Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №13

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы с открытой адресацией»

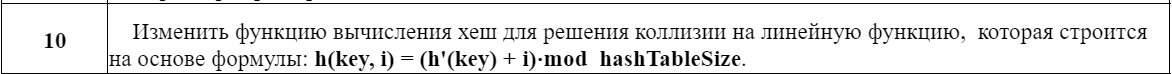
 Выполнила:

Студентка1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: асс. Андронова М.В.

Минск, 2024



#include <iostream>

using namespace std;

#define HASHDEL (void\*) -1

struct Object

{

void\*\* data;

Object(int, int(\*)(void\*));

int size;

int N;

int(\*getKey)(void\*);

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, int(\*getkey)(void\*));

#undef HASHDEL

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k;

mas = z;

}

AAA() {}

};

//-------------------------------

int key(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

//-------------------------------

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;

}

//-------------------------------

// Линейная хеш-функция для решения коллизий

int LinearHashFunction(int key, int size, int i)

{

return (key + i) % size; // h(key, i) = (h'(key) + i) % size

}

//-------------------------------

Object create(int size, int(\*getkey)(void\*))

{

return \*(new Object(size, getkey));

}

//-------------------------------

Object::Object(int size, int(\*getkey)(void\*))

{

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

//-------------------------------

bool Object::insert(void\* d)

{

bool b = false;

if (N != size)

for (int i = 0, t = getKey(d), j = LinearHashFunction(t, size, i);

i != size && !b; j = LinearHashFunction(t, size, ++i))

if (data[j] == NULL || data[j] == DEL)

{

data[j] = d;

N++;

b = true;

}

return b;

}

//-------------------------------

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

if (N != 0)

for (int i = 0, j = LinearHashFunction(key, size, i); data[j] != NULL && i != size && !b;

j = LinearHashFunction(key, size, ++i))

if (data[j] != DEL)

if (getKey(data[j]) == key)

{

t = j; b = true;

}

return t;

}

//-------------------------------

void\* Object::search(int key)

{

int t = searchInd(key);

return (t >= 0) ? (data[t]) : (NULL);

}

//-------------------------------

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//-------------------------------

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL);

}

//-------------------------------

void Object::scan(void(\*f)(void\*))

{

for (int i = 0; i < this->size; i++)

{

std::cout << " Элемент" << i;

if ((this->data)[i] == NULL)

std::cout << " пусто" << std::endl;

else

if ((this->data)[i] == DEL)

std::cout << " удален" << std::endl;

else

f((this->data)[i]);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

int siz = 10, choice, k;

cout << "Введите размер хеш-таблицы" << endl;

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl; cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: H.scan(AAA\_print); break;

case 2: { AAA\* a = new AAA;

char\* str = new char[20];

cout << "введите ключ" << endl;

cin >> k;

a->key = k;

cout << "введите строку" << endl;

cin >> str;

a->mas = str;

if (H.N == H.size)

cout << "Таблица заполнена" << endl;

else

H.insert(a);

} break;

case 3: { cout << "введите ключ для удаления" << endl;

cin >> k;

H.deleteByKey(k);

} break;

case 4: { cout << "введите ключ для поиска" << endl;

cin >> k;

if (H.search(k) == NULL)

cout << "Элемент не найден" << endl;

else

AAA\_print(H.search(k));

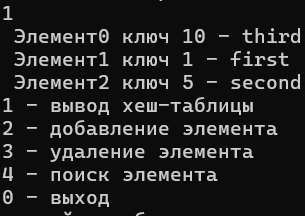
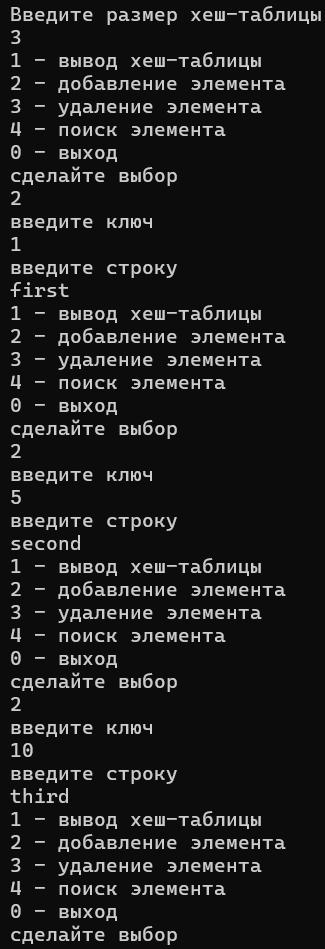
} break;

}

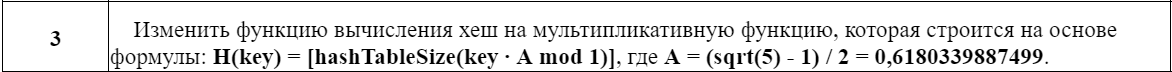
}

return 0;

}



**Дополнительные варианты:**

****  
#include <iostream>

#include <cmath>

using namespace std;

#define HASHDEL (void\*) -1

struct Object

{

void\*\* data;

Object(int, int(\*)(void\*));

int size;

int N;

int(\*getKey)(void\*);

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, int(\*getkey)(void\*));

#undef HASHDEL

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k;

mas = z;

}

AAA() {}

};

//-------------------------------

int key(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

//-------------------------------

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;

}

//-------------------------------

// Мультипликативная хеш-функция

int MultiplicativeHashFunction(int key, int size)

{

double A = 0.6180339887499;

return static\_cast<int>(size \* fmod(key \* A, 1));

}

//-------------------------------

Object create(int size, int(\*getkey)(void\*))

{

return \*(new Object(size, getkey));

}

//-------------------------------

Object::Object(int size, int(\*getkey)(void\*))

{

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

//-------------------------------

bool Object::insert(void\* d)

{

bool b = false;

if (N != size)

for (int i = 0, t = getKey(d), j = MultiplicativeHashFunction(t, size);

i != size && !b; j = (j + 1) % size, ++i)

if (data[j] == NULL || data[j] == DEL)

{

data[j] = d;

N++;

b = true;

}

return b;

}

//-------------------------------

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

if (N != 0)

for (int i = 0, j = MultiplicativeHashFunction(key, size); data[j] != NULL && i != size && !b;

j = (j + 1) % size, ++i)

if (data[j] != DEL)

if (getKey(data[j]) == key)

{

t = j; b = true;

}

return t;

}

//-------------------------------

void\* Object::search(int key)

{

int t = searchInd(key);

return (t >= 0) ? (data[t]) : (NULL);

}

//-------------------------------

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//-------------------------------

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL);

}

//-------------------------------

void Object::scan(void(\*f)(void\*))

{

for (int i = 0; i < this->size; i++)

{

std::cout << " Элемент" << i;

if ((this->data)[i] == NULL)

std::cout << " пусто" << std::endl;

else

if ((this->data)[i] == DEL)

std::cout << " удален" << std::endl;

else

f((this->data)[i]);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

int siz = 10, choice, k;

cout << "Введите размер хеш-таблицы" << endl;

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl; cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: H.scan(AAA\_print); break;

case 2: { AAA\* a = new AAA;

char\* str = new char[20];

cout << "введите ключ" << endl;

cin >> k;

a->key = k;

cout << "введите строку" << endl;

cin >> str;

a->mas = str;

if (H.N == H.size)

cout << "Таблица заполнена" << endl;

else

H.insert(a);

} break;

case 3: { cout << "введите ключ для удаления" << endl;

cin >> k;

H.deleteByKey(k);

} break;

case 4: { cout << "введите ключ для поиска" << endl;

cin >> k;

if (H.search(k) == NULL)

cout << "Элемент не найден" << endl;

else

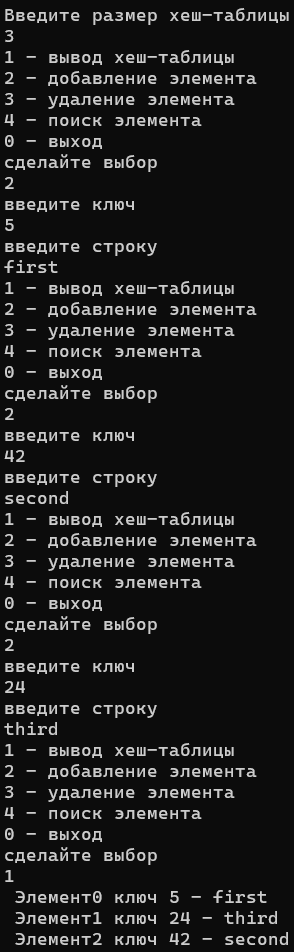
AAA\_print(H.search(k));

} break;

}

}

}



  
#include <iostream>

#include <cmath>

using namespace std;

#define HASHDEL (void\*) -1

struct Object

{

void\*\* data;

Object(int, int(\*)(void\*));

int size;

int N;

int(\*getKey)(void\*);

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, int(\*getkey)(void\*));

#undef HASHDEL

struct AAA

{

int key;

char\* mas;

AAA(int k, char\* z)

{

key = k;

mas = z;

}

AAA() {}

};

//-------------------------------

int key(void\* d)

{

AAA\* f = (AAA\*)d;

return f->key;

}

//-------------------------------

void AAA\_print(void\* d)

{

cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << endl;

}

//-------------------------------

// Квадратичная хеш-функция для решения коллизий

int QuadraticHashFunction(int key, int size, int i)

{

int c1 = 1; // коэффициенты c1 и c2 могут быть выбраны произвольно

int c2 = 3;

return (key + c1 \* i + c2 \* i \* i) % size; // h(key, i) = (h'(key) + с1\*i + c2\*i^2) % size

}

//-------------------------------

Object create(int size, int(\*getkey)(void\*))

{

return \*(new Object(size, getkey));

}

//-------------------------------

Object::Object(int size, int(\*getkey)(void\*))

{

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

//-------------------------------

bool Object::insert(void\* d)

{

bool b = false;

if (N != size)

for (int i = 0, t = getKey(d), j = QuadraticHashFunction(t, size, i);

i != size && !b; j = QuadraticHashFunction(t, size, ++i))

if (data[j] == NULL || data[j] == DEL)

{

data[j] = d;

N++;

b = true;

}

return b;

}

//-------------------------------

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

if (N != 0)

for (int i = 0, j = QuadraticHashFunction(key, size, i); data[j] != NULL && i != size && !b;

j = QuadraticHashFunction(key, size, ++i))

if (data[j] != DEL)

if (getKey(data[j]) == key)

{

t = j; b = true;

}

return t;

}

//-------------------------------

void\* Object::search(int key)

{

int t = searchInd(key);

return (t >= 0) ? (data[t]) : (NULL);

}

//-------------------------------

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

//-------------------------------

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL);

}

//-------------------------------

void Object::scan(void(\*f)(void\*))

{

for (int i = 0; i < this->size; i++)

{

std::cout << " Элемент" << i;

if ((this->data)[i] == NULL)

std::cout << " пусто" << std::endl;

else

if ((this->data)[i] == DEL)

std::cout << " удален" << std::endl;

else

f((this->data)[i]);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

int siz = 10, choice, k;

cout << "Введите размер хеш-таблицы" << endl;

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывод хеш-таблицы" << endl;

cout << "2 - добавление элемента" << endl;

cout << "3 - удаление элемента" << endl;

cout << "4 - поиск элемента" << endl;

cout << "0 - выход" << endl;

cout << "сделайте выбор" << endl; cin >> choice;

switch (choice)

{

case 0: exit(0);

case 1: H.scan(AAA\_print); break;

case 2: { AAA\* a = new AAA;

char\* str = new char[20];

cout << "введите ключ" << endl;

cin >> k;

a->key = k;

cout << "введите строку" << endl;

cin >> str;

a->mas = str;

if (H.N == H.size)

cout << "Таблица заполнена" << endl;

else

H.insert(a);

} break;

case 3: { cout << "введите ключ для удаления" << endl;

cin >> k;

H.deleteByKey(k);

} break;

case 4: { cout << "введите ключ для поиска" << endl;

cin >> k;

if (H.search(k) == NULL)

cout << "Элемент не найден" << endl;

else

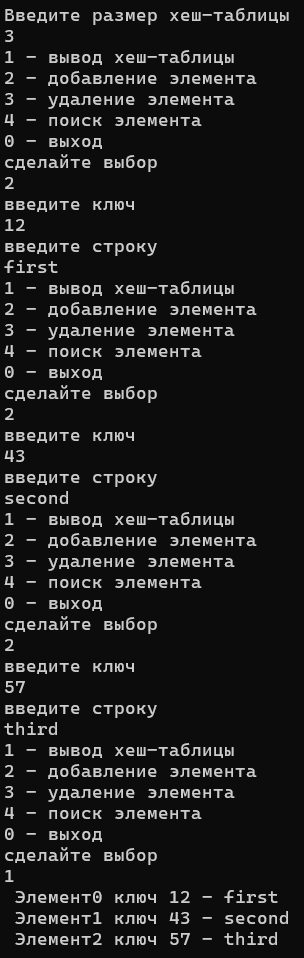
AAA\_print(H.search(k));

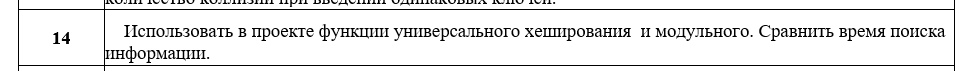
} break;

}

}

}



  
#include <iostream>

#include <ctime>

// Заголовочный файл Hash.h

#pragma once

#define HASHDEL (void\*) -1

struct Object {

void\*\* data;

Object(int, int(\*)(void\*));

int size;

int N;

int(\*getKey)(void\*);

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*));

};

static void\* DEL = (void\*)HASHDEL;

Object create(int size, int(\*getkey)(void\*));

#undef HASHDEL

// Функция для получения ключа из структуры AAA

int key(void\* d);

// Определение структуры AAA и функции для получения ключа

struct AAA {

int key; // Ключ

char\* mas; // Значение

AAA(int k, char\* z) {

key = k; mas = z;

}

AAA() {}

};

// Функция для получения ключа из структуры AAA

int key(void\* d) {

AAA\* f = (AAA\*)d;

return f->key;

}

// Функция для вывода элемента структуры AAA

void AAA\_print(void\* d) {

std::cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << std::endl;

}

// Хеш-функция для вычисления индекса

int HashFunction(int key, int size, int p) {

double key2 = 5 \* ((0.6180339887499 \* key) - int((0.6180339887499 \* key)));

return (p + key) % size;

}

// Вспомогательная функция для вычисления следующего индекса

int Next\_hash(int hash, int size, int p) {

return (hash + 5 \* p + 3 \* p \* p) % size;

}

// Создание объекта хеш-таблицы

Object create(int size, int(\*getkey)(void\*)) {

return \*(new Object(size, getkey));

}

// Конструктор объекта хеш-таблицы

Object::Object(int size, int(\*getkey)(void\*)) {

N = 0;

this->size = size;

this->getKey = getkey;

this->data = new void\* [size];

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

data[i] = NULL;

}

// Вставка элемента в хеш-таблицу

bool Object::insert(void\* d) {

bool b = false;

if (N != size) {

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0);

i != size && !b; j = Next\_hash(j, size, ++i)) {

if (data[j] == NULL || data[j] == DEL) {

data[j] = d;

N++;

b = true;

}

}

}

return b;

}

// Поиск индекса элемента по ключу

int Object::searchInd(int key) {

int t = -1;

bool b = false;

if (N != 0) {

for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i != size && !b; j = HashFunction(key, size, ++i)) {

if (data[j] != DEL)

if (getKey(data[j]) == key) {

t = j; b = true;

}

}

}

return t;

}

// Поиск элемента по ключу

void\* Object::search(int key) {

int t = searchInd(key);

return (t >= 0) ? (data[t]) : (NULL);

}

// Удаление элемента по ключу

void\* Object::deleteByKey(int key) {

int i = searchInd(key);

void\* t = data[i];

if (t != NULL) {

data[i] = DEL;

N--;

}

return t;

}

// Удаление элемента по значению

bool Object::deleteByValue(void\* d) {

return (deleteByKey(getKey(d)) != NULL);

}

// Сканирование и вывод элементов хеш-таблицы

void Object::scan(void(\*f)(void\*)) {

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

std::cout << " Элемент" << i;

if ((this->data)[i] == NULL)

std::cout << " пусто" << std::endl;

else if ((this->data)[i] == DEL)

std::cout << " удален" << std::endl;

else

f((this->data)[i]);

}

}

// Определение универсальной обертки для хеш-функции

int UniversalHashFunction(int key, int size, int p) {

return (key + p) % size;

}

// Определение обертки для модульной хеш-функции

int ModularHashFunction(int key, int size, int) {

return key % size;

}

// Обертка для универсальной функции хеширования, чтобы она соответствовала ожидаемой сигнатуре

int UniversalHashFunctionWrapper(void\* data) {

AAA\* d = (AAA\*)data;

return UniversalHashFunction(d->key, 10, 0); // Размер и параметр могут быть динамически изменены

}

// Обертка для модульной функции хеширования, чтобы она соответствовала ожидаемой сигнатуре

int ModularHashFunctionWrapper(void\* data) {

AAA\* d = (AAA\*)data;

return ModularHashFunction(d->key, 10, 0); // Размер и параметр могут быть динамически изменены

}

int main() {

setlocale(LC\_ALL, "ru");

int siz = 10;

std::cout << "Введите размер хеш-таблицы: ";

std::cin >> siz;

Object H = create(siz, UniversalHashFunctionWrapper);

int choice, k;

AAA\* a;

for (;;) {

std::cout << "1 - вывод хеш-таблицы" << std::endl;

std::cout << "2 - добавление элемента" << std::endl;

std::cout << "3 - удаление элемента" << std::endl;

std::cout << "4 - поиск элемента" << std::endl;

std::cout << "5 - тест времени поиска" << std::endl;

std::cout << "0 - выход" << std::endl;

std::cout << "сделайте выбор" << std::endl;

std::cin >> choice;

switch (choice) {

case 0: exit(0);

case 1: H.scan(AAA\_print); break;

case 2: {

a = new AAA;

char\* str = new char[20];

std::cout << "введите ключ" << std::endl;

std::cin >> k;

a->key = k;

std::cout << "введите строку" << std::endl;

std::cin >> str;

a->mas = str;

if (H.N == H.size)

std::cout << "Таблица заполнена" << std::endl;

else

H.insert(a);

} break;

case 3: {

std::cout << "введите ключ для удаления" << std::endl;

std::cin >> k;

H.deleteByKey(k);

} break;

case 4: {

std::cout << "введите ключ для поиска" << std::endl;

std::cin >> k;

if (H.search(k) == NULL)

std::cout << "Элемент не найден" << std::endl;

else

AAA\_print(H.search(k));

} break;

case 5: {

int testKey = 123;

// Тестирование времени поиска с универсальной хеш-функцией

H.getKey = UniversalHashFunctionWrapper;

clock\_t start = clock();

H.search(testKey);

clock\_t end = clock();

std::cout << "Время поиска (универсальная хеш-функция): " << (double)(end - start) / CLOCKS\_PER\_SEC << " секунд" << std::endl;

// Тестирование времени поиска с модульной хеш-функцией

H.getKey = ModularHashFunctionWrapper;

start = clock();

H.search(testKey);

end = clock();

std::cout << "Время поиска (модульная хеш-функция): " << (double)(end - start) / CLOCKS\_PER\_SEC << " секунд" << std::endl;

} break;

}

}

}

