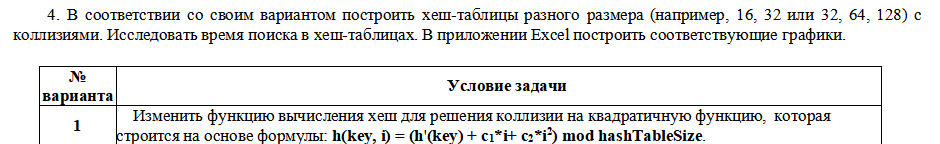
Банкузов Михаил 7 группа Лабораторная работа №14 Вариант 1



Tree.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

Hash.cpp

// Подключение заголовочного файла "Tree.h".

#include "Tree.h"

// Подключение заголовочного файла <iostream>.

#include <iostream>

// Функция Хеш-функции, которая принимает ключ, размер таблицы и параметр p.

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

{

return (p + key) % size;

}

// Функция Next\_hash, которая принимает хеш-код, размер таблицы и параметр p.

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

{

// Вариант 1 функции Next\_hash.

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

}

// Функция create для создания нового объекта хеш-таблицы с заданным размером и указателем на функцию getKey.

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

{

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

}

// Конструктор для объекта хеш-таблицы, который принимает размер и указатель на функцию getKey.

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

{

// Инициализация количества элементов и размера таблицы.

N = 0;

this->size = size;

// Указатель на функцию getKey.

this->getKey = getkey;

// Выделение памяти для массива данных и инициализация его значениями NULL.

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

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

data[i] = NULL;

}

// Функция insert для вставки элемента в хеш-таблицу.

bool Object::insert(void\* d)

{

bool b = false;

// Проверка, что количество элементов не превышает размер таблицы.

if (N != size)

// Поиск пустого слота в таблице для вставки элемента.

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)

{

// Вставка элемента в найденный слот.

data[j] = d;

N++;

b = true;

}

return b;

}

// Функция searchInd для поиска индекса элемента в хеш-таблице по ключу.

int Object::searchInd(int key)

{

// Инициализация индекса t и флага b.

int t = -1;

bool b = false;

// Проверка, что таблица не пуста.

if (N != 0)

// Поиск элемента в таблице по ключу.

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;

}

// Функция search для поиска элемента в хеш-таблице по ключу.

void\* Object::search(int key)

{

// Поиск индекса элемента в таблице.

int t = searchInd(key);

// Возврат элемента, если он найден, и NULL в противном случае.

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

}

// Функция deleteByKey для удаления элемента из хеш-таблицы по ключу.

void\* Object::deleteByKey(int key)

{

// Поиск индекса элемента в таблице по ключу.

int i = searchInd(key);

// Сохранение ссылки на элемент в переменную t.

void\* t = data[i];

// Если элемент найден, помечаем его как удаленный и уменьшаем количество элементов в таблице.

if (t != NULL)

{

data[i] = DEL;

N--;

}

// Возврат ссылки на удаленный элемент.

return t;

}

// Функция deleteByValue для удаления элемента из хеш-таблицы по значению.

bool Object::deleteByValue(void\* d)

{

// Удаление элемента из таблицы по ключу, полученному с помощью функции getKey.

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

}

// Функция scan для применения заданной функции к каждому элементу в хеш-таблице.

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

{

// Обход всех элементов в таблице.

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

{

// Вывод информации о текущем элементе.

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]);

}

}

Основной код

#include "Tree.h"

#include <iostream>

using namespace std;

struct AAA

{

int key;

std::string mas;

AAA(int k, const 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");

AAA a1(1, "one"), a2(2, "two"), a3(4, "three"), a4(2, "fo");

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;

std::string str;

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));

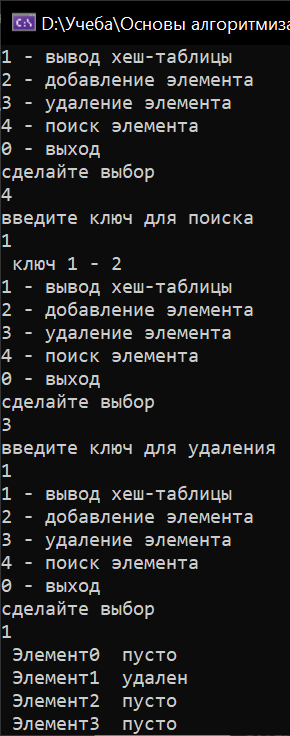
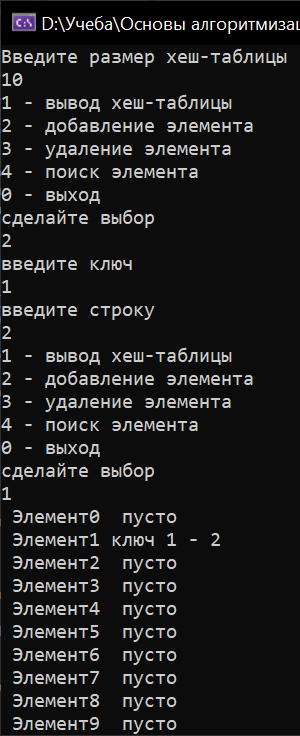
} break;

}

}

return 0;

}



Дополнительные задания

Вариант 9



Tree.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

Hash.cpp

// Подключение заголовочного файла "Tree.h".

#include "Tree.h"

// Подключение заголовочного файла <iostream>.

#include <iostream>

// Функция Хеш-функции, которая принимает ключ, размер таблицы и параметр p.

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

{

return (p + key) % size;

}

// Функция Next\_hash, которая принимает хеш-код, размер таблицы и параметр p.

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

{

// Вариант 1 функции Next\_hash.

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

}

// Функция create для создания нового объекта хеш-таблицы с заданным размером и указателем на функцию getKey.

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

{

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

}

// Конструктор для объекта хеш-таблицы, который принимает размер и указатель на функцию getKey.

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

{

// Инициализация количества элементов и размера таблицы.

N = 0;

this->size = size;

// Указатель на функцию getKey.

this->getKey = getkey;

// Выделение памяти для массива данных и инициализация его значениями NULL.

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

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

data[i] = NULL;

}

// Функция insert для вставки элемента в хеш-таблицу.

bool Object::insert(void\* d)

{

bool b = false;

// Проверка, что количество элементов не превышает размер таблицы.

if (N != size)

// Поиск пустого слота в таблице для вставки элемента.

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)

{

// Вставка элемента в найденный слот.

data[j] = d;

N++;

b = true;

}

return b;

}

// Функция searchInd для поиска индекса элемента в хеш-таблице по ключу.

int Object::searchInd(int key)

{

// Инициализация индекса t и флага b.

int t = -1;

bool b = false;

// Проверка, что таблица не пуста.

if (N != 0)

// Поиск элемента в таблице по ключу.

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;

}

// Функция search для поиска элемента в хеш-таблице по ключу.

void\* Object::search(int key)

{

// Поиск индекса элемента в таблице.

int t = searchInd(key);

// Возврат элемента, если он найден, и NULL в противном случае.

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

}

// Функция deleteByKey для удаления элемента из хеш-таблицы по ключу.

void\* Object::deleteByKey(int key)

{

// Поиск индекса элемента в таблице по ключу.

int i = searchInd(key);

// Сохранение ссылки на элемент в переменную t.

void\* t = data[i];

// Если элемент найден, помечаем его как удаленный и уменьшаем количество элементов в таблице.

if (t != NULL)

{

data[i] = DEL;

N--;

}

// Возврат ссылки на удаленный элемент.

return t;

}

// Функция deleteByValue для удаления элемента из хеш-таблицы по значению.

bool Object::deleteByValue(void\* d)

{

// Удаление элемента из таблицы по ключу, полученному с помощью функции getKey.

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

}

// Функция scan для применения заданной функции к каждому элементу в хеш-таблице.

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

{

// Обход всех элементов в таблице.

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

{

// Вывод информации о текущем элементе.

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]);

}

}

Основной код

#include "Tree.h"

#include <iostream>

using namespace std;

struct AAA

{

int key;

std::string mas;

AAA(int k, const 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");

AAA a1(1, "one"), a2(2, "two"), a3(4, "three"), a4(2, "fo");

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;

std::string str;

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));

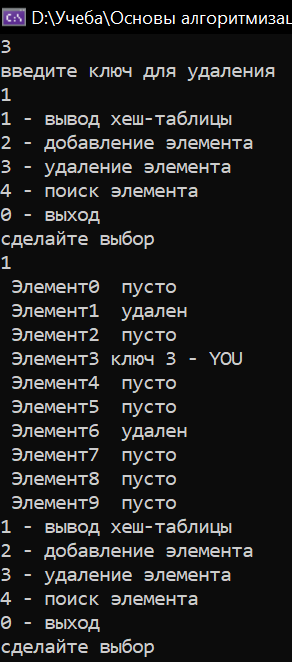
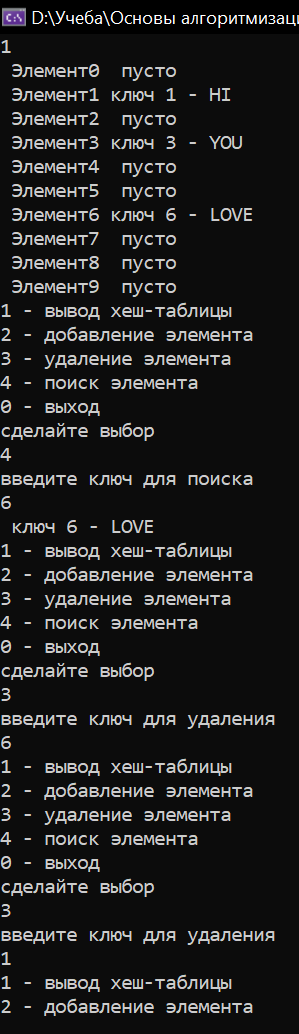
} break;

}

}

return 0;

}



Вариант 2



Tree.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

Hash.cpp

// Подключение заголовочного файла "Tree.h".

#include "Tree.h"

// Подключение заголовочного файла <iostream>.

#include <iostream>

// Функция Хеш-функции, которая принимает ключ, размер таблицы и параметр p.

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

{

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

return (p + key) % size;

}

// Функция Next\_hash, которая принимает хеш-код, размер таблицы и параметр p.

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

{

// Вариант 1 функции Next\_hash.

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

}

// Универсальная хеш-функция

int universal(int key, int size, int p)

{

// Инициализируем переменные h, a, b

int h = 0, a = 314445, b = 37542;

// Вычисляем значение h с помощью формулы (a \* h + key) % size

h = (a \* h + key) % size;

// Если значение h отрицательное, то прибавляем к нему значение size

return (h < 0) ? (h + size) : h;

}

// Функция create для создания нового объекта хеш-таблицы с заданным размером и указателем на функцию getKey.

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

{

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

}

// Конструктор для объекта хеш-таблицы, который принимает размер и указатель на функцию getKey.

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

{

// Инициализация количества элементов и размера таблицы.

N = 0;

this->size = size;

// Указатель на функцию getKey.

this->getKey = getkey;

// Выделение памяти для массива данных и инициализация его значениями NULL.

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

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

data[i] = NULL;

}

// Функция insert для вставки элемента в хеш-таблицу.

bool Object::insert(void\* d)

{

bool b = false;

// Если количество элементов N не равно размеру таблицы size, то добавляем новый элемент в таблицу

if (N != size)

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

}

// Возвращаем значение b

return b;

}

// Функция searchInd для поиска индекса элемента в хеш-таблице по ключу.

int Object::searchInd(int key)

{

int t = -1;

bool b = false;

// Если количество элементов N не равно нулю, то ищем элемент с заданным ключом в таблице

if (N != 0)

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

if (data[j] != DEL)

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

{

t = j; b = true;

}

// Возвращаем индекс найденного элемента или -1, если элемент не найден

return t;

}

// Функция search для поиска элемента в хеш-таблице по ключу.

void\* Object::search(int key)

{

// Поиск индекса элемента в таблице.

int t = searchInd(key);

// Возврат элемента, если он найден, и NULL в противном случае.

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

}

// Функция deleteByKey для удаления элемента из хеш-таблицы по ключу.

void\* Object::deleteByKey(int key)

{

// Поиск индекса элемента в таблице по ключу.

int i = searchInd(key);

// Сохранение ссылки на элемент в переменную t.

void\* t = data[i];

// Если элемент найден, помечаем его как удаленный и уменьшаем количество элементов в таблице.

if (t != NULL)

{

data[i] = DEL;

N--;

}

// Возврат ссылки на удаленный элемент.

return t;

}

// Функция deleteByValue для удаления элемента из хеш-таблицы по значению.

bool Object::deleteByValue(void\* d)

{

// Удаление элемента из таблицы по ключу, полученному с помощью функции getKey.

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

}

// Функция scan для применения заданной функции к каждому элементу в хеш-таблице.

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

{

// Обход всех элементов в таблице.

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

{

// Вывод информации о текущем элементе.

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]);

}

}

Основной код

#include "Tree.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");

AAA a1(1, (char\*)"one"), a2(2, (char\*)"two"), a3(4, (char\*)"three"), a4(2, (char\*)"fo");

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));

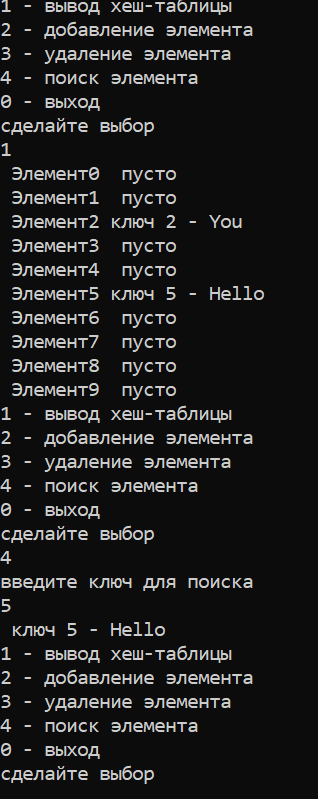
} break;

}

}

return 0;

}



Вариант 10

Просто изменить

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

{

return (hash + p) % size; // линейная функция

}