|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | МИНОБРНАУКИ РОССИИ | | | | 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 ЛАБОРАТОРНОЙ РАБОТЕ №5** | |
| **по дисциплине** | |
| **«Структуры и алгоритмы обработки данных»**  **По теме: «Структура данных - дерево»**  **Вариант 2** | |
|  | |
| Выполнил студент группы ИВБО-06-17 | Бикеев А. И. |
| Принял преподаватель | Скворцова Л.А. |

Москва 2019

**Оглавление**

[1. Задание 1. Разработайте шаблонный класс – хеш таблица, для поиска информации в коллекциях данных с помощью хеш функции. Хеш функцию подберите самостоятельно. 3](#_Toc26233195)

[1.1. Вариант задания 3](#_Toc26233196)

[1.2. Абстрактный тип данных (далее АТД) для варианта задания, включая список общих функций из задания. 3](#_Toc26233197)

[1.3. Реализация АТД 3](#_Toc26233198)

[Список модулей реализации АТД (или описать где расположена 4](#_Toc26233199)

[реализация АТД) 4](#_Toc26233200)

[1.4. Текст исходного кода (листинг) программы 4](#_Toc26233201)

[Задание 2. 5](#_Toc26233202)

[2.1. Вариант задания 5](#_Toc26233203)

[2.2. Реализация АТД 6](#_Toc26233204)

[2.3. Таблица тестов 10](#_Toc26233205)

[3. Дополнительные задания 10](#_Toc26233206)

[3.1. Вариант задания 10](#_Toc26233207)

[3.2. Вариант задания 10](#_Toc26233208)

[3.2. Код основной программы 11](#_Toc26233209)

[3.4. Общие контрольные прогоны программы 13](#_Toc26233210)

[Выводы 16](#_Toc26233211)

1. Задание 1. Разработайте шаблонный класс – хеш таблица, для поиска информации в коллекциях данных с помощью хеш функции. Хеш функцию подберите самостоятельно.

1.1. Вариант задания

Разработайте шаблонный класс – хеш таблица. В качестве хэш функции – подобрать самостоятельно

## 1.2. Абстрактный тип данных (далее АТД) для варианта задания, включая список общих функций из задания.

Абстрактный тип данных BanckAccount

**account\_number** – номер счета в банке

**FIO** – ФИО клиента банка

**address** – адрес клиента банка

Операции:

// Создает новый аккаунт для клиента банка по введенным в консоль данным

Node();

// Копирующий конструктор

Node(const Node& obj);

## 1.3. Реализация АТД

typedef struct Node

{

/// <summary>

/// Номер счета в банке

/// </summary>

int account\_number;

/// <summary>

/// ФИО - клиента банка

/// </summary>

string FIO;

/// <summary>

/// Адрес проживания клиента

/// </summary>

string address;

Node()

{

cout << "Введите номер счета в банке: ";

cin >> this->account\_number;

cout << "Введите ФИО: ";

cin >> this->FIO;

cout << "Введите адрес проживания: ";

cin >> this->address;

}

Node(const Node& obj) {

this->account\_number = obj.account\_number;

this->address = obj.address;

this->FIO = obj.FIO;

}

}BankAccount;

Список модулей реализации АТД (или описать где расположена

реализация АТД)

Объявление АТД находится в файле BanckAccount.h, а реализация в Operation.cpp

## 1.4. Текст исходного кода (листинг) программы

#include "Operation.h"

void PrintHeaderTable(string h1, string h2, string h3)

{

    cout << ".";

    for (size\_t i = 0; i < 60; i++) cout << "\_";

    cout << ".";

    cout << endl << "|" << setw(20) << left

        << h1 << setw(20) << left

        << "|" + h2 << setw(20)

        <<"|" + h3 << setw(20) << "|" << endl;

    cout << ".";

    for (size\_t i = 0; i < 60; i++) cout << "\_";

    cout << "."<<endl;

}

void CreateTable(HashTable<int, BankAccount\*> \*table, int n)

{

    BankAccount\* account = nullptr;

    for (size\_t i = 0; i < n; i++)

    {

        account = new BankAccount();

        table->Add(account->account\_number, account);

    }

    cout << "Хэш-таблицу сформирована" << endl;

}

void Print(HashTableNodePair<int, BankAccount\*> \*mass, int n)

{

    PrintHeaderTable("INDEX", "KEY", "VALUE");

    for (size\_t i = 0; i < n; i++)

    {

        if (!mass[i].GetIsVoid())

        {

            cout << endl << "|" << setw(20) << left

                << i << setw(20) << left

                << "|" + mass[i].GetKey() << setw(20)

                << "|" + mass[i].GetValue()->FIO << setw(20) << "|" << endl;

            cout << ".";  for (size\_t i = 0; i < 60; i++) cout << "\_"; cout << ".";

            cout << endl;

        }

        else {

            cout << endl << "|" << setw(20) << left

                << i << setw(20) << left

                << "|NULL" << setw(20)

                << "|NULL" << setw(20) << "|" << endl;

            cout << ".";  for (size\_t i = 0; i < 60; i++) cout << "\_"; cout << ".";

            cout << endl;

        }

    }

}

Задание 2.

Разработайте приложение, которое использует хеш таблицу для организации прямого доступа к элементам множества (массив данных), структура элементов которого приведена в варианте. Множество реализуйте через класс - шаблон с операциями вставки, удаления, поиска, вывода и включите в него хеш таблицу. Предусмотрите возможность создания наследников хеш-таблицы

2.1. Вариант задания

Требования

1. Метод хеширования (способ реализации коллизий) - С открытой адресацией(увеличение на номер выполняемого подбора)
2. Структура элемента множества(ключ – подчеркнутое поле) - Счет в банке: номер счета 7 разрядное число, ФИО, Адрес

## 2.2. Реализация АТД

#pragma once

/// <summary>

/// Шаблонный класс для узлов Хэш-Таблицы

/// TKey - тип для ключа

/// TValue - тип для значения

/// </summary>

template<class TKey, class TValue>

class HashTableNodePair

{

private:

    TKey key;

    TValue value;

    bool isVoid; // Указывает - удален ли элемент

public:

    HashTableNodePair()

    {

        this->isVoid = true;

    }

    /// <summary>

    /// Инициализирует новый элемент в хэш-таблице

    /// </summary>

    /// <param name="key">Ключ в хэш-таблице</param>

    /// <param name="value">Значение соответствующее данному ключу</param>

    HashTableNodePair(TKey key, TValue value)

    {

        this->key = key;

        this->value = value;

        this->isVoid = false;

    }

    TValue GetValue() {

        return value;

    }

    TKey GetKey() {

        return key;

    }

    bool GetIsVoid() {

        return this->isVoid;

    }

    void SetValue(TValue value) {

        this->value = value;

    }

    void SetIsVoid(bool value) {

        this->isVoid = value;

    }

};

/// <summary>

/// Шаблонный класс Хэш-Таблицы

/// TKey - тип для ключа

/// TValue - тип для значения

/// </summary>

template<class TKey, class TValue>

class HashTable

{

private:

    /// <summary>

    /// Массив узлов хэш-таблицы

    /// </summary>

    HashTableNodePair<TKey, TValue> \*mass;

    /// <summary>

    /// Размер хэш-таблицы

    /// </summary>

    int size;

    /// <summary>

    /// Кол-во непустых клеток в таблице

    /// </summary>

    int count\_not\_void;

    /// <summary>

    /// Хэщ-функция

    /// </summary>

    /// <param name="key">Ключ элемента из хэш-таблицы</param>

    /// <param name="attempt">Номер попытки</param>

    /// <returns></returns>

    int GetIndex(TKey key, int attempt)

    {

        return (sizeof(key) % 12 + attempt) % size;

    }

public:

    /// <summary>

    /// Инициализирует хэш-таблицу с заданным размером

    /// </summary>

    /// <param name="size">Размер хэш-таблицы</param>

    HashTable(int size)

    {

        this->size = size;

        this->count\_not\_void = size;

        mass = new HashTableNodePair<TKey, TValue>[size];

    }

    HashTableNodePair<TKey, TValue>\* GetMass()

    {

        return this->mass;

    }

    /// <summary>

    /// Возвращает кол-во пустых клеток в хэш-таблицы

    /// </summary>

    /// <returns></returns>

    int GetVoidCount()

    {

        return this->count\_not\_void;

    }

    /// <summary>

    /// Добавляет новый элемент в хэш таблицу

    /// </summary>

    /// <param name="key">Ключ элемента из хэш-таблицы</param>

    /// <param name="value">Значение соответствующее данному ключу</param>

    void Add(TKey key, TValue value)

    {

        // Если указанного элемента нет в таблице

        if (this->FindNode(key) == -1)

        {

            int attempt = 0; // кол-во попыток добавить новый элемент

            int index; // индекс для нового элемента

            HashTableNodePair<TKey, TValue> \*node = 0; // Новый элемент на добавление в таблицу

            do

            {

                index = this->GetIndex(key, attempt);

                node = &this->mass[index];

                attempt++;

            } while (node && node->GetIsVoid() == false && attempt != size); // пока существует по заданному ключу элемент

            if ( (node && node->GetIsVoid()) || !node) { // Если по заданному ключу - элемент удален

                this->mass[index] = HashTableNodePair<TKey, TValue>(key, value);

                this->count\_not\_void--;

            }

            else {

                throw exception("Таблица заполнена");

            }

        }

        else {

            throw exception("Элемент с указанным ключем и значением уже существует");

        }

    }

    /// <summary>

    /// Удаляет элемент из таблицы с заданным ключом и значением

    /// </summary>

    /// <param name="key">Ключ элемента из хэш-таблицы</param>

    /// <returns></returns>

    bool Remove(TKey key)

    {

        // Если в таблице есть элементы

        if (size)

        {

            int index = this->FindNode(key);

            // Если нашелся элемент с заданным ключем и значением в таблице

            if (index != -1)

            {

                this->mass[index].SetIsVoid(true); // Помечаем его удаленным

            }

            else {

                throw exception("Элемента с указанным ключем и значением - нет в таблице");

            }

        }

    }

    /// <summary>

    /// Возвращает индекс элемента с заданным ключем и значением в массиве хэш-таблицы

    /// или -1 в случаи если такой элемент не нашелся

    /// </summary>

    /// <param name="key">Ключ элемента из хэш-таблицы</param>

    /// <returns></returns>

    int FindNode(TKey key)

    {

        int attempt = 0; // кол-во попыток поиска ключа

        int index; // Индекс искомого элемента

        bool flag = false; // Флаг указывающий, что ключ подошел

        HashTableNodePair<TKey, TValue> \*node = 0;

        do

        {

            index = this->GetIndex(key, attempt);

            node = &this->mass[index];

            attempt++;

            if (typeid(TValue) == typeid(BankAccount\*))

            {

                flag = node && !node->GetIsVoid() ? (node->GetValue()->account\_number == key ? true : false) : false;

                if (flag) break;

            }

            else {

                flag = true;

            }

        } while (node && attempt != size); // пока не найдется нужный элемент с заданным ключем и значением

        // Если нашелся элемент с заданным ключем и значением в таблице

        if (node  && flag) {

            return index;

        }

        else {

            return -1;

        }

    }

};

Список модулей реализации АТД (или описать где расположена

реализация АТД)

Реализация АТД расположена в файле: HashTable.h

## 2.3. Таблица тестов

|  |  |  |
| --- | --- | --- |
| Номер теста | Исходные данные | Эталон результата |
| 1 | Укажите размер хэш-таблицы: 5 | Таблица успешно создана |
| 2 | Введите номер счета в банке: 4  Введите ФИО: 1  Введите адрес проживания: 11 | Данные успешно добавлены |
| 3 | Введите номер карты клиента: 4 | ФИО: 1  Адрес: 11 |

3. Дополнительные задания

3.1. Вариант задания

Разработайте такие тесты, чтобы возникли коллизии..

3.2. Вариант задания

Для разрешения коллизий использовать открытую адресацию с увеличением номера выполняемого подбора

3.2. Код основной программы

#include <windows.h>

#include "Operation.h"

int main()

{

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    short answer = 10;

    int n = 0;

    HashTable<int, BankAccount\*> \*table = nullptr;

    while (answer != 0) {

        system("cls");

        cout << "Лабораторная работа №5 Бикеева А. И. Вариант 2\n\n";

        cout << "Тема: Хеширование для организации быстрого поиска данных\n";

        cout << "Цель: Получить навыки по разработке хеш таблиц\n";

        cout << "\nМеню\n";

        cout << "1) Сформировать/пересоздать хэш-таблицу из n элементов\n";

        cout << "2) Вывести данные клиента по номеру карты\n";

        cout << "3) Вывести хэш-таблицу\n";

        cout << "4) Добавить инф. о новом клиенте хэш-таблицу\n";

        cout << "5) Удалить инф. о клиенте с указанным номером карты\n";

        cout << "0) Выход\n";

        cout << "\nВаш выбор: ";

        while ((!(cin >> answer)) || cin.get() != '\n') {     //делаем проверку на ввод букв

            cout << "ERROR\n";

            cin.clear(); // сбрасывает все биты потока, тоесть поток становится "хорошим"

            cin.sync();//Удалим данные из буффера

            cout << "Повторите ввод : ";

        }

        system("cls");

        cout << "Лабораторная работа №5 Бикеева А. И. Вариант 2\n\n";

        try

        {

            switch (answer)

            {

            case 1: {

                cout << "Укажите размер хэш-таблицы: ";

                cin >> n;

                table = new HashTable<int, BankAccount\*>(n);

                cout << "Таблица успешно создана\n";

                system("pause");

                break;

            }

            case 2: {

                if (table)

                {

                    cout << "Введите номер карты клиента: ";

                    int num = 0;

                    cin >> num;

                    int i = table->FindNode(num);

                    if (i != -1) {

                        HashTableNodePair<int, BankAccount\*>\* mass = table->GetMass();

                        cout << "ФИО: " << mass[i].GetValue()->FIO << endl;

                        cout << "Адрес: " << mass[i].GetValue()->address << endl;

                    }

                    else {

                        cout << "Клиента с указанным номером - нет в таблице\n";

                    }

                }

                else {

                    cout << "Хэш-таблица пустая" << endl;

                }

                system("pause");

                break;

            }

            case 3: {

                if (table)

                {

                    Print(table->GetMass(), n);

                }

                else {

                    cout << "Хэш-таблица пустая" << endl;

                }

                system("pause");

                break;

            }

            case 4: {

                if (table && table->GetVoidCount())

                {

                    BankAccount\* account = nullptr;

                    account = new BankAccount();

                    table->Add(account->account\_number, account);

                    Print(table->GetMass(), n);

                    cout << "Данные успешно добавлены" << endl;

                }

                else {

                    cout << "Хэш-таблица полностью заполнена\nДля повтора требуется пересоздать её" << endl;

                }

                system("pause");

                break;

            }

            case 5: {

                if (table && table->GetVoidCount() != n)

                {

                    cout << "Введите номер карты клиента: ";

                    int num = 0;

                    cin >> num;

                    if (table->Remove(num))

                        cout << "Информация о клиенте успешно удалена" << endl;

                    else

                        cout << "Данного клиента нет в хэш-таблице" << endl;

                } else {

                    cout << "Хэш-таблица пустая" << endl;

                }

                system("pause");

                break;

            }

            case 0: {

                system("pause");

                break;

            }

            default:

                break;

            }

        }

        catch (const std::exception& exc)

        {

            cout << "ERROR: " << exc.what() << endl;

            system("pause");

        }

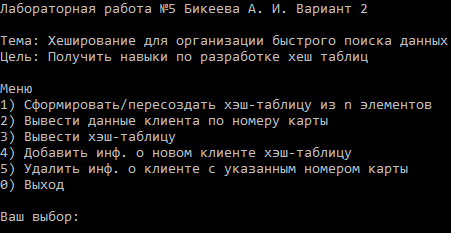
    }

    return 0;

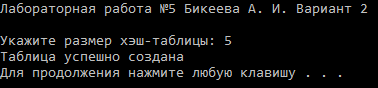
}

3.4. Общие контрольные прогоны программы

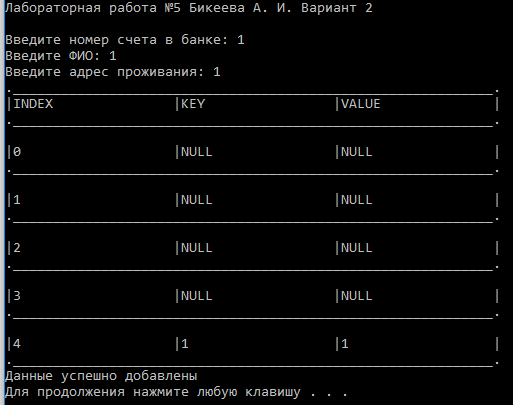
**Меню**



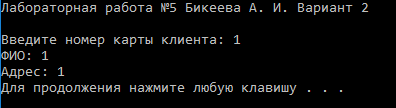
**1 пункт**



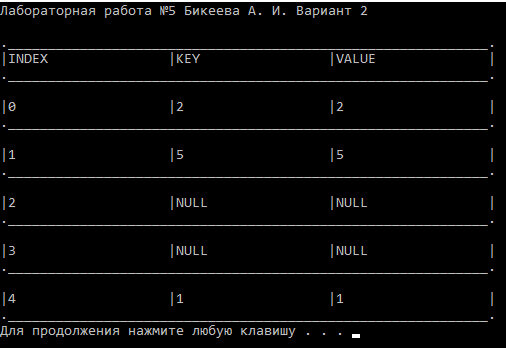
**4 пункт**



**2 пункт**



**4.3 пункт после добавления нескольких элементов**



Выводы

В результате выполнения лабораторной работы были получены навыки по разработке хэш таблиц.

В качестве хэш функции для хэш таблицы использовался и был изучен метод открытой адресации с способом реализации коллизий путем увеличения на номер выполняемого подбора.