**Лабораторная работа №13. Хэш-таблицы с открытой адресацией**

**Вариант 10**

Условие:

Изменить функцию вычисления хэш на мультипликативную функцию, которая строится на основе формулы:  **H(key) = [hashTableSize(key\*A mod 1)]**, где **key\*A mod 1** – дробная часть **key\*A**,

**A = (sqrt(5) - 1) / 2 = 0,6180339887499**

Код:

Главная функция:

#include "stdafx.h"

#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 \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

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

int siz = 10;

cout << "Введите размер хэш-таблицы" << endl; cin >> siz;

Object H = Create(siz, key); //создать

int choise=0; int k;

for (;;)

{

cout << "1 - вывод хэш-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 >> choise;

switch (choise)

{

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.Delete(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 "stdafx.h"

#include "Hash.h"

#include <iostream>

#include <math.h>

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

{

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

int h=fmod(key\*A,1)\*size;

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

return h; //return ((int)(p+fmod(((key\*(sqrt(5.0)-1))/2), 1)))%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 = HashFunction(t, Size, 0); i != Size && !b;

j = HashFunction(j, Size, ++i))

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

{ Data[j] = d; N++; b = true; }

else{

if(Data[j] != DEL||Data[j] != NULL)

{

while(Data[j]!=NULL)

j++;

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 = HashFunction(key, Size, 0); Data[j] != NULL && i != Size && !b; j = HashFunction(key, Size, ++i))

{

if(Data[j]==DEL)

{

j++;

while(GetKey(Data[j])!=key)

{ j++;

if (GetKey(Data[j]) == key)

{

t = j;

b = true;

return t;

}

}

}

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::Delete(int key)

{

int i = SearchInd(key);

void\* t = Data[i];

if (t != NULL)

{

Data[i] = DEL; N--;

}

return t;

}

bool Object::Delete(void\* d)

{

return(Delete(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]);

}

}

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\* Delete(int key); bool Delete(void\*);

void Scan(void(\*f)(void\*));

};

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

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

#undef HASHDEL

Скрин

