Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 13

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы c открытой адресацией»

Выполнил:

Студент 1 курса 10 группы

Жамойдо Артём Игоревич

Преподаватель: асс. Андронова М.В.

Минск, 2024

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 4**

Использовать в проекте функции универсального хеширования и модульного. Сравнить количество коллизий при введении одинаковых ключей.

**Hash.h**

#pragma once

#define HASHDEL (void\*) -1

using namespace std;

// Объявление структуры Object для работы с хеш-таблицей

struct Object

{

void\*\* data; // Указатель на данные

Object(int, int(\*)(void\*)); // Конструктор структуры

int size;

int N;

int(\*getKey)(void\*); // Указатель на функцию получения ключа

bool insert(void\*); // Метод вставки элемента

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*)); // Метод сканирования хеш-таблицы

};

static void\* DEL = (void\*)HASHDEL; // Статическая переменная для обозначения удаленного элемента

Object create(int size, int(\*getkey)(void\*)); // Функция создания объекта хеш-таблицы

#undef HASHDEL

**Hash.cpp**

#include "Hash.h"

#include <iostream>

// Функция универсального хеширования

int HashFunction(int key, int size)

{

// Генерация случайных коэффициентов a и b

int a = rand() % size + 1;

int b = rand() % size - 1;

int p = 10000001;

// Вычисление хеша с использованием универсальной хеш-функции

return ((a \* key + b) % p) % size;

}

//Функция модульного хеширования

//int HashFunction(int key, int size)

//{

// return key % 32;

//}

// Функция перехеширования

int Next\_hash(int hash, int size)

{

int p = 10000001;

// Вычисление нового хеша с использованием метода перехеширования

return (hash + 1) % size;

}

// Создание объекта хеш-таблицы

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

{

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

}

// Конструктор класса Object

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

{

N = 0;

this->size = size;

this->getKey = getkey;

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

// Инициализация хеш-таблицы пустыми указателями

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)

{

// Поиск свободного слота для вставки элемента

for (int i = 0, t = getKey(d), j = HashFunction(t, size); i != size && !b; j = Next\_hash(j, size))

{

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

{

data[j] = d; // Вставка элемента

N++; // Увеличение счетчика элементов

b = true;

}

}

}

return b; // Возвращение результата вставки

}

// Поиск индекса элемента по ключу

int Object::searchInd(int key)

{

bool find = false;

int hash\_index = HashFunction(key, size);

// Цикл поиска элемента

while (!find)

{

if (data[hash\_index] != NULL && data[hash\_index] != DEL && getKey(data[hash\_index]) == key)

{

find = true;

return hash\_index; // Возвращение найденного индекса

}

hash\_index = Next\_hash(hash\_index, size); // Переход к следующему хешу

}

}

// Поиск элемента по ключу

void\* Object::search(int key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента или NULL

}

// Удаление элемента по ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i]; // Сохранение удаленного элемента

if (t != NULL)

{

data[i] = DEL; // Пометка элемента как удаленного

N--; // Уменьшение счетчика элементов

}

return t; // Возвращение удаленного элемента

}

// Удаление элемента по значению

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу

}

// Просмотр всех элементов в хеш-таблице

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

{

int collisions = 0;

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

{

cout << " Элемент " << i;

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

{

cout << " Пусто" << endl; // Вывод информации о пустом слоте

}

else

{

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

{

cout << " Удален" << endl; // Вывод информации о удаленном элементе

}

else

{

// Проверка на коллизии

int index = HashFunction(getKey((this->data)[i]), size);

while (index != i)

{

if ((this->data)[index] != NULL && (this->data)[index] != DEL)

{

collisions++; // Увеличение количества коллизий

}

index = Next\_hash(index, size);

}

f((this->data)[i]); // Вызов пользовательской функции для распечатки элемента

}

}

}

cout << " (Коллизий: " << collisions << ")" << endl;

}

**Lab\_13.cpp**

#include "Hash.h"

#include <iostream>

#include <string>

#include <ctime>

// Объявление пользовательской структуры AAA

struct AAA

{

int key; // Целочисленное значение ключа

const char\* mas; // Указатель на строку

// Конструктор с параметрами

AAA(int k, const char z[])

{

key = k; mas = z;

} AAA() {} // Конструктор по умолчанию

};

// Функция вычисления ключа

int key(void\* d)

{

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

return f->key;

}

// Функция печати элемента AAA

void AAA\_print(void\* d)

{

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

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ru");

int siz, choice;

int k;

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

cin >> siz;

Object H = create(siz, key); // Создание хеш - таблицы с указанным размером и функцией хеширования

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: ";

cin >> choice;

switch (choice)

{

case 0: exit(0); // Выход из программы

case 1:

{

H.scan(AAA\_print); // Вывод содержимого хеш-таблицы

}

break;

case 2:

{

AAA\* a = new AAA;

char\* str = new char[20];

cout << "Введите ключ: ";

cin >> k;;

a->key = k;

cout << "Введите строку: ";

cin >> str;

a->mas = str;

if (H.N == H.size)

{

cout << "Таблица заполнена" << endl;

}

else

{

H.insert(a); // Добавление элемента в хеш-таблицу

}

}

break;

case 3:

{

cout << "Введите ключ для удаления: ";

cin >> k;

H.deleteByKey(k); // Удаление элемента по ключу

}

break;

case 4:

{

cout << "Введите ключ для поиска: ";

cin >> k;

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

{

cout << "Элемент не найден" << endl;

}

else

{

AAA\_print(H.search(k)); // Поиск и печать элемента по ключу

}

}

break;

}

}

return 0;

}

**Универсальное хеширование:**

****

****

****

****

**Модульное хеширование:**

****

****

****

****

Доп. задачи

**Вариант 14**

Использовать в проекте функции универсального хеширования и модульного. Сравнить время поиска информации.

**Hash.h**

#pragma once

#define HASHDEL (void\*) -1

using namespace std;

// Объявление структуры Object для работы с хеш-таблицей

struct Object

{

void\*\* data; // Указатель на данные

Object(int, int(\*)(void\*)); // Конструктор структуры

int size;

int N;

int(\*getKey)(void\*); // Указатель на функцию получения ключа

bool insert(void\*); // Метод вставки элемента

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*)); // Метод сканирования хеш-таблицы

};

static void\* DEL = (void\*)HASHDEL; // Статическая переменная для обозначения удаленного элемента

Object create(int size, int(\*getkey)(void\*)); // Функция создания объекта хеш-таблицы

#undef HASHDEL

**Hash.cpp**

#include "Hash.h"

#include <iostream>

// Функция универсального хеширования

int HashFunction(int key, int size)

{

// Генерация случайных коэффициентов a и b

int a = rand() % size + 1;

int b = rand() % size - 1;

int p = 10000001;

// Вычисление хеша с использованием универсальной хеш-функции

return ((a \* key + b) % p) % size;

}

//Функция модульного хеширования

//int HashFunction(int key, int size)

//{

// return key % 32;

//}

// Функция перехеширования

int Next\_hash(int hash, int size)

{

int p = 10000001;

// Вычисление нового хеша с использованием метода перехеширования

return (hash + 1) % size;

}

// Создание объекта хеш-таблицы

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

{

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

}

// Конструктор класса Object

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

{

N = 0;

this->size = size;

this->getKey = getkey;

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

// Инициализация хеш-таблицы пустыми указателями

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)

{

// Поиск свободного слота для вставки элемента

for (int i = 0, t = getKey(d), j = HashFunction(t, size); i != size && !b; j = Next\_hash(j, size))

{

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

{

data[j] = d; // Вставка элемента

N++; // Увеличение счетчика элементов

b = true;

}

}

}

return b; // Возвращение результата вставки

}

// Поиск индекса элемента по ключу

int Object::searchInd(int key)

{

bool find = false;

int hash\_index = HashFunction(key, size);

// Цикл поиска элемента

while (!find)

{

if (data[hash\_index] != NULL && data[hash\_index] != DEL && getKey(data[hash\_index]) == key)

{

find = true;

return hash\_index; // Возвращение найденного индекса

}

hash\_index = Next\_hash(hash\_index, size); // Переход к следующему хешу

}

}

// Поиск элемента по ключу

void\* Object::search(int key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента или NULL

}

// Удаление элемента по ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i]; // Сохранение удаленного элемента

if (t != NULL)

{

data[i] = DEL; // Пометка элемента как удаленного

N--; // Уменьшение счетчика элементов

}

return t; // Возвращение удаленного элемента

}

// Удаление элемента по значению

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу

}

// Просмотр всех элементов в хеш-таблице

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

{

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

{

cout << " Элемент " << i;

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

{

cout << " Пусто" << endl; // Вывод информации о пустом слоте

}

else

{

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

{

cout << " Удален" << endl; // Вывод информации о удаленном элементе

}

else

{

f((this->data)[i]); // Вызов пользовательской функции для распечатки элемента

}

}

}

}

**Lab\_13.cpp**

#include "Hash.h"

#include <iostream>

#include <string>

#include <ctime>

// Объявление пользовательской структуры AAA

struct AAA

{

int key; // Целочисленное значение ключа

const char\* mas; // Указатель на строку

// Конструктор с параметрами

AAA(int k, const char z[])

{

key = k; mas = z;

} AAA() {} // Конструктор по умолчанию

};

// Функция вычисления ключа

int key(void\* d)

{

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

return f->key;

}

// Функция печати элемента AAA

void AAA\_print(void\* d)

{

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

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ru");

int siz, choice;

int k;

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

cin >> siz;

Object H = create(siz, key); // Создание хеш - таблицы с указанным размером и функцией хеширования

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: ";

cin >> choice;

switch (choice)

{

case 0: exit(0); // Выход из программы

case 1:

{

H.scan(AAA\_print); // Вывод содержимого хеш-таблицы

}

break;

case 2:

{

AAA\* a = new AAA;

char\* str = new char[20];

cout << "Введите ключ: ";

cin >> k;;

a->key = k;

cout << "Введите строку: ";

cin >> str;

a->mas = str;

if (H.N == H.size)

{

cout << "Таблица заполнена" << endl;

}

else

{

H.insert(a); // Добавление элемента в хеш-таблицу

}

}

break;

case 3:

{

cout << "Введите ключ для удаления: ";

cin >> k;

H.deleteByKey(k); // Удаление элемента по ключу

}

break;

case 4:

{

cout << "Введите ключ для поиска: ";

cin >> k;

clock\_t start = clock(); // Начало отсчета времени

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

{

cout << "Элемент не найден" << endl;

}

else

{

AAA\_print(H.search(k)); // Поиск и печать элемента по ключу

}

clock\_t end = clock(); // Конец отсчета времени

cout << "Время: " << end - start << " мс" << endl;

}

break;

}

}

return 0;

}

**Универсальное хеширование:**

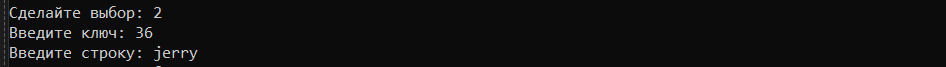
****

****

****

**Модульное хеширование:**

****

****

****

**Вариант 7**

Изменить функцию вычисления хеш на универсальную.

**Hash.h**

#pragma once

#define HASHDEL (void\*) -1

using namespace std;

// Объявление структуры Object для работы с хеш-таблицей

struct Object

{

void\*\* data; // Указатель на данные

Object(int, int(\*)(void\*)); // Конструктор структуры

int size;

int N;

int(\*getKey)(void\*); // Указатель на функцию получения ключа

bool insert(void\*); // Метод вставки элемента

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*)); // Метод сканирования хеш-таблицы

};

static void\* DEL = (void\*)HASHDEL; // Статическая переменная для обозначения удаленного элемента

Object create(int size, int(\*getkey)(void\*)); // Функция создания объекта хеш-таблицы

#undef HASHDEL

**Hash.cpp**

#include "Hash.h"

#include <iostream>

// Функция универсального хеширования

int HashFunction(int key, int size)

{

// Генерация случайных коэффициентов a и b

int a = rand() % size + 1;

int b = rand() % size - 1;

int p = 10000001;

// Вычисление хеша с использованием универсальной хеш-функции

return ((a \* key + b) % p) % size;

}

// Функция перехеширования

int Next\_hash(int hash, int size)

{

int p = 10000001;

// Вычисление нового хеша с использованием метода перехеширования

return (hash + 1) % size;

}

// Создание объекта хеш-таблицы

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

{

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

}

// Конструктор класса Object

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

{

N = 0;

this->size = size;

this->getKey = getkey;

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

// Инициализация хеш-таблицы пустыми указателями

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)

{

// Поиск свободного слота для вставки элемента

for (int i = 0, t = getKey(d), j = HashFunction(t, size); i != size && !b; j = Next\_hash(j, size))

{

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

{

data[j] = d; // Вставка элемента

N++; // Увеличение счетчика элементов

b = true;

}

}

}

return b; // Возвращение результата вставки

}

// Поиск индекса элемента по ключу

int Object::searchInd(int key)

{

bool find = false;

int hash\_index = HashFunction(key, size);

// Цикл поиска элемента

while (!find)

{

if (data[hash\_index] != NULL && data[hash\_index] != DEL && getKey(data[hash\_index]) == key)

{

find = true;

return hash\_index; // Возвращение найденного индекса

}

hash\_index = Next\_hash(hash\_index, size); // Переход к следующему хешу

}

}

// Поиск элемента по ключу

void\* Object::search(int key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента или NULL

}

// Удаление элемента по ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i]; // Сохранение удаленного элемента

if (t != NULL)

{

data[i] = DEL; // Пометка элемента как удаленного

N--; // Уменьшение счетчика элементов

}

return t; // Возвращение удаленного элемента

}

// Удаление элемента по значению

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу

}

// Просмотр всех элементов в хеш-таблице

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

{

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

{

cout << " Элемент " << i;

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

{

cout << " Пусто" << endl; // Вывод информации о пустом слоте

}

else

{

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

{

cout << " Удален" << endl; // Вывод информации о удаленном элементе

}

else

{

f((this->data)[i]); // Вызов пользовательской функции для распечатки элемента

}

}

}

}

**Lab\_13.cpp**

#include "Hash.h"

#include <iostream>

#include <string>

#include <ctime>

// Объявление пользовательской структуры AAA

struct AAA

{

int key; // Целочисленное значение ключа

const char\* mas; // Указатель на строку

// Конструктор с параметрами

AAA(int k, const char z[])

{

key = k; mas = z;

} AAA() {} // Конструктор по умолчанию

};

// Функция вычисления ключа

int key(void\* d)

{

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

return f->key;

}

// Функция печати элемента AAA

void AAA\_print(void\* d)

{

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

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ru");

int siz, choice;

int k;

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

cin >> siz;

Object H = create(siz, key); // Создание хеш - таблицы с указанным размером и функцией хеширования

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: ";

cin >> choice;

switch (choice)

{

case 0: exit(0); // Выход из программы

case 1:

{

H.scan(AAA\_print); // Вывод содержимого хеш-таблицы

}

break;

case 2:

{

AAA\* a = new AAA;

char\* str = new char[20];

cout << "Введите ключ: ";

cin >> k;;

a->key = k;

cout << "Введите строку: ";

cin >> str;

a->mas = str;

if (H.N == H.size)

{

cout << "Таблица заполнена" << endl;

}

else

{

H.insert(a); // Добавление элемента в хеш-таблицу

}

}

break;

case 3:

{

cout << "Введите ключ для удаления: ";

cin >> k;

H.deleteByKey(k); // Удаление элемента по ключу

}

break;

case 4:

{

cout << "Введите ключ для поиска: ";

cin >> k;

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

{

cout << "Элемент не найден" << endl;

}

else

{

AAA\_print(H.search(k)); // Поиск и печать элемента по ключу

}

}

break;

}

}

return 0;

}

****

****

****

**Вариант 16**

Изменить хеш-функцию в проекте на модульную.

**Hash.h**

#pragma once

#define HASHDEL (void\*) -1

using namespace std;

// Объявление структуры Object для работы с хеш-таблицей

struct Object

{

void\*\* data; // Указатель на данные

Object(int, int(\*)(void\*)); // Конструктор структуры

int size;

int N;

int(\*getKey)(void\*); // Указатель на функцию получения ключа

bool insert(void\*); // Метод вставки элемента

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

void scan(void(\*f)(void\*)); // Метод сканирования хеш-таблицы

};

static void\* DEL = (void\*)HASHDEL; // Статическая переменная для обозначения удаленного элемента

Object create(int size, int(\*getkey)(void\*)); // Функция создания объекта хеш-таблицы

#undef HASHDEL

**Hash.cpp**

#include "Hash.h"

#include <iostream>

//Функция модульного хеширования

int HashFunction(int key, int size)

{

return key % 32;

}

// Функция перехеширования

int Next\_hash(int hash, int size)

{

int p = 10000001;

// Вычисление нового хеша с использованием метода перехеширования

return (hash + 1) % size;

}

// Создание объекта хеш-таблицы

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

{

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

}

// Конструктор класса Object

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

{

N = 0;

this->size = size;

this->getKey = getkey;

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

// Инициализация хеш-таблицы пустыми указателями

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)

{

// Поиск свободного слота для вставки элемента

for (int i = 0, t = getKey(d), j = HashFunction(t, size); i != size && !b; j = Next\_hash(j, size))

{

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

{

data[j] = d; // Вставка элемента

N++; // Увеличение счетчика элементов

b = true;

}

}

}

return b; // Возвращение результата вставки

}

// Поиск индекса элемента по ключу

int Object::searchInd(int key)

{

bool find = false;

int hash\_index = HashFunction(key, size);

// Цикл поиска элемента

while (!find)

{

if (data[hash\_index] != NULL && data[hash\_index] != DEL && getKey(data[hash\_index]) == key)

{

find = true;

return hash\_index; // Возвращение найденного индекса

}

hash\_index = Next\_hash(hash\_index, size); // Переход к следующему хешу

}

}

// Поиск элемента по ключу

void\* Object::search(int key)

{

int t = searchInd(key);

return(t >= 0) ? (data[t]) : (NULL); // Возвращение найденного элемента или NULL

}

// Удаление элемента по ключу

void\* Object::deleteByKey(int key)

{

int i = searchInd(key);

void\* t = data[i]; // Сохранение удаленного элемента

if (t != NULL)

{

data[i] = DEL; // Пометка элемента как удаленного

N--; // Уменьшение счетчика элементов

}

return t; // Возвращение удаленного элемента

}

// Удаление элемента по значению

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Удаление элемента по ключу

}

// Просмотр всех элементов в хеш-таблице

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

{

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

{

cout << " Элемент " << i;

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

{

cout << " Пусто" << endl; // Вывод информации о пустом слоте

}

else

{

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

{

cout << " Удален" << endl; // Вывод информации о удаленном элементе

}

else

{

f((this->data)[i]); // Вызов пользовательской функции для распечатки элемента

}

}

}

}

**Lab\_13.cpp**

#include "Hash.h"

#include <iostream>

#include <string>

#include <ctime>

// Объявление пользовательской структуры AAA

struct AAA

{

int key; // Целочисленное значение ключа

const char\* mas; // Указатель на строку

// Конструктор с параметрами

AAA(int k, const char z[])

{

key = k; mas = z;

} AAA() {} // Конструктор по умолчанию

};

// Функция вычисления ключа

int key(void\* d)

{

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

return f->key;

}

// Функция печати элемента AAA

void AAA\_print(void\* d)

{

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

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "ru");

int siz, choice;

int k;

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

cin >> siz;

Object H = create(siz, key); // Создание хеш - таблицы с указанным размером и функцией хеширования

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: ";

cin >> choice;

switch (choice)

{

case 0: exit(0); // Выход из программы

case 1:

{

H.scan(AAA\_print); // Вывод содержимого хеш-таблицы

}

break;

case 2:

{

AAA\* a = new AAA;

char\* str = new char[20];

cout << "Введите ключ: ";

cin >> k;;

a->key = k;

cout << "Введите строку: ";

cin >> str;

a->mas = str;

if (H.N == H.size)

{

cout << "Таблица заполнена" << endl;

}

else

{

H.insert(a); // Добавление элемента в хеш-таблицу

}

}

break;

case 3:

{

cout << "Введите ключ для удаления: ";

cin >> k;

H.deleteByKey(k); // Удаление элемента по ключу

}

break;

case 4:

{

cout << "Введите ключ для поиска: ";

cin >> k;

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

{

cout << "Элемент не найден" << endl;

}

else

{

AAA\_print(H.search(k)); // Поиск и печать элемента по ключу

}

}

break;

}

}

return 0;

}

****

****

****