

Можна не трэба гэтыя графікі яшчэ будаваць? Я не умею так карыстацца экселем, гэтага мне яшчэ не хапала(



Source.cpp:

#include "Hash.h"

#include <iostream>

#include <Windows.h>

using namespace std;

// Структура AAA для прадстаўлення элементаў хэш-табліцы.

struct AAA

{

int key; // Ключ элемента.

char\* mas; // Масіў сімвалаў.

// Канструктар з параметрамі.

AAA(int k, char\* z)

{

key = k; // Прысвойванне ключа.

mas = z; // Прысвойванне масіва сімвалаў.

}

// Пусты канструктар (па змаўчанні).

AAA() {}

};

// Функцыя для атрымання ключа элемента AAA.

int key(void\* d)

{

AAA\* f = (AAA\*)d; // Прывядзенне паказальніка да тыпу AAA.

return f->key; // Вяртанне ключа элемента.

}

// Функцыя для вываду элемента AAA на экран.

void AAA\_print(void\* d)

{

std::cout << " ключ " << ((AAA\*)d)->key << " - " << ((AAA\*)d)->mas << std::endl; // Вывад ключа і масіва сімвалаў элемента AAA.

}

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

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int siz = 10, choice, k;

cout << "Увядзіце памер хэш-табліцы" << endl;

cin >> siz;

Object H = create(siz, key);

for (;;)

{

cout << "1 - вывад хэш-табліцы" << endl;

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

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

cout << "4 - пошук элемента" << endl;

cout << "0 - выхад" << endl;

cout << "зрабіце выбар" << 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 << "увядзіце ключ" << endl;

cin >> k;

a->key = k;

cout << "увядзіце радок" << endl;

cin >> str;

a->mas = str;

if (H.N == H.size)

cout << "Табліца запоўнена" << endl;

else

H.insert(a);

} break;

case 3: {

cout << "увядзіце ключ для выдалення" << endl;

cin >> k;

H.deleteByKey(k);

} break;

case 4: {

cout << "увядзіце ключ для пошуку" << endl;

cin >> k;

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

cout << "Элемент не знойдзены" << endl;

else

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

} break;

}

}

return 0;

}

Hash.h:

#pragma once

// Макрас для абазначэння выдаленага элемента ў хэш-табліцы.

#define HASHDEL (void\*) -1

// Структура для прадстаўлення хэш-табліцы.

struct Object

{

void\*\* data; // Масіў паказальнікаў на элементы хэш-табліцы.

int size; // Памер хэш-табліцы.

int N; // Колькасць элементаў у хэш-табліцы.

int(\*getKey)(void\*); // Паказальнік на функцыю для атрымання ключа элемента.

// Канструктар з параметрамі.

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

// Метады для працы з хэш-табліцай.

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

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

};

// Паказальнік на выдалены элемент.

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

// Функцыя для стварэння аб'екта хэш-табліцы.

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

// Адмена вызначэння макраса HASHDEL.

#undef HASHDEL

Hash.cpp:

#include "Hash.h"

#include <iostream>

// Функцыя вылічэнні хэша з выкарыстаннем мультыплікатыўнай хэш-функцыі.

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

const double A = 0.6180339887499; // Канстанта A для мультыплікатыўнай хэш-функцыі

double hashValue = key \* A; // Вылічэнне прамежкавага значэння хеша

hashValue -= int(hashValue); // Атрыманне дробавай часткі значэння хеша

return (p + int(size \* hashValue)) % size; // Улік параметру p пры вылічэнні канчатковага значэння хеша

}

// Функцыя для вылічэння наступнага значэння хеша.

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

{

return (hash + 5 \* p + 3 \* p \* p) % size; // Вылічэнне наступнага значэння хеша з улікам параметру p.

}

// Функцыя для стварэння аб'екта хэш-табліцы.

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

{

return \*(new Object(size, getkey)); // Стварэнне і вяртанне новага аб'екта хэш-табліцы.

}

// Канструктар аб'екта хэш-табліцы.

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

{

N = 0; // Ініцыялізацыя колькасці элементаў у табліцы.

this->size = size; // Заданне памеру табліцы.

this->getKey = getkey; // Усталяванне функцыі для атрымання ключа элемента.

this->data = new void\* [size]; // Вылучэнне памяці пад масіў паказальнікаў на элементы.

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

data[i] = NULL; // Ініцыялізацыя ўсіх элементаў масіва як NULL.

}

// Функцыя для ўстаўкі элемента ў хэш-табліцу.

bool Object::insert(void\* d)

{

bool b = false; // Сцяг для абазначэння паспяховасці ўстаўкі.

if (N != size) // Праверка на перапаўненне табліцы.

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

i != size && !b; j = Next\_hash(j, size, ++i)) // Цыкл пошуку месца для ўстаўкі.

if (data[j] == NULL || data[j] == DEL) // Праверка на пустэчу або аддаленасць вочка.

{

data[j] = d; // Устаўка элемента.

N++; // Павелічэнне колькасці элементаў у табліцы.

b = true; // Пазнака паспяховай устаўкі.

}

return b; // Вяртанне выніку ўстаўкі.

}

// Функцыя для пошуку індэкса элемента ў хэш-табліцы.

int Object::searchInd(int key)

{

int t = -1; // Ініцыялізацыя індэкса элемента.

bool b = false; // Сцяг для абазначэння паспяховасці пошуку.

if (N != 0) // Праверка на наяўнасць элементаў у табліцы.

for (int i = 0, j = HashFunction(key, size, 0); data[j] != NULL && i != size && !b; j = HashFunction(key, size, ++i)) // Цыкл пошуку элемента.

if (data[j] != DEL) // Праверка на аддаленасць ячэйкі.

if (getKey(data[j]) == key) // Праверка на адпаведнасць ключа элемента.

{

t = j; // Захаванне індэкса элемента.

b = true; // Пазнака паспяховага пошуку.

}

return t; // Вяртанне азначніка элемента (або -1, калі не знойдзены).

}

// Функцыя для пошуку элемента ў хэш-табліцы па ключы.

void\* Object::search(int key)

{

int t = searchInd(key); // Пошук індэкса элемента.

return (t >= 0) ? (data[t]) : (NULL); // Вяртанне знойдзенага элемента (або NULL, калі не знойдзены).

}

// Функцыя для выдалення элемента з хэш-табліцы па ключы.

void\* Object::deleteByKey(int key)

{

int i = searchInd(key); // Пошук індэкса элемента.

void\* t = data[i]; // Захаванне выдалянага элемента.

if (t != NULL) // Праверка на наяўнасць элемента для выдалення.

{

data[i] = DEL; // Пазнака вочка як выдаленай.

N--; // Памяншэнне колькасці элементаў у табліцы.

}

return t; // Вяртанне выдаленага элемента (або NULL, калі не знойдзены).

}

// Функцыя для выдалення элемента з хэш-табліцы па значэнні.

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Выдаленне элемента па ключы, атрыманым з значэння.

}

// Функцыя для сканавання змесціва хэш-табліцы і прымянення да кожнага элемента названай функцыі.

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

{

for (int i = 0; i < this->size; i++) // Цыкл сканавання ўсіх вочак табліцы.

{

std::cout << "Элемент" << i << '\t'; // Выснова нумара ячэйкі.

if ((this->data)[i] == NULL) // Праверка на пустэчу вочка.

std::cout << "пуста" << std::endl; // Выснова паведамлення аб пустым вочку.

else

if ((this->data)[i] == DEL) // Праверка на аддаленасць ячэйкі.

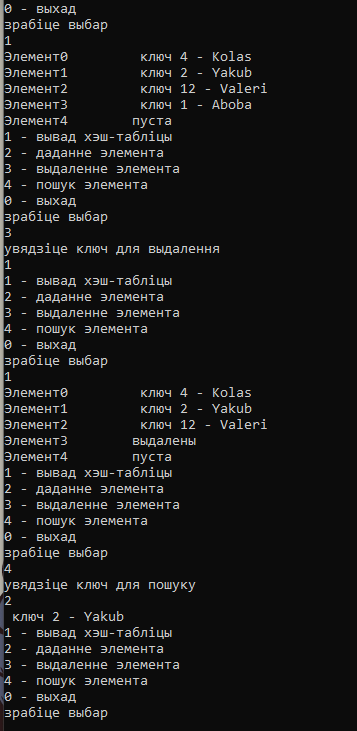
std::cout << "выдалены" << std::endl; // Выснова паведамлення аб выдаленым вочку.

else

f((this->data)[i]); // Ужыванне функцыі да элемента.

}

}



Дап. Варыянты



Зменены толькі 3 функцыі:  
// Функцыя вылічэння хэша з выкарыстаннем мультыплікатыўнай хэш-функцыі.

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

const double A = 0.6180339887499; // Канстанта A для мультыплікатыўнай хэш-функцыі

double hashValue = key \* A; // Вылічэнне прамежкавага значэння хеша

hashValue -= int(hashValue); // Атрыманне дробавай часткі значэння хеша

return (p + int(size \* hashValue) + i) % size; // Выкарыстанне лінейнай функцыі для ўліку калізіі

}

// Функцыя для ўстаўкі элемента ў хэш-табліцу.

bool Object::insert(void\* d)

{

bool b = false; // Сцяг для абазначэння паспяховай устаўкі.

if (N != size) // Праверка на перапаўненне табліцы.

{

for (int i = 0, t = getKey(d), j = HashFunction(t, size, 0, i); i != size && !b; j = Next\_hash(j, size, ++i)) { // Цыкл пошуку месца для ўстаўкі.

if (data[j] == NULL || data[j] == DEL) { // Праверка на пустату або аддаленасць вочка.

data[j] = d; // Устаўка элемента.

N++; // Павелічэнне колькасці элементаў у табліцы.

b = true; // Пазнака паспяховай устаўкі.

}

}

}

return b; // Вяртанне выніку ўстаўкі.

}

// Функцыя для пошуку індэкса элемента ў хэш-табліцы.

int Object::searchInd(int key)

{

int t = -1; // Ініцыялізацыя індэкса элемента.

bool b = false; // Сцяг для абазначэння паспяховага пошуку.

if (N != 0) // Праверка наяўнасці элементаў у табліцы.

{

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

if (data[j] != DEL) {// Праверка на аддаленасць ячэйкі.

if (getKey(data[j]) == key) { // Праверка на адпаведнасць ключа элемента.

t = j; // Захаванне індэкса элемента.

b = true; // Пазнака паспяховага пошуку.

}

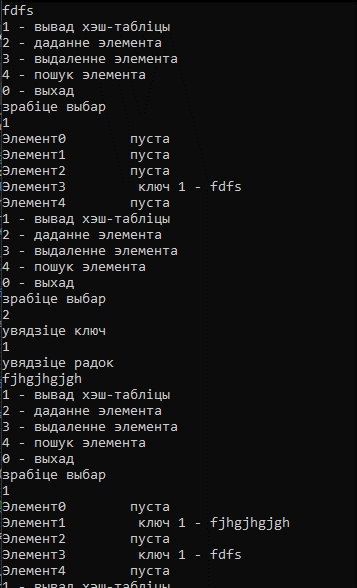
}

}

}

return t; // Вяртанне азначніка элемента (ці -1, калі не знойдзены).

}





Функцыі HashFunction, Next\_hash, create, searchInd, search, deleteByKey, deleteByValue, scan засталісь бяз змен

#include <iostream>

#include <Windows.h>

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) {

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

}

struct Object {

void\*\* data;

int size;

int N;

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

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

bool insert(void\*);

int searchInd(int key);

void\* search(int key);

void\* deleteByKey(int key);

bool deleteByValue(void\*);

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

};

// Функцыя вылічэння хэша з выкарыстаннем мультыплікатыўнай хэш-функцыі.

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

const double A = 0.6180339887499; // Канстанта A для мультыплікатыўнай хэш-функцыі

double hashValue = key \* A; // Вылічэнне прамежкавага значэння хеша

hashValue -= int(hashValue); // Атрыманне дробавай часткі значэння хеша

return (p + int(size \* hashValue) + i) % size; // Выкарыстанне лінейнай функцыі для ўліку калізіі

}

static void\* DEL = (void\*)-1;

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

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; // Ініцыялізацыя сцяга паспяховасці ўстаўкі

// Праверка, ці запоўнена табліца на 80% ці больш

if ((double)N / size >= 0.8) {

// Калі табліца запоўнена на 80% ці больш, вырабляем яе пашырэнне

// Вылічэнне новага памеру табліцы (падваенне бягучага памеру)

int newSize = size \* 2;

// Вылучэнне памяці пад новы масіў паказальнікаў на элементы для новай табліцы

void\*\* newData = new void\* [newSize];

// Ініцыялізацыя кожнага элемента новай табліцы значэннем NULL

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

newData[i] = NULL;

// Капіраванне элементаў з бягучай табліцы ў новую

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

if (data[i] != NULL && data[i] != DEL) {

// Атрыманне ключа бягучага элемента

int k = key(data[i]);

// Вылічэнне індэкса новага элемента ў новай табліцы

int index = k % newSize;

// Пошук пустога слота ў новай табліцы для ўстаўкі элемента

while (newData[index] != NULL)

index = (index + 1) % newSize;

// Устаўка элемента ў новую табліцу

newData[index] = data[i];

}

}

// Вызваленне памяці, занятай старой табліцай

delete[] data;

// Абнаўленне паказальніка на дадзеныя для захоўвання спасылкі на новую табліцу

data = newData;

// Абнаўленне памеру табліцы

size = newSize;

}

// Спроба ўстаўкі новага элемента ў бягучую табліцу

// Цыкл прабягае па слотах табліцы, пачынаючы з індэкса, вылічанага па хэш-функцыі

for (int i = 0, t = getKey(d), j = t % size; i!= size && !b; j = (j + 1) % size) {

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

// Калі бягучы слот пусты або выдалены, устаўляемы элемент

data[j] = d;

// Павялічваем лічыльнік колькасці элементаў у табліцы

N++;

// Усталёўваны сцяг паспяховай устаўкі

b = true;

}

}

// Вяртаем вынік спробы ўстаўкі (паспяхова ці не)

return b;

}

// Функцыя для вылічэння наступнага значэння хеша.

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

{

return (hash + 5 \* p + 3 \* p \* p) % size; // Вылічэнне наступнага значэння хеша з улікам параметру p.

}

// Функцыя для стварэння аб'екта хэш-табліцы.

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

{

return \*(new Object(size, getkey)); // Стварэнне і вяртанне новага аб'екта хэш-табліцы.

}

// Функцыя для пошуку індэкса элемента ў хэш-табліцы.

int Object::searchInd(int key)

{

int t = -1; // Ініцыялізацыя індэкса элемента.

bool b = false; // Сцяг для абазначэння паспяховага пошуку.

if (N != 0) // Праверка наяўнасці элементаў у табліцы.

{

for (int i = 0, j = HashFunction(key, size, 0, i); data[j] != NULL && i != size && !b; j = HashFunction(key, size, ++i, i)) // Цыкл пошуку элемента.

{

if (data[j] != DEL) // Праверка на аддаленасць ячэйкі.

{

if (getKey(data[j]) == key) // Праверка на адпаведнасць ключа элемента.

{

t = j; // Захаванне індэкса элемента.

b = true; // Пазнака паспяховага пошуку.

}

}

}

}

return t; // Вяртанне азначніка элемента (ці -1, калі не знойдзены).

}

// Функцыя для пошуку элемента ў хэш-табліцы па ключы.

void\* Object::search(int key)

{

int t = searchInd(key); // Пошук індэкса элемента.

return (t >= 0) ? (data[t]) : (NULL); // Вяртанне знойдзенага элемента (або NULL, калі не знойдзены).

}

// Функцыя для выдалення элемента з хэш-табліцы па ключы.

void\* Object::deleteByKey(int key)

{

int i = searchInd(key); // Пошук індэкса элемента.

void\* t = data[i]; // Захаванне выдалянага элемента.

if (t != NULL) // Праверка на наяўнасць элемента для выдалення.

{

data[i] = DEL; // Пазнака вочка як выдаленай.

N--; // Памяншэнне колькасці элементаў у табліцы.

}

return t; // Вяртанне выдаленага элемента (або NULL, калі не знойдзены).

}

// Функцыя для выдалення элемента з хэш-табліцы па значэнні.

bool Object::deleteByValue(void\* d)

{

return(deleteByKey(getKey(d)) != NULL); // Выдаленне элемента па ключы, атрыманым з значэння.

}

// Функцыя для сканавання змесціва хэш-табліцы і прымянення да кожнага элемента названай функцыі.

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

{

for (int i = 0; i < this->size; i++) // Цыкл сканавання ўсіх вочак табліцы.

{

std::cout << "Элемент" << i << '\t'; // Выснова нумара ячэйкі.

if ((this->data)[i] == NULL) // Праверка на пустэчу вочка.

std::cout << "пуста" << std::endl; // Выснова паведамлення аб пустым вочку.

else

if ((this->data)[i] == DEL) // Праверка на аддаленасць ячэйкі.

std::cout << "выдалены" << std::endl; // Выснова паведамлення аб выдаленым вочку.

else

f((this->data)[i]); // Ужыванне функцыі да элемента.

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int siz = 10, choice, k;

std::cout << "Увядзіце памер хэш-табліцы" << std::endl;

std::cin >> siz;

Object H = create(siz, key);

for (;;) {

std::cout << "1 - вывад хэш-табліцы" << std::endl;

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

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

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

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

std::cout << "зрабіце выбар" << std::endl;

std::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];

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

std::cin >> k;

a->key = k;

std::cout << "увядзіце радок" << std::endl;

std::cin >> str;

a->mas = str;

if (H.N == H.size)

std::cout << "Табліца запоўнена" << std::endl;

else

H.insert(a);

}

break;

case 3: {

std::cout << "увядзіце ключ для выдалення" << std::endl;

std::cin >> k;

H.deleteByKey(k);

}

break;

case 4: {

std::cout << "увядзіце ключ для пошуку" << std::endl;

std::cin >> k;

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

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

else

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

}

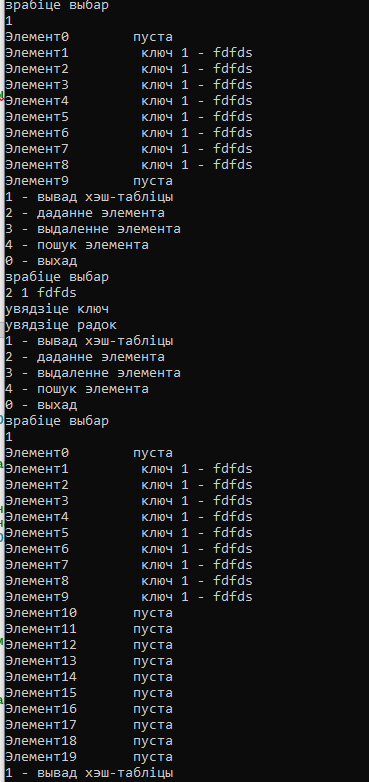
break;

}

}

return 0;

}





Адрозненне ад прошлага задання толькі у адным радку кода у функцыі insert, замест if ((double)N / size >= 0.8) { будзе if ((double)N / size >= 0.5) {

