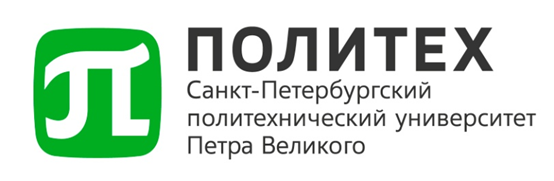
**Санкт-Петербургский политехнический университет Петра Великого**

**Институт компьютерных наук и технологий**

**Высшая школа программной инженерии**

****

**РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА**

**Алгоритмы работы со словарями**

по дисциплине «Алгоритмы и структуры данных»

Выполнил

студент гр. 3530904/00001 *<подпись>* Осипов А. А.

Руководитель *<подпись>* Надежда П.

«10» мая 2021 г.

Санкт-Петербург

2020 г

Оглавление

[**Введение. Общая постановка задачи** 3](#_Toc71510676)

[**Описание алгоритма решения и используемых структур данных** 4](#_Toc71510677)

[**Анализ алгоритма** 5](#_Toc71510678)

[**Описание спецификации программы** 6](#_Toc71510679)

[Решение проблемы с коллизиями 6](#_Toc71510680)

[Открытая адресация 7](#_Toc71510681)

[**Описание программы** 8](#_Toc71510682)

[Тесты 10](#_Toc71510683)

[**Заключение** 11](#_Toc71510684)

[**Список литературы** 12](#_Toc71510685)

[**Приложение 1. Текст кода программы.** 13](#_Toc71510686)

[Main.cpp: 13](#_Toc71510687)

[Table.hpp: 14](#_Toc71510688)

[Hash.hpp: 17](#_Toc71510689)

[Hash.cpp: 18](#_Toc71510690)

[**Приложение 2. Корректность работы программы** 19](#_Toc71510691)

# 

# **Введение. Общая постановка задачи**

Изначально задача была установлена подобным образом:

**Тема: Алгоритмы работы со словарями**

1. Для разрабатываемого словаря реализовать основные операции:
   1. INSERT(ключ, значение) — добавить запись с указанным ключом и значением;
   2. SEARCH(ключ) — найти запись с указанным ключом;
   3. DELETE(ключ) — удалить запись с указанным ключом.
2. Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием;
3. Программа должна быть написана в соответствии со стилем программирования C++ Programming Style Guidelines (<http://geosoft.no/development/cppstyle.html>);
4. Тесты должны учитывать как допустимые, так и недопустимые последовательности входных данных.

Вариант: 1.1.5. Англо-русский словарь. Хеш-таблица.

Разработать и реализовать алгоритм работы с англо-русским словарем, реализованным как хеш-таблица с разрешением коллизий внутри таблицы.

Ключом хеш-функции должно быть английское слово.

Строка хеш-таблицы должна содержать слово и ссылку на список, содержащий переводы английского слова, отсортированные по алфавиту (переводов слова может быть несколько).

В процессе выполнения работы оказалось недостаточно тех методов, что предлагались в изначальной постановке задачи. Для хеш-таблицы мне понадобится реализовать следующие функции:

1. Rehash() – функция позволяющая отчистить таблицу от удаленных элементов
2. Create() – функция-дополнение к функции добавления перевода – позволяет разделить создание ячейки и добавление перевода для отлова ошибки при крайне редком случае 3ной коллизии

# **Описание алгоритма решения и используемых структур данных**

При создании хэш таблицы была использована структура Node – являющаяся элементом первичного вектора, структура позволяет хранить переводы и статусы их наполнения\существования.

Изначально создается вектор содержащий n-ое количество структур Node, далее при добавлении нового перевода мы проверяем входные данные на исключительные ситуации, после проверок идет хеширование исходного слова двумя функциями и вызов функции создания ячейки (или же присваивание ей статуса существующей), в этой функции мы проверяем возможное существование тройной коллизии, если ее все же нет, то после смены статуса мы возвращаемся в метод добавления перевода и с помощью хэша полученного из hash\_horner() получаем нужную ячейку, в которой выбираем нужный вектор перевода путем получения хэша из второй функции, после чего и записываем наш перевод. Для поиска и вывода перевода создана метод Search(), вызывая ее, мы снова хэшируем исходное слово двумя функциями и получаем переводы. Метод Delete позволяет удалить какой-либо список переводов, изначально мы присваиваем данному списку статус удаленного и вызываем вспомогательную функцию rehash позволяющую отчистить таблицу от удаленных элементов.

# **Анализ алгоритма**

Анализ методов таблицы:

1. Конструктор - время О(N), память О(1).
2. Деструктор - время О(N), память О(1).
3. Add – время О(1), память О(1).
4. Search – время О(n), память О(1) – выполняется цикл выводящий все переводы слова.
5. Delete – время О(1), память О(1) .
6. Rehash - время О(N), память О(1).
7. Create - время О(1), память О(1).
8. Hash\_horner - время О(n), память О(1) – выполняется цикл проходящий по всем символам строки.
9. My\_hash - время О(n), память О(1) – выполняется цикл проходящий по всем символам строки.

# **Описание спецификации программы**

1. Создается хэш-таблица с указанным размером
   1. Если размер является некорректным, то переполнена вызывается исключение с текстом ошибки (“ Size must be greater than 0! ”)
2. Начинаем заполнять таблицу переводами, используя метод Add

2.1) Если таблица переполнена вызвается исключение с текстом ошибки (“Table overflow!”)

2.2) Если введенные данные не являются корректными, то вызывается исключение с текстом ошибки (“Word must be longer than 1 letter!”)

3) Для проверки наполнения нашей таблицы используем метод Search, передаем ключ (английское слово) для поиска

3.1) Если введенные данные не являются корректными, то вызывается исключение с текстом ошибки (“Word must be longer than 1 letter!”)

4) Delete – используем метод, для удаления какого-либо перевода по ключевому слову.

5) Rehash – дополнительный приватный метод, используемый для перестройки таблицы без учета удаленных элементов.

6) Create - дополнительный приватный метод, используемый для изменения статуса существования структуры Node

6.1) В случае повторения коллизии более 3х раз вызывается исключение с текстом ошибки (“Collision error, 3 hash function collision cases were expected, but the value is exceeded! ”)

## Решение проблемы с коллизиями

Такая структура как хэш-таблица имеет свои недостатки, а именно возможность коллизии, на выходе из хэш функции у двух разных слов могут получиться одинаковые коды. Решения для моего случая найдено не было, так как в моей работе таблица должна содержать множество переводов для одного слова. Однако для многократного сокращения возможных ошибок в следствии коллизии я ввел дополнительную функцию хэширования и добавил в структуру Node еще два списка содержащих переводы, дополнительное хэширование нужно для выбора нужного списка в случае коллизии полученной в первой функции хэширования.

Методы решения коллизии которые могли быть в моей работе (если бы не условие с множественным переводом):

## Открытая адресация

В другой стратегии, называемой **открытой адресацией**, все записи хранятся в самом массиве корзин (buckets). Когда необходимо вставить новую запись, корзины проверяются, начиная с ячейки хэшированного до и продолжаясь в некоторой последовательности проб, до тех пор, пока не будет найден незанятый слот. При поиске записи корзины сканируются в той же последовательности, пока не будет найдена либо целевая запись, либо не найден слот неиспользуемого массива, что указывает на то, что в таблице такого ключа нет. Название "открытая адресация" относится к тому факту, что местоположение ("адрес") элемента не определяется его хэш-значением. (Этот метод также называется закрытым хэшированием; его не следует путать с "открытым хэшированием" или "закрытой адресацией", которые обычно означают отдельное сцепление.)

#### Кукушечное хэширование

Другим альтернативным решением с открытой адресацией является хэширование кукушки, которое обеспечивает постоянное время поиска и удаления в худшем случае, а также постоянное амортизированное время для вставок (с низкой вероятностью, что будет встречаться худший случай). Оно использует две или более хэш-функций, что означает, что любая пара ключ/значение может находиться в двух или более местах. Для поиска используется первая хэш-функция; если ключ/значение не найдено, то используется вторая хэш-функция и т. д. Если во время вставки происходит коллизия, то ключ повторно хэшируется со второй хэш-функцией, чтобы сопоставить его с другой кориной. Если используются все хэш-функции и все еще существует коллизия, то ключ с которым произошла коллизия удаляется, чтобы освободить место для нового ключа, а старый ключ повторно хэшируется с одной из других хэш-функций, которая сопоставляет его с другой корзиной. Если это расположение также приводит к коллизии, то процесс повторяется до тех пор, пока не произойдет коллизия, или процесс не пройдет все корзины, после чего размер таблицы будет изменен. Комбинируя несколько хэш-функций с несколькими ячейками на сегмент, можно достичь очень высокого использования пространства.

# **Описание программы**

1. Для класса Table в файле Table.hpp определены и реализованы:
   1. Конструктор Table();
   2. Деструктор ~ Table ();
   3. Метод void Add(std::string en\_word, std::string ru\_word), который внутри себя проверяет корректность введенных слов, а после этого добавляет элемент с его переводами в таблицу;
   4. Метод bool search(std::string key), который сначала проверяет внутри себя исключительные ситуации, описанные в спецификации, а затем происходит обработка на то, найдено ли слово. Если оно не найдено, то метод возвращает false. В противном случае происходит вывод всех переводов и возвращается значение true;
   5. Метод Delete(std::string en\_word) сначала производит присваивание нужному списку переводов статус удаленного, а затем вызывает метод rehash очищающий таблицу от этих элементов.
   6. Приватную структуру node, которая содержит:

1.6.1) Поля

std::vector<std::string> first\_value;  
std::vector<std::string> second\_value;  
std::vector<std::string> third\_value;

– поля, которые будут хранить переводы слов.

1.6.2) Поля

bool first\_full;  
bool second\_full;  
bool third\_full;

- которые будут отвечать за статус наполнения векторов с переводами

1.6.3) Поля

bool first\_deleted;  
bool second\_deleted;  
bool third\_deleted;

- которые будут отвечать за статус удаления векторов с переводами

1.6.4) Поле

int create\_count;

- которое будет отвечать за количество произошедших коллизий внутри одной структуры node

1.6.5) Поле

bool status;

- отвечающее за статус использования данной ячейки ранее (пуста ли она)

1.6.6) Конструктор для структуры Node().

Node(){  
 status = false;  
 first\_deleted = false;  
 second\_deleted = false;  
 third\_deleted = false;  
 first\_full = false;  
 second\_full = false;  
 third\_full = false;  
 create\_count = 0;  
}

# Тесты

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название теста | Способ | Выход |
| 1 | Проверка конструктора | Table<> err\_1 (-1); | Size must be greater than 0! |
| 2 | Проверка метода добавления переводов | Table<> err\_2; err\_2.Add("a","а"); | Word must be longer than 1 letter! |
| 3 | Проверка метода поиска | Table<> err\_3; err\_3.Search("a"); | Word must be longer than 1 letter! |
| 4 | Проверка метода удаления перевода | tabl.Delete("cat"); if(tabl.Search("cat")){std::cout << "cat - NOT DELETED \n";}else{std::cout<<"cat - DELETED \n";} | cat - DELETED |
|  | Проверка метода добавления и поиска | tabl.Add("cat", "кот"); tabl.Add("cat", "кошак"); tabl.Add("cat", "кошка");  tabl.Search("cat"); | Translation cat to RU is кот  Translation cat to RU is кошак  Translation cat to RU is кошка |

# **Заключение**

В течение курса по «Алгоритмам и структурам данных», полученные мной знания были использованы в реализации данной работы. Была реализована хэш-таблица. Для тестирования корректности работы англо-русского словаря были написаны тесты в файле main.cpp. Я на практике разобрался со всем этим и смог написать алгоритм для решения поставленной задачи.

Такая структура данных, как хэш-таблица, может служить как вспомогательным, так и основным средством в составлении англо-русских словарей. Её достоинство перед другими структурами – это быстродействие, такой метод хранения данных хоть и не всегда надежен в виду коллизий, но очень быстр и малотребователен

В итоге, поставленная передо мной задача была выполнена, программа работает корректно и эффективно. Все вышеперечисленное обусловлено полученными знаниями за учебный год.

# **Список литературы**

1. Лекции и презентации по курсу «Алгоритмы и структуры данных»//Санкт-Петербургский политехнический университет Петра Великого (СПбПУ).- URL: <https://dl.spbstu.ru/course/view.php?id=513>. — (дата последнего обращения - 10.05.2021).
2. <https://alextoolsblog.blogspot.com/2019/12/hash-table-collisions.html>
3. <https://habr.com/ru/post/509220/>

# **Приложение 1. Текст кода программы.**

## Main.cpp:

#include <iostream>  
#include "hash.hpp"  
#include "table.hpp"  
#include <string>  
  
int main() {  
 setlocale(LC\_ALL, "Russian");  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Create table with default size: \n";  
 Table<> tabl;  
 std::cout<<"Create table with default size: completed! \n";  
 std::cout<<"------------------------------------------- \n";  
  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Fill table: \n";  
 tabl.Add("cat", "кот");  
 tabl.Add("cat", "кошак");  
 tabl.Add("cat", "кошка");  
 tabl.Add("dog", "собака");  
 tabl.Add("cloud", "облако");  
 tabl.Add("shake", "трясти");  
 tabl.Add("snake", "змея");  
 tabl.Add("snake", "змей");  
 std::cout<<"Fill table: completed! \n";  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Test Search function: \n";  
 tabl.Search("cat");  
 tabl.Search("shake");  
 tabl.Search("snake");  
 std::cout<<"Test Search function: completed! \n";  
 std::cout<<"------------------------------------------- \n";  
  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Check Errors: \n";  
 try{  
 Table<> err\_1 (-1);  
 }  
 catch (std::invalid\_argument &e){  
 std::cout<< e.what();  
 }  
 try{  
 Table<> err\_2;  
 err\_2.Add("a","а");  
 }  
 catch (std::invalid\_argument &e){  
 std::cout<< e.what();  
 }  
 try{  
 Table<> err\_3;  
 err\_3.Search("a");  
 }  
 catch (std::invalid\_argument &e){  
 std::cout<< e.what();  
 }  
 std::cout<<"Check Errors: completed! \n";  
 std::cout<<"------------------------------------------- \n";  
  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Check Delete function: \n";  
 tabl.Delete("cat");  
 if(tabl.Search("cat")){std::cout << "cat - NOT DELETED \n";}else{std::cout<<"cat - DELETED \n";}  
 std::cout<<"Check Delete function: completed! \n";  
 std::cout<<"------------------------------------------- \n";  
  
 std::cout<<"------------------------------------------- \n";  
 std::cout<<"Check destructor: \n";  
 tabl.~Table();  
 std::cout<<"Check destructor: completed!\n";  
 std::cout<<"------------------------------------------- \n";  
 return 0;  
}

## Table.hpp:

#ifndef KURSOVAYA\_TABLE\_HPP  
#define KURSOVAYA\_TABLE\_HPP  
#include "hash.hpp"  
#include <string>  
#include <vector>  
#include <exception>  
#include <algorithm>  
  
template <class t\_hash\_1 = hash\_1, class t\_hash\_2 = hash\_2>  
class Table  
{  
 static const int default\_size = 100;  
 struct Node  
 {  
 std::vector<std::string> first\_value;  
 std::vector<std::string> second\_value;  
 std::vector<std::string> third\_value;  
 bool first\_full;  
 bool second\_full;  
 bool third\_full;  
 int create\_count;  
 bool status;  
 bool first\_deleted;  
 bool second\_deleted;  
 bool third\_deleted;  
 Node(){  
 status = false;  
 first\_deleted = false;  
 second\_deleted = false;  
 third\_deleted = false;  
 first\_full = false;  
 second\_full = false;  
 third\_full = false;  
 create\_count = 0;  
 }  
 };  
 std::vector<Node> arr;  
 int buffer\_size;  
 int count;  
public:  
 Table(int size\_ = default\_size)  
 {  
 if(size\_ <= 0 ) {  
 throw std::invalid\_argument("Size must be greater than 0! \n");  
 }  
 count =0;  
 buffer\_size = size\_;  
 for(int i =0; i<buffer\_size; i++) {  
 arr.push\_back(Node());  
 }  
 }  
 ~Table()  
 {  
 for(int i=0; i<buffer\_size; i++) {  
 arr[i].first\_value.clear();  
 arr[i].second\_value.clear();  
 arr[i].third\_value.clear();  
 }  
 arr.clear();  
 }  
  
bool Add(std::string en\_word, std::string ru\_word, const t\_hash\_1& hash1 = t\_hash\_1(), const t\_hash\_2& hash2 = t\_hash\_2()) {  
 if(count == buffer\_size - 1) {  
 throw std::runtime\_error("Table overflow! \n");  
 }  
 if(en\_word.size() <= 1) {  
 throw std::invalid\_argument("Word must be longer than 1 letter! \n");  
 }  
 int h1 = hash1(en\_word, buffer\_size);  
 int h2 = hash2(en\_word);  
 create(h1);  
 switch (h2) {  
 case 0:  
 arr[h1].first\_value.push\_back(ru\_word);  
 arr[h1].status = true;  
 arr[h1].first\_full = true;  
 sort(arr[h1].first\_value.begin(), arr[h1].first\_value.end());  
 break;  
 case 1:  
 arr[h1].second\_value.push\_back(ru\_word);  
 arr[h1].status = true;  
 arr[h1].second\_full = true;  
 sort(arr[h1].second\_value.begin(), arr[h1].second\_value.end());  
 break;  
 case 2:  
 arr[h1].third\_value.push\_back(ru\_word);  
 arr[h1].status = true;  
 arr[h1].third\_full = true;  
 sort(arr[h1].third\_value.begin(), arr[h1].third\_value.end());  
 break;  
 default:  
 throw std::runtime\_error("Error with second hash function! \n");  
 break;  
 }  
 return true;  
}  
  
 bool Search(std::string en\_word, const t\_hash\_1& hash1 = t\_hash\_1(), const t\_hash\_2& hash2 = t\_hash\_2()) {  
 if(en\_word.size() <= 1) {  
 throw std::invalid\_argument("Word must be longer than 1 letter! \n");  
 }  
 int h1 = hash1(en\_word, buffer\_size);  
 int h2 = hash2(en\_word);  
 switch (h2) {  
 case 0:  
 if(arr[h1].status && !arr[h1].first\_deleted && arr[h1].first\_full){  
 for(int i=0; i<arr[h1].first\_value.size(); i++){  
 std::cout<<"Translation " << en\_word << " to RU is "<< arr[h1].first\_value[i]<< "\n";  
 }  
 return true;  
 }else{return false;}  
 break;  
 case 1:  
 if(arr[h1].status && !arr[h1].second\_deleted && arr[h1].second\_full){  
 for(int i=0; i<arr[h1].second\_value.size(); i++){  
 std::cout<<"Translation " << en\_word << " to RU is "<< arr[h1].second\_value[i]<< "\n";  
 }  
 return true;  
 }else{return false;}  
 break;  
 case 2:  
 if(arr[h1].status && !arr[h1].third\_deleted && arr[h1].third\_full){  
 for(int i=0; i<arr[h1].third\_value.size(); i++){  
 std::cout<<"Translation " << en\_word << " to RU is "<< arr[h1].third\_value[i]<< "\n";  
 }  
 return true;  
 }else{return false;}  
 break;  
 default:  
 throw std::runtime\_error("Error with second hash function! \n");  
 break;  
 }  
 }  
 void Delete(std::string en\_word, const t\_hash\_1& hash1 = t\_hash\_1(), const t\_hash\_2& hash2 = t\_hash\_2()) {  
 int h1 = hash1(en\_word, buffer\_size);  
 int h2 = hash2(en\_word);  
 switch (h2) {  
 case 0:  
 arr[h1].first\_deleted = true;  
 break;  
 case 1:  
 arr[h1].second\_deleted = true;  
 break;  
 case 2:  
 arr[h1].third\_deleted = true;  
 break;  
 default:  
 throw std::runtime\_error("Error with second hash function! \n");  
 break;  
 }  
 rehash();  
 count--;  
 }  
private:  
 void rehash() {  
 std::vector<Node> temp (arr);  
 arr.clear();  
 for(int i =0; i<buffer\_size; i++) {  
 if(temp[i].first\_deleted){  
 temp[i].first\_value.clear();  
 temp[i].first\_deleted = false;  
 temp[i].first\_full = false;  
 }  
 if(temp[i].second\_deleted){  
 temp[i].second\_value.clear();  
 temp[i].second\_deleted = false;  
 temp[i].second\_full = false;  
 }  
 if(temp[i].third\_deleted){  
 temp[i].third\_value.clear();  
 temp[i].third\_deleted = false;  
 temp[i].third\_full = false;  
 }  
 arr.push\_back(temp[i]);  
 }  
 temp.clear();  
 }  
 void create(int hash){  
 if(arr[hash].create\_count<3){  
 arr[hash].create\_count++;  
 count++;  
 }else{  
 throw std::runtime\_error("Collision error, 3 hash function collision cases were expected, but the value is exceeded! \n");  
 }  
 }  
};  
  
#endif

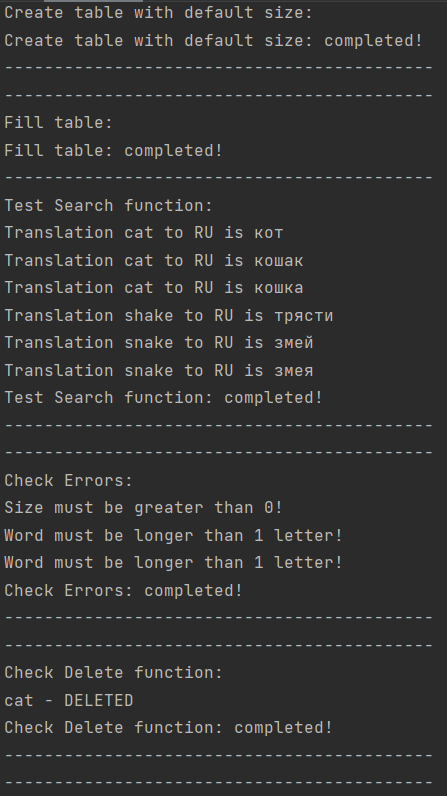
## Hash.hpp:

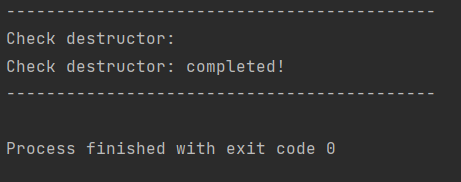
#ifndef KURSOVAYA\_HASH\_HPP  
#define KURSOVAYA\_HASH\_HPP  
#include <iostream>  
#include <string>  
  
int hash\_horner(const std::string& s, int table\_size, const int key);  
int my\_hash(const std::string& s, int table\_size);  
struct hash\_1  
{  
 int operator()(const std::string& s, int table\_size) const  
 {  
 return hash\_horner(s, table\_size, table\_size - 1);  
 }  
};  
struct hash\_2  
{  
 int operator()(const std::string& s) const  
 {  
 return my\_hash(s, 3);  
 }  
};  
#endif

## Hash.cpp:

#include "hash.hpp"  
#include <iostream>  
#include <string>  
  
int hash\_horner(const std::string& s, int table\_size, const int key){  
 int hash\_result = 0;  
 for (int i = 0; i < s.size(); ++i){  
 hash\_result += (key \* hash\_result + s[i]) % table\_size;  
 }  
 hash\_result = (hash\_result \* 2 + 1) % table\_size;  
 return hash\_result;  
}  
int my\_hash(const std::string& s, int table\_size){  
 int hash\_result = 0;  
 int max\_simv = 0;  
 int sred\_simv = 0;  
 for(int i =0; i<s.size();i++){  
 if(max\_simv<s[i]){  
 max\_simv = s[i];  
 }  
 sred\_simv += s[i];  
 }  
 sred\_simv /= s.size();  
 hash\_result = (max\_simv + s.size() - sred\_simv)%table\_size;  
 return hash\_result;  
}

# **Приложение 2. Корректность работы программы**





Сама программа также заканчивает работу с выходным кодом 0, что свидетельствует об отсутствии ошибок и корректной работе.