

inputEl->setDisabled(true);

nextStepButton->setDisabled(false);

prevStepButton->setDisabled(false);

showAllButton->setDisabled(false);

TableHandler::stepLoopSwitcher = true;

connect(prevStepButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::baseDrawStepBack);

connect(nextStepButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::reconnectForStepByStep);

connect(showAllButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::showAllSteps);

}

}

void DrawingWindow::closeEvent(QCloseEvent \*event) {

emit closeSignal();

TableHandler::disableLoop();

event->accept();

}

void DrawingWindow::setUpDrawingTools() {

qDebug() << "setUpDrawingTools()" << endl;

color.setRgb(0, 0, 0);

pen.setColor(color);

brush.setColor(color);

brush.setStyle(Qt::SolidPattern);

font.setFamily("Roboto");

font.setPointSize(8);

pen.setWidth(3);

}

void DrawingWindow::reconnectForStepByStep() {

disconnect(nextStepButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::reconnectForStepByStep);

connect(nextStepButton, &QtMaterialFlatButton::clicked, this, &TableHandler::disableLoop);

drawRectsStepByStep();

}

void DrawingWindow::reconectForAddDel() {

addElButton->setDisabled(false);

delElButton->setDisabled(false);

inputEl->setDisabled(false);

nextStepButton->setDisabled(true);

prevStepButton->setDisabled(true);

showAllButton->setDisabled(true);

connect(addElButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::addEl);

connect(delElButton, &QtMaterialFlatButton::clicked, this, &DrawingWindow::delEl);

}

# Приложение Ж.

# Исходный код программы. functionsfordrawing.cpp

#include "drawingwindow.h"

#include "allheaders.h"

#include "TableHandler.h"

void DrawingWindow::drawNodesStartGrid(int levelsCount) {

for (int i = 0; i < levelsCount; i++) {

mainGraphicsScene->addRect(startDrawPos.x(), startDrawPos.y() + i \* (nodeHight + spaceBetNodes), \

nodeWidth, nodeHight, pen);

}

// nodes numbers

std::vector<QGraphicsTextItem\*> numbers;

numbers.reserve(static\_cast<unsigned long>(levelsCount));

for (unsigned long i{0}; i < static\_cast<unsigned long>(levelsCount); i++) {

numbers[i] = new QGraphicsTextItem;

numbers[i]->setPlainText(QString::number(i));

numbers[i]->setPos(startDrawPos.x() + nodeWidth / 4, \

static\_cast<unsigned long>(startDrawPos.y() + nodeHight) / 4 + \

i \* static\_cast<unsigned long>(nodeHight + spaceBetNodes));

numbers[i]->setFont(font);

numbers[i]->setDefaultTextColor(QColor::fromRgb(0, 0, 0));

mainGraphicsScene->addItem(numbers[i]);

}

mainTextOutput->setPlainText("Starting grid was generated!\nHash table key size is: " + \

QString::number(levelsCount) + "\n");

}

/\*\*

\* @brief DrawingWindow::makeNodesRects

\*

\* Recalc all nodes coords and positions

\*/

void DrawingWindow::makeNodesRects(lrstruct::HashTable<std::string> \*wTable, std::vector<nodeRect>& wTableNodeRect) {

nodeRect newRect;

int i = 0;

for(auto iter = workTable->begin(); iter != workTable->end(); ++iter) {

newRect.data = \*iter;

wTable->getPosOfEl(newRect.data, newRect.level, newRect.posInLevel);

newRect.geomPar.setX(startDrawPos.x() + nodeWidth + arrowLength + newRect.posInLevel \* (arrowLength + rectWidth));

newRect.geomPar.setY(startDrawPos.y() + newRect.level \* (nodeHight + spaceBetNodes) - (nodeHight - rectHieght) / 2);

newRect.geomPar.setHeight(rectHieght);

newRect.geomPar.setWidth(rectWidth);

i++;

wTableNodeRect.push\_back(newRect);

}

}

void DrawingWindow::drawBaseRects() {

for(const auto& rect : tableNodesRect) {

mainGraphicsScene->addRect(rect.geomPar, pen);

}

QPen arrowPen(color, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

int arrowX{0}, arrowY{0};

QPolygon polygon; // Using Polygon class, to draw the triangle

// data out

std::vector<QGraphicsTextItem\*> strings;

strings.reserve(tableNodesRect.size());

for (unsigned long i = 0; i < tableNodesRect.size(); i++) {

polygon.clear();

TableHandler::separateNames(tableNodesRect[i].data);

strings[i] = new QGraphicsTextItem;

strings[i]->setPlainText(QString::fromUtf8(tableNodesRect[i].data.c\_str()));

strings[i]->setFont(font);

strings[i]->setDefaultTextColor(QColor::fromRgb(0, 0, 0));

strings[i]->setPos(startDrawPos.x() + nodeWidth + arrowLength + \

tableNodesRect[i].posInLevel \* (rectWidth + arrowLength), \

startDrawPos.y() + \

tableNodesRect[i].level \* (rectHieght + spaceBetNodes));

arrowX = startDrawPos.x() + nodeWidth + tableNodesRect[i].posInLevel \* (rectWidth + arrowLength);

arrowY = startDrawPos.y() + nodeHight / 2 + tableNodesRect[i].level \* (rectHieght + spaceBetNodes);

mainGraphicsScene->addLine(arrowX, arrowY, arrowX + arrowLength, arrowY, arrowPen);

// arrow cup drawing

polygon << QPoint(arrowX + arrowLength - 1, arrowY) << QPoint(arrowX + arrowLength - 1 - arrowTriangleLength, arrowY + arrowTriangleHight) << \

QPoint(arrowX + arrowLength - 1 - arrowTriangleLength, arrowY - arrowTriangleHight);

mainGraphicsScene->addPolygon(polygon, pen, brush);

mainGraphicsScene->addItem(strings[i]);

}

mainTextOutput->setPlainText("All steps was shown.\nIt's full hash table.\nSize is " + \

QString::number(inpStrs.size()) + " elements.");

reconectForAddDel();

}

void DrawingWindow::drawRect(nodeRect& rectNode, QPen pen = QPen(), QBrush brush = QBrush()) {

mainGraphicsScene->addRect(rectNode.geomPar, pen);

QPen arrowPen(color, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin);

int arrowX{0}, arrowY{0};

QPolygon polygon; // Using Polygon class, to draw the triangle

TableHandler::separateNames(rectNode.data);

QGraphicsTextItem\* text = new QGraphicsTextItem;

text = new QGraphicsTextItem;

text->setPlainText(QString::fromUtf8(rectNode.data.c\_str()));

text->setFont(font);

text->setDefaultTextColor(pen.color());

text->setPos(startDrawPos.x() + nodeWidth + arrowLength + \

rectNode.posInLevel \* (rectWidth + arrowLength), \

startDrawPos.y() + \

rectNode.level \* (rectHieght + spaceBetNodes));

arrowX = startDrawPos.x() + nodeWidth + rectNode.posInLevel \* (rectWidth + arrowLength);

arrowY = startDrawPos.y() + nodeHight / 2 + rectNode.level \* (rectHieght + spaceBetNodes);

mainGraphicsScene->addLine(arrowX, arrowY, arrowX + arrowLength, arrowY, arrowPen);

// arrow cup drawing

polygon << QPoint(arrowX + arrowLength - 1, arrowY) << QPoint(arrowX + arrowLength - 1 - arrowTriangleLength, arrowY + arrowTriangleHight) << \

QPoint(arrowX + arrowLength - 1 - arrowTriangleLength, arrowY - arrowTriangleHight);

mainGraphicsScene->addPolygon(polygon, pen, brush);

mainGraphicsScene->addItem(text);

}

void DrawingWindow::drawRectsStepByStep() {

std::vector<nodeRect> currentNodeRects;

std::vector<nodeRect> firstEmptyEl;

tableBackUp.push\_back(firstEmptyEl);

// making backup

for(stepsCount = static\_cast<int>(inpStrs.size() - 1); stepsCount >= 0; stepsCount--) {

for(auto& rect : tableNodesRect) {

if(TableHandler::connectNames(rect.data) == inpStrs[static\_cast<unsigned long>(stepsCount)]) {

currentNodeRects.push\_back(rect);

tableBackUp.push\_back(currentNodeRects);

break;

}

}

}

if(DEBUG) {

qDebug() << "Backup: " << endl;

for(auto& rectH : tableBackUp) {

for(auto& rect : rectH) {

qDebug() << QString::fromUtf8(rect.data.c\_str());

}

qDebug() << endl;

}

}

for(stepsCount = static\_cast<int>(inpStrs.size() - 1); stepsCount >= 0; stepsCount--) {

for(auto& rect : tableNodesRect) {

if(TableHandler::connectNames(rect.data) == inpStrs[static\_cast<unsigned long>(stepsCount)]) {

if(DEBUG)qDebug() << "Next step: " << stepsCount << ", " << QString::fromUtf8(TableHandler::connectNames(rect.data).c\_str()) << endl;

mainTextOutput->setPlainText("Next step for: " + QString::fromUtf8(TableHandler::connectNames(rect.data).c\_str()));

drawRect(rect, pen, brush);

TableHandler::loopLatency();

break;

}

}

}

mainTextOutput->setPlainText("Full hash table was drawn!\nStep button reconnected!");

reconectForAddDel();

}

void DrawingWindow::baseDrawStepBack() {

if(stepsCount <= static\_cast<int>(tableBackUp.size())) {

for(auto& rect : tableNodesRect) {

if(TableHandler::connectNames(rect.data) == inpStrs[static\_cast<unsigned long>(stepsCount)]) {

if(DEBUG)qDebug() << "Back step: " << stepsCount << ", " << QString::fromUtf8(TableHandler::connectNames(rect.data).c\_str()) << endl;

if(!tableBackUp.empty()) {

stepsCount++;

mainGraphicsScene->clear();

drawNodesStartGrid(levelsCount);

std::vector<nodeRect> prevRectDraw;

prevRectDraw = tableBackUp[tableBackUp.size() - static\_cast<unsigned long>(stepsCount) - 1];

for(auto& rect : prevRectDraw) {

drawRect(rect, pen, brush);

}

}

mainTextOutput->setPlainText("Back step for: " + QString::fromUtf8(TableHandler::connectNames(rect.data).c\_str()));

break;

}

}

}

}

void DrawingWindow::showAllSteps() {

TableHandler::disableLoop();

mainGraphicsScene->clear();

drawNodesStartGrid(levelsCount);

drawBaseRects();

nextStepButton->disconnect();

prevStepButton->disconnect();

reconectForAddDel();

}

void DrawingWindow::addEl() {

std::string inputStr = inputEl->text().toStdString();

int level, posInLevel;

if(inputStr.empty()) {

mainTextOutput->setPlainText("Please, enter the string for adding.");

return;

}

if(workTable->findEl(inputStr) == true) {

workTable->getPosOfEl(inputStr, level, posInLevel);

mainTextOutput->setPlainText("Element already exists!\nPosition: level = " \

+ QString::number(level) + ", pos = " + QString::number(posInLevel));

return;

}

workTable->insert(inputStr);

workTable->getPosOfEl(inputStr, level, posInLevel);

tableNodesRect.clear();

makeNodesRects(workTable, tableNodesRect);

mainGraphicsScene->clear();

drawNodesStartGrid(levelsCount);

drawBaseRects();

mainTextOutput->setPlainText("Element: " + QString::fromUtf8(inputStr.c\_str()) + " was added.\n" + \

"Position: level = " + QString::number(level) + ", pos = " + QString::number(posInLevel));

}

void DrawingWindow::delEl() {

std::string inputStr = inputEl->text().toStdString();

int level, posInLevel;

if(inputStr.empty()) {

mainTextOutput->setPlainText("Please, enter the string for adding.");

return;

}

workTable->getPosOfEl(inputStr, level, posInLevel);

workTable->erase(inputStr);

tableNodesRect.clear();

makeNodesRects(workTable, tableNodesRect);

mainGraphicsScene->clear();

drawNodesStartGrid(levelsCount);

drawBaseRects();

mainTextOutput->setPlainText("Element: " + QString::fromUtf8(inputStr.c\_str()) + " was deleted.\nIt had position: " + \

QString::number(level) + ", pos = " + QString::number(posInLevel));

}

# Приложение И.

# Исходный код программы. mainwindow.cpp

#include "mainwindow.h"

#include "allheaders.h"

#include "TableHandler.h"

#include "HashTable.h"

#include "drawingwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

fileOpenButton(new QtMaterialFlatButton),

fileWayLabel(new QLabel),

runButton(new QtMaterialFlatButton),

stepByStepMode(new QtMaterialRadioButton),

allInMomentMode(new QtMaterialRadioButton)

{

setUpUI();

}

MainWindow::~MainWindow() {

}

void MainWindow::onFileOpenButtonClicked() {

qDebug() << "onFileOpenlicked();" << endl;

fileWayLabel->clear();

QString fileName = QFileDialog::getOpenFileName(this,

tr("Open TXT File"), QDir::homePath(),

tr("TXT text (\*.txt);;All Files (\*)"));

if (fileName == nullptr) {

qDebug() << "No file was choosen!" << endl;

fileWayLabel->setText("No file was choosen!");

return;

}

QFile file(fileName);

if(file.open(QIODevice::ReadOnly | QIODevice::Text)) {

QTextStream stream(&file);

foreach (QString i, QString(stream.readAll()).split(QRegExp("[\n]"), \

QString::SkipEmptyParts))

inputDataStr.push\_back(i.toUtf8().constData());

}

fileWayLabel->setText(fileName);

// working function activate

runButton->setDisabled(false);

file.close();

qDebug() << "End of onFileOpenlicked()" << endl;

}

void MainWindow::onRunButtonClicked() {

qDebug() << "onRunButtonClicked();" << endl;

if(inputDataStr.empty()) {

fileWayLabel->setText("No input data!");

return;

}

drawingWindow = new DrawingWindow(inputDataStr, inputDataStr, stepByStepMode->isChecked() == true);

connect(drawingWindow, &DrawingWindow::closeSignal, this, &MainWindow::activateWindow);

drawingWindow->show();

setDisabled(true);

}

void MainWindow::activateWindow() {

this->setDisabled(false);

}

# Приложение К.

# Исходный код программы. setupui.cpp

#include "mainwindow.h"

#include "allheaders.h"

#include "TableHandler.h"

#include "drawingwindow.h"

void MainWindow::setUpUI() {

qDebug() << "main window UI set up" << endl;

setWindowTitle(tr("Course work"));

// working zone - main widget

QWidget \*canvas = new QWidget;

QVBoxLayout \*layout = new QVBoxLayout;

// set main vertical layout

canvas->setLayout(layout);

// sets the stretch factor at position index

// and aligning of main widget

layout->setStretch(1, 2);

layout->setAlignment(Qt::AlignCenter);

// set main widget of window

canvas->setStyleSheet("QWidget { background: white; }");

setCentralWidget(canvas);

// set up buttons

runButton->setText("Run");

runButton->setToolTip("Start hash table visualization");

fileOpenButton->setText("Open file");

fileOpenButton->setToolTip("File with strings (names) for hash table");

fileOpenButton->setIcon(QtMaterialTheme::icon("file", "folder\_open"));

fileOpenButton->setIconSize(QSize(35, 35));

fileWayLabel->setStyleSheet("QLabel {color : black; }");

fileWayLabel->setText("Please, open the file");

stepByStepMode->setText("Step by step building");

stepByStepMode->setToolTip("New element insert - one step");

allInMomentMode->setText("Automate building");

allInMomentMode->setToolTip("Full table will be built in a moment");

QVBoxLayout \*modeChooseLayout = new QVBoxLayout;

modeChooseLayout->addWidget(stepByStepMode);

modeChooseLayout->addWidget(allInMomentMode);

QVBoxLayout \*fileRunLayout = new QVBoxLayout;

fileRunLayout->addWidget(fileOpenButton);

fileRunLayout->addWidget(fileWayLabel);

fileRunLayout->addWidget(runButton);

QGridLayout \*mainLayout = new QGridLayout;

mainLayout->addLayout(fileRunLayout, 0, 0, 3, 1, Qt::AlignLeft);

mainLayout->addLayout(modeChooseLayout, 0, 2, 2, 3, Qt::AlignRight);

layout->addLayout(mainLayout);

// set up minimum window size

int height = mainLayout->geometry().height();

if (height != minimumHeight() || height != maximumHeight()) {

setMinimumHeight(height);

setMaximumHeight(height);

}

int width = mainLayout->geometry().width();

if (width != minimumWidth() || height != maximumWidth()) {

setMinimumWidth(width);

setMaximumWidth(width);

}

// default conditions

runButton->setDisabled(true);

/\* SIGNALS AND SLOTS CONNECTIONS \*/

connect(runButton, &QtMaterialFlatButton::clicked, this, &MainWindow::onRunButtonClicked);

connect(fileOpenButton, &QtMaterialFlatButton::clicked, this, &MainWindow::onFileOpenButtonClicked);

}

void DrawingWindow::setUpUI() {

qDebug() << "drawing window UI set up" << endl;

setWindowTitle(tr("Hash Table Visualization"));

mainGraphicsView->setAlignment(Qt::AlignTop | Qt::AlignLeft);

mainGraphicsView->setScene(mainGraphicsScene);

mainGraphicsView->setRenderHints(QPainter::Antialiasing);

QScreen \*screen = QGuiApplication::primaryScreen();

QRect screenGeometry = screen->geometry();

int height = screenGeometry.height();

int width = screenGeometry.width();

this->resize(static\_cast<int>(width), static\_cast<int>(height));

QVBoxLayout \*layout = new QVBoxLayout;

// set main vertical layout

setLayout(layout);

// sets the stretch factor at position index

// and aligning of main widget

layout->setStretch(1, 2);

layout->setAlignment(Qt::AlignCenter);

setStyleSheet("QWidget { background: white; }");

// init and set up buttons

nextStepButton->setIcon(QtMaterialTheme::icon("navigation", "arrow\_forward"));

nextStepButton->setToolTip("Next step");

nextStepButton->setIconSize(QSize(35, 35));

prevStepButton->setIcon(QtMaterialTheme::icon("navigation", "arrow\_back"));

prevStepButton->setToolTip("Step back");

prevStepButton->setIconSize(QSize(35, 35));

addElButton->setIcon(QtMaterialTheme::icon("content", "add"));

addElButton->setToolTip("Add new string in hash table");

addElButton->setIconSize(QSize(35, 35));

delElButton->setIcon(QtMaterialTheme::icon("content", "delete\_sweep"));

delElButton->setIconSize(QSize(35, 35));

delElButton->setToolTip("Delete string from hash table");

showAllButton->setText("Show all");

showAllButton->setToolTip("Stop step by step mode and show everything");

inputEl->setLabel("Enter the string for adding / deleting");

inputEl->setTextColor(QColor::fromRgb(0, 0, 0));

inputEl->setMinimumWidth(200);

inputEl->setMaximumWidth(300);

mainTextOutput->setReadOnly(true);

mainTextOutput->setAlignment(Qt::AlignCenter);

mainTextOutput->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

mainTextOutput->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

mainTextOutput->setTextColor(QColor::fromRgb(0, 0, 0));

QHBoxLayout \*stepsLayout = new QHBoxLayout;

stepsLayout->addWidget(prevStepButton);

stepsLayout->addWidget(nextStepButton);

stepsLayout->addWidget(showAllButton);

QHBoxLayout \*elManipLayout = new QHBoxLayout;

elManipLayout->addWidget(inputEl);

elManipLayout->addWidget(addElButton);

elManipLayout->addWidget(delElButton);

QHBoxLayout \*allButtonLayout = new QHBoxLayout;

QSpacerItem \*centralSpacer = new QSpacerItem(1500, 50, QSizePolicy::Maximum, QSizePolicy::Minimum);

allButtonLayout->addLayout(stepsLayout);

allButtonLayout->addSpacerItem(centralSpacer);

allButtonLayout->addLayout(elManipLayout);

QHBoxLayout \*textOutLayout = new QHBoxLayout;

QSpacerItem \*leftTextSpacer = new QSpacerItem(200, 50, QSizePolicy::Preferred, QSizePolicy::Minimum);

QSpacerItem \*rightTextSpacer = new QSpacerItem(200, 50, QSizePolicy::Preferred, QSizePolicy::Minimum);

textOutLayout->addSpacerItem(leftTextSpacer);

textOutLayout->addWidget(mainTextOutput);

textOutLayout->addSpacerItem(rightTextSpacer);

mainTextOutput->setMaximumHeight(50);

mainTextOutput->setMaximumWidth(200);

mainGraphicsView->setMinimumHeight(200);

layout->addLayout(allButtonLayout);

layout->addWidget(mainGraphicsView);

layout->addLayout(textOutLayout);

}

# Приложение Л.

# Исходный код программы. setupui.cpp

#include "allheaders.h"

#include "HashTable.h"

#include "TableHandler.h"

std::vector<std::string> convertStrToWords(std::string str) {

// Used to split string around spaces.

std::istringstream ss(str);

std::vector<std::string> res;

// Traverse through all words

do {

// Read a word

std::string word;

ss >> word;

res.push\_back(word);

} while (ss); // While there is more to read

return res;

}

void TableHandler::makeHashTable(std::vector<std::string> inpStrs, lrstruct::HashTable<std::string>\*& workTable) {

workTable = new lrstruct::HashTable<std::string>();

/\*

\* FIXME: out of range error

for(auto const& value: inpStrs) {

workTable.insert(value);

}

\*/

for(size\_t i = 0; i < inpStrs.size(); i++) {

workTable->insert(inpStrs.at(i));

}

}

void TableHandler::makeHashTableDump(const lrstruct::HashTable<std::string> \*workTable) {

std::fstream fs;

fs.open (LOG\_FILE\_WAY, std::ios::out | std::ios::trunc);

workTable->dump(fs);

fs.close();

}

void TableHandler::separateNames(std::string &data) {

// separate name and surname for tests

size\_t spaceFindPose;

for(size\_t i = 0; i < data.size(); i++) {

spaceFindPose = data.find(" ");

if (spaceFindPose != std::string::npos)

data.replace(spaceFindPose, 1, "\n");

}

}

std::string TableHandler::connectNames(std::string data) {

size\_t spaceFindPose;

std::string res = data;

for(size\_t i = 0; i < res.size(); i++) {

spaceFindPose = res.find("\n");

if (spaceFindPose != std::string::npos)

res.replace(spaceFindPose, 1, " ");

}

return res;

}

void TableHandler::loopLatency() {

qDebug() << "Loop latency for step-by-step started" << endl;

for( ; ; ) {

QApplication::processEvents();

if(stepLoopSwitcher == false) break;

}

stepLoopSwitcher = true;

}

void TableHandler::disableLoop() {

stepLoopSwitcher = false;

}

# Приложение М.

# Исходный код программы. tablehandler.cpp

#include "allheaders.h"

#include "HashTable.h"

#include "TableHandler.h"

std::vector<std::string> convertStrToWords(std::string str) {

// Used to split string around spaces.

std::istringstream ss(str);

std::vector<std::string> res;

// Traverse through all words

do {

// Read a word

std::string word;

ss >> word;

res.push\_back(word);

} while (ss); // While there is more to read

return res;

}

void TableHandler::makeHashTable(std::vector<std::string> inpStrs, lrstruct::HashTable<std::string>\*& workTable) {

workTable = new lrstruct::HashTable<std::string>();

/\*

\* FIXME: out of range error

for(auto const& value: inpStrs) {

workTable.insert(value);

}

\*/

for(size\_t i = 0; i < inpStrs.size(); i++) {

workTable->insert(inpStrs.at(i));

}

}

void TableHandler::makeHashTableDump(const lrstruct::HashTable<std::string> \*workTable) {

std::fstream fs;

fs.open (LOG\_FILE\_WAY, std::ios::out | std::ios::trunc);

workTable->dump(fs);

fs.close();

}

void TableHandler::separateNames(std::string &data) {

// separate name and surname for tests

size\_t spaceFindPose;

for(size\_t i = 0; i < data.size(); i++) {

spaceFindPose = data.find(" ");

if (spaceFindPose != std::string::npos)

data.replace(spaceFindPose, 1, "\n");

}

}

std::string TableHandler::connectNames(std::string data) {

size\_t spaceFindPose;

std::string res = data;

for(size\_t i = 0; i < res.size(); i++) {

spaceFindPose = res.find("\n");

if (spaceFindPose != std::string::npos)

res.replace(spaceFindPose, 1, " ");

}

return res;

}

void TableHandler::loopLatency() {

qDebug() << "Loop latency for step-by-step started" << endl;

for( ; ; ) {

QApplication::processEvents();

if(stepLoopSwitcher == false) break;

}

stepLoopSwitcher = true;

}

void TableHandler::disableLoop() {

stepLoopSwitcher = false;

}

# Приложение Н.

# Исходный код программы. Allheaders.h

#ifndef BASICHEADERS\_H

#define BASICHEADERS\_H

#include <iostream>

#include <vector>

#include <queue>

#include <map>

#include <fstream>

#include <algorithm>

#include <memory>

#include <cstdint>

#include <cstring>

#include <string>

#include <cstdlib>

#include <unistd.h>

#include <exception>

#include <cstdio>

#include <cassert>

#include <regex>

#include <cmath>

#include <unistd.h>

#include <string>

#include <initializer\_list>

#include <stdexcept>

#include <functional>

#include <sstream>

#include <type\_traits>

#include <QObject>

#include <QMessageBox>

#include <QDebug>

#include <QString>

#include <QFileDialog>

#include <QGraphicsItem>

#include <QtGui>

#include <QDialog>

#include <QColorDialog>

#include <QString>

#include <QDebug>

#include <QPainter>

#include <QComboBox>

#include <QLabel>

#include <QPushButton>

#include <QFile>

#include <QWidget>

#include <QVBoxLayout>

#include <QPushButton>

#include <QLabel>

#include <QLineEdit>

#include <QGroupBox>

#include <QRadioButton>

#include <QTextEdit>

#include <QEventLoop>

#include <QTimer>

#include <QColor>

#include <QDebug>

#include <QGraphicsView>

#include <QFormLayout>

#include <QApplication>

#include <QSignalMapper>

#include <QStackedLayout>

#include <QMouseEvent>

#include <QCloseEvent>

#include <QRectF>

#define LOG\_FILE\_WAY "../../log.txt"

#define DEBUG 1

#endif // BASICHEADERS\_H