Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Тема: “Хэш-таблица"

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Мальчиков Дмитрий Григорьевич

Проверил

Доцент кафедры ИТАС

Полякова Ольга Андреевна

Г. Пермь-2023

Постановка задачи

1. Создать динамический массив из записей (в соответствии с вариантом), содержащий не менее 100 элементов. Для заполнения элементов массива использовать ДСЧ.
2. Предусмотреть сохранение массива в файл и загрузку массива из файла.
3. Предусмотреть возможность добавления и удаления элементов из массива (файла).
4. Выполнить поиск элемента в массиве по ключу в соответствии с вариантом. Для поиска использовать хэш-таблицу.
5. Подсчитать количество коллизий при размере хэш-таблицы 40, 75 и 90 элементов.

Вариант 20:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Данные** | **Ключ (string)** | **Хэш-функция** | **Метод рехеширования** |
| 20 | ФИО, дата\_рождения, №паспорта, | дата\_рождения | H(k)= [M (kAmod1)], 0<A<1, mod1 – получение дробной части, [] – получение целой части | Метод цепочек |

Анализ задачи

По условию задачи необходимо создать хэш-таблицу, т.е. структуру данных, которая позволяет хранить пару “ключ-значение” и выполнять функцию добавления элемента, поиска элемента по ключу и удаление элемента.

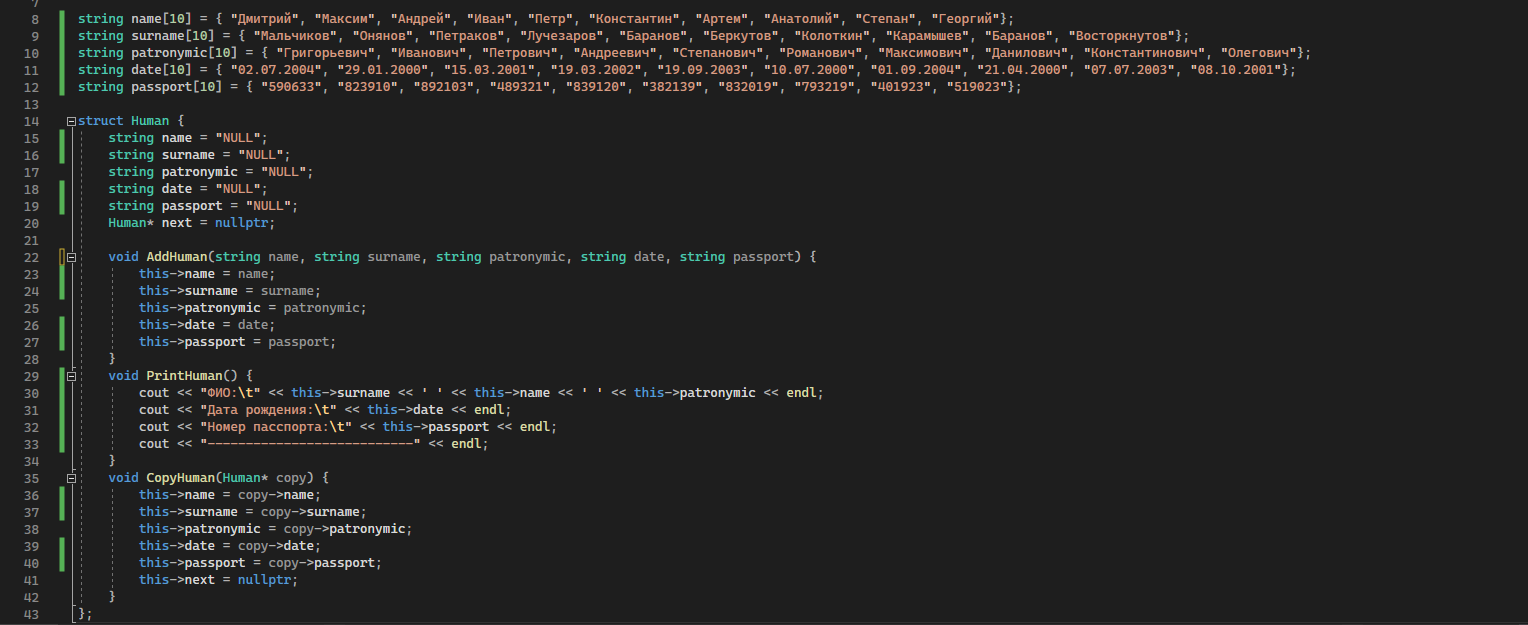
Перед тем, как создавать хэш-таблицу, создадим сначала структуру Human, которая позволяет создать информацию о человеке: ФИО, дата рождения и номер паспорта. В этой же структуре напишем функции AddHuman и CopyHuman для добавления элемента (понадобится в списках) и копирования элемента (понадобится в хэшировании методом цепочек).

Для работы хэш-таблицей необходимо реализовать односвязный список, а также добавление в начало, середину конец списка + удаление + вывод списка.

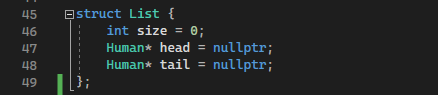
Для реализации методом хэширования (цепочки + открытая адресация) напишем отдельные функции, которые работают с хэш-таблицей (открытая адресация - работает с хэш-таблицей и массивами, цепочки - с хэш-таблицей с односвязными списками).

Проектирование функций

1. Структура struct Human() и её функции SetHuman() и CopyHuman(). Создаётся отдельная структура для работы с данными о людях. SetHuman() - конструктор, PrintHuman() - вывод информации о человеке.



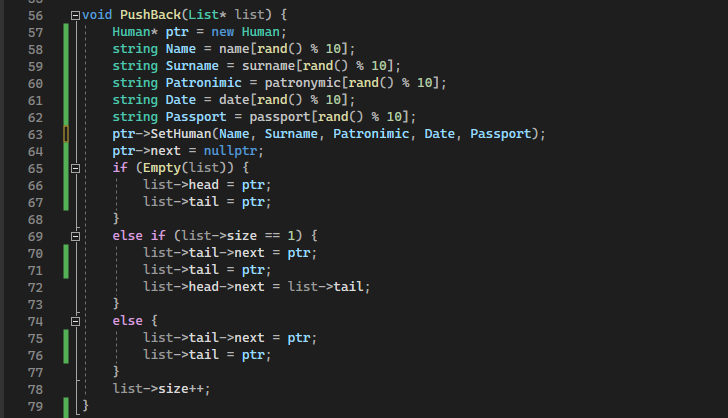
1. Структура struct List - структура однонаправленного списка.



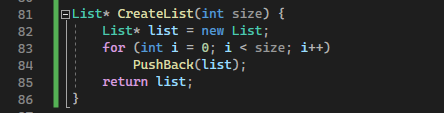
1. Функция bool Empty() - проверяет список на его пустоту. Нужна в дальнейшем для удобства (чтобы не нагромождать код).



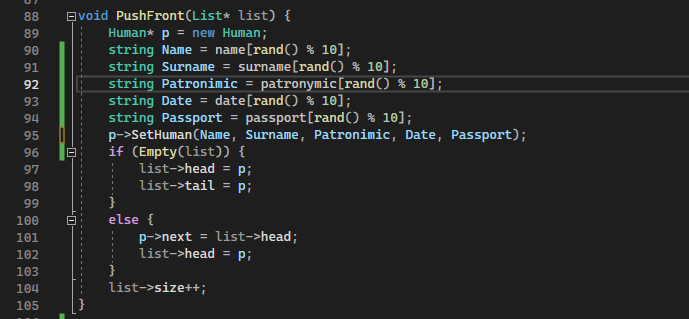
1. Функция PushBack() - добавление элемента в конец однонаправленного списка. Нужна для задачи “добавление в конец списка” и создание нового списка.



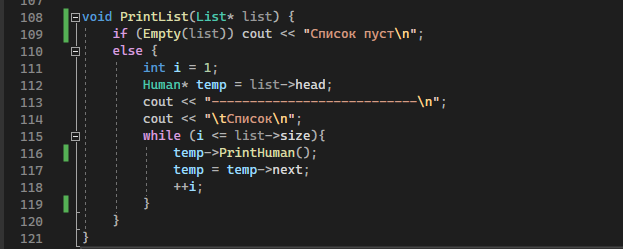
1. Функция CreateList() - создание списка. На вход поступает размер списка и через функцию PushBack() реализуется его создание. Нужна для создания списка на начальном этапе выполнения программы (чтобы много раз самому не вызывать PushBack()).



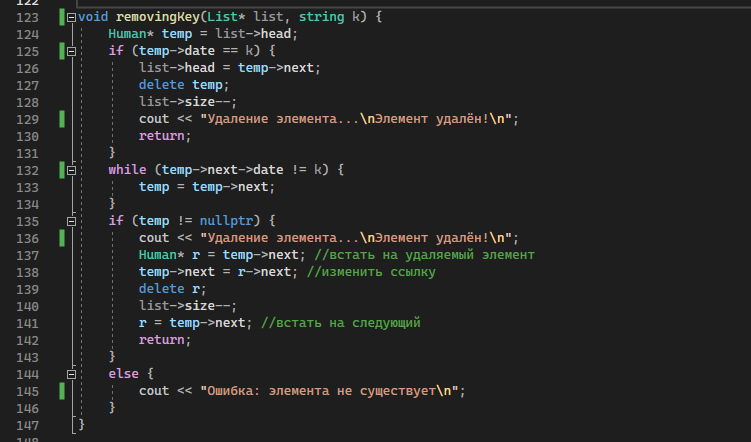
1. Функция PushFront() - добавление элемента в начало списка. Нужна для решения задачи “добавление в начало списка”.



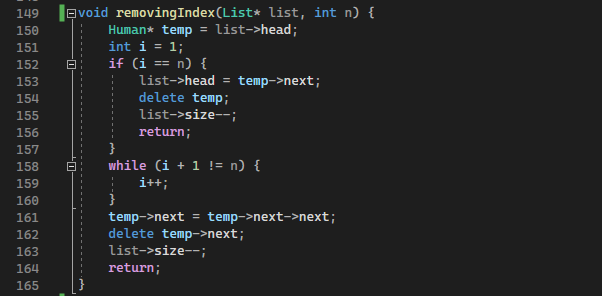
1. Функция PrintList() - вывод всего списка. Нужна для решения задачи “вывести все элементы”.



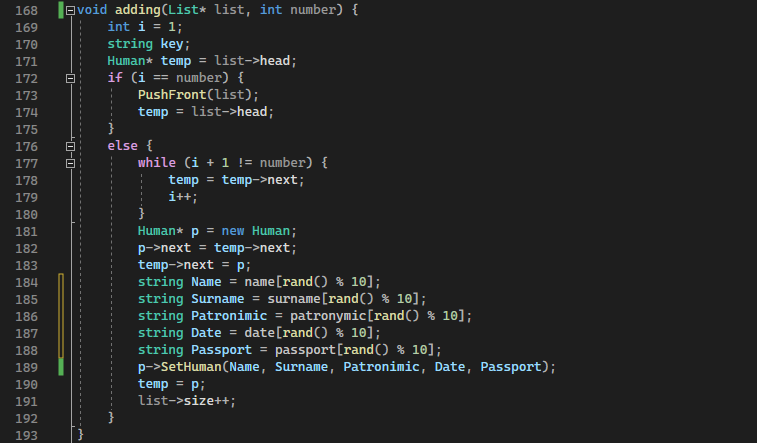
1. Функция removingKey() - удаление элемента из списка по ключу. На вход поступает список и ключ. Через цикл проходимся через весь односвязный список и, если найден необходимый ключ (в нашем случае - дата рождения), то удаляем элемент из списка. Иначе же выведется “Ошибка: элемент не существует”.



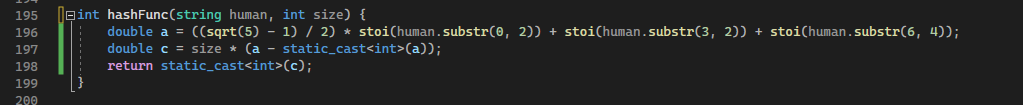
1. Функция removingIndex() - удаление элемента по индексу. На вход поступает список и индекс элемента. Создаём счетчик и проходимся по всему списку через цикл, каждый раз увеличивая счетчик. И, когда счетчик будет равен поступившему индексу, то удаляем тот элемент списка, на котором остановились.



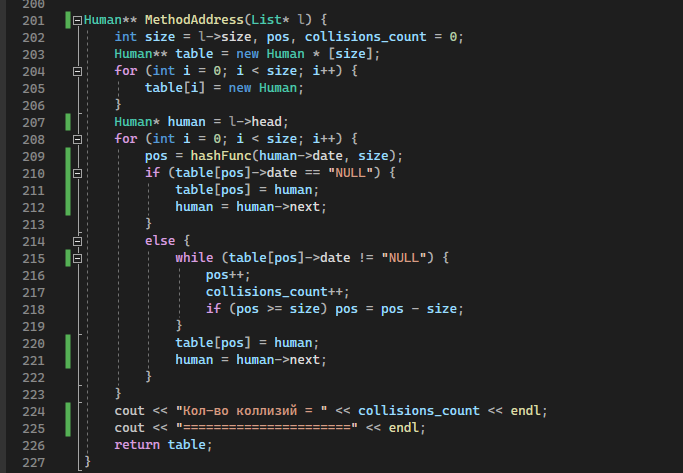
1. Функция adding() - добавление элемент на заданную позицию (номер). На вход поступает список и номер. Если номер = 1, то элемент добавится в начало списка. Если же номер > 1, то элемент добавляется на заданный номер так, что поступаемый номер = номеру нового элемента в списке.



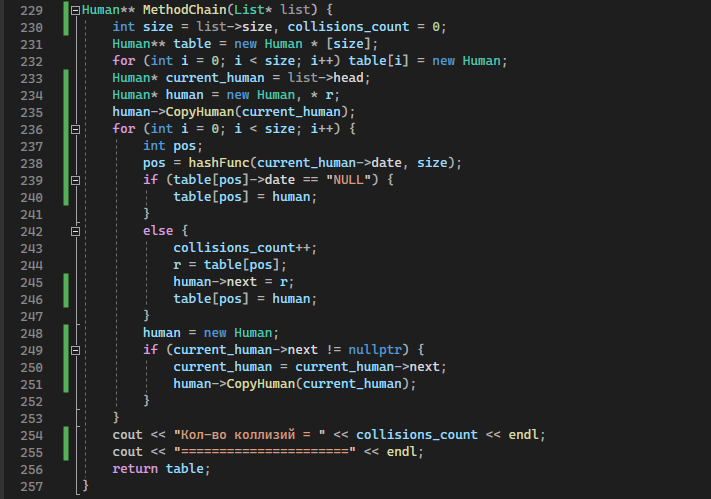
1. Функция hashFunc() - создание хэш-функции. Возвращает значение, которое получается при вычислении хэш-функции



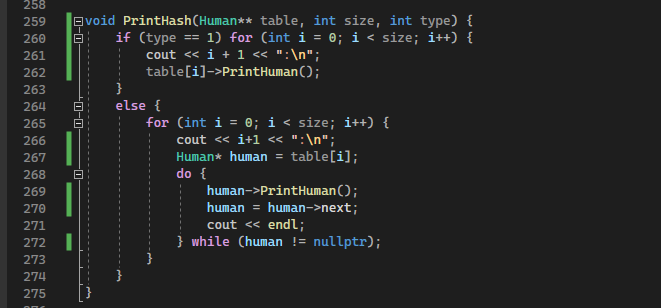
1. Функция MethodAddress() - хэширование методом открытой адресации. На вход поступает указатель на список. Создаем двумерный массив table, указатель human, указывающий на элементы списка (изначально - голова) и размер поступающего списка. В цикле for (int i = 0; i < size; i++) проверяем элементы на пустоту. Если элемент пустой, то просто туда записываем данные из human. Если не пустой, то возникла коллизия, поэтому добавляем к счётчику коллизий +1, перескакиваем элементы, пока не найдётся нулевой и, когда нулёвой всё-таки нашелся, записываем туда данные из human. В цикле каждый раз меняем указатель human: он будет указывать каждый раз на следующий элемент изначального списка. В итоге получаем количество коллизий и возвращаем table для дальшейшей работы с поиском элемента (по задаче)



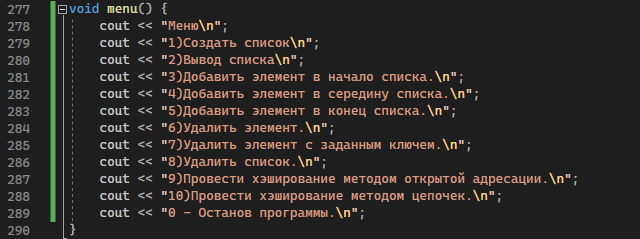
1. Функция MethodChain() - хэширование методом цепочек. На вход поступает указатель на список. Создаём двумерный массив table, указатель current\_human - указатель на элемент в списке, а также переменную human, куда запишутся данные из current\_human. Далее создаем цикл for(int i = 0; i < size; i++). В нём переменная pos принимает значение хэш-функции. В условном операторе проверяется, пустая ли ячейка для записи элемента. Если пуста, то записываем туда скопированные данные переменной human. Если же не пустая, то возникла коллизия, поэтому к счётчику коллизий прибавляем +1 и в этой ячейке будет список, длина (количество элементов) которого будет увеличиваться на 1 и элемент добавляется в конец этого списка, пока не встретится новый пустой элемент в table. Каждый раз в цикле переменная human меняется: она копирует информацию из списка через переменную current\_human. В конце концов выводится количество коллизий и возвращается массив table для дальнейшей работы с поиском элемента (по задаче).



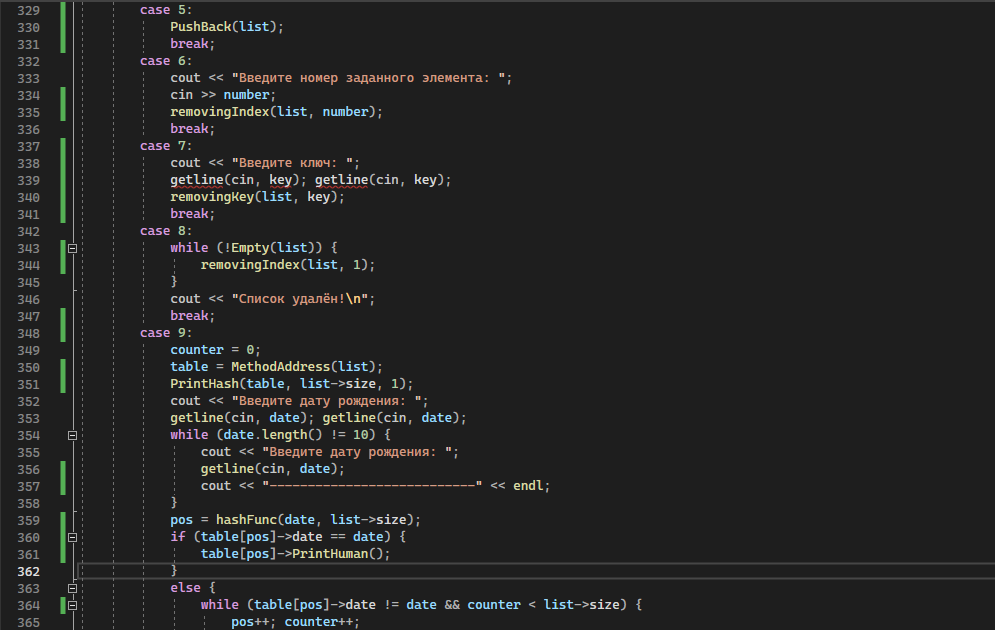
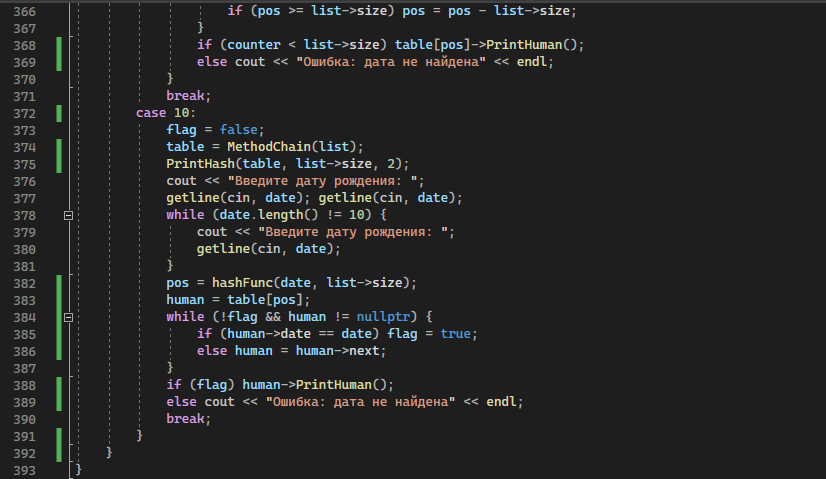
1. Функция PrintHash() - вывод хэш-таблицы. На вход поступают полученный методом хэширования двойной указатель table, размер списка и тип вывода (для метода цепочек). Через цикл проходимся по всей таблице и выводим её.



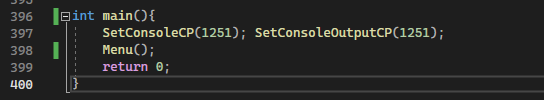
1. Функция menu() - просто вывод интерактивного меню на экран.



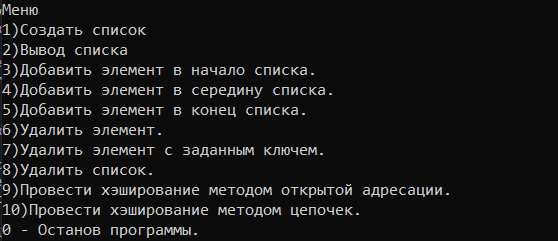
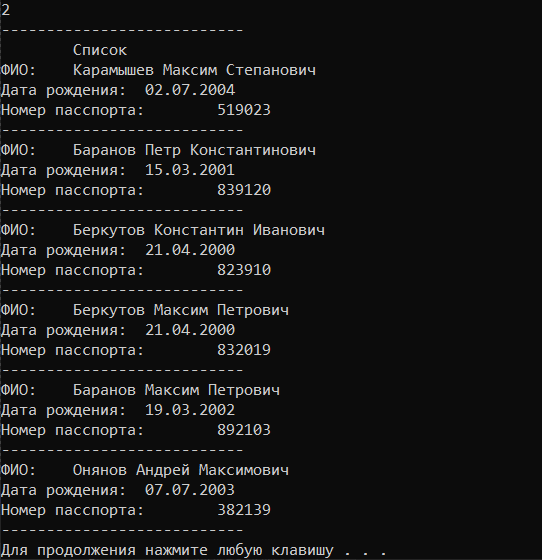
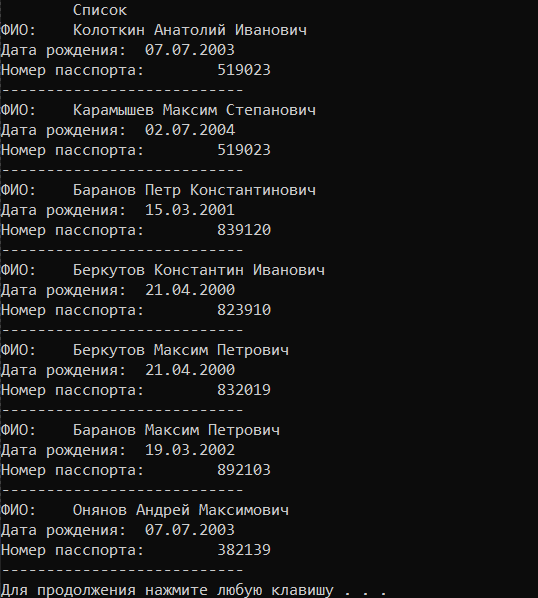
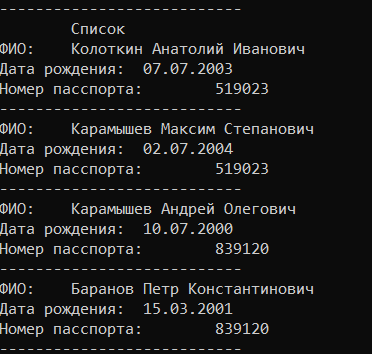
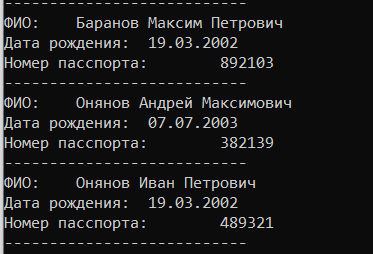
1. Функция Menu() - работа с меню. Работаем со switch(): если 1 - создаем список (вызываем CreateList()), если 2 - выводим список (вызываем PrintList()) и т.д. Программа завершит работу, если пользователь введёт 0.

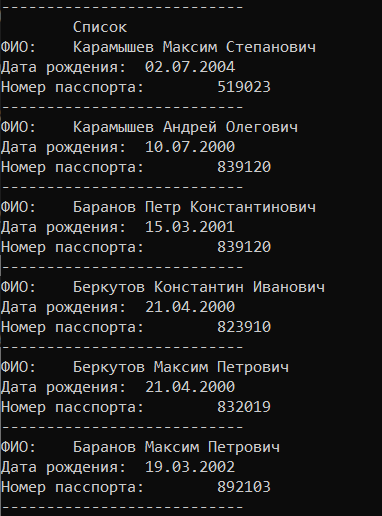
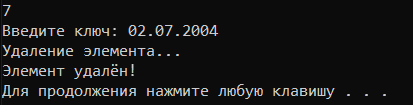
  
  


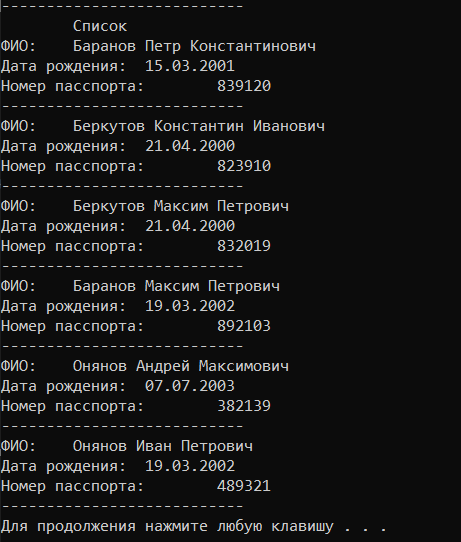
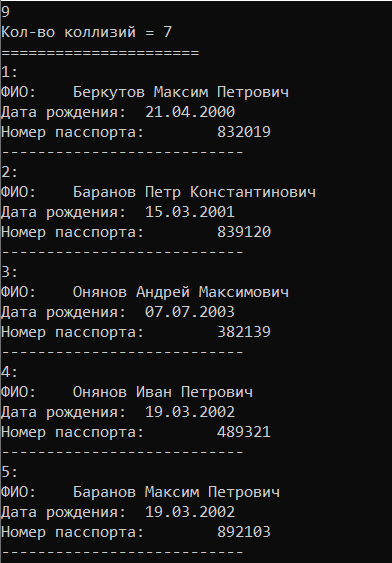
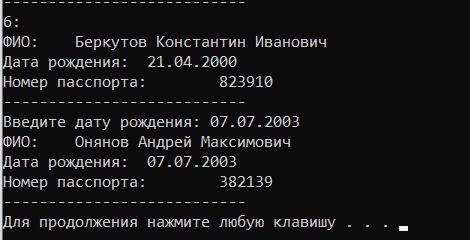
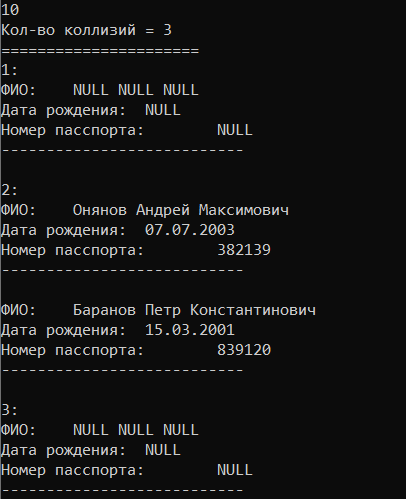
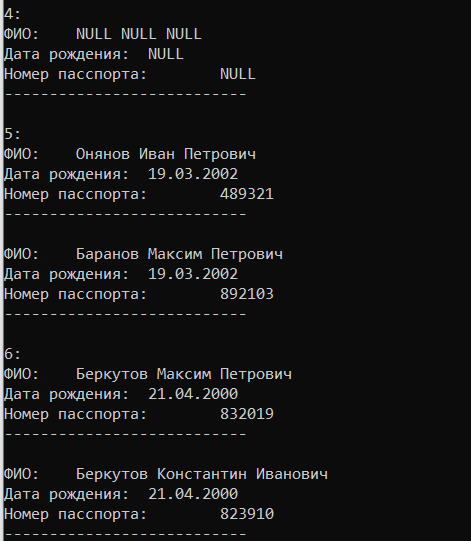
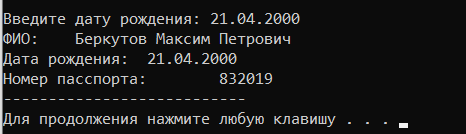
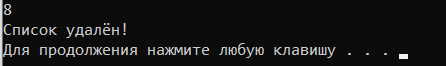
1. В главной функции int main() настраиваем консоль на работу с русскими символами и вызываем функцию Menu() (в дальнейшем эта функция может вызвать все остальные).



Тестирование работы программы.

Меню -   
1 -   
2 -   
3 -   
3 (вывод) -  
4 -   
4 (вывод) -   
5 -   
5 (вывод) - 

6 -   
6 (вывод) -   
7 - 

7 (вывод) -   
9 + вывод -   
   
10 + вывод -   
  
  
8 -   
8 (вывод) -   
0 - 