Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 8

По дисциплине «Структуры данных»

На тему «Фильтрация данных»

Выполнила:

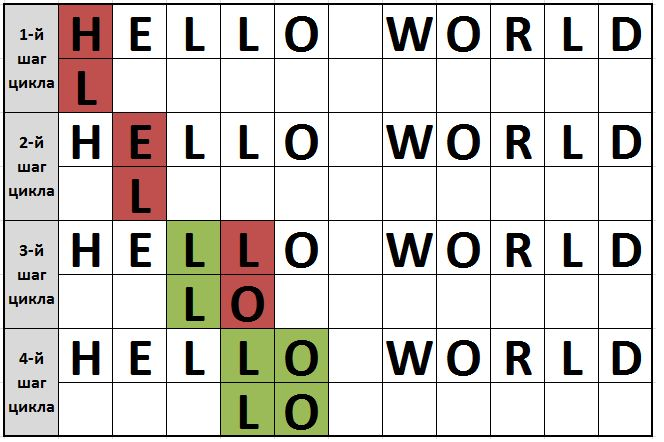
Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: Север А.С.

2023, Минск

**Проиллюстрируем поиск подстроки в строке:**

****

**Задание к лабораторной работе №8:**

Для вашей БД из лабораторной работы № 2 реализовать поиск подстроки в строке (фильтрацию, например по фамилии).

Задание из лабораторной работы №2

Вы являетесь сотрудником коммерческого отдела компании, продающей различные товары через Интернет. Вашей задачей является отслеживание финансовой составляющей работы компании.  
Работа Вашей компании организована следующим образом: на Интернет-сайте компании представлены (выставлены на продажу) некоторые товары. Каждый из них имеет некоторое название, цену и единицу измерения (штуки, килограммы, литры).  
Для проведения исследований и оптимизации работы магазина Вы пытаетесь собирать данные с Ваших клиентов. При этом для Вас определяющее значение имеют стандартные анкетные данные, а также телефон и адрес электронной почты для связи.  
По каждому факту продажи Вы автоматически фиксируете клиента, товары, количество, дату продажи, дату доставки.

|  |
| --- |
| Код программы |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <windows.h>  # define str\_len 40  # define SIZE 40  using namespace std;  struct Initial {  string fio;  char telephone[str\_len];  char email[str\_len];  char product[str\_len];  int numProduct;  char dateSale[str\_len];  char dateDelivery[str\_len];  };  vector<Initial> initial;  struct Initial bad;  int Search\_Binary(int left, int right, int key) {  int midd = 0;  while (1)  {  midd = (left + right) / 2;  if (key < initial[midd].numProduct) // если искомое меньше значения в ячейке  right = midd - 1; // смещаем правую границу поиска  else if (key > initial[midd].numProduct) // если искомое больше значения в ячейке  left = midd + 1; // смещаем левую границу поиска  else // иначе (значения равны)  return midd; // функция возвращает индекс ячейки  if (left > right) // если границы сомкнулись  return -1;  }  }  void filterStore(string ch) {  for (auto item : initial) {  int i = 0;  for (i = item.fio.find(ch, i++); i != string::npos; i = item.fio.find(ch, i + 1)) {  cout << i << endl;  }  }  }  void SelectionSort() // следует на каждой итерации найти элемент с наибольшим значением. С ним нужно поменять местами последний элемент  {  int smallest\_id;  for (int i = 0; i < initial.size(); i++) {  smallest\_id = i;  for (int j = i + 1; j < initial.size(); j++) {  if (initial[j].numProduct < initial[smallest\_id].numProduct)  smallest\_id = j;  }  swap(initial[smallest\_id], initial[i]);  }  }  void BubleSort() { // сравниваются соседние элементы и меняются местами, если следующий элемент меньше предыдущего  for (int i = 0; i < initial.size(); i++) {  for (int j = i + 1; j < initial.size(); j++) {  if (initial[i].numProduct > initial[j].numProduct)  swap(initial[i], initial[j]);  }  }  }  void insertionSort() // массив разбивается на две области: упорядоченную и и неупорядоченную  {  int key = 0;  int i = 0;  for (int j = 1; j < initial.size(); j++) {  key = initial[j].numProduct;  i = j - 1;  while (i >= 0 && initial[i].numProduct > key) {  swap(initial[i + 1], initial[i]);  i = i - 1;  initial[i + 1].numProduct = key;  }  }  }  void merge\_sort(vector<Initial>& v, int begin, int end) { // разбивается на мелкие кусочки - на первом этапе - на состоящие из одного элемента. Затем эти кусочки объединяются в более крупные кусочки  // Для отрезков длиной 1 не выполняем никаких операций.  // Это так называемый "базовый" (крайний) случай рекурсии, который  // гарантирует, что алгоритм закончит своё выполнение.  if (begin == end) {  return;  }  int mid = (begin + end) / 2;  // Рекурсивные вызовы алгоритма.  merge\_sort(v, begin, mid);  merge\_sort(v, mid + 1, end);  // Собственно слияние.  vector<Initial> t;  for (int i = begin, j = mid + 1; i <= mid || j <= end;) {  // Если одна из частей закончилась, добавляем элемент из другой  if (i > mid) {  t.push\_back(v[j++]);  }  else if (j > end) {  t.push\_back(v[i++]);  // Иначе добавляем меньший из текущих элементов  }  else if (v[i].numProduct <= v[j].numProduct) {  t.push\_back(v[i++]);  }  else {  t.push\_back(v[j++]);  }  }  // Копируем отсортированный отрезок из временного массива в v.  for (int i = 0; i < t.size(); i++) {  v[begin + i] = t[i];  }  }  void QuickSort(vector<Initial>& in, int left, int right)  {  if (right <= left)  return;  int ipi = left + rand() % (right - left + 1);  swap(in[ipi], in[right]);  int x = in[right].numProduct;  int m = left;  for (int i = left; i < right; i++)  if (in[i].numProduct <= x)  swap(in[i], in[m++]);  swap(in[m], in[right]);  QuickSort(in, left, m);  QuickSort(in, m + 1, right);  }  void ReadingData(string fileName)  {  cout << "=======================================\n";  for (auto client : initial)  {  cout << "Фамилия клиента: " << client.fio << "\n";  cout << "Почта клиента: " << client.email << "\n";  cout << "Телефон клиента: " << client.telephone << "\n";  cout << "Купленный товар: " << client.product << "\n";  cout << "Количество товаров: " << client.numProduct << "\n";  cout << "Дата продажи: " << client.dateSale << "\n";  cout << "Дата доставки: " << client.dateDelivery << "\n";  cout << "=======================================\n";  }  }  void ReadFromFile(string fileName) {  ifstream read(fileName, ios::app);  if (read) {  while (!read.eof())  {  Initial temp;  read >> temp.fio;  read >> temp.email;  read >> temp.telephone;  read >> temp.product;  read >> temp.numProduct;  read >> temp.dateSale;  read >> temp.dateDelivery;  initial.push\_back(temp);  }  }  else  cout << "Ошибка открытия файла!" << endl;  read.close();  }  void WriteInFile(string fileName) {  ofstream record(fileName, ios::app);  if (record) {  for (auto client : initial)  {  record << client.fio << "\n";  record << client.email << "\n";  record << client.telephone << "\n";  record << client.product << "\n";  record << client.numProduct << "\n";  record << client.dateSale << "\n";  record << client.dateDelivery << "\n";  }  }  else  cout << "Ошибка открытия файла!" << endl;  record.close();  }  void SavingData(string fileName)  {  ofstream record(fileName, ios::app);  if (record) {  cout << "Ввод информации" << endl;  Initial list\_of\_clients;  cout << endl << "Фамилия клиента: " << endl;  cin >> list\_of\_clients.fio;  cout << "Номер телефона: " << endl;  cin >> list\_of\_clients.telephone;  cout << "Электронная почта: " << endl;  cin >> list\_of\_clients.email;  cout << "Товар: " << endl;  cin >> list\_of\_clients.product;  cout << "Количество товаров: " << endl;  cin >> list\_of\_clients.numProduct;  cout << "Дата продажи: " << endl;  cin >> list\_of\_clients.dateSale;  cout << "Дата доставки: " << endl;  cin >> list\_of\_clients.dateDelivery;  initial.push\_back(list\_of\_clients);  }  else  cout << "Ошибка открытия файла!" << endl;  record.close();  }  void Change()  {  int ind;  cout << "Input index for changing\n";  cin >> ind;  cout << endl << "Фамилия клиента: " << endl;  cin >> initial[ind].fio;  cout << "Номер телефона: " << endl;  cin >> initial[ind].telephone;  cout << "Электронная почта: " << endl;  cin >> initial[ind].email;  cout << "Товар: " << endl;  cin >> initial[ind].product;  cout << "Количество товаров: " << endl;  cin >> initial[ind].numProduct;  cout << "Дата продажи: " << endl;  cin >> initial[ind].dateSale;  cout << "Дата доставки: " << endl;  cin >> initial[ind].dateDelivery;  }  void delInFile(string fileName) {  ofstream file("Output.txt", ios\_base::trunc);  }  void DeleteData()  {  int d;  cout << "\n 1 - Удалить определённый фрагмент\n 2 - Удалить всё ";  cin >> d;  auto iter = initial.cbegin();  switch (d)  {  case 1:  int ind;  cout << "Выберете фрагмент от 0 до " << initial.size() << "\n";  cin >> ind;  initial.erase(ind + iter);  break;  case 2:  initial.clear();  break;  default:  break;  }  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  setlocale(LC\_ALL, ".1251");  int filevar = NULL, var = NULL, num, number;  string fileName, fiosearch;  do {  cout << "ФАЙЛ: \n 1-ваше имя файла\n 2-по стандарту\n 0 - выход\n";  cin >> filevar;  switch (filevar)  {  case 1:  {  cout << "введите ваше имя файла\n";  cin >> fileName;  do {  cout << "\n1-вывод в консоль" << endl;  cout << "2-для ввода новой записи" << endl;  cout << "3-запись в файл" << endl;  cout << "4-чтение из файла" << endl;  cout << "5-удаление всего содержимого файла" << endl;  cout << "6-частичное удаление информации" << endl;  cout << "7-изменение информации" << endl;  cout << "8 - сортировка пузырьком по количеству продуктов по возрастанию" << endl;  cout << "9 - сортировка выбором" << endl;  cout << "10 - сортировка вставками" << endl;  cout << "11 - сортировка слиянием" << endl;  cout << "12 - быстрая сортировка" << endl;  cout << "13 - бинарный поиск по количеству товаров" << endl;  cout << "14 - фильтрация" << endl;  cout << "0-для выхода" << endl;  cin >> var;  switch (var)  {  case 1: ReadingData(fileName); break;  case 2: SavingData(fileName); break;  case 3: WriteInFile(fileName); break;  case 4: ReadFromFile(fileName); break;  case 5: delInFile(fileName); break;  case 6: DeleteData(); break;  case 7: Change(); break;  case 8: BubleSort(); break;  case 9: SelectionSort(); break;  case 10: insertionSort(); break;  case 11: merge\_sort(initial, 0, initial.size() - 1); break;  case 12: QuickSort(initial, 0, initial.size() - 1); break;  case 13:  {  cout << "количество товаров, по которой искать" << endl;  cin >> num;  number = Search\_Binary(0, initial.size() - 1, num);  if (number == -1) cout << "не найдено!" << endl;  else cout << "клиент с " << num << " количеством товаров под индексом " << number;  }  case 14:  {  cout << "введите фамилию для поиска ";  cin >> fiosearch;  cout << "индекс равен ";  filterStore(fiosearch);  break;  }  default:  break;  }  } while (var != 0);  }  case 2:  {  do  {  cout << "\n1-вывод в консоль" << endl;  cout << "2-для ввода новой записи" << endl;  cout << "3-запись в файл" << endl;  cout << "4-чтение из файла" << endl;  cout << "5-удаление всего содержимого файла" << endl;  cout << "6-частичное удаление информации" << endl;  cout << "7-изменение информации" << endl;  cout << "8 - сортировка пузырьком по количеству продуктов по возрастанию" << endl;  cout << "9 - сортировка выбором" << endl;  cout << "10 - сортировка вставками" << endl;  cout << "11 - сортировка слиянием" << endl;  cout << "12 - быстрая сортировка" << endl;  cout << "13 - бинарный поиск по количеству товаров" << endl;  cout << "14 - фильтрация" << endl;  cout << "0-для выхода" << endl;  cin >> var;  switch (var)  {  case 1: ReadingData("Output.txt"); break;  case 2: SavingData("Output.txt"); break;  case 3: WriteInFile("Output.txt"); break;  case 4: ReadFromFile("Output.txt"); break;  case 5: delInFile("Output.txt"); break;  case 6: DeleteData(); break;  case 7: Change(); break;  case 8: BubleSort(); break;  case 9: SelectionSort(); break;  case 10: insertionSort(); break;  case 11: merge\_sort(initial, 0, initial.size() - 1); break;  case 12: QuickSort(initial, 0, initial.size() - 1); break;  case 13:  {  cout << "количество товаров, по которой искать" << endl;  cin >> num;  number = Search\_Binary(0, initial.size() - 1, num);  if (number == -1) cout << "не найдено!" << endl;  else cout << "клиент с " << num << " количеством товаров под индексом " << number;  }  case 14:  {  cout << "введите фамилию для поиска ";  cin >> fiosearch;  cout << "индекс равен ";  filterStore(fiosearch);  break;  }  default:  break;  }  } while (var != 0);  }  default:  break;  }  } while (filevar != 0);  } |
| Данные из консоли |
|  |