Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 15

По дисциплине «Основы алгоритмизации и программирования»

На тему «Хеш-таблицы c цепочками»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: асс. Андронова М.В.

2023, Минск

5. В соответствии со своим вариантом разработать проект для условия из таблицы, представленной ниже. Построить хеш-таблицы разного размера с коллизиями.

Для вариантов с 1 по 8 вычисление хеш-функции произвести по методу универсального хеширования. Для вариантов с 9 по 16 при вычислении хеш-функции использовать алгоритм на основе исключающего ИЛИ для поля строки данных.

Исследовать время поиска информации.

**Вариант 2.**

**Список электронных адресов.** Создать хеш-таблицу со следующими полями: адрес, фамилия абонента, год создания. Ключ – год создания.

|  |
| --- |
| Главный файл |
| // Список электронных адресов. Создать хеш-таблицу со следующими полями: адрес, фамилия абонента, год создания. Ключ – год создания.  #include "Hash\_Twin\_Chain.h"  #include <iostream>  #include <windows.h>  #include <tchar.h>  #include <time.h>  using namespace std;  const char\* NML[15] = { "Яглинская", "Пивоваров", "Лупинка", "Петрович", "Полторан",  "Михедов", "Дубина", "Кашков", "Щипер", "Зданевич",  "Никитенко", "Жигадло", "Курило", "Нестерова", "Дайтбегова" };  const char\* ADR[15] = { "Пушкина 32", "Строителей 3", "Ленина 52", "Революционная 161", "Октябрьская 11",  "Фрунзе 21", "Пушкина 55", "Студенческая 50", "Кирова 10", "Хоружей 55",  "Советская 1", "Интернациональная 67", "Пролетарская 11", "Мира 8", "Юности 5"  };  struct AAA  {  int key;  const char\* mas;  const char\* adr;  AAA(int k, const char\* z, const char\* l)  {  key = k;  mas = z;  adr = l;  }  AAA()  {  key = 0;  mas = NULL;  adr = NULL;  }  };  //-------------------------------  int hf(void\* d) //Функция создания указателя как информационного элемента списка  {  AAA\* f = (AAA\*)d;  return f->key;  }  //-------------------------------  void AAA\_print(listx::Element\* e)  {  std::cout << ((AAA\*)e->data)->key << '\t' << ((AAA\*)e->data)->mas << "\t" << ((AAA\*)e->data)->adr << " / ";  }  //-------------------------------  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus"); //Установление кодировки  SetConsoleOutputCP(1251);  SetConsoleCP(1251);  srand(time(0));  clock\_t start, end; // для времени  double searching\_time;  int current\_size = 0;  //Переменная, определяющая размер таблицы  cout << "Введите размер хеш-таблицы: "; //Сообщение о вводе размера таблицы  cin >> current\_size; //Ввод пользователем размера таблицы  hashTC::Object H = hashTC::create(current\_size, hf);  int choice;  int k;  for (;;) //Меню  {  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента" << endl;  cout << "3 - удаление элемента" << endl;  cout << "4 - поиск элемента" << endl;  cout << "5 - заполнение хэш-таблицы" << endl;  cout << "0 - выход" << endl;  cout << "сделайте выбор" << endl;  cin >> choice;  switch (choice)  {  case 0: exit(0);  case 1: H.Scan();  break;  case 2: { AAA\* a = new AAA; //Создание нового элемента списка. Список - элемент таблицы  char\* str = new char[100]; //Создание новой строки как данных, хранящихся в поле элемента списка  char\* str2 = new char[100];  cout << "Введите год(начало c 0): ";  cin >> k;  k = k;//Ввод ключа, хранящегося в поле элемента списка  a->key = k; //Присваивание полю "ключ" списка вводимый ключ  cout << "Введите фамилию: ";  cin.ignore();  cin.getline(str, 100);  a->mas = str; //Присваивание полю "строка" списка вводимую строку  cout << "Введите адрес: ";  cin.ignore();  cin.getline(str2, 100);  a->adr = str2;  H.insert(a); //Вставка элемента в объкт-таблицу  }  break;  case 3: { AAA\* b = new AAA;  cout << "Введите год: ";  cin >> k;  b->key = k;  H.deleteByData(b);  }  break;  case 4: {AAA\* c = new AAA;  cout << "Введите год: ";  cin >> k;  c->key = k;  start = clock();  if (H.search(c) == NULL)  cout << "Элемент не найден" << endl;  else  {  cout << "Первый элемент с данным годом: ";  AAA\_print(H.search(c));  cout << endl;  }  end = clock();  searching\_time = (double)(end - start) / CLOCKS\_PER\_SEC;  cout << "время поиска: " << searching\_time << endl;  }  break;  case 5:  {  for (int i = 0; i < H.sizeO; i++)  {  AAA\* a = new AAA;  k = 0 + rand() % H.sizeO;  a->key = k;  a->mas = NML[0 + rand() % 13];  a->adr = ADR[0 + rand() % 13];  H.insert(a);  }  }  break;  }  }  return 0;  } |
| Файл Hash\_Table.cpp |
| #include "List.h"  #include "Hash\_Twin\_Chain.h"  #include <iostream>  struct AAA  {  int key;  const char\* mas;  const char\* adr;  AAA(int k, char\* z, char\* l)  {  key = k;  mas = z;  adr = l;  }  };  namespace hashTC  {  Object create(int size, int(\*f)(void\*)) //Функция создания нового объекта - таблицы  {  return \*(new Object(size, f)); //Новый объект(размер объекта, указатель)  }  int Object::hashFunction(void\* data) //универсальное хеширование  {  int h = 0, a = 314445, b = 37542;  h = (a \* h + FunKey(data)) % sizeO;  return (h < 0) ? (h + sizeO) : h;  };  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 < sizeO; i++)  {  std::cout << i << ". ";  Hash[i].scan();  std::cout << '\n';  }  };  } |
| Файл List.cpp |
| #include "List.h"  #include <iostream>  struct AAA //элемент таблицы  {  int key;  char\* mas;  char\* adr;  };  namespace listx  {  bool Object::insert(void\* data)  {  bool rc = NULL;  if (head == NULL) //Голова списка пустая?  head = new Element(NULL, data, head); //Создание нового списка  else  head = (head->prev = new Element(NULL, data, head)); //Вставка элемента с данными в предыдущую позицию, сдвигая голову вперёд  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(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  std::cout << "Год: " << ((AAA\*)e->data)->key << ";\t" << " фамилия: " << ((AAA\*)e->data)->mas << ";\t" << " адрес: " << ((AAA\*)e->data)->adr << ";\t "; //Поочерёдный вывод элементов списка. Список - это отдельный элемент таблицы  e = e->getNext(); //Получение адреса следующего элемента списка  k = 1;  };  if (!k)  {  std::cout << "Пустая ячейка";  }  }  } |
| Заголовочный файл Hash\_Twin\_Chain.h |
| #pragma once  #include "List.h"  namespace hashTC  {  struct Object //Структура "объект"  {  int sizeO; //Размер объекта  int(\*FunKey)(void\*);  listx::Object\* Hash;  Object(int size, int(\*f)(void\*)) //Метод создания нового объекта (размер, указатель)  {  sizeO = size; //Размер нового объекта  FunKey = f; //Указатель на первый список-элемент таблицы  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\*));  } |
| Заголовочный файл List.h |
| #pragma once  #define LISTNIL (Element\*)-1  namespace listx  {  struct Element  {  Element\* prev; //Указатель на предыдущий элемент списка  Element\* next; //Указатель на следующий элемент списка  void\* data; //Поле, содержащее данные  Element(Element\* prevO, void\* dataO, Element\* nextO) //Конструктор создания списка  {  prev = prevO;  data = dataO;  next = nextO;  }  Element\* getNext() //Метод получения указателя на следующий элемент списка  {  return next;  };  Element\* getPrev() //Метод получения указателя на предыдущий элемент списка  {  return prev;  };  };  static Element\* NIL = 0; //Создание пустого списка как элемент таблицы  struct Object  {  Element\* head;  Object()  {  head = NIL; //Создание указателей на списки-члены таблицы  };  Element\* getFirst() //Метод получения указателя на первый список-элемент таблицы  {  return head;  };  Element\* getLast(); //Метод получения указателя на последний список-элемент таблицы  Element\* search(void\* data); //Метод поиска списка-элемента таблицы  bool insert(void\* data); //Метод вставки нового списка-элемента в таблицу  bool deleteByElement(Element\* e); //Метод удаления списка-элемента таблицы  bool deleteByData(void\* data); //Метод удаления элемента списка-элемента таблицы по значению информации  void scan(); //Метод обхода всех списков-элементов и их самостоятельных элементов  };  Object create();  }  #undef LISTNIL #pragma once |
| Результат в консоли |
|  |

Дополнительные задания.

1. Составить хеш-таблицу, содержащую буквы и количество их вхождений во введенной строке. Вывести таблицу на экран. Осуществить поиск введенной буквы в хеш-таблице.

|  |
| --- |
| Главный файл |
| #include "Hash\_Twin\_Chain.h"  #include <iostream>  #include <windows.h>  #include <tchar.h>  #include <time.h>  #include "List.h"  using namespace std;  struct AAA  {  int key;  const char\* mas;  AAA(int k, const char\* z)  {  key = k;  mas = z;  }  AAA()  {  key = 0;  mas = NULL;  }  };  //-------------------------------  int hf(void\* d) //Функция создания указателя как информационного элемента списка  {  AAA\* f = (AAA\*)d;  return f->key;  }  //-------------------------------  void AAA\_print(listx::Element\* e)  {  std::cout << ((AAA\*)e->data)->key << '-' << ((AAA\*)e->data)->mas << " / ";  }  //-------------------------------  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus"); //Установление кодировки  SetConsoleOutputCP(1251);  SetConsoleCP(1251);  srand(time(0));  int current\_size = 0; //Переменная, определяющая размер таблицы  cout << "Введите размер хеш-таблицы: "; //Сообщение о вводе размера таблицы  cin >> current\_size; //Ввод пользователем размера таблицы  hashTC::Object H = hashTC::create(current\_size, hf);  int choice;  int k = 0;  for (;;) //Меню  {  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; //Создание нового элемента списка. Список - элемент таблицы  char\* str = new char[100]; //Создание новой строки как данных, хранящихся в поле элемента списка //Ввод ключа, хранящегося в поле элемента спи //Присваивание полю "ключ" списка вводимый ключ  cout << "Введите строку из символа: ";  cin.ignore();  cin.getline(str, 100);  char\* str1 = str;  k = 0;  char only[1];  only[0] = str1[0];  do  {  if (\*only == \*str1)  {  k++;  }  } while (strcmp(str1++, ""));  a->key = k;  a->mas = str; //Присваивание полю "строка" списка вводимую строку  H.insert(a); //Вставка элемента в объкт-таблицу  }  break;  case 1: H.Scan();  break;  case 3: { AAA\* b = new AAA;  cout << "Введите ключ:";  cin >> k;  b->key = k;  H.deleteByData(b);  }  break;  case 4:  AAA \* c = new AAA;  char only[1];  cout << "Введите букву: ";  cin.ignore();  cin.getline(only, 2);  c->mas = only;  H.Scan1(c);  cout << '\n';  break;  }  }  return 0;  } |
| Файл Hash\_Table.cpp |
| #include "Hash\_Twin\_Chain.h"  #include "List.h"  #include <iostream>  struct AAA  {  int key;  const char\* mas;  AAA(int k, char\* z)  {  key = k;  mas = z;  }  };  namespace hashTC  {  Object create(int size, int(\*f)(void\*))  {  return \*(new Object(size, f));  }  int Object::hashFunction(void\* data)  {  return (FunKey(data) % sizeO);  };  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 < sizeO; i++)  {  std::cout << i << ". ";  Hash[i].scan();  std::cout << '\n';  }  };  void Object::Scan1(void\* data)  {  std::cout << "Найденные записи: \t";  for (int i = 0; i < sizeO; i++)  {  Hash[i].scan1(data);  }  std::cout << "\n";  };  } |
| Файл List.cpp |
| #include <iostream>  #include "List.h"  struct AAA //элемент таблицы  {  int key;  char\* mas;  };  namespace listx  {  bool Object::insert(void\* data)  {  bool rc = NULL;  if (head == NULL) //Голова списка пустая?  head = new Element(NULL, data, head); //Создани нового списка  else  head = (head->prev = new Element(NULL, data, head)); //Вставка элемента с данными в предыдущую позицию, сдвигая голову вперёд  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 << "Ýëåìåíò óäàëåí" << 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(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  std::cout << "Ключ (=кол-во вхождений): " << ((AAA\*)e->data)->key << "; " << "Строка: " << ((AAA\*)e->data)->mas << "\t"; //Поочерёдный вывод элементов списка. Список - это отдельный элемент таблицы  e = e->getNext(); //Получение адреса следующего элемента списка  k = 1;  };  if (!k)  {  std::cout << "Пустая ячейка";  }  }  void Object::scan1(void\* data)  {  listx::Element\* e = this->getFirst(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  if (((((AAA\*)data)->mas) = ((AAA\*)e->data)->mas))  {  std::cout << "Ключ: " << ((AAA\*)e->data)->key << "; " << "Строка: " << ((AAA\*)e->data)->mas << "\t";  }  e = e->getNext();  };  }  } |
| Заголовочный файл Hash\_Twin\_Chain.h |
| #pragma once  #include "List.h"  namespace hashTC  {  struct Object //Структура "объект"  {  int sizeO; //Размер объекта  int(\*FunKey)(void\*);  listx::Object\* Hash;  Object(int size, int(\*f)(void\*)) //(размер, указатель)  {  sizeO = size; //Размер нового объекта  FunKey = f; //Указатель на первый список-элемент таблицы  Hash = new listx::Object[size];  };  int hashFunction(void\* data);  bool insert(void\* data);  listx::Element\* search(void\* data);  bool deleteByData(void\* data);  void Scan();  void Scan1(void\* data);  };  Object create(int size, int(\*f)(void\*));  } |
| Заголовочный файл List.h |
| #pragma once  #define LISTNIL (Element\*)-1  namespace listx  {  struct Element  {  Element\* prev; //Указатель на предыдущий элемент списка  Element\* next; //Указатель на следующий элемент списка  void\* data; //Поле, содержащее данные  Element(Element\* prevO, void\* dataO, Element\* nextO) //Конструктор создания списка  {  prev = prevO;  data = dataO;  next = nextO;  }  Element\* getNext() //Метод получения указателя на следующий элемент списка  {  return next;  };  Element\* getPrev() //Метод получения указателя на предыдущий элемент списка  {  return prev;  };  };  static Element\* NIL = 0; //Создание пустого списка как элемент таблицы  struct Object  {  Element\* head;  Object()  {  head = NIL; //Создание указателей на списки-члены таблицы  };  Element\* getFirst() //Метод получения указателя на первый список-элемент таблицы  {  return head;  };  Element\* getLast(); //Метод получения указателя на последний список-элемент таблицы  Element\* search(void\* data); //Метод поиска списка-элемента таблицы  bool insert(void\* data); //Метод вставки нового списка-элемента в таблицу  bool deleteByElement(Element\* e); //Метод удаления списка-элемента таблицы  bool deleteByData(void\* data); //Метод удаления элемента списка-элемента таблицы по значению информации  void scan();  void scan1(void\* data); //Метод обхода всех списков-элементов и их самостоятельных элементов  };  Object create();  }  #undef LISTNIL |
| Результат в консоли |
|  |

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.

|  |
| --- |
| Главный файл |
| #include "Header.h"  #include "Lists.h"  #include <iostream>  #include <windows.h>  #include <tchar.h>  #include <time.h>  #include <fstream>  using namespace std;  const char\* NML[15] = { "Яглинская", "Пивоваров", "Лупинка", "Петрович", "Полторан",  "Михедов", "Дубина", "Кашков", "Щипер", "Зданевич",  "Никитенко", "Жигадло", "Курило", "Нестерова", "Дайтбегова" };  struct AAA  {  int key;  char mas[50];  AAA(int k, char\* z)  {  key = k;  strcpy\_s(mas, 50, z);  }  AAA()  {  key = 0;  strcpy\_s(mas, 50, "\0");  }  };  //-------------------------------  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 << " / ";  }  //-------------------------------  void foutF(ofstream& f, char\* name);  int \_tmain(int argc, \_TCHAR\* argv[])  {  ofstream input;  ifstream output;  setlocale(LC\_ALL, "rus"); //Óñòàíîâëåíèå êîäèðîâêè  SetConsoleOutputCP(1251);  SetConsoleCP(1251);  srand(time(0));  int current\_size = 0; //Переменная, определяющая размер таблицы  cout << "Введите размер хеш-таблицы: "; //Сообщение о вводе размера таблицы  cin >> current\_size; //Ввод пользователем размера таблицы  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 << "5 - заполнение хэш-таблицы из файла" << endl;  cout << "6 - заполнение файла словами" << endl;  cout << "0 - выход" << endl;  cout << "Сделайте выбор" << endl;  cin >> choice;  switch (choice)  {  case 0: exit(0);  break;  case 1: H.Scan();  break;  case 2: { AAA\* a = new AAA; //Создание нового элемента списка. Список - элемент таблицы  char\* str = new char[100]; //Создание новой строки как данных, хранящихся в поле элемента списка  cout << "Введите ключи: ";  cin >> k; //Ввод ключа, хранящегося в поле элемента списка  a->key = k; //Присваивание полю "ключ" списка вводимый ключ  cout << "Введите слово: ";  cin.ignore();  cin.getline(str, 100);  strcpy\_s(a->mas, 50, str);//Присваивание полю "строка" списка вводимую строку  H.insert(a); //Вставка элемента в объкт-таблицу  }  case 3: { AAA\* b = new AAA;  char letter[1];  cout << "Введите букву:";  cin.ignore();  cin.getline(letter, 2);  strcpy\_s(b->mas, 2, letter);  H.deleteByData(b);  }  break;  case 4: {AAA\* c = new AAA;  char only[50];  cout << "Введите слово: ";  cin.ignore();  cin.getline(only, 50);  strcpy\_s(c->mas, 50, only);  H.Scan1(c);  }  break;  case 5:  {  char\* str = new char[100];  char words[14][50];  char buf[25];  cout << "Введите имя файла для чтения: ";  cin.ignore();  cin.getline(str, 100);  output.open(str);  if (!output.is\_open())  cout << "Файл не может быть открыт!\n";  else  {  int i = 0;  output.getline(buf, 25);  do  {  strcpy\_s(\*(words + i), 50, buf);  i++;  } while (output.getline(buf, 50));  for (int i = 0; i < H.sizeO - 1; i++)  {  AAA\* a = new AAA;  k = 0 + rand() % H.sizeO;  a->key = k;  strcpy\_s(a->mas, 50, \*(words + (0 + rand() % 14)));  H.insert(a);  }  output.close();  }  }  break;  case 6:  {  char\* str = new char[100];  cout << "Введите имя файла для записи: ";  cin.ignore();  cin.getline(str, 100);  foutF(input, str);  }  break;  }  }  return 0;  }  void foutF(ofstream& f, char\* name)  {  f.open(name);  for (int i = 0; i < 14; i++)  {  f << NML[i] << endl;  }  f.close();  } |
| Файл Hash\_Table.cpp |
| #include "Header.h"  #include <iostream>  #include "Lists.h"  struct AAA  {  int key;  char mas[50];  AAA(int k, char\* z)  {  key = k;  strcpy\_s(mas, 50, z);  }  };  namespace hashTC  {  Object create(int size, int(\*f)(void\*)) //Функция создания нового объекта - таблицы  {  return \*(new Object(size, f)); //Новый объект(размер объекта, указатель)  }  int Object::hashFunction(void\* data)  {  return (FunKey(data) % sizeO);  };  bool Object::insert(void\* data)  {  return (Hash[hashFunction(data)].insert(data));  };  bool Object::deleteByData(void\* data)  {  for (int i = 0; i < sizeO; i++)  {  listx::Element\* e = Hash[i].getFirst(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  if (\*(((AAA\*)data)->mas) == \*(((AAA\*)e->data)->mas))  {  return(Hash[i].deleteByElement(e));  }  e = e->getNext();  };  }  };  listx::Element\* Object::search(void\* data)  {  return Hash[hashFunction(data)].search(data);  };  void Object::Scan()  {  for (int i = 0; i < sizeO; i++)  {  std::cout << i + 1 << ". ";  Hash[i].scan();  std::cout << '\n';  }  };  void Object::Scan1(void\* data)  {  std::cout << "Найденные записи: " << std::endl;  for (int i = 0; i < sizeO; i++)  {  Hash[i].scan1(data);  }  };  } |
| Файл Lists.cpp |
| #include "Header.h"  #include "Lists.h"  #include <iostream>  struct AAA  {  int key;  char mas[50];  };  namespace listx  {  bool Object::insert(void\* data)  {  bool rc = NULL;  if (head == NULL)  //Голова списка пустая?  head = new Element(NULL, data, head); //Создание нового списка  else  head = (head->prev = new Element(NULL, data, head)); //Вставка элемента с данными в предыдущую позицию, сдвигая голову вперёд  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(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  std::cout << "Ключ: " << ((AAA\*)e->data)->key << "; " << "Слово: " << ((AAA\*)e->data)->mas << ";\t"; //Поочерёдный вывод элементов списка. Список - это отдельный элемент таблицы  e = e->getNext(); //Получение адреса следующего элемента списка  k = 1;  };  if (!k)  {  std::cout << "Пустая ячейка";  }  }  void Object::scan1(void\* data)  {  listx::Element\* e = this->getFirst(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  if (strcmp((((AAA\*)data)->mas), ((AAA\*)e->data)->mas) == 0)  {  std::cout << "Ключ: " << ((AAA\*)e->data)->key << "; " << "Слово: " << ((AAA\*)e->data)->mas << ";\t" << std::endl; //Поочерёдный вывод элементов списка. Список - это отдельный элемент таблицы  }  e = e->getNext();  };  }  } |
| Заголовочный файл Header.h |
| #pragma once  #include "Lists.h"  namespace hashTC  {  struct Object //Структура "объект"  {  int sizeO; //Размер объекта  int(\*FunKey)(void\*);  listx::Object\* Hash;  Object(int size, int(\*f)(void\*)) //Метод создания нового объекта (размер, указатель)  {  sizeO = size; //Размер нового объекта  FunKey = f; //Указатель на первый список-элемент таблицы  Hash = new listx::Object[size];  };  int hashFunction(void\* data);  bool insert(void\* data);  listx::Element\* search(void\* data);  bool deleteByData(void\* data);  void Scan();  void Scan1(void\* data);  };  Object create(int size, int(\*f)(void\*));  }  #pragma once |
| Заголовочный файл Lists.h |
| #pragma once  #define LISTNIL (Element\*)-1  namespace listx  {  struct Element  {  Element\* prev; //Указатель на предыдущий элемент списка  Element\* next; //Указатель на следующий элемент списка  void\* data; //Поле, содержащее данные  Element(Element\* prevO, void\* dataO, Element\* nextO) //Конструктор создания списка  {  prev = prevO;  data = dataO;  next = nextO;  }  Element\* getNext() //Метод получения указателя на следующий элемент списка  {  return next;  };  Element\* getPrev() //Метод получения указателя на предыдущий элемент списка  {  return prev;  };  };  static Element\* NIL = 0; //Создание пустого списка как элемент таблицы  struct Object  {  Element\* head;  Object()  {  head = NIL; //Создание указателей на списки-члены таблицы  };  Element\* getFirst() //Метод получения указателя на первый список-элемент таблицы  {  return head;  };  Element\* getLast(); //Метод получения указателя на последний список-элемент таблицы  Element\* search(void\* data); //Метод поиска списка-элемента таблицы  bool insert(void\* data); //Метод вставки нового списка-элемента в таблицу  bool deleteByElement(Element\* e); //Метод удаления списка-элемента таблицы  bool deleteByData(void\* data); //Метод удаления элемента списка-элемента таблицы по значению информации  void scan();  void scan1(void\* data);  };  Object create();  }  #undef LISTNIL  #pragma once |
| Результат в консоли |
|  |
| Файл h.txt |
|  |

4. В текстовом файле содержатся целые числа. Построить хеш-таблицу из чисел файла. Осуществить поиск введенного целого числа в хеш-таблице. Сравнить результаты количества сравнений при различном наборе данных в файле.

|  |
| --- |
| Главный файл |
| #include "Header.h"  #include "Lists.h"  #include <iostream>  #include <windows.h>  #include <tchar.h>  #include <time.h>  #include <fstream>  using namespace std;  struct AAA  {  int key;  int mas;  AAA(int k, int z)  {  key = k;  mas = z;  }  AAA()  {  key = 0;  mas = 0;  }  };  //-------------------------------  int hf(void\* d) //Функция создания указателя как информационного элемента списка  {  AAA\* f = (AAA\*)d;  return f->key;  }  //-------------------------------  void AAA\_print(listx::Element\* e)  {  std::cout << ((AAA\*)e->data)->key << '-' << ((AAA\*)e->data)->mas << " / ";  }  //-------------------------------  void foutF(ofstream& f, char\* name);  int \_tmain(int argc, \_TCHAR\* argv[])  {  ofstream input;  ifstream output;  setlocale(LC\_ALL, "rus"); //Установление кодировки  SetConsoleOutputCP(1251);  SetConsoleCP(1251);  srand(time(0));  int current\_size = 0; //Переменная, определяющая размер таблицы  cout << "Введите размер хеш-таблицы: "; //Сообщение о вводе размера таблицы  cin >> current\_size; //Ввод пользователем размера таблицы  hashTC::Object H = hashTC::create(current\_size, hf);  int choice;  int k;  for (;;) //Меню  {  cout << "1 - вывод хеш-таблицы" << endl;  cout << "2 - добавление элемента" << endl;  cout << "3 - удаление элемента" << endl;  cout << "4 - поиск элемента" << endl;  cout << "5 - заполнение хэш-таблицы из файла" << endl;  cout << "0 - выход" << endl;  cout << "Сделайте выбор" << endl;  cin >> choice;  switch (choice)  {  case 0: exit(0);  case 2: { AAA\* a = new AAA; //Создание нового элемента списка. Список - элемент таблицы  char\* str = new char[100]; //Создание новой строки как данных, хранящихся в поле элемента списка  cout << "Введите ключ: ";  cin >> k; //Ввод ключа, хранящегося в поле элемента списка  a->key = k; //Присваивание полю "ключ" списка вводимый ключ  cout << "Введите число: ";  cin >> k;  a->mas = k;  H.insert(a); //Вставка элемента в объкт-таблицу  }  break;  case 1: H.Scan();  break;  case 3: { AAA\* b = new AAA;  cout << "Введите число: ";  cin >> k;  b->mas = k;  H.deleteByData(b);  }  break;  case 4: {AAA\* c = new AAA;  cout << "Введите число: ";  cin >> k;  c->mas = k;  H.Scan1(c);  }  break;  case 5:  {  char\* str = new char[100];  char words[14][50];  char buf[25];  cout << "Введите имя файла для чтения: ";  cin.ignore();  cin.getline(str, 100);  output.open(str);  if (!output.is\_open())  cout << "Файл не может быть открыт!\n";  else  {  int i = 0;  output.getline(buf, 25);  do  {  strcpy\_s(\*(words + i), 50, buf);  i++;  } while (output.getline(buf, 50));  for (int i = 0; i < H.sizeO - 1; i++)  {  AAA\* a = new AAA;  k = 0 + rand() % H.sizeO;  a->key = k;  a->mas = atoi(\*(words + (0 + rand() % 14)));  H.insert(a);  }  output.close();  }  }  break;  case 6:  {  char\* str = new char[100];  cout << "Введите имя файла для записи: ";  cin.ignore();  cin.getline(str, 100);  foutF(input, str);  }  break;  }  }  return 0;  }  void foutF(ofstream& f, char\* name)  {  f.open(name);  for (int i = 0; i < 14; i++)  {  f << 0 + rand() % 1000 << endl;  }  f.close();  } |
| Файл Hash\_Table.cpp |
| #include "Header.h"  #include <iostream>  #include "Lists.h"  struct AAA  {  int key;  int mas;  AAA(int k, int z)  {  key = k;  mas = z;  }  };  namespace hashTC  {  Object create(int size, int(\*f)(void\*)) //Функция создания нового объекта - таблицы  {  return \*(new Object(size, f)); //Новый объект(размер объекта, указатель)  }  int Object::hashFunction(void\* data)  {  return (FunKey(data) % sizeO);  };  bool Object::insert(void\* data)  {  return (Hash[hashFunction(data)].insert(data));  };  bool Object::deleteByData(void\* data)  {  for (int i = 0; i < sizeO; i++)  {  listx::Element\* e = Hash[i].getFirst(); //е - это указатель на голову отдельного элемента-списка таблицы  bool k = 0;  while (e != NULL)  {  if ((((AAA\*)data)->mas) == (((AAA\*)e->data)->mas))  {  return(Hash[i].deleteByElement(e));  }  e = e->getNext();  };  }  };  listx::Element\* Object::search(void\* data)  {  return Hash[hashFunction(data)].search(data);  };  void Object::Scan()  {  for (int i = 0; i < sizeO; i++)  {  std::cout << i + 1 << ". ";  Hash[i].scan();  std::cout << '\n';  }  };  void Object::Scan1(void\* data)  {  std::cout << "Найденные записи: " << std::endl;  for (int i = 0; i < sizeO; i++)  {  Hash[i].scan1(data);  }  };  } |
| Файл Lists.cpp |
| #include "Lists.h"  #include"Header.h"  #include <iostream>  struct AAA  {  int key;  int 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 << "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();  bool k = 0;  while (e != NULL)  {  std::cout << "ключ: " << ((AAA\*)e->data)->key << "; " << "число: " << ((AAA\*)e->data)->mas << "\t";  e = e->getNext();  k = 1;  };  if (!k)  {  std::cout << "Пустая ячейка";  }  }  void Object::scan1(void\* data)  {  listx::Element\* e = this->getFirst();  bool k = 0;  while (e != NULL)  {  if ((((AAA\*)data)->mas) == ((AAA\*)e->data)->mas)  {  std::cout << "ключ: " << ((AAA\*)e->data)->key << "; " << "число: " << ((AAA\*)e->data)->mas << "\t" << std::endl;  }  e = e->getNext();  };  }  } |
| Заголовочный файл Header.h |
| #pragma once  #include "Lists.h"  namespace hashTC  {  struct Object  {  int sizeO;  int(\*FunKey)(void\*);  listx::Object\* Hash;  Object(int size, int(\*f)(void\*))  {  sizeO = size;  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();  void Scan1(void\* data);  };  Object create(int size, int(\*f)(void\*));  }  #pragma once |
| Заголовочный файл Lists.h |
| #pragma once  #define LISTNIL (Element\*)-1  namespace listx  {  struct Element  {  Element\* prev; //Указатель на предыдущий элемент списка  Element\* next; //Указатель на следующий элемент списка  void\* data; //Поле, содержащее данные  Element(Element\* prevO, void\* dataO, Element\* nextO) //Конструктор создания списка  {  prev = prevO;  data = dataO;  next = nextO;  }  Element\* getNext() //Метод получения указателя на следующий элемент списка  {  return next;  };  Element\* getPrev() //Метод получения указателя на предыдущий элемент списка  {  return prev;  };  };  static Element\* NIL = 0; //Создание пустого списка как элемент таблицы  struct Object  {  Element\* head;  Object()  {  head = NIL; //Создание указателей на списки-члены таблицы  };  Element\* getFirst() //Метод получения указателя на первый список-элемент таблицы  {  return head;  };  Element\* getLast(); //Метод получения указателя на последний список-элемент таблицы  Element\* search(void\* data); //Метод поиска списка-элемента таблицы  bool insert(void\* data); //Метод вставки нового списка-элемента в таблицу  bool deleteByElement(Element\* e); //Метод удаления списка-элемента таблицы  bool deleteByData(void\* data); //Метод удаления элемента списка-элемента таблицы по значению информации  void scan();  void scan1(void\* data);  };  Object create();  }  #undef LISTNIL  #pragma once |
| Результат в консоли |
|  |
| Файл h.txt |
|  |