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»

Выполнил(а):

Студент(ка) 1 курса 8 группы

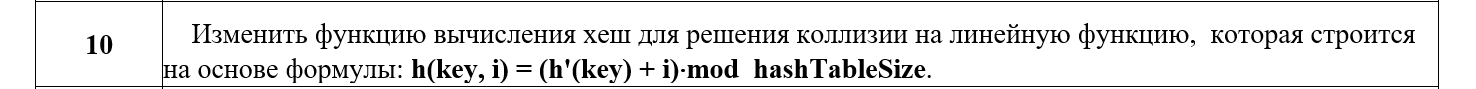
Семёнов Даниил Вячеславович

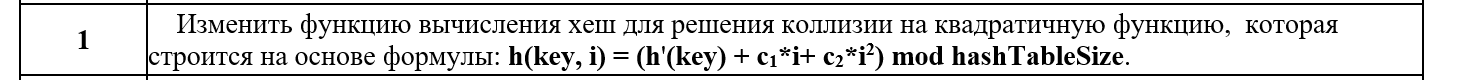
Преподаватель: асс. Андронова М.В.

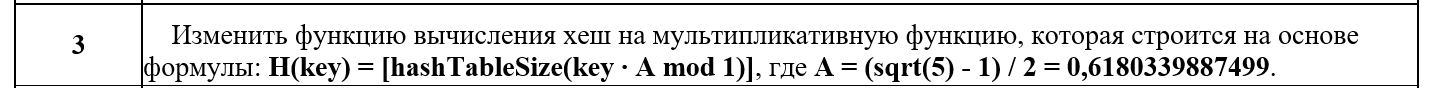
Минск, 2024

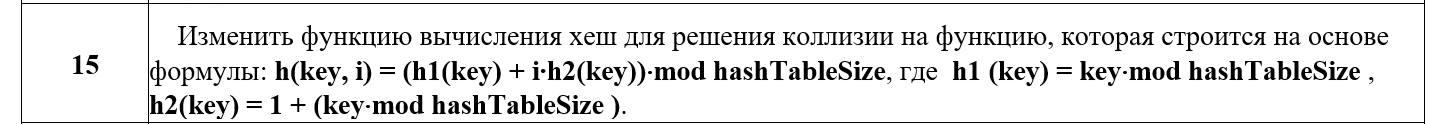
5.  . В соответствии со своим вариантом построить ***хеш-таблицы*** с ***открытой*** адресацией разного размера, например, 16, 32 или 32, 64, 128 с коллизиями. В таблице **h**'**(key)** −значение хеш-функции, приведшее к коллизии.

Вариант 10 + Доп задания Вариант 1 + Вариант 3 + Варинат 15









Main.cpp

// Во всех вариантах нужно изменить либо функцию вычисления хеша либо функцию вычисления хеша для решения коллизии, поэтому они все реализованы в одной программе(функции закомментированы чтобы не сломать программу).

#include "Hash.h"

#include <iostream>

using namespace std;

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;

}

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

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;

}

Hash.cpp  
  
#include "Hash.h"

#include <iostream>

int HashFunction(int key, int size, int p) //Хеш-функция

{

// функция возвращает одно и то же, что абсолютно всегда приводит к коллизии

double key2 = (int)(size \* (key \* 0.6180339887499)) % 1; // доп вариант 3

return key2;

}

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

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

{

return (key % size + p \* (1 + (key % size))); // доп вариант 15

//return (hash + 1 \* p + 2 \* p \* p) % size; // доп вариант 1

//return (hash + p) % size; // изменили на линейную функцию Основной вариант 10

}

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

Object create(int size, int(\*getkey)(void\*)) // создание объекта хеш функции

{

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

}

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

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

{

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) //вставка элемента

{

bool b = false;

if (N != size)

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

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

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

{

data[j] = d;

N++;

b = true;

}

return b;

}

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

int Object::searchInd(int key) //поиск элемента по ключу

{

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;

}

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

void\* Object::search(int key) //поиск

{

int t = searchInd(key);

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

}

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

void\* Object::deleteByKey(int key) // удаление по ключу

{

int i = searchInd(key);

void\* t = data[i];

if (t != NULL)

{

data[i] = DEL;

N--;

}

return t;

}

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

bool Object::deleteByValue(void\* d) //удаление по значению

{

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

}

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

void Object::scan(void(\*f)(void\*)) //просмотр содержимого хеш-таблицы

{

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]);

}

}

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\*));

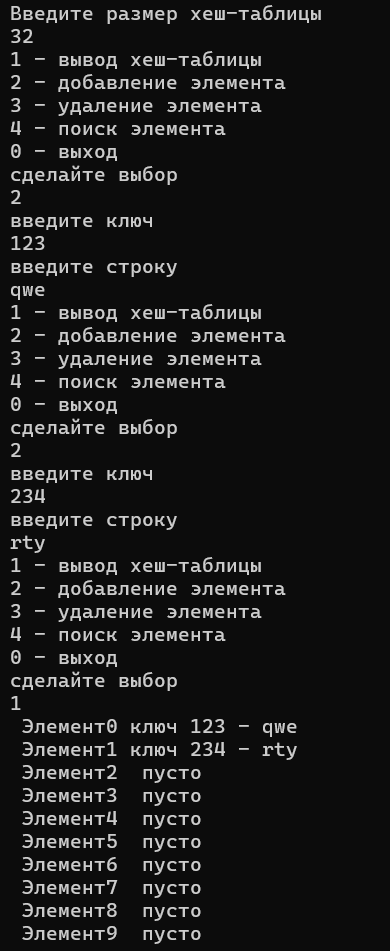
};

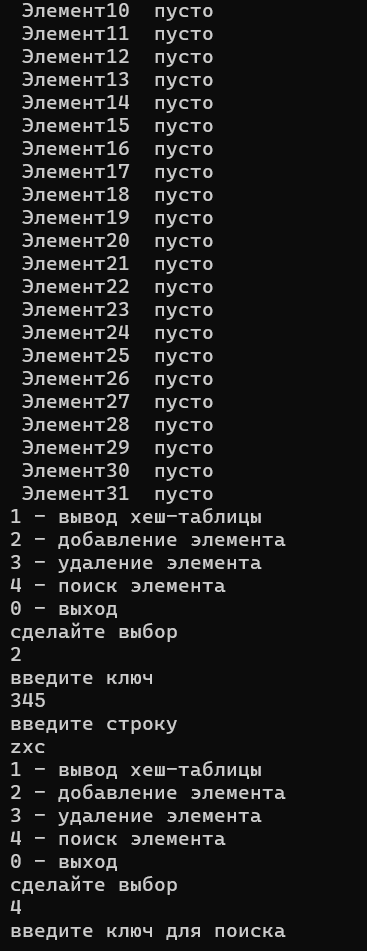
static void\* DEL = (void\*)HASHDEL;

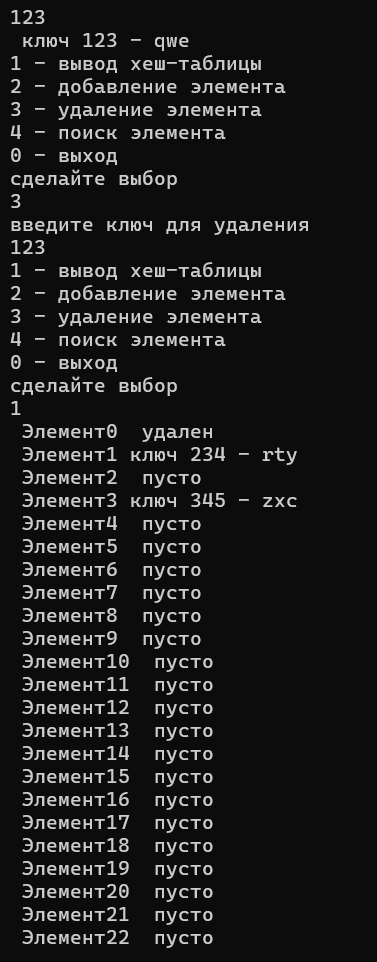
Object create(int size, int(\*getkey)(void\*));

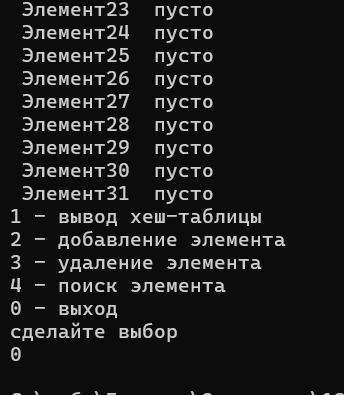
#undef HASHDEL

Внизу скриншоты работы при функции 3 Варианта и Вараинта 10

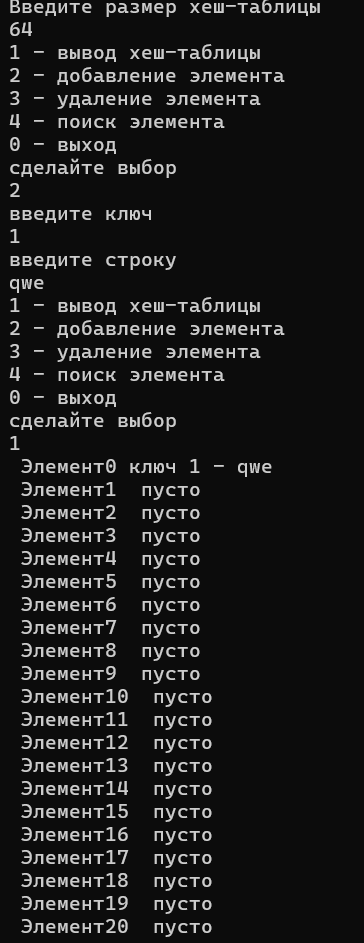


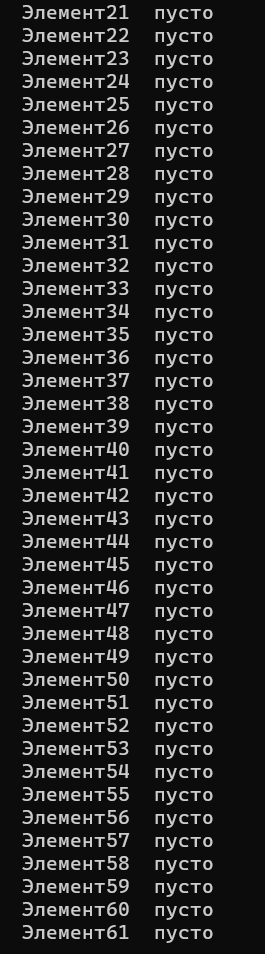


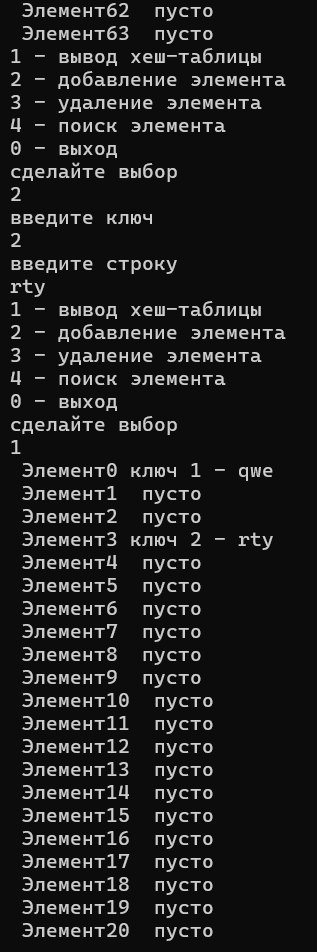


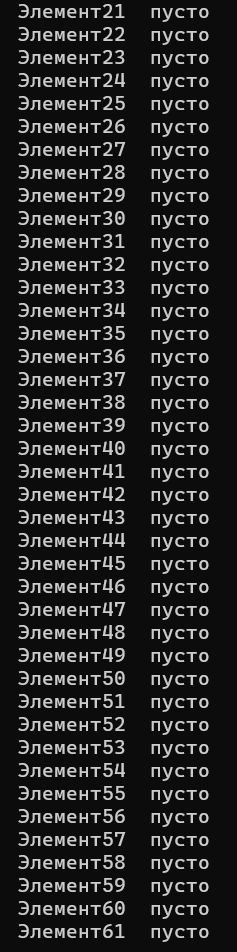


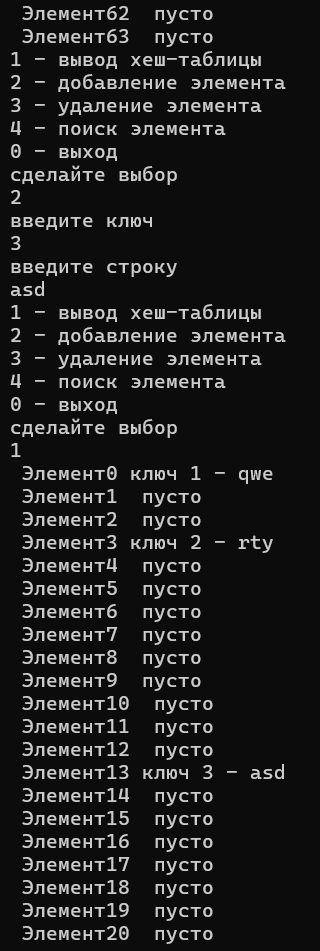
Ниже скриншоты работы программа при функции 3 Варианта и Варианта 1

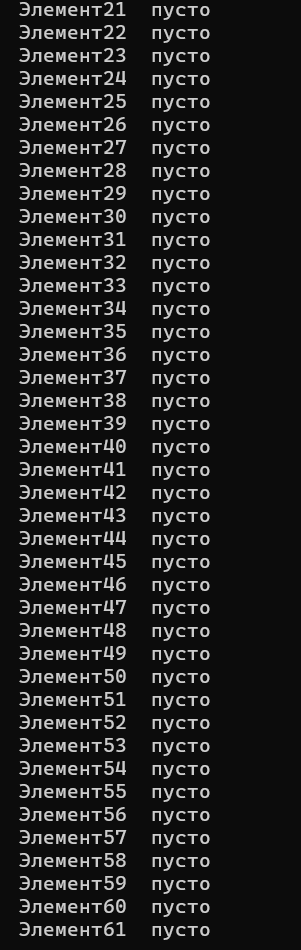


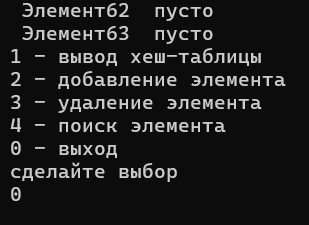












Внизу скриншоты работы программы при функции 3 Варианта и Варианта 15

