МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**ОТЧЕТ**

по учебной практике

**Таблицы**

**Выполнил:** студент группы 381906-3  
Захаров А. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород

2021

**Содержание**

[**Введение** 3](#_Toc72772276)

[**1** **Постановка задачи** 4](#_Toc72772277)

[**2** **Руководство пользователя** 4](#_Toc72772278)

[**3** **Руководство программиста** 4](#_Toc72772279)

[**3.1** **Описание структуры программы** 4](#_Toc72772280)

[**3.2** **Описание структуры данных** 4](#_Toc72772281)

[**3.3** **Описание программной реализации** 6](#_Toc72772282)

[**Заключение** 10](#_Toc72772283)

[**Список литературы** 11](#_Toc72772284)

# **Введение**

Данная работа направлена на изучение способов организации таблиц и начальное знакомство с принципами проектирования структуры хранения, используемых в методах решения прикладных задач. Эта программа должна выполнять стандартные для любой таблицы действия: поиск, вставка, удаление. Основная цель лабораторной работы – это понять в каких ситуациях хороша каждая конкретная таблица, где выгоднее по скорости использовать сортированную, а где хэш и т.д. В ходе выполнения задания будут написаны четыре вида таблицы: обычная, сортированная, на дереве, хэш, а также их методы.

# **Постановка задачи**

*Цель:*

Разработать и реализовать программу, в которой будут хранится четыре вида таблицы и их методы. Каждая из них может как минимум искать, добавлять и удалять элемент. Таблицы – это последовательность строк (записей), структура строк может быть различной, но обязательным является поле, задающее имя (ключ) записи. Обычная таблица является набором не структурированных строк. Сортированная же, как следует из названия, является сортированным набором строк. На дереве – это структурированный особым образом набор строк. Хэш тоже самое, что и на дереве. В данной работе обычная, сортированная, а также хэш таблица имеют свободную адресацию. Для написания программы используется язык C++.

# **Руководство пользователя**

В данной программе нет интерфейса, поэтому обычному пользователю будет невозможно ею воспользоваться.

# **Руководство программиста**

## **Описание структуры программы**

Программа состоит из одного проекта.

**Tables** – консольное приложение.

Source.cpp – файл, который содержит функцию main().

fileWorker.h – заголовочный файл, который содержит реализацию работы с .txt файлом.

AbstractTable.h – заголовочный файл, в котором реализован абстрактный класс Table.

ScanTable.h - заголовочный файл, который содержит реализацию обычной таблицы.

SortTable.h – заголовочный файл, который содержит реализацию сортированной таблицы.

TreeTable.h – заголовочный файл, который содержит реализацию таблицы на дереве.

HashTable.h – заголовочный файл, который содержит реализацию хэш-таблицы.

## **Описание структуры данных**

В программе используются тип данных string из стандартной библиотеки, а также стек, вектор, очередь и лист также из стандартных библиотек.

Ещё была создана структура узел (node) для работы с деревом.

Наследование происходит таким образом: от абстрактной таблицы наследуется обычная, а все остальные наследуются от обычной.

**Алгоритмы:**

1. *AbstractTable.h*

Конструктор – использует функцию из заголовочного файла fileWorker.h, чтобы посимвольно считать из .txt файла текст и построчно сохранить его в массиве.

1. *ScanTable.h*

Конструктор – строковой массив из абстрактной таблицы преобразует в двойной строчный массив наподобие exel таблицы. Строки – люди, колонки - предметы.

Деструктор – сохраняет двойной массив в .txt файл с помощью функции из fileWorker.h, а затем удаляет массив.

Inserter – данный метод принимает строку и ключ. Ключ отвечает за то, куда пойдет строка: в колонну или в линию. После этого проверяет один из двух стеков на наличие в них элементов. Эти элементы являются индексами удаленных строк иди колонок из таблицы. В случае наличия – записывается на место удаленной строки или колонки. Если стек пустой, то расширяется массив и закидывается в конец массива.

Finder – сначала проверяет переданную строку: является ли каждый символ цифрой. Если да то пытается найти человека по этому индексу (данный метод находит только строчки, если передаётся число), предварительно превратив строчку в целое число. Если по этому индексу находится удаленный человек, то берется следующий человек. В случае, когда передаются не цифры, то ищется сначала такая строка, если не находится, то ищется колонка.

Deleter – удаление работает примерно также как поиск, только после нахождения элемента записывает вместо названия слово “deleted”.

printArray – выводит весь массив, из абстрактной таблицы.

printMatrix – выводит весь двойной массив (в отличие от finder выводит и удаленные элементы).

1. *SortTable.h*

binarySearch\_line – ищет бинарно индекс человека в массиве (поиск идет по сравнению строк. Например, АБА > ААЯ. Также функция работает рекурсивно).

binarySearch\_column – делает тоже самое, что и binarySearch\_line однако ищет предмет.

Конструктор – сортирует людей и предметы в двойном массиве.

Deleter – сначала данный метод смотрит является ли первый символ “\*”, если да, то он ищет бинарно колонку и удаляет предмет, если нет то ищет бинарно человека и удаляет его. Удаление происходит также, как и в обычной таблице: заменой на слово “deleted”.

Finder – работает также, как и finder в обычной таблице только поиск по строчке (не состоит из цифр) происходит бинарно.

Inserter – происходит также, как и в обычной таблице, только после добавления происходит сортировка.

1. *TreeTable.h*

(Класс написан как авл-дерево.)  
Конструктор – с помощью insert раскидывает по дереву строчки.

Деструктор – удаляет все элементы дерева.

Finder – ищет по ключу элемент в дереве. Ключ – это ФИО. Поиск идет по сравнению строк.

Inserter – реализация по большей части написана в insert. Insert – рекурсивно ищет свободный узел (node), чтобы в нем создать нового человека с оценками, после вызывает балансировку.

Deleter – также состоит по большей степени из del. Del – сначала находит такого человека, а потом происходит один из трех сценариев. Если удаляемый элемент лист, то он просто удаляется. Если есть слева или справа еще элемент, то сначала пытается поменяться местами с правым узлом, если справа ничего нет, то меняется с левым. Далее элемент удаляется (так как после обмена она стал листом). Если и слева, и с права есть узлы, то ищется след по величине элемент, меняется местами с удаляемым. А далее проверяется наличие справа или слева узлов и, в общем, повторяется начальные действия.

1. *HashTable.h*

Hash – хэш-функция. Принимает строчку, которая должна состоять из трех заглавных букв. Далее преобразует строчку в целое число от 0 да Hash(“ЯЯЯ”) и возвращает это число.

Конструктор – резервирует место в векторе. И заполняет его строчками из массива из обычной таблицы.

Finder – принимает ФИО человека и берет из него три заглавных буквы. Если в начале строки есть цифра, то после нахождения человека пропускает его и ищет дальше, это нужно для нахождения людей с одинаковым ФИО. После взятия трех букв с помощью хэш-функции находит индекс, а далее по листу ищет человека, если он есть.

Deleter – находит человека также как finder. Далее вырезает его с помощью метода листа erase. Далее вызывает удаление из обычной таблицы, это нужно, чтобы изменения сохранились в .txt файл.

Inserter – с помощью хэш-функции находится индекс и закидывается в лист человек.

## **Описание программной реализации**

**Table:**

*Поля:*

int stringCount; - количество строк

string\* array; - массив строк

*Методы:*

Table(string path); - конструктор

~Table();- деструктор

virtual void inserter(string str, string key) = 0; - чистая виртуальная вставка

virtual string finder(string key) = 0; - чисто виртуальный поиск

virtual bool deleter(string key) = 0; - чисто виртуальное удаление

**TScanTable:**

*Поля:*

string\*\* tableMatrix = new string \* [stringCount]; - двойной массив строк

int ammountTableColumns; - количество колонок

stack<int> deleted\_column; - стек с индексами удаленных колонок

stack<int> deleted\_line; - стек с индексами удаленных строк

*Методы:*

TScanTable(string path); - конструктор

~TScanTable();- деструктор

void inserter(string str, string key = "-1") override; - перегруженная вставка

string finder(string key) override; - перегруженный поиск

bool deleter(string key) override; - перегруженное удаление

void printArray(); - вывести массив

void printMatrix(); - вывести двойной массив

**TSortTable:**

*Методы:*

int binarySearch\_line(string key, int min, int max); - бинарный поиск в строку

int binarySearch\_column(string key, int min, int max); - бинарный поиск в колонку

TSortTable(string path); - конструктор

bool deleter(string key) override; - перегруженное удаление

string finder(string key) override; - перегруженный поиск

void inserter(string str, string key = "-1") override; - перегруженная вставка

**TTreeTable:**

*Классы:*

struct node; - элемент дерева

*Поля:*

node\* head; - начало дерево (корень)

int ammountMarks; - количество оценок

string subjects; - строчка хранит в себе предметы (математика, русский и т.д.)

*Методы:*

int height(node\* p); - высота узла

int bfactor(node\* p); - разница высот справа и слева от узла

void fixheight(node\* p); - починка высоты

node\* rotateRight(node\* p); - поворот в право (балансировка)

node\* rotateLeft(node\* q); - поворот в лево (балансировка)

node\* balance(node\* p); - балансировка

node\* insert(node\* p, string k); - добавление узла

void swaper(node\* first, node\* second); - меняет местами ключ и человека (они хранятся в узлах)

bool del(node\* p, string key); - удаление узла

TTreeTable(string path); - конструктор

string finder(string key) override; - перегруженный поиск

void inserter(string key, string marks) override; - перегруженная вставка

bool deleter(string key) override; - перегруженное удаление

~TTreeTable(); - деструктор

**THashTable:**

*Поля:*

vector<list<string>> hashTable; - вектор, состоящий из листов, листы состоят из строк

*Методы:*

THashTable(string path); - конструктор

string finder(string key) override; - перегруженный поиск

bool deleter(string key) override; - перегруженное удаление

void inserter(string person, string marks) override; - перегруженная вставка

# **Заключение**

В лабораторной работе были реализованы четыре вида таблицы и их методы. Также я узнал, что такое таблицы на дереве, хэш, хэш-таблицы. И что когда стоит использовать.

# **Список литературы**

1. <https://habr.com/ru/post/150732/> - авл-деревья
2. <https://www.youtube.com/watch?v=rVr1y32fDI0> – хэш-таблицы
3. <https://www.youtube.com/watch?v=BtBxtoXBI54&t=423s> – хэш-таблицы
4. <https://www.youtube.com/watch?v=9g3xte3Lf_M> – хэш-таблицы
5. <https://www.youtube.com/watch?v=ujRnhGc1GNw> – деревья
6. <https://www.youtube.com/watch?v=aFoaeA2tsVQ&t=1s> – бинарный поиск