**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра АПУ**

**Пояснительная записка**

**к практической работе**

**на тему: «Реализация хеш-таблицы с использованием закрытого хеширования и алгоритма поиска по фрагменту строки (Рабина-Карпа)»**

**по дисциплине «Алгоритмы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3391 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Николаев В.Ю. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Ряскова Е.Б. |

Санкт-Петербург

2024 г.

Оглавление

[Введение 3](#_Toc178090119)

[Задание на практическую работу 3](#_Toc178090120)

[Описание программы 3](#_Toc178090121)

[Описание структуры данных 4](#_Toc178090122)

[Описание реализованных функций 4](#_Toc178090123)

[Тестирование программы 4](#_Toc178090124)

[Заключение 7](#_Toc178090125)

[Список использованной литературы 7](#_Toc178090126)

[Листинг кода 7](#_Toc178090127)

[bike.h 7](#_Toc178090128)

[validation.h 8](#_Toc178090129)

[validation.cpp 8](#_Toc178090130)

[search.h 8](#_Toc178090131)

[search.cpp 8](#_Toc178090132)

[hash\_table.h 9](#_Toc178090133)

[hash\_table.cpp 9](#_Toc178090134)

[main.cpp 11](#_Toc178090135)

# Введение

В данной работе требуется реализовать хеш-таблицу с закрытым хешированием и линейным опробованием для хранения данных о велосипедах в велопрокате. Основная задача программы — организовать эффективное хранение и поиск данных по ключевому полю (инвентарный номер) и по фрагменту модели с использованием алгоритма Рабина-Карпа.

Хеш-таблицы являются одним из самых быстрых методов поиска данных, когда речь идет о больших объемах информации, и важным аспектом их работы является эффективное разрешение коллизий. Для работы с текстовыми данными был выбран алгоритм Рабина-Карпа, который оптимально подходит для поиска подстрок в строках.

Цель работы — разработка программы, которая реализует хеш-таблицу и функции для управления данными о велосипедах: добавление, удаление, поиск по ключу и по фрагменту модели.

# Задание на практическую работу

Вариант 5: данные о велосипедах в велопрокате.

Каждая запись содержит:

* Инвентарный номер — строка формата «NNNNN», где N — цифры;
* Цвет — строка;
* Модель — строка;
* Дата поставки — строка формата DD.MM.YYYY.

Необходимо реализовать следующие функции:

* Добавление велосипеда.
* Удаление велосипеда по инвентарному номеру.
* Поиск велосипеда по инвентарному номеру.
* Поиск велосипеда по фрагменту модели с использованием алгоритма Рабина-Карпа.
* Вывод всех записей.
* Очистка таблицы.

Методы хеширования: закрытое хеширование с линейным опробованием для разрешения коллизий.

# Описание программы

Программа написана на языке C++ и разделена на несколько файлов:

**bike.h/bike.cpp**: Определение структуры Bike, которая содержит данные о каждом велосипеде.

**hash\_table.h/hash\_table.cpp**: Реализация хеш-таблицы с закрытым хешированием и линейным опробованием для разрешения коллизий.

**search.h/search.cpp**: Реализация алгоритма Рабина-Карпа для поиска по фрагменту строки.

**validation.h/validation.cpp**: Функции для проверки корректности вводимых данных.

**main.cpp**: Основная программа, предоставляющая интерфейс взаимодействия с пользователем через меню.

# Описание структуры данных

Для хранения данных о велосипедах используется структура Bike, которая содержит:

* **inventory\_number** — строка, содержащая инвентарный номер;
* **color** — строка, содержащая цвет велосипеда;
* **model** — строка, содержащая модель велосипеда;
* **supply\_date** — строка, содержащая дату поставки в формате DD.MM.YYYY.

Хеш-таблица реализована с закрытым хешированием и линейным опробованием:

Хеш-функция вычисляется по инвентарному номеру с использованием метода деления (остаток от деления суммы цифр на размер таблицы).

Для разрешения коллизий применяется линейное опробование (при коллизии проверяются последующие ячейки).

# Описание реализованных функций

**Добавление записи**: Пользователь может добавить новый велосипед в хеш-таблицу. Перед добавлением проверяется корректность введённых данных: инвентарный номер должен состоять из цифр, а дата должна быть в формате DD.MM.YYYY.

**Удаление записи**: Пользователь может удалить запись о велосипеде по инвентарному номеру. Для этого выполняется поиск элемента по ключевому полю.

**Поиск записи по инвентарному номеру**: Реализован метод поиска велосипеда по ключевому полю — инвентарному номеру. Если велосипед найден, программа выводит его данные.

**Поиск по фрагменту модели**: Реализован алгоритм Рабина-Карпа, который позволяет находить велосипеды по фрагменту названия модели.

**Показ всех записей**: Выводятся все записи, хранящиеся в хеш-таблице.

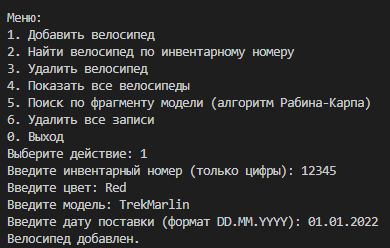
**Удаление всех записей**: Метод clear() очищает всю таблицу, удаляя все записи.

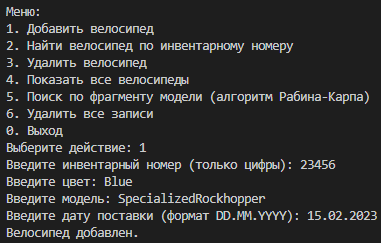
**Валидация данных**: Для всех полей проводится проверка на корректность: инвентарный номер должен содержать только цифры, дата поставки должна соответствовать формату DD.MM.YYYY.

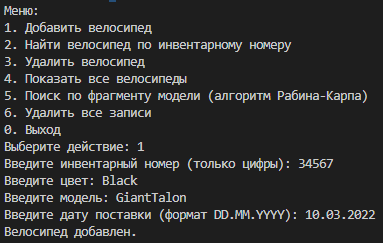
# Тестирование программы

Для тестирования программы были проведены следующие операции:

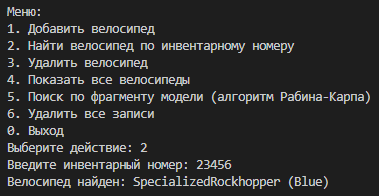
Добавление велосипедов:



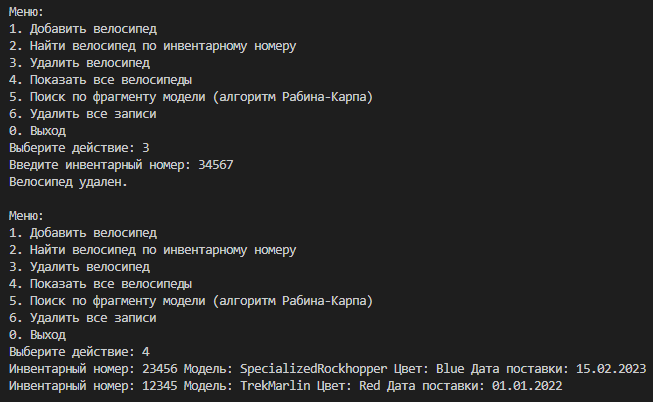




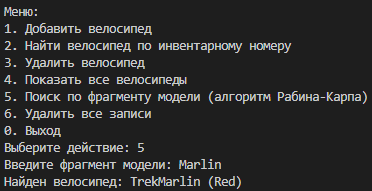
Поиск по инвентарному номеру:



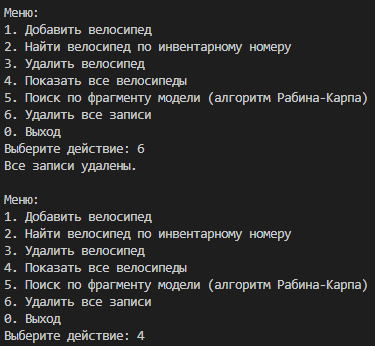
Удаление записи:



Поиск по фрагменту модели:



Очистка таблицы:



Все тесты были проведены успешно, и программа работает согласно заявленным требованиям.

# Заключение

В результате выполнения практической работы была разработана программа для управления данными велосипедов в велопрокате с использованием хеш-таблицы и алгоритма Рабина-Карпа для поиска по фрагменту строки. Программа успешно реализует добавление, удаление и поиск данных, а также проверку корректности вводимых данных.

Цель работы достигнута: программа соответствует требованиям задания и продемонстрировала правильную работу в ходе тестирования.

# Список использованной литературы

Сайт с теорией по хеш-таблицам: <https://kvodo.ru/hash-table.html>

# Листинг кода

## bike.h

|  |
| --- |
| #pragma once  #include <string>  struct Bike {      std::string inventory\_number; // Инвентарный номер (NNNNN)      std::string color;            // Цвет      std::string model;            // Модель      std::string supply\_date;      // Дата поставки (DD.MM.YYYY)        Bike(std::string inv\_num = "", std::string col = "", std::string mod = "", std::string date = "") :          inventory\_number(inv\_num), color(col), model(mod), supply\_date(date) {}  }; |

## validation.h

|  |
| --- |
| #pragma once  #include <string>  bool isNumber(const std::string& str);  bool isNotEmpty(const std::string& str);  bool isValidDate(const std::string& date); |

## validation.cpp

|  |
| --- |
| #include "validation.h"  #include <regex>  bool isNumber(const std::string& str) {      return !str.empty() && str.find\_first\_not\_of("0123456789") == std::string::npos;  }  bool isNotEmpty(const std::string& str) {      return !str.empty();  }  bool isValidDate(const std::string& date) {      std::regex date\_pattern(R"(\d{2}\.\d{2}\.\d{4})");      return std::regex\_match(date, date\_pattern);  } |

## search.h

|  |
| --- |
| #pragma once  #include <string>  bool rabinKarpSearch(const std::string& text, const std::string& pattern); |

## search.cpp

|  |
| --- |
| #include "search.h"  const int prime = 101;  bool rabinKarpSearch(const std::string& text, const std::string& pattern) {      int m = pattern.size();      int n = text.size();      if (m > n) return false;      long long patternHash = 0, textHash = 0, h = 1;        for (int i = 0; i < m - 1; i++)          h = (h \* prime);      for (int i = 0; i < m; i++) {          patternHash = (prime \* patternHash + pattern[i]);          textHash = (prime \* textHash + text[i]);      }      for (int i = 0; i <= n - m; i++) {          if (patternHash == textHash) {              int j;              for (j = 0; j < m; j++) {                  if (text[i + j] != pattern[j])                      break;              }              if (j == m)                  return true;          }          if (i < n - m) {              textHash = (prime \* (textHash - text[i] \* h) + text[i + m]);          }      }        return false;  } |

## hash\_table.h

|  |
| --- |
| #pragma once  #include <vector>  #include "bike.h"  class HashTable {  private:      std::vector<Bike> table;      int size;      int hashFunction(const std::string& key);      int linearProbe(int hash\_value, int step);    public:      HashTable(int table\_size);      bool insert(const Bike& bike);      bool remove(const std::string& inventory\_number);      Bike\* search(const std::string& inventory\_number);      std::vector<Bike> searchByModelFragment(const std::string& modelFragment);      void display();      void clear();  }; |

## hash\_table.cpp

|  |
| --- |
| #include "hash\_table.h"  #include "search.h"  #include <iostream>  int HashTable::hashFunction(const std::string& key) {      int sum = 0;      for (char ch : key) {          sum += ch;      }      return sum % size;  }  int HashTable::linearProbe(int hash\_value, int step) {      return (hash\_value + step) % size;  }  HashTable::HashTable(int table\_size) : size(table\_size) {      table.resize(size);  }  bool HashTable::insert(const Bike& bike) {      int hash\_value = hashFunction(bike.inventory\_number);      int step = 0;      while (!table[hash\_value].inventory\_number.empty() && step < size) {          hash\_value = linearProbe(hash\_value, ++step);      }      if (step >= size) {          std::cout << "Хеш-таблица заполнена\n";          return false;      }      table[hash\_value] = bike;      return true;  }  bool HashTable::remove(const std::string& inventory\_number) {      int hash\_value = hashFunction(inventory\_number);      int step = 0;      while (!table[hash\_value].inventory\_number.empty() && step < size) {          if (table[hash\_value].inventory\_number == inventory\_number) {              table[hash\_value] = Bike();              return true;          }          hash\_value = linearProbe(hash\_value, ++step);      }      return false;  }  Bike\* HashTable::search(const std::string& inventory\_number) {      int hash\_value = hashFunction(inventory\_number);      int step = 0;      while (!table[hash\_value].inventory\_number.empty() && step < size) {          if (table[hash\_value].inventory\_number == inventory\_number) {              return &table[hash\_value];          }          hash\_value = linearProbe(hash\_value, ++step);      }      return nullptr;  }  std::vector<Bike> HashTable::searchByModelFragment(const std::string& modelFragment) {      std::vector<Bike> results;      for (const auto& bike : table) {          if (!bike.inventory\_number.empty() && rabinKarpSearch(bike.model, modelFragment)) {              results.push\_back(bike);          }      }      return results;  }  void HashTable::display() {      for (const auto& bike : table) {          if (!bike.inventory\_number.empty()) {              std::cout << "Инвентарный номер: " << bike.inventory\_number                        << " Модель: " << bike.model                        << " Цвет: " << bike.color                        << " Дата поставки: " << bike.supply\_date << '\n';          }      }  }  void HashTable::clear() {      table.clear();      table.resize(size);  } |

## main.cpp

|  |
| --- |
| #include <iostream>  #include "hash\_table.h"  #include "validation.h"  int main() {      HashTable bikeTable(10);      int choice;      std::string inventory\_number, color, model, supply\_date, modelFragment;      do {          std::cout << "\nМеню:\n";          std::cout << "1. Добавить велосипед\n";          std::cout << "2. Найти велосипед по инвентарному номеру\n";          std::cout << "3. Удалить велосипед\n";          std::cout << "4. Показать все велосипеды\n";          std::cout << "5. Поиск по фрагменту модели (алгоритм Рабина-Карпа)\n";          std::cout << "6. Удалить все записи\n";          std::cout << "0. Выход\n";          std::cout << "Выберите действие: ";          std::cin >> choice;          switch (choice) {          case 1:              std::cout << "Введите инвентарный номер (только цифры): ";              std::cin >> inventory\_number;              if (!isNumber(inventory\_number)) {                  std::cout << "Ошибка: Инвентарный номер должен состоять только из цифр.\n";                  break;              }              std::cout << "Введите цвет: ";              std::cin >> color;              if (!isNotEmpty(color)) {                  std::cout << "Ошибка: Цвет не может быть пустым.\n";                  break;              }              std::cout << "Введите модель: ";              std::cin >> model;              if (!isNotEmpty(model)) {                  std::cout << "Ошибка: Модель не может быть пустой.\n";                  break;              }              std::cout << "Введите дату поставки (формат DD.MM.YYYY): ";              std::cin >> supply\_date;              if (!isValidDate(supply\_date)) {                  std::cout << "Ошибка: Дата должна быть в формате DD.MM.YYYY.\n";                  break;              }              bikeTable.insert(Bike(inventory\_number, color, model, supply\_date));              std::cout << "Велосипед добавлен.\n";              break;          case 2:              std::cout << "Введите инвентарный номер: ";              std::cin >> inventory\_number;              if (Bike\* bike = bikeTable.search(inventory\_number)) {                  std::cout << "Велосипед найден: " << bike->model << " (" << bike->color << ")\n";              } else {                  std::cout << "Велосипед не найден.\n";              }              break;          case 3:              std::cout << "Введите инвентарный номер: ";              std::cin >> inventory\_number;              if (bikeTable.remove(inventory\_number)) {                  std::cout << "Велосипед удален.\n";              } else {                  std::cout << "Велосипед не найден.\n";              }              break;          case 4:              bikeTable.display();              break;          case 5:              std::cout << "Введите фрагмент модели: ";              std::cin >> modelFragment;              {                  std::vector<Bike> results = bikeTable.searchByModelFragment(modelFragment);                  if (results.empty()) {                      std::cout << "Велосипеды не найдены.\n";                  } else {                      for (const auto& bike : results) {                          std::cout << "Найден велосипед: " << bike.model << " (" << bike.color << ")\n";                      }                  }              }              break;          case 6:              bikeTable.clear();              std::cout << "Все записи удалены.\n";              break;          case 0:              std::cout << "Выход из программы.\n";              break;          default:              std::cout << "Неверный выбор!\n";          }      } while (choice != 0);      return 0;  } |