|  |  |
| --- | --- |
| **Задание** | **Краткие теоретические сведения** |
| 1. Пункты 1, 2, 3 содержат программный код *проекта*, в котором реализована хеш-таблица с *открытой адресацией.*  В правой части данного пункта записан *заголовочный* файл **Hash.h**.  Написать комментарии к программному коду. | **#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** | |
| 2. В правой части представлена структура, *главная функция* проекта и две вспомогательные функции.  Написать комментарии к программе. | **#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;**  **}** | |
| 3. В правой части записан программный модуль **Hash.cpp**.  Написать комментарии к программному коду.  Сформировать программный код в один проект и выполнить его. | #include "Hash.h"  #include <iostream>  // Хеш-функция с использованием метода умножения  int HashFunction(int key, int size, int p)  {  // Вычисление дробной части ключа \* (sqrt(5)-1)/2  double key2 = 5 \* ((0.6180339887499 \* key) - int((0.6180339887499 \* key)));  return (p + key) % size; // Комбинированный хеш  }  //-------------------------------  // Функция разрешения коллизий (квадратичное пробирование)  int Next\_hash(int hash, int size, int p)  {  return (hash + 5 \* p + 3 \* p \* p) % size; // Квадратичная последовательность  }  //-------------------------------  // Фабричный метод создания таблицы  Object create(int size, int(\*getkey)(void\*))  {  return \*(new Object(size, getkey));  }  //-------------------------------  // Конструктор объекта таблицы  Object::Object(int size, int(\*getkey)(void\*))  {  N = 0; // Инициализация счетчика элементов  this->size = size; // Установка размера таблицы  this->getKey = getkey;// Сохранение функции получения ключа  this->data = new void\* [size]; // Выделение памяти  for (int i = 0; i < size; ++i)  data[i] = NULL; // Инициализация ячеек  }  //-------------------------------  // Метод вставки элемента  bool Object::insert(void\* d)  {  bool b = false;  if (N != size) // Проверка на переполнение  // Цикл пробирования с использованием хеш-функций  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) // Поиск свободной ячейки  {  data[j] = d; // Вставка элемента  N++; // Увеличение счетчика  b = true; // Флаг успешной вставки  }  return b;  }  //-------------------------------  // Поиск индекса элемента по ключу  int Object::searchInd(int key)  {  int t = -1; // Индекс не найден  bool b = false;  if (N != 0) // Проверка на пустую таблицу  // Цикл пробирования с двойным хешированием  for (int i = 0, j = HashFunction(key, size, 0);  data[j] != NULL && i != size && !b;  j = HashFunction(key, size, ++i))  if (data[j] != DEL) // Пропуск удаленных элементов  if (getKey(data[j]) == key) // Проверка совпадения ключа  {  t = j; b = true; // Найденный индекс  }  return t;  }  //-------------------------------  // Поиск элемента по ключу и возвращение его данных  void\* Object::search(int key)  {  int t = searchInd(key); // Получаем индекс через searchInd  return(t >= 0) ? (data[t]) : (NULL); // Возвращаем данные или NULL  }  //-------------------------------  // Удаление элемента по ключу (логическое удаление)  void\* Object::deleteByKey(int key)  {  int i = searchInd(key); // Поиск индекса элемента  void\* t = data[i]; // Сохранение указателя на данные  if (t != NULL)  {  data[i] = DEL; // Помечаем ячейку как удаленную  N--; // Уменьшаем счетчик элементов  }  return t; // Возвращаем удаленные данные (или NULL)  }  //-------------------------------  // Удаление элемента по значению  bool Object::deleteByValue(void\* d)  {  // Используем deleteByKey, преобразуя значение в ключ  return(deleteByKey(getKey(d)) != NULL);  }  //-------------------------------  // Вывод содержимого таблицы с пользовательским форматированием  void Object::scan(void(\*f)(void\*))  {  for (int i = 0; i < this->size; i++)  {  std::cout << " Элемент " << i;  // Обработка трех состояний ячейки:  if ((this->data)[i] == NULL)  std::cout << " пусто" << std::endl; // Пустая ячейка  else if ((this->data)[i] == DEL)  std::cout << " удален" << std::endl; // Удаленный элемент  else  f((this->data)[i]); // Вызов пользовательской функции печати  }  } |

4. 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.

Исследовать время поиска в хеш-таблицах.

|  |
| --- |
| Вариант 10  Изменить функцию вычисления хеш для решения коллизии на линейную функцию, которая строится на основе формулы: h(key, i) = (h'(key) + i) mod hashTableSize. |
| int Next\_hash(int hash, int size, int p)  {  return (hash + p) % size; // Линейная последовательность  }  #include "Hash.h"  #include <iostream>  #include <chrono>  #include <cstdlib>  #include <cstdio>  #define \_CRT\_SECURE\_NO\_WARNINGS  using namespace std;  struct AAA {  int key;  char\* mas;  AAA(int k, char\* z) : key(k), mas(z) {}  AAA() : key(0), mas(nullptr) {}  };  int key(void\* d) {  return ((AAA\*)d)->key;  }  int main() {  setlocale(LC\_ALL, "rus");  const int tableSizes[] = { 16, 32, 64, 128 };  const int numSizes = sizeof(tableSizes) / sizeof(tableSizes[0]);  const int numElements = 1000;  const int collisionsFactor = 5;  for (int s = 0; s < numSizes; s++) {  int size = tableSizes[s];  Object H = create(size, key);  int\* keys = new int[numElements];  int keysCount = 0;  srand(time(0));  for (int i = 0; i < numElements; ++i) {  AAA\* a = new AAA;  a->key = rand() % (size / collisionsFactor);  a->mas = new char[20];  snprintf(a->mas, 20, "Element%d", i);  if (H.insert(a)) {  if (keysCount < numElements) {  keys[keysCount++] = a->key;  }  }  else {  delete[] a->mas;  delete a;  }  }  auto start = chrono::high\_resolution\_clock::now();  for (int i = 0; i < keysCount; ++i) {  H.search(keys[i]);  }  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> duration = end - start;  cout << "Размер таблицы: " << size  << " | Элементов: " << H.N  << " | Время поиска: " <<fixed<< double(duration.count()) << " сек\n";  }  return 0;  } |
| Вариант 1  Изменить функцию вычисления хеш для решения коллизии на квадратичную функцию, которая строится на основе формулы: **h(key, i) = (h**'**(key) + с1\*i+ c2\*i2) mod hashTableSize**. |
| int Next\_hash(int hash, int size, int p)  {  return (hash + 2\*p + 11\*p\*p) % size;  } |
| Вариант 2  Изменить функцию вычисления хеш для решения коллизии на квадратичную функцию, которая строится на основе формулы: **h(key, i) = (h**'**(key) + с1\*i+ c2\*i2) mod hashTableSize**. |
|  |