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**

НА КУРСОВУЮ РАБОТУ **2**

АННОТАЦИЯ **3**

SUMMARY **3**

СОДЕРЖАНИЕ **4**

ВВЕДЕНИЕ **6**

Цель работы 6

Основные задачи 6

Методы решения 6

1. ЗАДАНИЕ **7**

2. ОПИСАНИЕ ПРОГРАММЫ **8**

2.1. Описание интерфейса пользователя 8

2.3. Описание основных полей и методов класса графического отображения 9

2.4 Описание методов базового класса TableHandler 12

3. ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ РАБОТЫ **14**

3.1 Общее описание работы хеш-таблицы с цепочками (хеширование) 14

3.2 Реализация методов вставки и удаления элемента 16

4. ДЕМОНСТРАЦИЯ **17**

4.1 Окна интерфейса программы 17

4.2 Результаты графической обработки таблицы 17

ЗАКЛЮЧЕНИЕ **20**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **21**

ПРИЛОЖЕНИЕ А. **22**

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.C **22**

ПРИЛОЖЕНИЕ Б. **23**

ИСХОДНЫЙ КОД ПРОГРАММЫ. DRAWING.H **23**

ПРИЛОЖЕНИЕ В. **26**

ИСХОДНЫЙ КОД ПРОГРАММЫ. HASHTABLE.H **26**

ПРИЛОЖЕНИЕ Г. **36**

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H **36**

ПРИЛОЖЕНИЕ Д. 37

ИСХОДНЫЙ КОД ПРОГРАММЫ. TABLERHANDLER.H 37

ПРИЛОЖЕНИЕ Е. 38

ИСХОДНЫЙ КОД ПРОГРАММЫ. DRAWING.CPP 38

ПРИЛОЖЕНИЕ Ж. 41

ИСХОДНЫЙ КОД ПРОГРАММЫ. FUNCTIONSFORDRAWING.CPP 41

ПРИЛОЖЕНИЕ И. 47

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP 47

ПРИЛОЖЕНИЕ К. 49

ИСХОДНЫЙ КОД ПРОГРАММЫ. SETUPUI.CPP 49

ПРИЛОЖЕНИЕ Л. 53

ИСХОДНЫЙ КОД ПРОГРАММЫ. SETUPUI.CPP 53

ПРИЛОЖЕНИЕ М. 55

ИСХОДНЫЙ КОД ПРОГРАММЫ. TABLEHANDLER.CPP 55

ПРИЛОЖЕНИЕ Н. 57

ИСХОДНЫЙ КОД ПРОГРАММЫ. ALLHEADERS.H 57

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.

1. 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 $O(1)$. 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 i массива H содержит указатель на начало списка всех элементов, хеш-код которых равен i, 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 $O(1)$. 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 $O(n)$.

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

Удаления элемента может быть выполнено за $O(1)$, 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 $O(1)$, где l - длина списка, то с некоторого момента выгодно увеличить размер хеш-таблицы, чтобы поддерживать амортизационную стоимость операции $O(1)$.

Рассмотрим следующий алгоритм перехеширование: когда в хеш-таблицу добавлено $\frac{4}{3}$ элементов, где n - размер хеш-таблицы, создадим новую хеш-таблицу размера $2n$, и последовательно переместим в нее все элементы первой таблицы. При этом, сменим хеш-функцию так, чтобы она выдавала значения $[0..2n-1]$.

Найдем амортизационную стоимость добавления, после которого было сделано перехеширование, используя метод предоплаты. С момента последнего перехеширования было произведено не менее $\frac{2n}{3}$ операций добавления, так как изначально в массиве находится $\frac{2n}{3}$ элементов (или 0 в начале работы), а перехеширование происходит при наличии $\frac{4n}{3}$ элементов.

Для проведения перехеширования необходимо произвести $\frac{4n}{3}$ операций добавления, средняя стоимость которых составляет $O(1)$, потратить $\frac{4n}{3}$ 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 $O(1)$, операция перехеширования будет полностью предоплачена. Значит,

амортизационная стоимость перехеширования при открытом типе хеш-таблицы равна $O(1)$.

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.

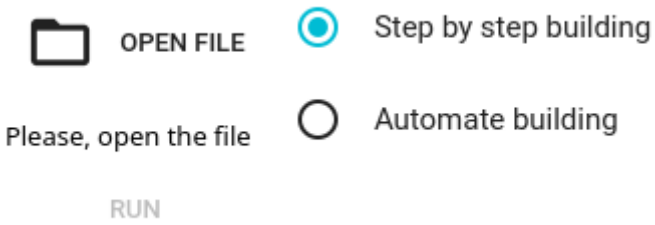

Рисунок 1 – стартовое меню программы

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.

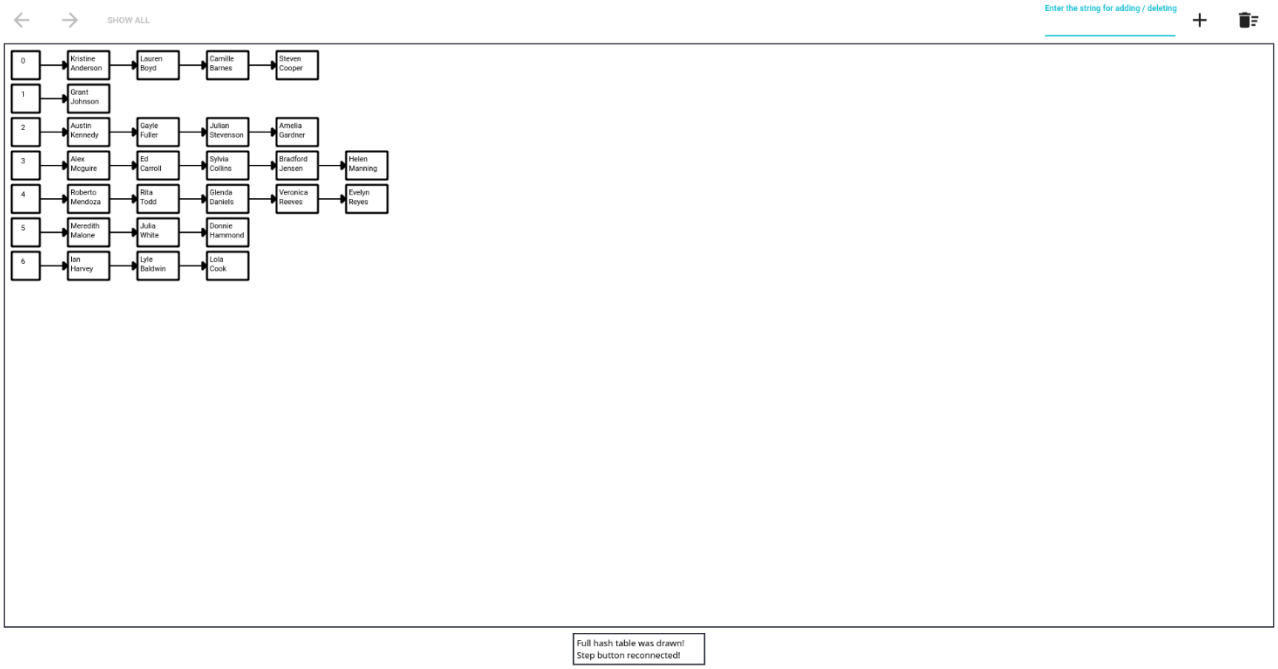

Рисунок 3 – полностью построенная хеш-таблица для 25 значений

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

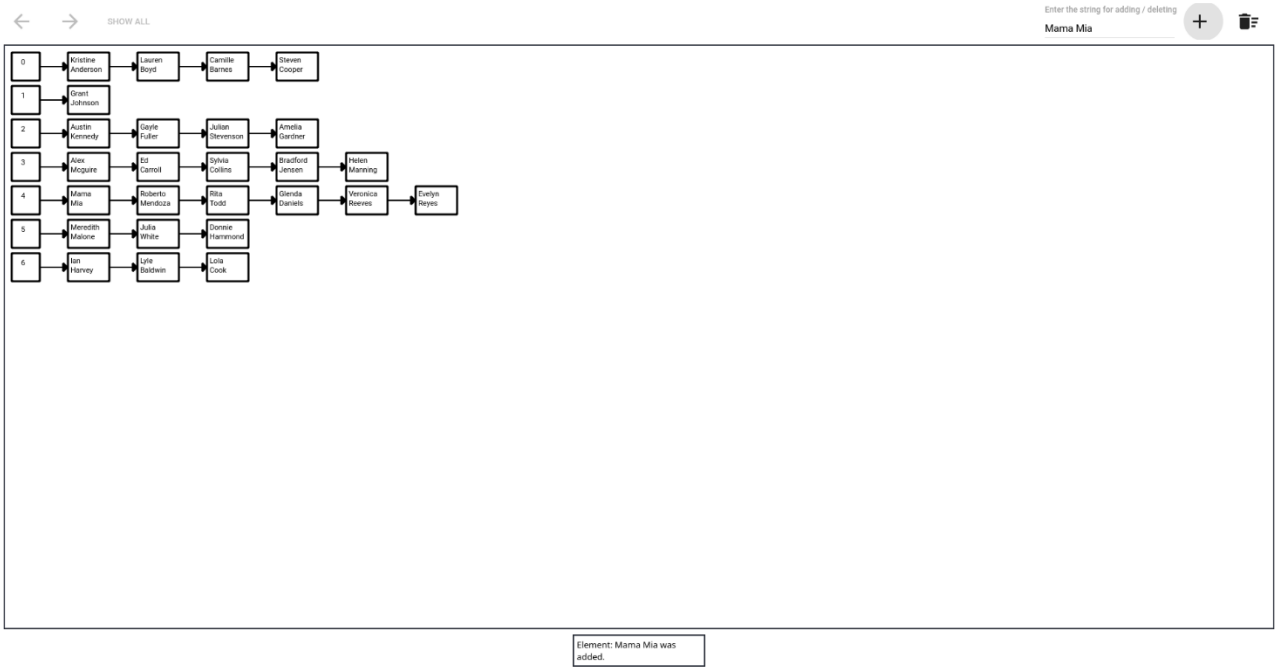

Рисунок 4 – результат добавления в таблицу элемента Mama Mia

На рис. 5 представлен результат добавленного элемента из таблицы.

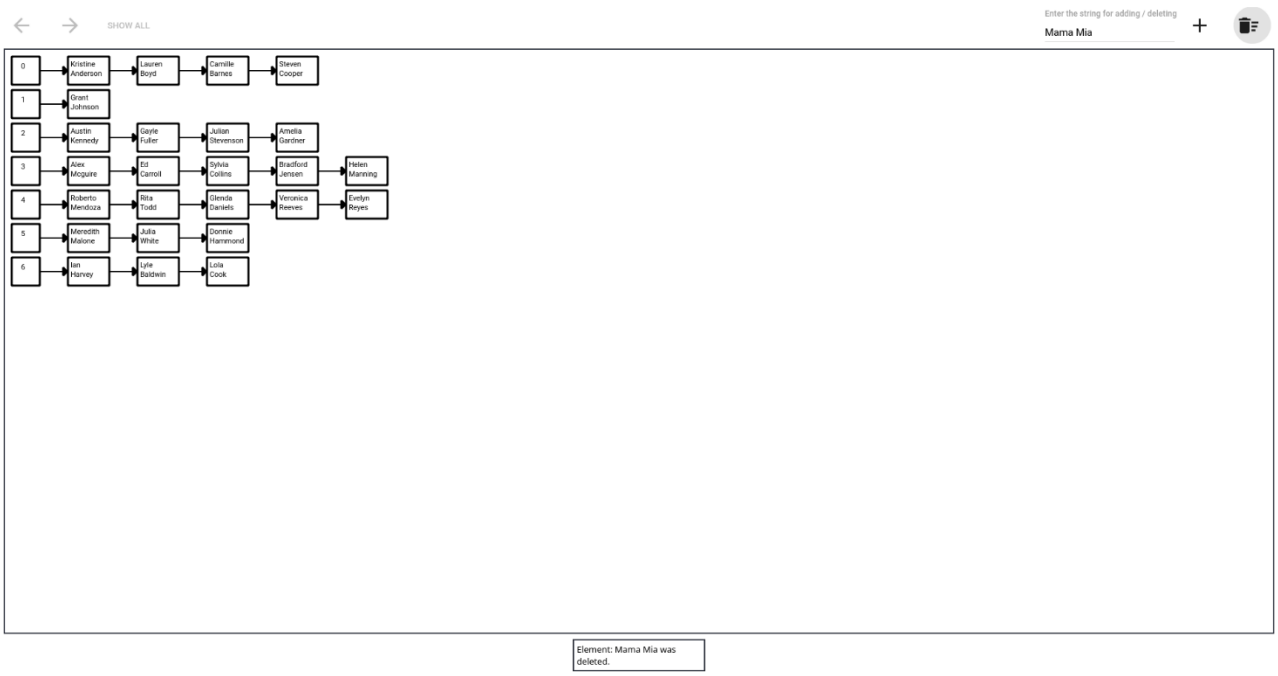

Рисунок 5 – результат удаления элемента Mama Mia

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

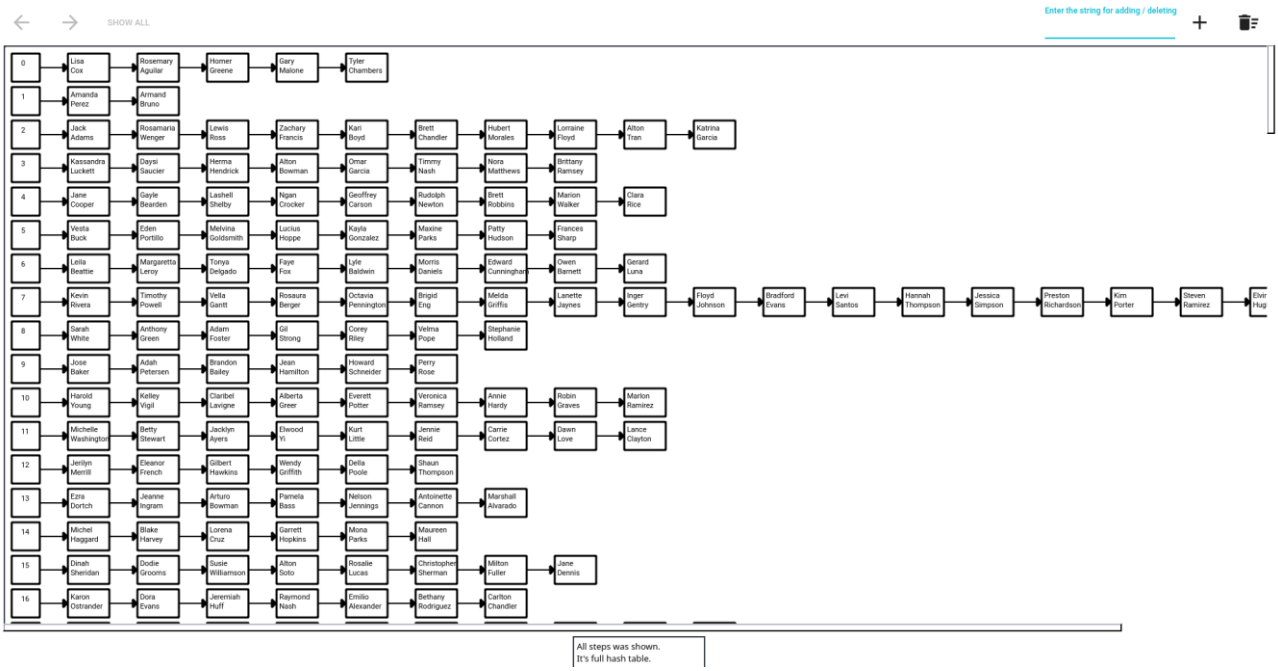

Рисунок 6 – таблица для 1000 элементов

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++. M.: Addison-Wesley, 2018. 217 с.
2. Макс Шлее. Qt5.10. Профессиональное программирование на C++. М.: BHV-СПб, 2018, 513 с.
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
```