

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема. Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА 3](#_Toc148659380)

[2 ПОДХОД К РЕШЕНИЮ 4](#_Toc148659381)

[2.1 ДВОИЧНЫЙ ФАЙЛ ИЗ ЗАПИСЕЙ ФИКСИРОВАННОГО РАЗМЕРА 4](#_Toc148659382)

[2.1.1 Структура записи 4](#_Toc148659383)

[2.1.2 Прототипы функций 4](#_Toc148659384)

[2.2 ХЕШ-ТАБЛИЦА 5](#_Toc148659385)

[2.2.1 Структура элемента таблицы 5](#_Toc148659386)

[2.2.2 Код элемента таблицы, реализация структуры таблицы 6](#_Toc148659387)

[2.2.3 Прототипы функций 6](#_Toc148659388)

[2.3 АЛГОРИТМ ПОИСКА ЗАПИСИ С ЗАДАННЫМ КЛЮЧЕМ В ФАЙЛЕ 7](#_Toc148659389)

[3 КОД ПРИЛОЖЕНИЯ 8](#_Toc148659390)

[3.1 КОД ЗАГОЛОВОЧНЫХ МОДУЛЕЙ 8](#_Toc148659391)

[3.1.1 Код файла HashTable.h 8](#_Toc148659392)

[3.1.2 Код файла BinHash.h 8](#_Toc148659393)

[3.1.3 Код файла testBinF.h 9](#_Toc148659394)

[3.2 КОД ФАЙЛОВ РЕАЛИЗАЦИИ 9](#_Toc148659395)

[3.2.1 Код файла HashTable.cpp 9](#_Toc148659396)

[3.2.2 Код файла BinHash.cpp 13](#_Toc148659397)

[3.2.3 Код файла testBinF.cpp 14](#_Toc148659398)

[3.3 КОД ОСНОВНОЙ ПРОГРАММЫ 16](#_Toc148659399)

[4 ТЕСТИРОВАНИЕ ПРОГРАММЫ 22](#_Toc148659402)

[4.1 ТЕСТИРОВАНИЕ ФУНКЦИЙ ХЕШ-ТАБЛИЦЫ 22](#_Toc148659403)

[4.2 ТЕСТИРОВАНИЕ ФУНКЦИЙ УПРАВЛЕНИЯ БИНАРНЫМ ФАЙЛОМ ПОСРЕДСТВОМ ХЕШ-ТАБЛИЦЫ 28](#_Toc148659404)

[5 ЧТЕНИЕ ЗАПИСИ ИЗ БОЛЬШОГО ФАЙЛА 33](#_Toc148659405)

[6 ВЫВОДЫ 34](#_Toc148659406)

[7 ПРИЛОЖЕНИЕ 35](#_Toc148659407)

[8 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 36](#_Toc148659408)

# 1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА

Разработать приложение, которое использует хеш-таблицу для организации прямого доступа к записям файла, структура записи которого приведена в варианте. Задание варианта:

|  |  |
| --- | --- |
| № | Метод хеширования (способ реализации коллизий) |
| 13 | Открытый адрес (двойное хеширование) |

Дано: Файл двоичный с записями фиксированной длины. Структура записи файла согласно варианту:

|  |  |  |  |
| --- | --- | --- | --- |
| Назначение | Номер зачетной книжки | Номер группы | ФИО |
| Поле | studentID | group | full\_name |

Результат: Хеш-таблица

# 2 ПОДХОД К РЕШЕНИЮ

## 2.1 ДВОИЧНЫЙ ФАЙЛ ИЗ ЗАПИСЕЙ ФИКСИРОВАННОГО РАЗМЕРА

### 2.1.1 Структура записи

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 1 – Структура записи файла

Произведем расчет размера одной структуры в байтах: 40 \* 1 + 40 \* 1 + 80 \* 1 = 160 байт.

Системный расчет размера одной структуры в байтах представлен на рисунке № 2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2 - Системный расчет размера одной структуры

### 2.1.2 Прототипы функций

1. Создание бинарного файла из текстового

Предусловие. Объект класса ifstream, передаваемый по ссылке, объект класса ofstream, передаваемый по ссылке.

Постусловие. Нет возвращаемого значения.

**void create\_bin\_file(ifstream& output, ofstream& input);**

1. Сохранение данных двоичного файла в текстовом

Предусловие. Объект класса ifstream, передаваемый по ссылке, объект класса ofstream, передаваемый по ссылке.

Постусловие. Заполняет текстовый файл, нет возвращаемого значения.

**void write\_from\_bin\_to\_text(ifstream& output, ofstream& input);**

1. Вывод всех записей двоичного файла

Предусловие. Объект класса ifstream, передаваемый по ссылке.

Постусловие. Вывод записей файла в консоль. Нет возвращаемого значения.

**void output\_bin(ifstream& output);**

1. Возвращение записи по индексу

Предусловие. Объект класса ifstream, передаваемый по ссылке, объект структуры Student, целочисленное значение n – индекс.

Постусловие. Объект класса Student.

**Student get\_student(ifstream& output, int n, Student& stud);**

1. Удаление записи по ключу

Предусловие. Объект класса fstream, передаваемый по ссылке, целочисленная переменная n, строковая переменная bin\_file.

Постусловие. Указатель на объект класса Student.

**Student\* delete\_student(fstream& f, int n, string bin\_file);**

## 2.2 ХЕШ-ТАБЛИЦА

### 2.2.1 Структура элемента таблицы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Поле | Строковое значение key | Целочисленное значение pos | Булево значение is\_deleted | Булево значение is\_busy | Целочисленное значение attempt |
| Назначение | Номер зачетки | Номер записи в файле | Удалена ли запись | Занята ли ячейка | Попытка обратиться к элементу |

## 2.2.2 Код элемента таблицы, реализация структуры таблицы

Изображение выглядит как текст, снимок экрана, Шрифт, дисплей

Автоматически созданное описание

Рисунок 3 - Код элемента таблицы, реализация структуры таблицы

### 2.2.3 Прототипы функций

1. Вставка ключа в таблицу

Предусловие. Объект структуры HashTable, передаваемый по ссылке, объект структуры element elem.

Постусловие. Нет возвращаемого значения.

**void add\_student(HashTable& table, element elem);**

1. Поиск записи по ключу в таблице и возвращение номера записи в файле

Предусловие. Объект структуры HashTable, передаваемый по ссылке, строковое значение key.

Постусловие. Номер записи в файле.

**long long int find\_student(string key, HashTable& table);**

1. Удаление элемента из хеш-таблицы

Предусловие. Объект структуры HashTable, передаваемый по ссылке, строковое значение key.

Постусловие. Номер записи в файле, -1 при отсутствии.

**long long int delete\_student(string key, HashTable& table);**

## 2.3 АЛГОРИТМ ПОИСКА ЗАПИСИ С ЗАДАННЫМ КЛЮЧЕМ В ФАЙЛЕ

Эта операция имеет следующий алгоритм:

1. Получение ключа поиска К.

2. Определение индекса элемента в таблице по ключу К посредством хеш-функции, которая использовалась при вставке значений в таблицу.

3. Проверка, содержит ли элемент, ключ поиска:

3.1. Если этот элемент содержит введенный ключ, то поиск завершается и возвращаются данные записи;

3.2. Если элемент таблицы по этому индексу не содержит заданный ключ и индекс не вышел за границу таблицы, то осуществляется алгоритм подбора нового индекса по тому же алгоритму, который использовался при выполнении операции вставки в случае коллизии, пока не будет найден элемент с таким ключом, или не будет найдена открытый адрес. Если найден открытый адрес, то это означает, что такого ключа в таблице нет.

Также при поиске записи с заданным ключом, в случае свободной ячейки надо проверять и значение поля Признак удаления.

# 3 КОД ПРИЛОЖЕНИЯ

## 3.1 КОД ЗАГОЛОВОЧНЫХ МОДУЛЕЙ

### 3.1.1 Код файла HashTable.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

struct element

{

string key;

long long int pos;

bool is\_busy = 0;

bool is\_deleted = 0;

int attempt = 0;

element();

element(string key, long long int pos);

};

struct HashTable

{

long long int total = 0;

long long int size = 10;

element\* data = new element[size];

HashTable();

};

int hash\_func1(string key, long long int size);

int hash\_func2(string key, long long int size);

void add\_student(HashTable& table, element elem);

void print\_hash\_table(const HashTable& table);

long long int find\_student(string key, HashTable& table);

long long int delete\_student(string key, HashTable& table);

void rehash\_table(HashTable& table);

### 3.1.2 Код файла BinHash.h

#pragma once

#include "HashTable.h"

#include "testBinF.h"

void create\_hash\_table(HashTable& table, ifstream& file);

Student find\_el(HashTable& table, ifstream& file, string key);

void delete\_el(HashTable& table, fstream& file, string key, string filename);

### 3.1.3 Код файла testBinF.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <io.h>

#include <windows.h>

#include <fcntl.h>

#include <share.h>

#include <sys/stat.h>

#include <string>

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

#include <string.h>

#include <chrono>

using namespace std;

struct Student

{

char studentID[40];

char group[40];

char full\_name[80];

Student();

~Student();

};

void create\_bin\_file(ifstream& output, ofstream& input);

void write\_from\_bin\_to\_text(ifstream& output, ofstream& input);

void output\_bin(ifstream& output);

Student get\_student(ifstream& output, int n,Student& stud);

Student\* delete\_student(fstream& f, long long int n, string bin\_file);

## 3.2 КОД ФАЙЛОВ РЕАЛИЗАЦИИ

### 3.2.1 Код файла HashTable.cpp

#include "HashTable.h"

element::element()

{

}

element::element(string key, long long int pos)

{

this->key = key;

this->pos = pos;

}

HashTable::HashTable()

{

}

int hash\_func1(string key, long long int size)

{

long long index = 0;

long long p = 1;

for (char symbol : key)

{

index += abs(symbol) \* p;

p \*= 31;

}

return abs(index % size);

}

int hash\_func2(string key, long long int size)

{

long long index = 0;

long long p = 1;

for (char symbol : key)

{

index += abs(symbol) \* p;

p \*= 31;

}

if (abs(index / size) % size == 0)

{

return abs(index / size) + 1;

}

return abs(index / size);

}

void add\_student(HashTable& table, element elem)

{

table.total++;

while (true)

{

long long int hash = (hash\_func1(elem.key, table.size) + elem.attempt \* hash\_func2(elem.key, table.size)) % table.size;

if (not table.data[hash].is\_busy)

{

table.data[hash] = elem;

table.data[hash].is\_busy = 1;

break;

}

else

{

elem.attempt++;

}

}

if (table.total > 0.75 \* table.size)

{

rehash\_table(table);

}

}

void print\_hash\_table(const HashTable& table)

{

for (int i = 0; i < table.size; i++)

{

cout << i << ": ";

cout << table.data[i].key;

cout << "\n";

}

}

long long int find\_student(string key, HashTable& table)

{

int attempt = 0;

while (true)

{

int hash = (hash\_func1(key, table.size) + attempt \* hash\_func2(key, table.size)) % table.size;

if (not table.data[hash].is\_busy && not table.data[hash].is\_deleted)

{

return -1;

}

else

{

if (table.data[hash].key == key)

return table.data[hash].pos;

attempt++;

}

}

}

long long int delete\_student(string key, HashTable& table)

{

int attempt = 0;

while (true)

{

int hash = (hash\_func1(key, table.size) + attempt \* hash\_func2(key, table.size)) % table.size;

if (not table.data[hash].is\_busy && not table.data[hash].is\_deleted)

{

return -1;

}

else

{

if (table.data[hash].key == key)

{

table.data[hash].is\_deleted = 1;

table.data[hash].is\_busy = 0;

table.data[hash].key = "";

return 0;

}

attempt++;

}

}

}

void rehash\_table(HashTable& table)

{

cout << " Rehash: \n";

HashTable new\_table;

new\_table.size = table.size \* 2;

new\_table.data = new element[new\_table.size];

for (int i = 0; i < table.size; i++)

{

element elem;

elem.key = table.data[i].key;

elem.pos = table.data[i].pos;

while (true)

{

long long int hash = (hash\_func1(elem.key, new\_table.size) + elem.attempt \* hash\_func2(elem.key, new\_table.size)) % new\_table.size;

if (not new\_table.data[hash].is\_busy)

{

new\_table.data[hash] = elem;

new\_table.data[hash].is\_busy = 1;

break;

}

else

{

elem.attempt++;

}

}

}

delete[] table.data;

table = new\_table;

}

### 3.2.2 Код файла BinHash.cpp

#include "BinHash.h"

#include "HashTable.h"

void create\_hash\_table(HashTable& table, ifstream& file)

{

Student Vlad;

element elem;

int position = 0;

file.read((char\*)&Vlad, sizeof(Student));

while (!file.eof())

{

elem.pos = position;

elem.key = Vlad.studentID;

add\_student(table, elem);

position++;

file.read((char\*)&Vlad, sizeof(Student));

}

}

Student find\_el(HashTable& table, ifstream& file, string key)

{

Student stud;

long long int n = find\_student(key,table);

return get\_student(file, n,stud);

}

void delete\_el(HashTable& table, fstream& file, string key, string filename)

{

long long int n = delete\_student(key,table);

Student\* Vlad = delete\_student(file, n, filename);

if (Vlad == nullptr)

{

return;

}

if (Vlad->studentID != key)

{

element elem(Vlad->studentID, n);

delete\_student(elem.key,table);

add\_student(table, elem);

}

}

### 3.2.3 Код файла testBinF.cpp

#include "testBinF.h"

Student::Student()

{

this->studentID[0] = '\0';

this->group[0] = '\0';

this->full\_name[0] = '\0';

}

Student::~Student()

{

this->studentID[0] = '\0';

this->group[0] = '\0';

this->full\_name[0] = '\0';

}

void create\_bin\_file(ifstream& output, ofstream& input)

{

while (!output.eof())

{

Student Vlad;

char symbol;

output.getline(Vlad.studentID, 40);

output.getline(Vlad.group, 40);

output.getline(Vlad.full\_name, 80);

input.write((char\*)&Vlad, sizeof(Student));

}

}

void write\_from\_bin\_to\_text(ifstream& output, ofstream& input)

{

Student Vlad;

output.read((char\*)&Vlad, sizeof(Vlad));

while (!output.eof())

{

input << Vlad.studentID << '\n'

<< Vlad.group << '\n'

<< Vlad.full\_name;

output.read((char\*)&Vlad, sizeof(Vlad));

if (!output.eof())

{

input << '\n';

}

}

}

void output\_bin(ifstream& output)

{

Student Vlad;

output.read((char\*)&Vlad, sizeof(Student));

while (!output.eof())

{

cout << Vlad.studentID << " | " << Vlad.group << " | " << Vlad.full\_name << "\n";

output.read((char\*)&Vlad, sizeof(Vlad));

}

}

Student get\_student(ifstream& output, int n,Student& stud)

{

output.seekg(n \* sizeof(Student), ios::beg);

if (!output.bad())

{

output.read((char\*)&stud, sizeof(Student));

return Student();

}

return stud;

}

Student\* delete\_student(fstream& f, long long int n, string bin\_file)

{

f.seekg(0, ios::end);

int size = f.tellg();

if (n \* sizeof(Student) >= size && n >= 0)

{

return nullptr;

}

Student\* lastRecord = new Student(), \* record = new Student();

f.seekg((-1) \* sizeof(Student), ios::end);

f.read((char\*)lastRecord, sizeof(Student));

f.seekg(n \* sizeof(Student), ios::beg);

f.read((char\*)record, sizeof(Student));

record = lastRecord;

f.seekg((-1) \* sizeof(Student), ios::cur);

f.write((char\*)record, sizeof(Student));

f.close();

int desc = \_open(bin\_file.c\_str(), O\_RDWR);

\_chsize\_s(desc, size - sizeof(Student));

\_close(desc);

f.open(bin\_file, ios::binary);

return lastRecord;

}

## 3.3 КОД ОСНОВНОЙ ПРОГРАММЫ

### 3.3.1 Код файла Table\_main.cpp

#include "HashTable.h"

#include "testBinF.h"

int main()

{

system("chcp 1251");

HashTable table;

int num;

long long int position = 0;

long long int res = 0;

string key;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Добавить элемент в таблицу\n";

cout << " 2. Вывести хеш-таблицу\n";

cout << " 3. Найти элемент по ключу в таблице\n";

cout << " 4. Удалить элемент по ключу из таблицы\n";

cout << " 5. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите ключ записи: ";

cin.get();

getline(cin, key);

element elem(key, position++);

add\_student(table, elem);

break;

}

case 2:

{

print\_hash\_table(table);

break;

}

case 3:

{

cout << " Введите ключ для поиска: ";

cin >> key;

res = find\_student(key, table);

if (res == -1)

cout << " Элемент не найден";

else cout << res << endl;

break;

}

case 4:

{

cout << " Введите ключ для поиска: ";

cin >> key;

res = find\_student(key, table);

if (res == -1)

cout << " Элемент не найден";

break;

}

case 5:

{

return 0;

}

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

### 3.3.2 Код файла Table\_BINmain.cpp

#include "BinHash.h"

#include "HashTable.h"

int main()

{

system("chcp 1251");

HashTable table;

int num;

ifstream output;

ofstream input;

string first\_file, second\_file;

fstream f;

string key;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Преобразовать текстовый файл в двоичный\n";

cout << " 2. Перевод двоичного файла в хэш-таблицу\n";

cout << " 3. Поиск записи\n";

cout << " 4. Удаление записи\n";

cout << " 5. Вывести хэш-таблицу\n";

cout << " 6. Вывести содержимое двоичного файла в консоль\n";

cout << " 7. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите название текстового файла: ";

cin >> first\_file;

cout << " Введите название двоичного файла: ";

cin >> second\_file;

output.open(first\_file);

if (!output.is\_open())

{

cout << " Текстовый файл не открыт\n";

}

else

{

input.open(second\_file, ios::binary);

create\_bin\_file(output, input);

if (!input.good())

{

cout << " Ошибка записи в двоичный файл\n";

return -1;

}

else

{

output.close();

input.close();

}

}

}

case 2:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

create\_hash\_table(table, output);

if (output.bad())

{

cout << " Возникла ошибка создания хэш-таблицы\n";

return 1;

}

else

{

cout << " Создание хэш-таблицы произошло успешно\n";

output.close();

}

}

else

{

cout << " Двоичного файла не существует\n";

}

break;

}

case 3:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (output.is\_open())

{

cout << "Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

auto start = chrono::high\_resolution\_clock::now();

Student stud = find\_el(table, output, key);

auto end = chrono::high\_resolution\_clock::now();

cout << " Время поиска: " << chrono::duration\_cast<chrono::nanoseconds>(end - start).count() / 1e6 << " мс\n";

if (output.bad())

{

cout << " Возникла ошибка чтения\n";

return 1;

}

else

{

if (stud.studentID != "")

cout << stud.studentID << " | " << stud.group << " | " << stud.full\_name << "\n";

else

cout << " Запись не найдена\n";

output.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 4:

{

cout << "Введите название двоичного файла: ";

cin >> first\_file;

f.open(first\_file, ios::in | ios::out | ios::binary);

if (f.is\_open())

{

string key;

cout << " Введите номер зачетной книжки: ";

cin.get();

getline(cin, key);

delete\_el(table, f, key, first\_file);

if (output.bad())

{

cout << " Возникла ошибка при удалении\n";

return 1;

}

else

{

cout << " Запись успешно удалена!\n";

f.close();

}

}

else

{

cout << " Двоичный файл не существует\n";

}

break;

}

case 5:

{

print\_hash\_table(table);

break;

}

case 6:

{

cout << " Введите название двоичного файла: ";

cin >> first\_file;

output.open(first\_file, ios::binary);

if (!output.is\_open())

{

cout << " Двоичный файл не открыт\n";

}

else

{

output\_bin(output);

if (output.bad())

{

cout << " Ошибка при чтении из двоичного файла\n";

return 1;

}

else

{

output.close();

}

}

break;

}

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

# 4 ТЕСТИРОВАНИЕ ПРОГРАММЫ

## 4.1 ТЕСТИРОВАНИЕ ФУНКЦИЙ ХЕШ-ТАБЛИЦЫ

1) Вставка элемента в хеш-таблицу без коллизии

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 – Тестирование функции вставки элемента в хеш-таблицу без коллизии

2) Вставка элемента в хеш-таблицу с коллизией

Подберем ключи с одинаковым хешем, например:



Рисунок 5 – Две записи с одинаковым хешем

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

Рисунок 6 – Хеш записей

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 - Тестирование функции вставки элемента в хеш-таблицу с коллизией

3) Добавление элемента с последующим рехешированием

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 8 - Тестирование функции вставки элемента в хеш-таблицу с последующим рехешированием

4) Удаление ключа из таблицы

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 9 – Результат тестирования функции удаления записи

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 10 - Результат тестирования функции удаления записи

5) Поиск ключа в таблице

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 11 - Результат тестирования функции поиска записи

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 12 – Результат тестирования функции поиска записи

6) Поиск после удаления

Добавим две записи с одинаковым хешем, представленные выше, удалим первую добавленную из них.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 13 - Результат тестирования функции поиска записи после удаления

## 4.2 ТЕСТИРОВАНИЕ ФУНКЦИЙ УПРАВЛЕНИЯ БИНАРНЫМ ФАЙЛОМ ПОСРЕДСТВОМ ХЕШ-ТАБЛИЦЫ

1) Обработка коллизии в хеш-таблице

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 14 – Создание хеш-таблицы на основе двоичного файла

2) Поиск элемента в таблице

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 15 – Содержание файла

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 16 – Успешный поиск элемента

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 17 – Безуспешный поиск

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 18 – Удаление записи

# 5 ЧТЕНИЕ ЗАПИСИ ИЗ БОЛЬШОГО ФАЙЛА

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 19 – Поиск последней записи

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 20 – Поиск первой записи

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 21 – Поиск произвольной записи в середине

# 6 ВЫВОДЫ

В ходе выполнения практической работы были получены навыки работы с хеш-таблицей. Время доступа для всех записей в хеш-таблице одинаково.

# 7 ПРИЛОЖЕНИЕ

Для генерации файла из миллиона записей был написан генератор на языке Python для «случайного» номера зачетки, группы и фио.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 22 – Генератор записей в файле

# 8 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 1)
2. Приложение к практическим работам – СДО (online-edu.mirea.ru)