Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

им. В.И. Ульянова (Ленина)»

кафедра Апу

**ОТЧЕТ**

**по курсовой работе**

Выполнили:

Саватин В. А.

Панфёров М. А.

Группа №3371

Преподаватель: Ряскова Е. Б.

Санкт-Петербург

2024

Содержание

Оглавление

[Введение 3](#_Toc184574860)

[Постановка задачи 4](#_Toc184574861)

[Листинг программы 5](#_Toc184574862)

[Код и пример работы программы 10](#_Toc184574863)

# Введение

С учетом стремительного развития информационных технологий и увеличения объема обрабатываемых данных, эффективное управление информацией становится одной из ключевых задач в различных сферах деятельности. Эта проблема затрагивает сервисные компании, такие как салоны красоты, которые располагают обширными базами данных о своих услугах и сотрудниках. Для оптимизации обработки и хранения такой информации, необходимо внедрение современных алгоритмов и структур данных, таких как хэш-таблицы.

Хэш-таблицы представляют собой структуры данных, позволяющие быстро осуществлять поиск, добавление и удаление элементов. Используя хэш-функции для преобразования ключей в индексы, эта структура обеспечивает быстрый доступ к элементам, что особенно полезно в контексте динамически изменяемой информации, характерной для салонов услуг.

Цель данной курсовой работы заключается в разработке программы, использующей хэш-таблицы для хранения и управления данными о услугах салона. В процессе работы над проектом будет выполнен анализ структуры данных, оценка производительности хэш-таблиц и демонстрация их применения на практике.

# Постановка задачи

1. Организуйте данные из предметной области в виде хеш-таблицы (*первичный ключ хеш-таблицы выделен курсивом*). Методом закрытого хеширования с квадратичным опробованием. В качестве ключевого значения для расчета хеш-функции берётся число, состоящее из цифр кода услуги. Обратите внимание, что для вычисления хеш-функции значение ключевого поля должно быть целым.

2. Реализуйте следующие функции:

* Меню для выбора функции (используя конструкцию switch — case).
* Добавление записи (в случае закрытого хеширования необходимо отслеживать заполненность хеш-таблицы).
* Проверку корректности вводимой информации.
* Удаление записи.
* Просмотр всех имеющихся записей.
* Удаление всех записей.
* Поиск записи по ключевому полю.
* Поиск данных по фрагменту одного из не ключевых полей (см. вариант задания).

В данном случае задание 6 варианта. Нужно сделать программу хэш-таблицы, где хранятся данные об услугах слона красоты. Это данные:

Код услуги – строка формата «TNN» ,где T – вид услуги, N – цифра от 1 до 6.

Название услуги – строка.

ФИО мастера – строка.

Стоимость в рублях – число.

Применяется метод закрытого хеширования с квадратичным опробованием. В качестве ключевого значения для расчета хеш-функции берётся число, состоящее из цифр кода услуги. Хэш-значение генерируется методом умножения. Поиск услуг ведётся по фрагменту названия Кнута-Морриса-Пратта.

# Листинг программы

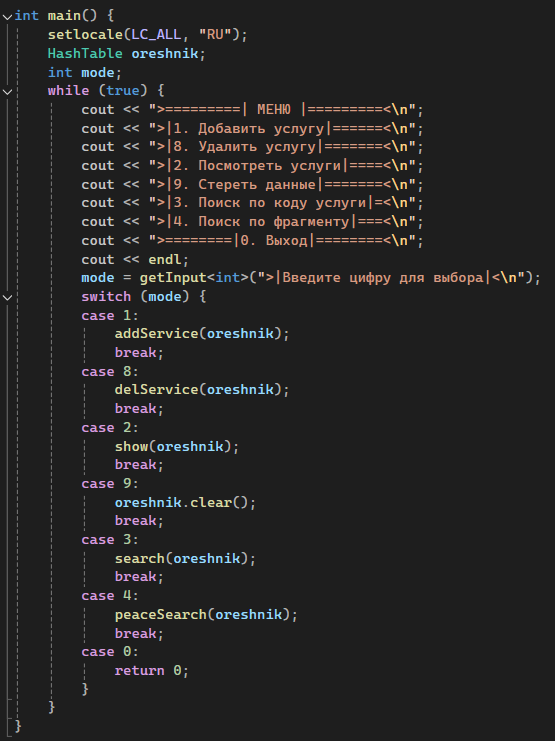


Рис.1 – Гавная функция.

На рис.1 изображена главная функция программы. Тут создаётся хэш-таблица и находится меню. Выбор функции осущетвлён при помощи switch() case.

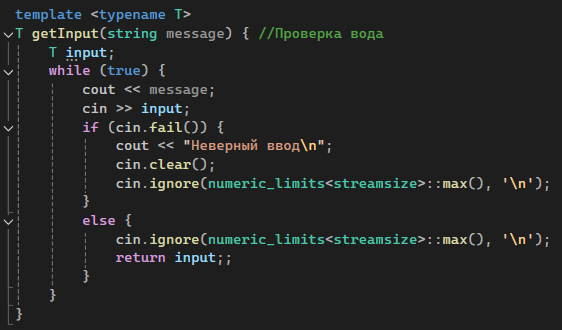


Рис.2 – Функция проверки корректности ввода данных.

На рис.2 изображена функция проверки корректности вводимых данных. Любой ввод в программе проходит через неё. Если вводятся корректные данные, то программа будет работать дальше, в ином случае будет сообщено о неверном вводе. Программа будет запрашивать корректные данные до тех пор, пока они не будут введены.



Рис.3 – Функция добавления услуги

На рис.3 изображена функция добавления услуги. В начале заращивается ввод кода услуги. После идёт проверка корректности введёных чисел, при неверном вводе программа попросит их ввести заново и так до тех пор, пока не будут введены корректные числа. После вводятся названия услуги, ФИО мастера и стоимость. После этот элемент добавляется в таблицу, иначе нет.

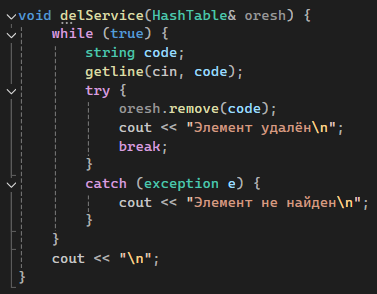
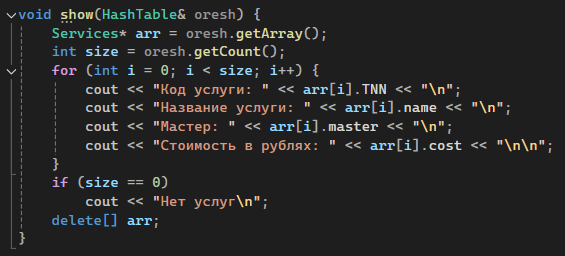


Рис.4 – Функция удаления элемента.

На рис.4 изображена функция удаления элемента. Вводится код, после программа удаляет элемент, иначе сообщает, что элемент не найден.

Рис.5 – Функция вывода услуг.

На рис.5 изображена функция вывода услуг. Если услуги есть, то они будут выведены, иначе будет сообщено, что услуг нет.

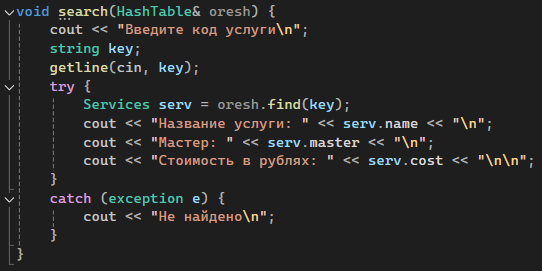


Рис.6 – Функция поиска.

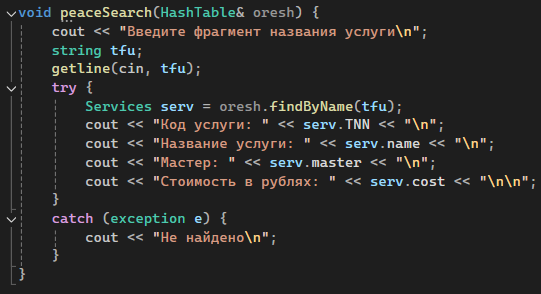
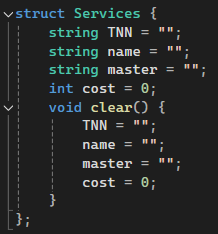
****На рис.6 изображена функция поиска. Вводится код услуги, если услуга с таким кодом есть, то будет выведены название услуги, ФИО мастера и стоимость, иначе программа сообщит, что ничего не найдено.

Рис.6 – Функция поиска по фрагменту названия услуги.

На рис.6 изображена функция поиска по фрагменту названия услуги. Вводится фрагмент названия услуги, если услуга с таким фрагментом есть, то будет выведены код услуги, название услуги, ФИО мастера и стоимость, иначе программа сообщит, что ничего не найдено.

 Рис.8 – Структура услуги

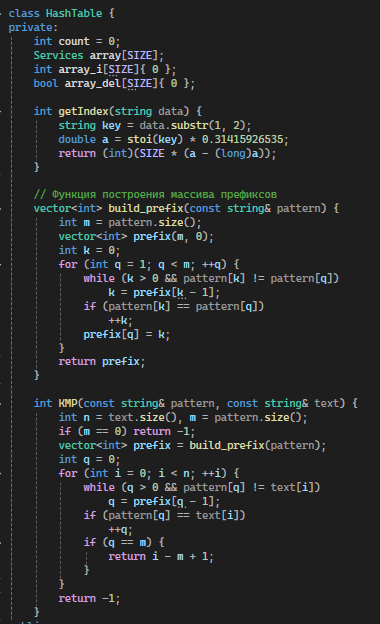
На рис.8 изображена структура услуги. В некё указан код, название, ФИО мастера и стоимость. Также есть значения, для удаления услуги.

Рис.9 – класс хэш-таблицы, приват

На рис.9 изображена приватная часть класса хэш-таблицы. Тут создаётся массив на 100 услуг, находится алоритм хэширования, функция создания массива префиксов для алгоритма Кнута-Морриса-Пратта и сам алгоритм поиска.

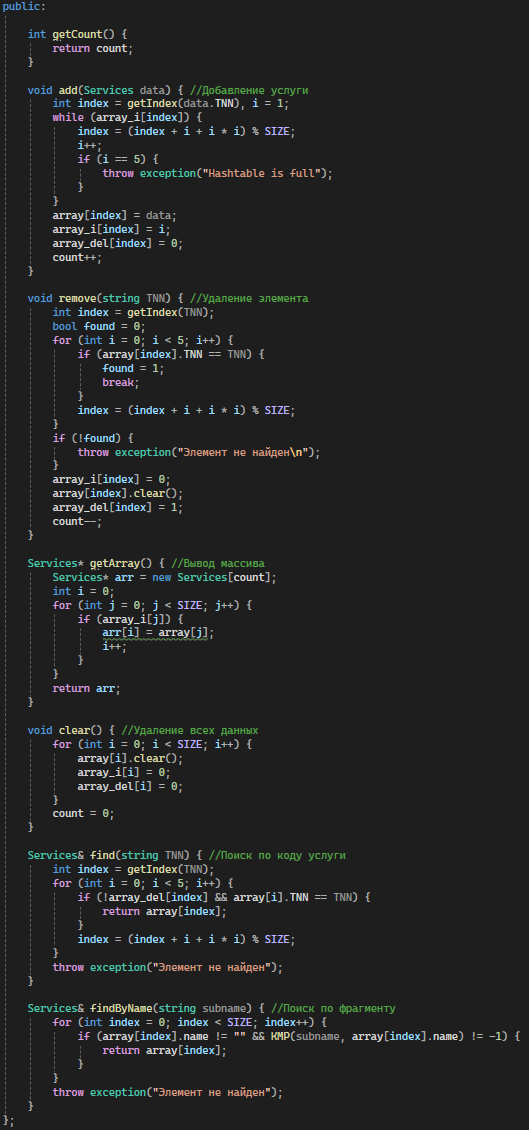


Рис.10 – класс хэш-таблица, паблик

На рис.10 изображена публичная часть класса хэш-таблицы. Тут находятся функции добавления и удаления элемента, поиск по коду и фрагменту названия услуги, вывода всех услуг.

# Код и пример работы программы

Coursivia\_AiSD.cpp:

#include <iostream>

#include <string>

#include <vector>

using namespace std;

#define SIZE 100

struct Services {

string TNN = "";

string name = "";

string master = "";

int cost = 0;

void clear() {

TNN = "";

name = "";

master = "";

cost = 0;

}

};

class HashTable {

private:

int count = 0;

Services array[SIZE];

int array\_i[SIZE]{ 0 };

bool array\_del[SIZE]{ 0 };

int getIndex(string data) {

string key = data.substr(1, 2);

double a = stoi(key) \* 0.31415926535;

return (int)(SIZE \* (a - (long)a));

}

// Функция построения массива префиксов

vector<int> build\_prefix(const string& pattern) {

int m = pattern.size();

vector<int> prefix(m, 0);

int k = 0;

for (int q = 1; q < m; ++q) {

while (k > 0 && pattern[k] != pattern[q])

k = prefix[k - 1];

if (pattern[k] == pattern[q])

++k;

prefix[q] = k;

}

return prefix;

}

int KMP(const string& pattern, const string& text) {

int n = text.size(), m = pattern.size();

if (m == 0) return -1;

vector<int> prefix = build\_prefix(pattern);

int q = 0;

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

while (q > 0 && pattern[q] != text[i])

q = prefix[q - 1];

if (pattern[q] == text[i])

++q;

if (q == m) {

return i - m + 1;

}

}

return -1;

}

public:

int getCount() {

return count;

}

void add(Services data) { //Добавление услуги

int index = getIndex(data.TNN), i = 1;

while (array\_i[index]) {

index = (index + i + i \* i) % SIZE;

i++;

if (i == 5) {

throw exception("Hashtable is full");

}

}

array[index] = data;

array\_i[index] = i;

array\_del[index] = 0;

count++;

}

void remove(string TNN) { //Удаление элемента

int index = getIndex(TNN);

bool found = 0;

for (int i = 0; i < 5; i++) {

if (array[index].TNN == TNN) {

found = 1;

break;

}

index = (index + i + i \* i) % SIZE;

}

if (!found) {

throw exception("Элемент не найден\n");

}

array\_i[index] = 0;

array[index].clear();

array\_del[index] = 1;

count--;

}

Services\* getArray() { //Вывод массива

Services\* arr = new Services[count];

int i = 0;

for (int j = 0; j < SIZE; j++) {

if (array\_i[j]) {

arr[i] = array[j];

i++;

}

}

return arr;

}

void clear() { //Удаление всех данных

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

array[i].clear();

array\_i[i] = 0;

array\_del[i] = 0;

}

count = 0;

}

Services& find(string TNN) { //Поиск по коду услуги

int index = getIndex(TNN);

for (int i = 0; i < 5; i++) {

if (!array\_del[index] && array[i].TNN == TNN) {

return array[index];

}

index = (index + i + i \* i) % SIZE;

}

throw exception("Элемент не найден");

}

Services& findByName(string subname) { //Поиск по фрагменту

for (int index = 0; index < SIZE; index++) {

if (array[index].name != "" && KMP(subname, array[index].name) != -1) {

return array[index];

}

}

throw exception("Элемент не найден");

}

};

template <typename T>

T getInput(string message) { //Проверка вода

T input;

while (true) {

cout << message;

cin >> input;

if (cin.fail()) {

cout << "Неверный ввод\n";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

else {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return input;;

}

}

}

void addService(HashTable& oresh) {

Services serv;

cout << "Введите код услуги\n";

cout << "Формат кода - <Символ><1-6><1-6>\n";

string key;

while (true) {

getline(cin, serv.TNN);

key = serv.TNN.substr(1, 2);

try {

int a = stoi(key);

if (a / 10 > 6 || a % 10 > 6 || a / 10 < 1 || a % 10 < 1)

cout << "Неверный формат кода\n";

else

break;

}

catch (exception e) {

continue;

}

}

cin.ignore();

cout << "Введите название услуги\n";

getline(cin, serv.name);

cout << "Введите ФИО мастера\n";

getline(cin, serv.master);

serv.cost = getInput<int>("Введите стоимость в рублях\n");

try {

oresh.add(serv);

}

catch (exception e) {

cout << "Невозможно вставить элемент в таблицу\n";

}

cout << "\n";

}

void delService(HashTable& oresh) {

while (true) {

string code;

getline(cin, code);

try {

oresh.remove(code);

cout << "Элемент удалён\n";

break;

}

catch (exception e) {

cout << "Элемент не найден\n";

}

}

cout << "\n";

}

void show(HashTable& oresh) {

Services\* arr = oresh.getArray();

int size = oresh.getCount();

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

cout << "Код услуги: " << arr[i].TNN << "\n";

cout << "Название услуги: " << arr[i].name << "\n";

cout << "Мастер: " << arr[i].master << "\n";

cout << "Стоимость в рублях: " << arr[i].cost << "\n\n";

}

if (size == 0)

cout << "Нет услуг\n";

delete[] arr;

}

void search(HashTable& oresh) {

cout << "Введите код услуги\n";

string key;

getline(cin, key);

try {

Services serv = oresh.find(key);

cout << "Название услуги: " << serv.name << "\n";

cout << "Мастер: " << serv.master << "\n";

cout << "Стоимость в рублях: " << serv.cost << "\n\n";

}

catch (exception e) {

cout << "Не найдено\n";

}

}

void peaceSearch(HashTable& oresh) {

cout << "Введите фрагмент названия услуги\n";

string tfu;

getline(cin, tfu);

try {

Services serv = oresh.findByName(tfu);

cout << "Код услуги: " << serv.TNN << "\n";

cout << "Название услуги: " << serv.name << "\n";

cout << "Мастер: " << serv.master << "\n";

cout << "Стоимость в рублях: " << serv.cost << "\n\n";

}

catch (exception e) {

cout << "Не найдено\n";

}

}

int main() {

setlocale(LC\_ALL, "RU");

HashTable oreshnik;

int mode;

while (true) {

cout << ">=========| МЕНЮ |=========<\n";

cout << ">|1. Добавить услугу|======<\n";

cout << ">|8. Удалить услугу|=======<\n";

cout << ">|2. Посмотреть услуги|====<\n";

cout << ">|9. Стереть данные|=======<\n";

cout << ">|3. Поиск по коду услуги|=<\n";

cout << ">|4. Поиск по фрагменту|===<\n";

cout << ">========|0. Выход|========<\n";

cout << endl;

mode = getInput<int>(">|Введите цифру для выбора|<\n");

switch (mode) {

case 1:

addService(oreshnik);

break;

case 8:

delService(oreshnik);

break;

case 2:

show(oreshnik);

break;

case 9:

oreshnik.clear();

break;

case 3:

search(oreshnik);

break;

case 4:

peaceSearch(oreshnik);

break;

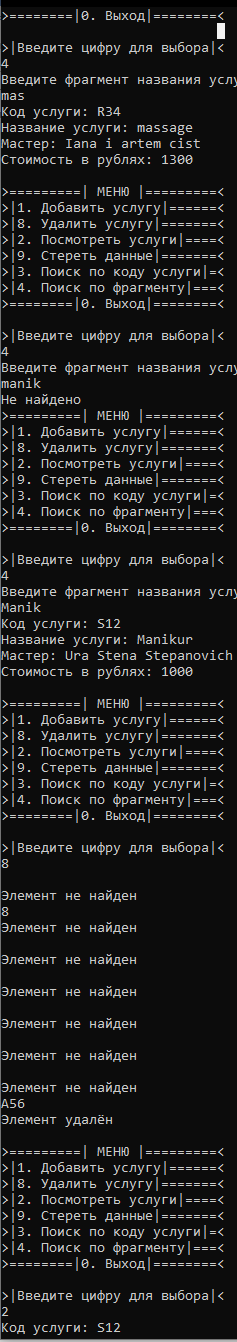
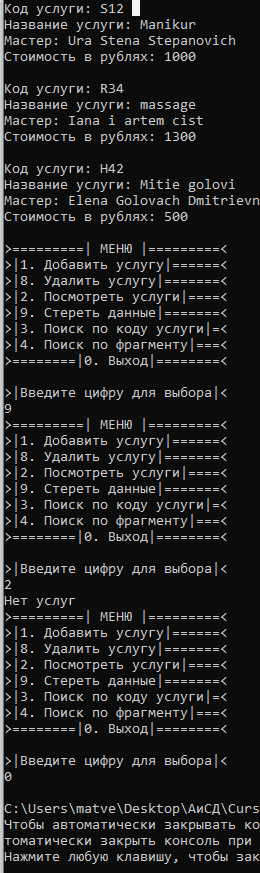
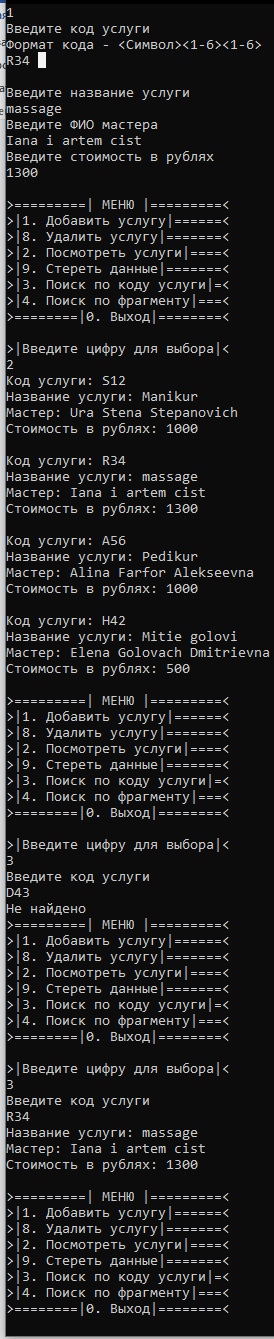
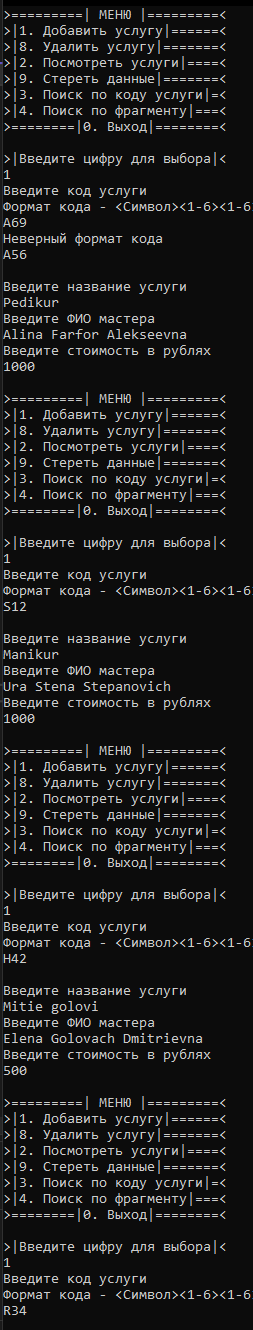
case 0:

return 0;

}

}

}

Пример работы программы:

Вывод: в процессе создания курсовой работы была создана программа хэш-таблицы данных салона красоты. Для выполнения были изучены алгоритм Кнута-Морриса-Пратта, алгоритмы и методы хеширования и принцип работы хэш-таблиц. В итоге вышла программа создания и хранения услуг, она соответствует всем указанным условиям и требованиям, работает корректно и исправно.