МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

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

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Структура хранения данных: хеш-таблицы с разрешением коллизий перемешиванием и списками»**

**Выполнил:** студент группы 381706-1

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2019.

Содержание

[1. Введение 3](#_Toc9191464)

[2. Постановка задачи 4](#_Toc9191465)

[3. Руководство пользователя 5](#_Toc9191466)

[4. Руководство программиста 7](#_Toc9191467)

[4.1 Описание структуры программы 7](#_Toc9191468)

[4.2 Описание структур данных 7](#_Toc9191469)

[4.3 Описание алгоритмов 10](#_Toc9191470)

[5. Заключение 13](#_Toc9191471)

[6. Литература 14](#_Toc9191472)

# Введение

Лабораторная работа направлена на практическое освоение структуры данных под названием хеш-таблица.

Таблица – способ структурирования данных. Представляет собой распределение данных по однотипным строкам и столбцам.

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Существуют два основных варианта хеш-таблиц: с цепочками и открытой адресацией. Хеш-таблица содержит некоторый массив *mas*, элементы которого есть пары (хеш-таблица с открытой адресацией) или списки пар (хеш-таблица с цепочками).

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение *hashvalue = Hash(key)* играет роль индекса в массиве *mas*. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива *mas[hashvalue]*.

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Такие события не так уж и редки — например, при вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии уже превысит 50% (если каждый элемент может равновероятно попасть в любую ячейку) Поэтому механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

В данной лабораторной работе будут рассмотрены те два варианта хеш-таблиц, о которых шла речь выше: таблицы с открытой адресацией, где разрешение коллизий производится путем перемешивания, а также таблицы с цепочками, где проблема коллизий решена путем списков элементов для одного и того же значения хеша *hashvalue.*

# Постановка задачи

В рамках лабораторной работы ставится задача реализации структуры данных – хеш-таблица.

Для работы с хеш-таблицей будут реализованы:

* возможность создания хеш-таблицы с разрешением коллизий перемешиванием,
* возможность создания хеш-таблицы с разрешением коллизий списками,
* метод добавление и удаления элементов в таблицу,
* метод поиска элемента в таблице,
* метод, позволяющий вывести таблицу на экран.

Программное решение будет выглядеть следующим образом:

1. Класс хеш-таблицы с разрешением коллизий перемешиванием– THashTable
2. Класс хеш-таблицы с разрешением коллизий списками – THashTableList..
3. Класс элемента хеш-таблицы для пункта 1– TElem.
4. Класс элемента хеш-таблицы для пункта 2 – TElemL.
5. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
6. Программа, демонстрирующая работу классов THashTable и THashTableList.
7. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

В качестве примера использования упорядоченной таблицы предлагается следующее.

Создается пустая таблица размером 5 элементов с методом разрешения коллизий перемешиванием. Выводится на консоль.

В таблицу добавляются новые элементы и при каждом добавлении производится вывод измененной таблицы на консоль. Увеличение размера таблицы происходит в связи с используемым алгоритмом хеширования и добавления новых элементов.

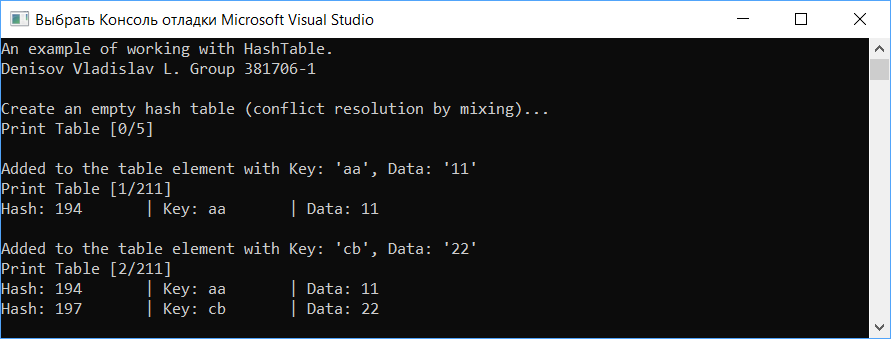


Рисунок 1 Пример работы демонстрационной программы.

Затем в таблице производится поиск элемента в таблицы и полученные сведения выводятся на консоль. Этот же элемент удаляется из таблицы, измененная таблица отображается на консоль. Производится повторная попытка поиска только что удаленного элемента.

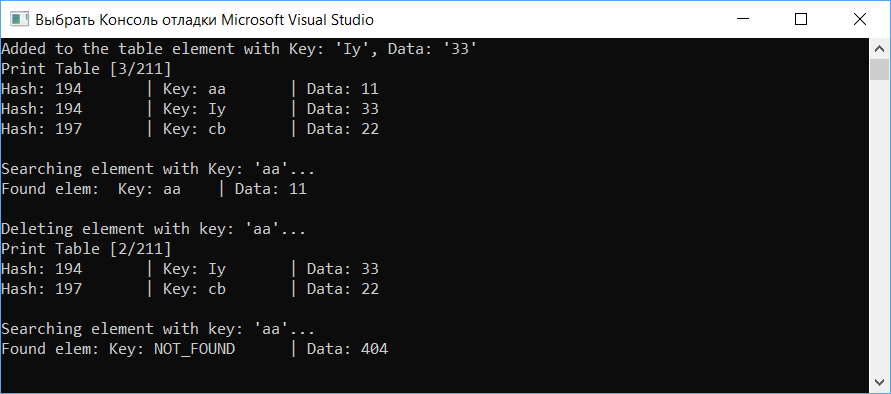


Рисунок 2 Пример работы демонстрационной программы.

Стоит обратить внимание на то, что ключи ‘aa’ и ‘Iy’ при вычислении значения хеш-функции дают одинаковое значение хеша, однако проблема коллизии была успешно решена, работа с таблицей не нарушена.

После этого создается пустая таблица размером 5 элементов с разрешением коллизий списками. В таблице добавляются элементы аналогично предыдущему демонстрационному варианту.

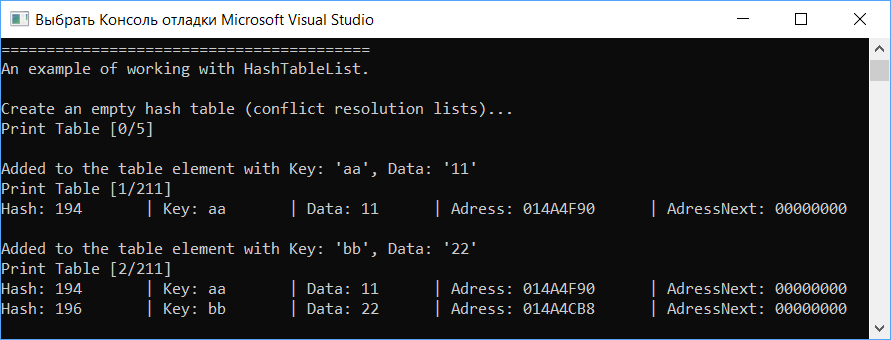


Рисунок 3 Пример работы демонстрационной программы.

В случае добавления элемента с ключом ‘Iy’ при уже существующем ключе ‘aa’, производится создание списка из этих двух элементов, к которому можно будет получить доступ по их общему значению хеша.

Выполняется поиск элемента в таблице, его удаление и повторный поиск. Все проходит успешно на примере все тех же элементов с проблемой коллизии.

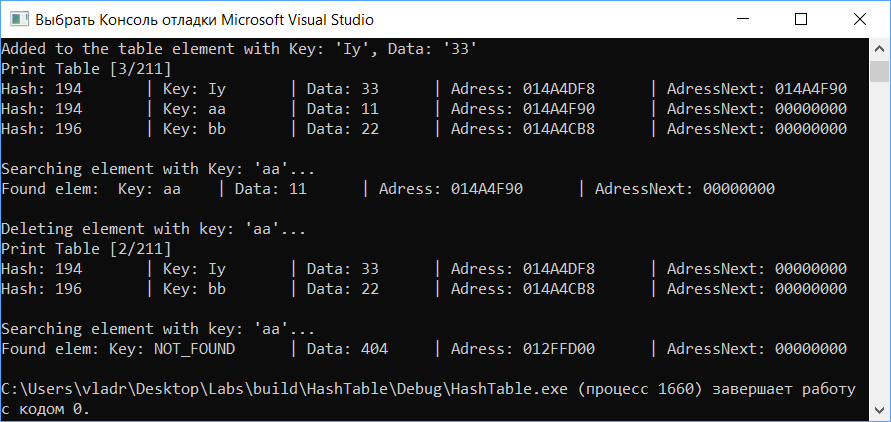


Рисунок 4 Пример работы демонстрационной программы.

На этом работа демонстрационной программы завершается.

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль HashTable. Содержит пример использования упорядоченной таблицы. Реализация в файле *main\_HashTable.cpp.*
* Модуль HashTableLib – статическая библиотека. Содержит файлы *HashTable.h* и *HashTableList.h,* в которых описан интерфейс и реализация шаблонных классов *THashTable* и *THashTableList* соответственно.
* Модуль HashTableTest. Содержит 26 тестов (по 13 для каждой из таблиц со своим способом разрешения коллизий), описанных в файлах *HashTableTest.cpp* и *HashTableListTest.cpp* иразработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TException – класс исключений.

Класс содержит 1 **private** поле *std::string msg* – переменная, хранящая сообщение об ошибке в виде строки.

И содержит 2 **public** элемента:

*TException(std::string \_str) : msg(\_msg)* – конструктор с одним параметром.

*void Print()* – метод отображения ошибки на консоль.

#### Классы TElem и TElemL– классы элементов каждой из таблиц.

Рассмотрим класс *TElem* подробно.

template <class T> class TElem {…} – класс элемента хеш-таблицы является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

string key – значение ключа.

T data – хранимая информация.

**Элементы класса, объявленные со спецификатором public:**

TElem(string \_key = "", T \_data = 0) – конструктор инициализатор.

TElem(TElem<T> &copy) – конструктор копирования.

void SetKey(string \_key) – метод изменения значения ключа.

string GetKey() – метод получения значения ключа.

void SetData(T \_data) – метод изменения значения хранимой информации.

T GetData() – метод получения значения хранимой информации.

T& GetDataAddress() – метод возвращающий адрес на хранимую информацию.

**Перегруженные операторы:**

TElem<T>& operator=(TElem<T> &elem) – оператор присваивания.

bool operator==(TElem<T> &elem) – оператор проверки на равенство элементов таблицы.

bool operator!=(TElem<T> &elem) – оператор проверки на неравенство элементов таблицы.

**Дружественная функция:**

friend ostream& operator<<(ostream& out, TElem<T>& elem) – вывод элемента таблицы на консоль.

Рассмотрим класс *TElemL* подробно.

template <class T> class TElemL {…} – класс элемента хеш-таблицы является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

string key – значение ключа.

T data – хранимая информация.

TElem<T> \*next – указатель на следующий хранимый элемент.

**Элементы класса, объявленные со спецификатором public:**

TElemL(string \_key = "", T \_data = 0, TElemL<T> \*\_next) – конструктор инициализатор.

TElemL(TElemL<T> &copy) – конструктор копирования.

void SetKey(string \_key) – метод изменения значения ключа.

string GetKey() – метод получения значения ключа.

void SetData(T \_data) – метод изменения значения хранимой информации.

T GetData() – метод получения значения хранимой информации.

T& GetDataAddress() – метод возвращающий адрес на хранимую информацию.

void SetNext(TElemL<T> \*\_next) – метод изменения указателя на следующий элемент.

TElemL<T>\* GetNext() – метод получения указателя на следующий элемент.

**Перегруженные операторы:**

TElemL<T>& operator=(TElemL<T> &elem) – оператор присваивания.

bool operator==(TElemL<T> &elem) – оператор проверки на равенство элементов таблицы.

bool operator!=(TElemL<T> &elem) – оператор проверки на неравенство элементов таблицы.

**Дружественная функция:**

friend ostream& operator<<(ostream& out, TElemL<T>& elem) – вывод элемента таблицы на консоль.

#### Класс THashTable – класс хеш-таблицы с разрешением коллизий перемешиванием.

Рассмотрим класс *THashtTable* подробно.

template <class T> class THashTable {…} – класс хеш-таблицы является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

TElem<T> notFound – отсутствующий в таблице элемент.

TElem<T> \*mas – указатель на область памяти, выделенную под элементы таблицы.

int size – максимальный размер таблицы и число занятых элементов.

Int count – число занятых элементов.

const int m – шаг, с которым производится обход массива.

unsigned Hash(string \_key) – функция, вычисляющая значение хеш.

void Resize(int \_size) – функция увеличения размера таблицы.

**Элементы класса, объявленные со спецификатором public:**

THashTable(int \_size = 5) – конструктор-инициализатор.

THashTable(THashTable<T> &hashtable) – конструктор копирования.

~THashTable() – деструктор.

int GetSize() – получение максимального числа элементов в таблице.

int GetCount() – получение числа добавленных элементов.

void Put(string \_key, T \_data) – добавление элемента.

void Put(TElem<T>& elem) – добавление элемента.

bool Del(string \_key) – удаление элемента.

TElem<T>& Search(string \_key) – поиск элемента.

bool IsSimple(const int nubmer) – проверка на то, является ли число простым.

**Дружественная функция:**

friend ostream& operator<<(ostream& out, THashTable& hashtable) – вывод на консоль.

#### Класс THashTableList – класс хеш-таблицы с разрешением коллизий списками.

Рассмотрим класс *THashTableList* подробно.

template <class T> class THashTableList {…} – класс хеш-аблицы является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

TElem<T> notFound – отсутствующий в таблице элемент.

TElem<T> \*\*mas – указатель на область памяти, выделенную под элементы таблицы.

int size – максимальный размер таблицы и число занятых элементов.

unt count – число занятых элементов.

unsigned Hash(string \_key) – функция, вычисляющая значение хеш.

void Resize(int \_size) – функция увеличения размера таблицы.

**Элементы класса, объявленные со спецификатором public:**

THashTableList(int \_size = 5) – конструктор-инициализатор.

THashTableList(THashTableList<T> &hashtable) – конструктор копирования

~THashTableList() – деструктор.

int GetSize() – получение максимального числа элементов в таблице.

int GetCount() – получение числа добавленных элементов.

void Put(string \_key, T \_data, TElemL<T> \*\_next = NULL) – добавление элемента.

void Put(TElemL<T>& elem) – добавление элемента.

bool Del(string \_key) – удаление элемента.

TElemL<T>& Search(string \_key) – поиск элемента.

bool IsSimple(const int nubmer) – проверка на то, является ли число простым.

**Дружественная функция:**

friend ostream& operator<<(ostream& out, TTableList<T>& hashtable) – вывод на консоль.

## Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Добавление элемента в таблицу с разрешением коллизий перемешиванием.**

Выполняется проверка на наличие свободных ячеек в таблице, если их нет, то увеличиваем ее максимальный размер в 2 раза. Вычисляем значение хеша для вставляемого ключа с помощью хеш-функции. Если полученное значение больше размера таблицы, то увеличиваем размер таблицы до значения этого хеша.

Затем выполняем просмотр массива с заданным по умолчанию шагом обхода и ищем первую незанятую ячейку. Выполняем вставку нового элемента в найденную позицию и увеличиваем число элементов на единицу.

**Добавление элемента в таблицу с разрешением коллизий списками.**

Выполняется проверка на наличие свободных ячеек в таблице, если их нет, то увеличиваем ее максимальный размер в 2 раза. Вычисляем значение хеша для вставляемого ключа с помощью хеш-функции. Если полученное значение больше размера таблицы, то увеличиваем размер таблицы до значения этого хеша.

Если в той ячейке массива, индекс которой мы получили с помощью хеша, еще нет ни одного элемента, то выделяем память под новый элемент и записываем указатель на него в эту ячейку.

В противном случае выделяем память под новый элемент, сохраняем указатель на выделенную память. Для добавляемого элемента в качестве следующего устанавливаем адрес того элемента, который сейчас расположен в требуемой ячейке. В ячейку записываем сохраненный указатель. Увеличиваем число элементов на единицу.

**Удаление элемента из таблицы с разрешением коллизий перемешиванием.**

Вычисляем значение хеша от полученного для удаления ключа. Если значение хеша больше максимального числа элементов в таблице, то элемента нет в таблице. Возвращается false – элемент не был удален.

Если проверка прошла успешно, то работаем в цикле, пока не будет найдена подходящая ячейка массива по вычисленному значению хеша. А именно, ключ очередного элемента массива не совпадет со значением удаляемого ключа.

При этом, запоминаем значение индекса в массиве, с которого начинается поиск удаляемого элемента. В цикле выполняется просмотр массива с заданным по умолчанию шагом обхода.

В случае, если нашли подходящую позицию, то в нее записывается значение notFound, а число элементов уменьшается на единицу. Возвращается значение true, сигнализирующее об успешном удалении элемента. Если же цикл завершился в связи тем, что произошел возврат на начало поиска, то возвращается значение false – элемент не был удален.

**Удаление элемента из таблицы с разрешением коллизий списками.**

Вычисляем значение хеша от полученного для удаления ключа. Если значение хеша больше максимального числа элементов в таблице, то элемента нет в таблице. Возвращается false – элемент не был удален. Создаем итератор и указатель на предыдущий элемент, на котором был итератор.

Если в ячейке массива с индексом, полученным по значению хеша, отсутствует элемент, то возвращается false – элемент не был удален.

Если в этой ячейке был единственный элемент, то память по указателю из этой ячейки очищается, значение текущего числа элементов уменьшается на единицу, а в ячейку устанавливается адрес системного элемента notFound. Возвращается true – элемент был удален.

Если в ячейке массива был список из нескольких элементов, дающих один и тот же хеш, то работаем в цикле, пока не достигнем конца этого списка или пока не будет обнаружен элемент, ключ которого совпадает с удаляемым.

Если нашли элемент, ключ которого совпадает с удаляемым, то для родителя устанавливаем следующим тот элемент, который стоял за удаляемым. Очищаем память, которая была выделена для удаляемого элемента. Уменьшаем число элементов в таблице. Возвращаем true – элемент был удален.

В противном случае элемент не был найден в этом списке – возвращается false.

**Поиск элемента в таблице с разрешением коллизий перемешиванием**

Вычисляем значение хеша от полученного для поиска ключа. Если значение хеша больше максимального числа элементов в таблице, то элемента нет в таблице. Возвращается системное значение notFound – элемент отсутствует в таблице.

Выполняем поиск элемента – алгоритм аналогичен поиску при удалении. В случае, если нашли подходящую позицию, то возвращается элемент из этой ячейки. Если нет – возвращается системное значение notFound.

**Поиск элемента в таблице с разрешением коллизий списками.**

Вычисляем значение хеша от полученного для поиска ключа. Если значение хеш больше максимального числа элементов в таблице, то элемента нет в таблице. Возвращается системное значение notFound, что означает отсутствие элемента в таблице.

Если в ячейке массива с полученным значением хеша значение ключа совпадает с искомым ключом, то возвращается элемент из этой ячейки.

В противном случае, если в этой ячейке хранится не один элемент, а список из нескольких элементов с одним и тем же значением хеша, то создаем итератор по этому списку. В цикле, пока не достигнут конец списка, сравниваем значение ключа очередного элемента с искомым.

Если ключи совпали, то возвращается найденный элемент. В противном случае цикл завершается и возвращается системное значение notFound.

# Заключение

В результате лабораторной работы была разработана библиотека, реализующая шаблонные классы хеш-таблиц с двумя способами разрешения коллизий: перемешиванием и списками. Она позволяет создать объект класса хеш-таблицы первого или второго вида и выполнить с ним простейшие операции, задача реализации которых была поставлена в начале данной лабораторной работы.

Были разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над хеш-таблицей. Описание примера работы с хеш-таблицей было представлено в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: <https://ru.wikipedia.org/wiki/Таблица_(значения)>
2. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: <https://ru.wikipedia.org/wiki/Хеш-таблица>