МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное автономное образовательное учреждение высшего образования

«Новосибирский государственный технический университет»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет по лабораторной работе №1

«Коллекция данных – хеш таблица»

Вариант 8

Выполнила студентка группы ДТ-160

Буянкина Елизавета Алексеевна

Проверил преподаватель

Романенко Т.А.

Новосибирск – 2024

# Задание

Спроектировать, реализовать интерфейс коллекции «Хеш-таблица», содержащей объекты произвольного типа. Тип объектов задаётся клиентской программой. АТД «Хеш-таблица» представляет ассоциативное, неупорядоченное множество элементов, доступ к которым выполняется с использованием ключа.

Интерфейс АТД «Хеш-таблица» включает следующие операции:

• опрос количества элементов таблице ,

• опрос ёмкости таблицы,

• опрос пустоты таблицы,

• очистка таблицы,

• поиск элемента по ключу,

• вставка элемента по ключу,

• удаление элемента по ключу,

• вывод структуры хеш-таблицы на экран,

• опрос числа проб, выполненных предыдущей операцией.

• создание итератора таблицы

• интерфейс итератора:

o конструктор,

o оператор \* для доступа по чтению записи к данным текущего элемента

o оператор ++ для перехода к следующему элементу в таблице,

o оператор == для сравнения двух итераторов

Вариант 8

Форма представления: хеш – таблица с открытой адресацией

Тип ключа k - строка текста произвольной длины (символы – заглавные буквы кириллицы).

Преобразование k к натуральному значению k’ по схеме Горнера.

Метод хеширования k’- мультипликативный.

Метод разрешения коллизий - линейный

# Формат АТД «Хэш - таблица»

**Общая характеристика:**

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива. Она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию удаления и операцию поиска пары по ключу. Оптимальная трудоемкость доступа по ключу на структуре О(1). Хеш-таблица с открытой адресацией представляет собой массив Т, в который помещаются элементы таблицы по индексу, равному хеш-значению ключа.

**Параметры:**

***ValueType*** – тип данных, хранящихся в хеш таблице

***size*** – количество элементов, размещенных в хеш таблице

***capacity*** – максимальное количество элементов, которые могут храниться в хеш таблице

**Операции:**

1. Конструктор с заданным предельным количеством элементов в хеш таблице Hashtable(size\_t size)

**Вход**: size – предельное количество элементов в хеш таблице

**Предусловия**: нет

**Процесс:** создание хеш таблицы с заданным предельным количеством элементов

**Выход**: нет

**Постусловия:** создана хеш таблица с заданным предельным количеством элементов

1. Вставка элемента в хеш таблицу Insert(string key, ValueType data)

**Вход**: key – ключ элемента, data – значение элемента

**Предусловия**: ключ key не содержится в хеш таблице

**Процесс:** добавление в хеш таблицу элемента со значением data с ключом key

**Выход**: булевое значение false, если предусловие не выполнено, иначе true

**Постусловия:** добавлен в хеш таблицу элемент со значением data с ключом key, size = size + 1

1. Удаление элемента из хеш таблицы Delete(string key)

**Вход**: key – ключ элемента

**Предусловия**: ключ key содержится в хеш таблице

**Процесс:** удаление в хеш таблицу элемента с ключом key

**Выход**: булевое значение false, если предусловие не выполнено, иначе true

**Постусловия** удален в хеш таблицу элемента с ключом key, size = size – 1

1. Поиск элемента хеш таблицы по ключу Search(string key)

**Вход**: key – ключ элемента

**Предусловия**: ключ key содержится в хеш таблице

**Процесс:** поиск элемента хеш таблицы по ключу key

**Выход**: возврат текущего значения size - количества элементов, хранящихся в хеш таблице или генерация исключения, если предусловие не выполнено

**Постусловия:** нет

1. Опрос количества элементов хеш таблицы getSize()

**Вход**: нет

**Предусловия**: нет

**Процесс:** возврат текущего значения size - количества элементов, хранящихся в хеш таблице

**Выход**: количество элементов, хранящихся в хеш таблице size

**Постусловия:** нет

1. Опрос емкости хеш таблицы getCapacity()

**Вход**: нет

**Предусловия**: нет

**Процесс:** возврат текущего значения capacity максимальное количество элементов, которые могут храниться в хеш таблице

**Выход**: емкость хеш таблицы capacity

**Постусловия:** нет

1. Опрос числа проб, выполненных предыдущей операциейgetProbesForPrevOp()

**Вход**: нет

**Предусловия**: нет

**Процесс:** возврат числа проб, выполненных предыдущей операцией в хеш таблице

**Выход**: числа зондирований probes

**Постусловия:** нет

1. Проверка хеш таблицы на пустоту isEmpty()

**Вход**: нет

**Предусловия**: нет

**Процесс:** проверка наличия хотя бы одного элемента в хеш таблице

**Выход**: булевое значение true, если хеш таблица хранит хотя бы один элемент, иначе false

**Постусловия:** нет

1. Запрос прямого итератора begin( )

**Вход**: нет

**Предусловия**: хеш таблица не пуста

**Процесс:** формирование итератора, установленного на первый элемент, содержащийся в хеш таблице, или «неустановленный» итератор ***end***( ) при невыполнении предусловия

**Выход:** итератор для доступа к элементам ***begin***( ) или «неустановленный» итератор ***end***( )

**Постусловия:** нет

12) Запрос «неустановленного» прямого итератора end( )

**Вход**: нет

**Предусловия**: нет

**Процесс:** формирование «неустановленного» обратного итератора, указывающего на позицию после последнего элемента хеш таблицы.

**Выход:** «неустановленный» итератор произвольного доступа *rend*( )

**Постусловия:** нет

1. Очистка хеш таблицы Clear()

**Вход**: нет

**Предусловия**: нет

**Процесс:** удаление всех элементов

**Выход**: нет

**Постусловия:** удалены все элементы, число элементов ***size*** = 0

# Формат АТД «Итератор хеш-таблицы»

1. Доступ к данным текущего элемента для чтения/записи operator\* ()

**Вход**: нет

**Предусловия**: итератор установлен

**Процесс:** получение значения данных текущего элемента, на который установлен итератор, по ссылке

**Выход**: нет

**Постусловия:** значения данных текущего элемента, на который установлен итератор, по ссылке или генерация исключения, если предусловие не выполнено

1. Инкремент итератора operator ++()

**Вход**: нет

**Предусловия**: нет

**Процесс:** Инкремент итератора на следующий элемент хеш таблицы

**Выход**: нет

**Постусловия:** нет

1. Проверка равенства итераторов operator == (const Iterator& other)

**Вход**: ссылка на объект АТД Iterator other

Предусловия: нет

**Процесс:** сравнение итераторов

**Выход**: булевое значение false, если итераторы не равны, иначе true

Постусловия: нет

1. Проверка неравенства итераторов operator != (const Iterator& other)

**Вход**: ссылка на объект АТД Iterator other

Предусловия: нет

**Процесс:** сравнение итераторов

**Выход**: булевое значение false, если итераторы равны, иначе true

Постусловия: нет

# Справочные определения классов для коллекций «Хэш - таблица»

template <class ValueType>

class Hashtable {

public:

Hashtable(size\_t size); // Конструктор с заданным предельным количеством элементов

bool Insert(string key, ValueType data);// Вставка элемента

bool Delete(string key); // Удаление элемента

ValueType Search(string key); // Поиск элемента хеш таблицы по ключу

size\_t getCapacity(); // Опрос емкости

size\_t getSize(); // Опрос количества элементов

size\_t getProbesForPrevOp(); // Опрос числа проб, выполненных предыдущей операцией

bool IsEmpty(); // Проверка хеш таблицы на пустоту

void Clear(); // Очистка хеш таблицы

Iterator begin(); // Запрос прямого итератора

Iterator end();//Запрос «неустановленного» прямого итератора

};

# Справочные определения классов для коллекций «Итератор хеш-таблицы»

template <class ValueType>

class Iterator {

public:

ValueType& operator\* (); // Доступ к данным текущего элемента для чтения/записи

Iterator& operator ++(); // Инкремент итератора на следующий элемент

bool operator == (const Iterator& other); // проверка равенства итераторов

bool operator != (const Iterator& other); // проверка неравенства итераторов

};

# Список использованной литературы

1. Романенко, Т. А. Программные коллекции данных. Проектирование и реализация : учебник для вузов / Т. А. Романенко. — Санкт-Петербург : Лань, 2021. — 152 с.

2. Фрэнк М. Каррано, Джанет Дж. Причард. Абстракция данных и решение задач на С++. Стены и зеркала. - М. - СПб – Киев: «Вильямс», 2003 г. – 848 с.

3. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы. Анализ и построение. - М: «БИНОМ», 2000 г. – 960 с.

4. Кубенский А.А. Структуры и алгоритмы обработки данных: объектно-ориентированный подход и реализация на С++. – СПб.: БХВ-Петербург, 2004 г. – 464 с.

5. Коллинз У.Дж. Структуры данных и стандартная библиотека шаблонов. – М.: ООО «Бином-Пресс», 2004. – 624 с.

6. Роберт Сэджвик. Фундаментальные алгоритмы на С++. Части 1-5. - М: «DiaSoft», 2001 г. – 688 с.

# Приложение с текстами программ

## Hashtable.h

#include <vector>

#include <iostream>

using namespace std;

template <class ValueType>

class Hashtable {

private:

enum class State { FREE, BUSY, DELETED };

class Node {

public:

State state = State::FREE;

string key;

ValueType value{};

Node() {}

Node(string key, ValueType value) : state(State::BUSY), key(key), value(value) {}

void clear() {

state = State::FREE;

key = "";

value = ValueType{};

}

};

size\_t size = 0;

size\_t probes = 0;

vector<Node> data;

public:

Hashtable(size\_t size);

bool Insert(string key, ValueType data);

bool Delete(string key);

void Show();

ValueType Search(string key);

size\_t getCapacity();

size\_t getSize();

size\_t getProbesForPrevOp();

bool IsEmpty();

void Clear();

class Iterator {

private:

Hashtable\* table = nullptr;

Node\* current = nullptr;

size\_t currentIndex = 0;

public:

Iterator(Hashtable\* table, bool isBegin) {

this->table = table;

if (!isBegin) {

currentIndex = table->getCapacity();

current = nullptr;

return;

}

currentIndex = 0;

while (currentIndex < table->getCapacity() && table->data[currentIndex].state != State::BUSY) {

currentIndex++;

}

if (currentIndex == table->getCapacity() || table->data[currentIndex].state != State::BUSY) {

current = nullptr;

}

else {

current = &table->data[currentIndex];

}

}

ValueType& operator\* () {

if (current == nullptr) {

throw exception("Вышли за пределы таблицы");

}

return current->value;

}

Iterator& operator ++() {

if (current != nullptr) {

currentIndex++;

while (currentIndex < table->getCapacity() && table->data[currentIndex].state != State::BUSY) {

currentIndex++;

}

if (currentIndex == table->getCapacity() || table->data[currentIndex].state != State::BUSY) {

current = nullptr;

}

else {

current = &table->data[currentIndex];

}

}

return \*this;

}

bool operator == (const Iterator& other) { // проверка равенства

return table == other.table && current == other.current;

}

bool operator != (const Iterator& other) { // проверка неравенства

return table == other.table && current != other.current;

}

};

Iterator begin() {

return Iterator(this, true);

}

Iterator end() {

return Iterator(this, false);

}

private:

size\_t hashcode(string key, size\_t i = 0);

};

template<class ValueType>

Hashtable<ValueType>::Hashtable(size\_t size) {

size\_t capacity = 1;

size\_t minCapacity = size \* 2;

while (capacity < minCapacity) {

capacity \*= 2;

}

data.clear();

data.resize(capacity, Node());

}

template<class ValueType>

bool Hashtable<ValueType>::Insert(string key, ValueType value) {

probes = 0;

int pos = -1;

size\_t i = 0;

size\_t j;

do {

j = hashcode(key, i);

if (data[j].state == State::DELETED && pos == -1) {

pos = j;

}

else if (data[j].key == key && data[j].state == State::BUSY) {

return false;

}

i++;

} while (!(i == getCapacity() || data[j].state == State::FREE));

if (i == getCapacity() && pos == -1) {

probes = i;

return false;

}

if (pos == -1) {

pos = j;

}

data[pos] = Node(key, value);

size++;

probes = i;

return true;

}

template<class ValueType>

bool Hashtable<ValueType>::Delete(string key) {

probes = 0;

size\_t i = 0;

size\_t j;

do {

j = hashcode(key, i);

if (data[j].key == key && data[j].state == State::BUSY) {

size--;

probes = i + 1;

if (size == 0) {

Clear();

return true;

}

data[j].clear();

data[j].state = State::DELETED;

return true;

}

i++;

} while (!(data[j].state == State::FREE || i == getCapacity()));

probes = i;

return false;

}

template<class ValueType>

void Hashtable<ValueType>::Show() {

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

cout << i << " ";

switch (data[i].state) {

case State::FREE: {

cout << "f ";

break;

};

case State::BUSY: {

cout << "b ";

cout << data[i].key;

break;

};

case State::DELETED: {

cout << "d ";

break;

}

}

cout << endl;

}

}

template<class ValueType>

ValueType Hashtable<ValueType>::Search(string key) {

probes = 0;

size\_t i = 0;

size\_t j;

do {

j = hashcode(key, i);

if (data[j].key == key && data[j].state == State::BUSY) {

probes = i+1;

return data[j].value;

}

i++;

} while (!(data[j].state == State::FREE || i == getCapacity()));

probes = i;

throw exception("Ключ не существует!");

}

template<class ValueType>

size\_t Hashtable<ValueType>::hashcode(string s, size\_t probe) {

int k = 31, mod = 1e13;

unsigned long long m = 1;

unsigned long long hash\_result = 0;

double A = (sqrt(5) - 1) / 2;

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

int \_s = s[i] - 'А' + 7;

hash\_result = hash\_result \* (int)pow(2, s.size());

hash\_result = (hash\_result % mod + m \* \_s) % mod;

m = (m \* k) % mod;

}

double buf = A \* hash\_result;

buf = buf - (size\_t)buf;

hash\_result = ((size\_t)(getCapacity() \* buf) + probe) % getCapacity();

cout << "Сгенерированный индекс: " << hash\_result << endl;

return hash\_result;

}

template<class ValueType>

size\_t Hashtable<ValueType>::getCapacity() {

return data.size();

}

template<class ValueType>

size\_t Hashtable<ValueType>::getSize() {

return size;

}

template<class ValueType>

size\_t Hashtable<ValueType>::getProbesForPrevOp() {

return probes;

}

template<class ValueType>

bool Hashtable<ValueType>::IsEmpty() {

return size == 0;

}

template<class ValueType>

void Hashtable<ValueType>::Clear() {

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

data[i].clear();

}

size = 0;

}

## Menu.cpp

#include <iostream>

#include <windows.h>

#include "Hashtable.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "RU");

Hashtable<int> hashtable = Hashtable<int>(0);

auto it = hashtable.begin();

int n;

while (true) {

cout << endl << "Меню. Выберите действие:" << endl << endl;

cout << "0. Создать хеш-таблицу с заданным размером" << endl;

cout << "1. Вставка" << endl;

cout << "2. Удаление" << endl;

cout << "3. Емкость" << endl;

cout << "4. Размер" << endl;

cout << "5. Проверка на пустоту" << endl;

cout << "6. Поиск элемента по ключу" << endl;

cout << "7. Вывод структуры на экран" << endl;

cout << "8. Очистка" << endl;

cout << "9. Опрос числа проб, выполненных предыдущей операцией" << endl;

cout << "10. Итератор на начало" << endl;

cout << "11. Показать содержание итератора" << endl;

cout << "12. Перевести итератор на следующий элемент" << endl;

cout << "13. Присвоить итератору новое значение" << endl;

cout << endl;

cin >> n;

cout << endl;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

try {

switch (n) {

case 0: {

int s;

cin >> s;

hashtable = Hashtable<int>(s);

break;

}

case 1: {

string key;

int value;

cin >> key;

cin >> value;

cout << hashtable.Insert(key, value) << endl;

break;

}

case 2: {

string key;

cin >> key;

cout << hashtable.Delete(key) << endl;

break;

}

case 3: {

cout << hashtable.getCapacity() << endl;

break;

}

case 4: {

cout << hashtable.getSize() << endl;

break;

}

case 5: {

cout << hashtable.IsEmpty() << endl;

break;

}

case 6: {

string key;

cin >> key;

cout << hashtable.Search(key) << endl;

break;

}

case 7: {

hashtable.Show();

break;

}

case 8: {

hashtable.Clear();

break;

}

case 9: {

cout << hashtable.getProbesForPrevOp() << endl;

break;

}

case 10: {

it = hashtable.begin();

break;

}

case 11: {

cout << \*it << endl;

break;

}

case 12: {

++it;

break;

}

case 13: {

cin >> \*it;

break;

}

default: {

cout << "Попробуйте еще раз" << endl;

break;

}

}

}

catch (exception e) {

cout << e.what() << endl;

}

}

}