Лабораторная работа 15

Хеш-таблицы с цепочкой

|  |  |
| --- | --- |
|  | Hash\_Chain.h  #pragma once //Заголовочный файл Hash\_Chain.h  #include "Lists.h"  namespace hashTC  {  struct Object  {  int size;  int(\*FunKey)(void\*);  listx::Object\* Hash;  Object(int siz, int(\*f)(void\*))  {  size = siz;  FunKey = f;  Hash = new listx::Object[size];  };  int hashFunction(void\* data);  bool insert(void\* data);  listx::Element\* search(void\* data);  bool deleteByData(void\* data);  void Scan();  };  Object create(int size, int(\*f)(void\*));  }  Lists.h  #pragma once //Заголовочный файл Lists.h  #define LISTNIL (Element\*)-1  namespace listx  {  struct Element  {  Element\* prev;  Element\* next;  void\* data;  Element(Element\* pr, void\* dat, Element\* nt)  {  prev = pr;  data = dat;  next = nt;  }  Element\* getNext()  {  return next;  };  Element\* getPrev()  {  return prev;  };  };  static Element\* NIL = nullptr;  struct Object  {  Element\* head;  Object()  {  head = NIL;  };  Element\* getFirst()  {  return head;  };  Element\* getLast();  Element\* search(void\* data);  bool insert(void\* data);  bool deleteByElement(Element\* e);  bool deleteByData(void\* data);  void scan();  };  Object create();  }  #undef LISTNIL  Hash\_table.cpp  #include "Hash\_Chain.h"  #include <iostream>  namespace hashTC  {  Object create(int size, int(\*f)(void\*))  {  return \*(new Object(size, f));  }  int Object::hashFunction(void\* data)  {  return (FunKey(data) % size);  };  bool Object::insert(void\* data)  {  return (Hash[hashFunction(data)].insert(data));  };  bool Object::deleteByData(void\* data)  {  return (Hash[hashFunction(data)].deleteByData(data));  };  listx::Element\* Object::search(void\* data)  {  return Hash[hashFunction(data)].search(data);  };  void Object::Scan()  {  for (int i = 0; i < size; i++)  {  Hash[i].scan();  std::cout << '\n';  }  };  }  Lists.cpp  #include "Lists.h"  #include <iostream>  struct AAA //элемент таблицы  {  int key;  char\* mas;  };  namespace listx  {  bool Object::insert(void\* data)  {  bool rc = NULL;  if (head == NULL)  head = new Element(NULL, data, head);  else  head = (head->prev = new Element(NULL, data, head));  return rc;  }  //-------------------------------  Element\* Object::search(void\* data)  {  Element\* rc = head;  while ((rc != NULL) && ((((AAA\*)rc->data)->key) != ((AAA\*)data)->key))  rc = rc->next;  return rc;  }  //-------------------------------  bool Object::deleteByElement(Element\* e)  {  bool rc = NULL;  if (rc = (e != NULL))  {  if (e->next != NULL)  e->next->prev = e->prev;  if (e->prev != NULL)  e->prev->next = e->next;  else  head = e->next;  delete e;  }  std::cout << "Элемент удален" << std::endl;  return rc;  }  //-------------------------------  bool Object::deleteByData(void\* data)  {  return deleteByElement(search(data));  }  //-------------------------------  Element\* Object::getLast()  {  listx::Element\* e = this->getFirst(), \* rc = this->getFirst();  while (e != NULL)  {  rc = e;  e = e->getNext();  };  return rc;  }  Object create()  {  return \*(new Object());  };  //-------------------------------  void Object::scan()  {  listx::Element\* e = this->getFirst();  while (e != NULL)  {  std::cout << ((AAA\*)e->data)->key << '-' << ((AAA\*)e->data)->mas << " / ";  e = e->getNext();  };  }  }  Main.cpp  #include "Hash\_Chain.h"  #include <iostream>  using namespace std;  struct AAA  {  int key;  char\* mas;  AAA(int k, char\* z)  {  key = k;  mas = z;  }  AAA()  {  key = 0;  mas = nullptr;  }  };  int hf(void\* d)  {  AAA\* f = (AAA\*)d;  return f->key;  }  //-------------------------------  void AAA\_print(listx::Element\* e)  {  std::cout << ((AAA\*)e->data)->key << '-' << ((AAA\*)e->data)->mas << " / ";  }  //-------------------------------  int main()  {  setlocale(LC\_ALL, "rus");  int current\_size = 7;  cout << "Введите размер хеш-таблицы" << endl;  cin >> current\_size;  hashTC::Object H = hashTC::create(current\_size, hf);  int choice;  int k;  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 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;  H.insert(a);  }  break;  case 1: H.Scan();  break;  case 3: {  AAA\* b = new AAA;  cout << "введите ключ" << endl;  cin >> k;  b->key = k;  H.deleteByData(b);  }  break;  case 4: AAA \* c = new AAA;  cout << "введите ключ" << endl;  cin >> k;  c->key = k;  if (H.search(c) == NULL)  cout << "Элемент не найден" << endl;  else  {  cout << "Первый элемент с данным ключом" << endl;  AAA\_print(H.search(c));  cout << endl;  }  break;  }  }  return 0;  } |
| В соответствии со своим вариантом разработать проект для условия из таблицы, представленной ниже. Построить ***хеш-таблицы с цепочками*** разного размера с коллизиями.  При вычислении хеш-функции для вариантов с 9 по 16 использовать алгоритм на основе исключающего ИЛИ для поля строки данных. | |
| **Вариант 10 Клиенты банка**. Реализовать хеш-таблицу со следующими полями: номер счета, ФИО. Ключ – номер счета. | Lists,h  #pragma once // Заголовочный файл для реализации связного списка  #define LISTNIL (Element\*)-1  namespace listx  {  // Элемент связного списка  struct Element  {  Element\* prev; // Указатель на предыдущий элемент  Element\* next; // Указатель на следующий элемент  void\* data; // Указатель на данные  // Конструктор элемента списка  Element(Element\* pr, void\* dat, Element\* nt)  {  prev = pr;  data = dat;  next = nt;  }  // Получение следующего элемента  Element\* getNext() { return next; };  // Получение предыдущего элемента  Element\* getPrev() { return prev; };  };  static Element\* NIL = nullptr; // Специальное значение "NIL" (пустой указатель)  // Структура, представляющая связный список  struct Object  {  Element\* head; // Указатель на голову списка  // Конструктор списка  Object() { head = NIL; };  // Получение первого элемента  Element\* getFirst() { return head; };  // Получение последнего элемента  Element\* getLast();  // Поиск элемента по данным  Element\* search(void\* data);  // Вставка элемента в список  bool insert(void\* data);  // Удаление элемента по указателю  bool deleteByElement(Element\* e);  // Удаление элемента по данным  bool deleteByData(void\* data);  // Вывод содержимого списка  void scan();  };  // Функция создания списка  Object create();  }  Hach\_chain  #pragma once // Заголовочный файл для хеш-таблицы с цепочками (метод разрешения коллизий)  #include "Lists.h"  namespace hashTC  {  // Структура, представляющая хеш-таблицу  struct Object  {  int size; // Размер хеш-таблицы  int(\*FunKey)(void\*); // Указатель на хеш-функцию  listx::Object\* Hash; // Массив связных списков для хранения данных  // Конструктор хеш-таблицы  Object(int siz, int(\*f)(void\*))  {  size = siz;  FunKey = f;  Hash = new listx::Object[size]; // Инициализация массива списков  };  int hashFunction(void\* data); // Вычисление хеша для данных  bool insert(void\* data); // Вставка элемента  listx::Element\* search(void\* data); // Поиск элемента  bool deleteByData(void\* data); // Удаление элемента по данным  void Scan(); // Вывод содержимого таблицы  };  // Функция создания хеш-таблицы  Object create(int size, int(\*f)(void\*));  }  Hash\_table.cpp  #include "Hach\_chain.h"  #include <iostream>  namespace hashTC  {  // Создание хеш-таблицы  Object create(int size, int(\*f)(void\*))  {  return \*(new Object(size, f));  }  // Вычисление хеш-функции  int Object::hashFunction(void\* data)  {  return (FunKey(data) % size); // Остаток от деления на размер таблицы  };  // Вставка элемента в хеш-таблицу  bool Object::insert(void\* data)  {  return (Hash[hashFunction(data)].insert(data)); // Вставка в соответствующий список  };  // Удаление элемента из хеш-таблицы  bool Object::deleteByData(void\* data)  {  return (Hash[hashFunction(data)].deleteByData(data)); // Удаление из соответствующего списка  };  // Поиск элемента в хеш-таблице  listx::Element\* Object::search(void\* data)  {  return Hash[hashFunction(data)].search(data); // Поиск в соответствующем списке  };  // Вывод содержимого хеш-таблицы  void Object::Scan()  {  for (int i = 0; i < size; i++)  {  Hash[i].scan(); // Вывод каждого списка  std::cout << '\n';  }  };  Lists.cpp  #include "Lists.h"  #include <cstring>  #include <iostream>  // Структура данных, хранимая в таблице (ключ + строка)  struct AAA  {  char\* mas; // номер счета  char\* fio; // ФИО (добавленное поле)  AAA(char\* account, char\* name) : mas(account), fio(name) {}  AAA() : mas(nullptr), fio(nullptr) {}  };  namespace listx  {  // Вставка элемента в список  bool Object::insert(void\* data)  {  bool rc = NULL;  if (head == NULL)  head = new Element(NULL, data, head); // Вставка в пустой список  else  head = (head->prev = new Element(NULL, data, head)); // Вставка в начало  return rc;  }  // Поиск элемента в списке по ключу  Element\* Object::search(void\* data)  {  Element\* rc = head;  AAA\* searchData = (AAA\*)data;  while (rc != nullptr)  {  AAA\* currentData = (AAA\*)rc->data;  if (strcmp(currentData->mas, searchData->mas) == 0)  break;  rc = rc->next;  }  return rc;  }  // Вывод содержимого списка  void Object::scan()  {  Element\* e = head;  while (e != nullptr)  {  AAA\* item = (AAA\*)e->data;  std::cout << item->mas << " - " << item->fio << " / ";  e = e->next;  }  }  // Удаление элемента по указателю  bool Object::deleteByElement(Element\* e)  {  if (e == nullptr) return false;  if (e->next) e->next->prev = e->prev;  if (e->prev) e->prev->next = e->next;  else head = e->next;  AAA\* data = (AAA\*)(e->data);  delete[] data->mas; // Освобождение номера счета  delete[] data->fio; // Освобождение ФИО (добавлено)  delete data; // Освобождение структуры  delete e; // Освобождение элемента  std::cout << "Элемент удален" << std::endl;  return true;  }  // Удаление элемента по данным  bool Object::deleteByData(void\* data)  {  return deleteByElement(search(data));  }  // Получение последнего элемента списка  Element\* Object::getLast()  {  listx::Element\* e = this->getFirst(), \* rc = this->getFirst();  while (e != NULL)  {  rc = e;  e = e->getNext();  };  return rc;  }  // Создание нового списка  Object create()  {  return \*(new Object());  };  }  Main.cpp  #include "Hach\_chain.h"  #include <iostream>  using namespace std;  // Структура данных для хранения в хеш-таблице  struct AAA  {  char\* mas; // номер счета  char\* fio; // ФИО  AAA(char\* account = nullptr, char\* name = nullptr) : mas(account), fio(name) {}  };  // Хеш-функция (возвращает ключ)  int hf(void\* d)  {  AAA\* f = (AAA\*)d;  char\* s = f->mas;  unsigned int hash = 0;  // Вычисление XOR-хеша  for (int i = 0; s[i] != '\0'; i++)  {  hash = (hash << 4) ^ (hash >> 28) ^ s[i]; // Комбинация сдвигов и XOR  }  return (int)(hash & 0x7FFFFFFF); // Гарантия положительного числа  }  // Вывод элемента списка  void AAA\_print(listx::Element\* e)  {  std::cout << hf(((AAA\*)e->data)) << '-' << ((AAA\*)e->data)->mas << " / ";  }  int main()  {  setlocale(LC\_ALL, "rus");  int current\_size;  cout << "Введите размер хеш-таблицы" << endl;  cin >> current\_size;  hashTC::Object H = hashTC::create(current\_size, hf);  int choice;  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(); break;  case 2: { // Добавлен ввод ФИО  char\* account = new char[20];  char\* fio = new char[100];  cout << "Введите номер счета: ";  cin >> account;  cout << "Введите ФИО: ";  cin.ignore(); // Очистка буфера  cin.getline(fio, 100); // Чтение с пробелами  H.insert(new AAA(account, fio));  break;  }  case 3: { // Удаление по номеру счета  char\* str = new char[20];  cout << "Введите номер счета: ";  cin >> str;  AAA\* temp = new AAA(str);  H.deleteByData(temp);  delete temp;  delete[] str;  break;  }  case 4: { // Поиск по номеру счета  char\* str = new char[20];  cout << "Введите номер счета: ";  cin >> str;  AAA\* temp = new AAA(str);  listx::Element\* result = H.search(temp);  if (result) {  AAA\* found = (AAA\*)result->data;  cout << "Найден: " << found->mas << " - " << found->fio << endl;  }  else {  cout << "Не найден" << endl;  }  delete temp;  delete[] str;  break;  }  }  }  return 0;  } |
| 1. Составить хеш-таблицу, содержащую буквы и количество их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в хеш-таблице. | #include <iostream>  #include <unordered\_map>  #include <string>  #include <Windows.h>  using namespace std;  void createHashTable(unordered\_map<char, int>& hashTable, const string& input) {  for (char ch : input) {  if (isalpha(ch)) {  hashTable[ch]++;  }  }  }  void printHashTable(const unordered\_map<char, int>& hashTable) {  for (const auto& entry : hashTable) {  cout << entry.first << ": " << entry.second << endl;  }  }  void searchLetter(const unordered\_map<char, int>& hashTable, char letter) {  auto it = hashTable.find(letter);  if (it != hashTable.end()) {  cout << "Буква '" << letter << "' найдена с частотой: " << it->second << endl;  }  else {  cout << "Буква '" << letter << "' не найдена в хеш-таблице." << endl;  }  }  int main() {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  string input;  cout << "Введите строку: ";  getline(cin, input);  unordered\_map<char, int> hashTable;  createHashTable(hashTable, input);  cout << "Построенная хеш-таблица:" << endl;  printHashTable(hashTable);  char letter;  cout << "Введите букву для поиска: ";  cin >> letter;  searchLetter(hashTable, letter);  } |
| 2. Построить хеш-таблицу из слов произвольного текстового файла, задав ее размерность с экрана. Вывести построенную таблицу слов на экран. Осуществить поиск введенного слова. Выполнить программу для различных размерностей таблицы и сравните количество сравнений. Удалить все слова, начинающиеся на указанную букву, выведите таблицу. | #include <iostream>  #include <fstream>  #include <unordered\_map>  #include <string>  using namespace std;  void createHashTable(unordered\_map<string, int>& hashTable, const string& filename) {  ifstream file(filename);  if (!file.is\_open()) {  cerr << "Не удалось открыть файл." << endl;  return;  }  string word;  while (file >> word) {  hashTable[word]++;  }  file.close();  }  void printHashTable(const unordered\_map<string, int>& hashTable) {  for (const auto& entry : hashTable) {  cout << entry.first << ": " << entry.second << endl;  }  }  void searchWord(const unordered\_map<string, int>& hashTable, const string& word) {  auto it = hashTable.find(word);  if (it != hashTable.end()) {  cout << "Слово '" << word << "' найдено с частотой: " << it->second << endl;  }  else {  cout << "Слово '" << word << "' не найдено в хеш-таблице." << endl;  }  }  void deleteWordsStartingWith(unordered\_map<string, int>& hashTable, char letter) {  for (auto it = hashTable.begin(); it != hashTable.end(); ) {  if (it->first[0] == letter) {  it = hashTable.erase(it);  }  else {  ++it;  }  }  }  int main() {  int tableSize;  cout << "Введите предполагаемое количество различных слов в файле (размерность хеш-таблицы): ";  cin >> tableSize;  unordered\_map<string, int> hashTable;  hashTable.reserve(tableSize);  string filename;  cout << "Введите имя файла: ";  cin >> filename;  createHashTable(hashTable, filename);  cout << "Построенная хеш-таблица:" << endl;  printHashTable(hashTable);  string word;  cout << "Введите слово для поиска: ";  cin >> word;  searchWord(hashTable, word);  char letter;  cout << "Введите букву для удаления всех слов, начинающихся на нее: ";  cin >> letter;  deleteWordsStartingWith(hashTable, letter);  cout << "Хеш-таблица после удаления слов, начинающихся на букву '" << letter << "':" << endl;  printHashTable(hashTable);  return 0;  } |