МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЁН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., к.т.н. |  |  |  | А. В. Туманова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРОГРАММЫ «СПИСОК КОНТАКТОВ» |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 |  |  |  |  |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 4189 | |  |  |  |

Санкт-Петербург 2021

Содержание

[1. Постановка задачи 3](#_Toc62581724)

[2. Описание структур данных 3](#_Toc62581725)

[3. Описание программы и созданных функций 4](#_Toc62581726)

[3.1. Функции вызываемые напрямую из главного меню программы. 5](#_Toc62581727)

[3.1.1 find\_element 5](#_Toc62581728)

[3.1.2 add\_node 5](#_Toc62581729)

[3.1.3 del\_node 5](#_Toc62581730)

[3.1.4 my\_shaker 6](#_Toc62581731)

[3.1.5 print\_list 6](#_Toc62581732)

[3.1.6 save\_list 6](#_Toc62581733)

[3.1.7 load\_list 7](#_Toc62581734)

[3.1.8 is\_string\_correct 8](#_Toc62581735)

[3.1.9 find\_family 8](#_Toc62581736)

[3.2. Функции, вызываемые не из основного меню. 9](#_Toc62581737)

[3.2.1 is\_unique 9](#_Toc62581738)

[3.2.2 load\_node2 9](#_Toc62581739)

[3.2.3 int\_input 10](#_Toc62581740)

[3.2.4 phone\_input 10](#_Toc62581741)

[3.2.5 birth\_date\_input 10](#_Toc62581742)

[3.2.6 is\_date\_correct 10](#_Toc62581743)

[3.2.7 is\_phone\_correct 11](#_Toc62581744)

[3.3. Общие пояснения по исполнению основных функций приложения 12](#_Toc62581745)

[4. Описание пользовательского интерфейса 12](#_Toc62581746)

[5. Результаты тестирования программы 15](#_Toc62581747)

[5.1. Тестирование готового приложения 15](#_Toc62581748)

[Заключение 20](#_Toc62581749)

[Приложение. Листинг программы. 20](#_Toc62581750)

[Использованная литература 62](#_Toc62581751)

# Постановка задачи

Задачей курсового проекта является разработка программы реализующей список контактов.

# Описание структур данных

2.1 Данные о контакте хранятся в структуре Contact\_List, на ее основе реализован

циклический двусвязный список.

Структура несет в себе поля:

Contact\_List\* next – указатель на следующий элемент списка.

Contact\_List\* prev – указатель на предыдущий элемент списка.

int birth\_date[3] – целочисленный массив из 3 элементов для хранения даты рождения. Выбраны следующие ограничения:

- Год рождения ограничен от 1920 до 2020 года.

- Месяц ограничен от 1 до 12.

- День месяца ограничивается в соответствием с ограничениями, накладываемыми календарем для выбранного периода.

int phone[11] – целочисленный массив из 11 элементов для хранения телефонного номера.

Первый элемент ограничен цифрой 8, ибо в наших реалиях все номера начинаются с 8.

Элементы со 2 по 11 ограничены выбором значений от 0 до 9.

char family[20] – символьный массив из 20 элементов для хранения фамилии.

Элементы массива ограничены списком кириллических символов верхнего и нижнего регистра.

char name[20] – символьный массив из 20 элементов для хранения имени.

Элементы массива ограничены списком кириллических символов верхнего и нижнего регистра.

2.2 Для передачи данных внутри программы используется вспомогательная структура

load\_struct.

Структура несет в себе поля:

int birth\_date[3] – целочисленный массив из 3 элементов для хранения даты рождения. Выбраны следующие ограничения:

- Год рождения ограничен от 1920 до 2020 года.

- Месяц ограничен от 1 до 12.

- День месяца ограничивается в соответствием с ограничениями, накладываемыми календарем для выбранного периода.

int phone[11] – целочисленный массив из 11 элементов для хранения телефонного номера.

Первый элемент ограничен цифрой 8, ибо в наших реалиях все номера начинаются с 8.

Элементы со 2 по 11 ограничены выбором значений от 0 до 9.

char family[20] – символьный массив из 20 элементов для хранения фамилии.

Элементы массива ограничены списком кириллических символов верхнего и нижнего регистра.

char name[20] – символьный массив из 20 элементов для хранения имени.

Элементы массива ограничены списком кириллических символов верхнего и нижнего регистра.

2.3 База данных хранится на диске в виде файла, используется формат файла “csv”, разделителями служат «;». Для работы с БД в процессе выполнения программы

используется циклический список. Комбинации «фамилия-имя» в программе

являются уникальными.

# Описание программы и созданных функций

Программа реализована на языке C++ в виде консольного приложения.

В программе реализована следующая функциональность:

* Добавление записи
* Отображение списка контактов
* Редактирование записи
* Удаление записи
* Чтение данных из файла в CSV-формате
* Запись данных в файл в CSV-формате
* Поиск в записях по Фамилии и Имени
* Сортировка по Фамилии в алфавитном порядке.

## Функции вызываемые напрямую из главного меню программы.

### 3.1.1 find\_element

Назначение: поиск и(или) редактирование элемента списка по комбинации «фамилия-имя».

Входные данные:

- redact – целочисленное. ОДЗ ={0,1}

При передаче нуля осуществляется поиск элемента с указанными фамилией и именем и вывод на экран результата.

При передаче единицы осуществляется поиск элемента с указанными фамилией и именем и в случае нахождения его редактирование, в случае ненахождения элемента выводится извещение об этом на экран.

- family – символьный массив, несет в себе фамилию для поиска.

- name – символьный массив, несет в себе имя для поиска.

Возвращаемые данные: отсутствуют.

### 3.1.2 add\_node

Назначение: добавление элемента в список.

Входные данные: отсутствуют.

Возвращаемые данные: отсутствуют.

Особенности: в функции присутствует проверка уникальности элемента по связке «фамилия-имя», в случае неуникальности элемент не добавляется, выводится сообщение о неуникальности.

### 3.1.3 del\_node

Назначение: удаление элемента из списка

Входные данные:

- family\_d – символьный массив, несет в себе фамилию для поиска.

- name\_d – символьный массив, несет в себе имя для поиска.

Возвращаемые данные: отсутствуют.

Особенности: в случае не нахождения в списке элемента с такими именем и фамилией выдается сообщение об отсутствии подходящего элемента. В случае нахождения подходящего элемента он удаляется.

### 3.1.4 my\_shaker

Назначение: сортировка списка по алфавиту, опираясь на фамилию контакта.

Входные данные: отсутствуют.

Возвращаемые данные: отсутствуют.

### 3.1.5 print\_list

Назначение: вывод списка на экран в форматированном виде.

Входные данные: отсутствуют.

Возвращаемые данные: отсутствуют.

Особенности: в случае пустоты списка выдается сообщение об этом на экран.

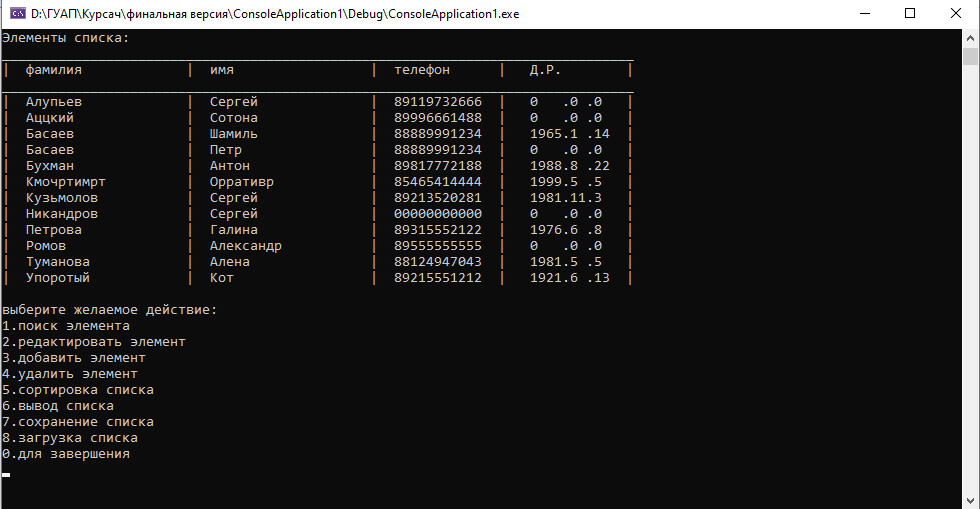


Рис 1. Пример вывода на экран содержимого списка.

### 3.1.6 save\_list

Назначение: сохранение списка в файл формате csv.

Входные данные: отсутствуют.

Возвращаемые данные: отсутствуют.

Особенности: в случае пустоты списка выдается сообщение об этом на экран. Разделителями служат «;» как стандартно используемые сейчас в «csv».

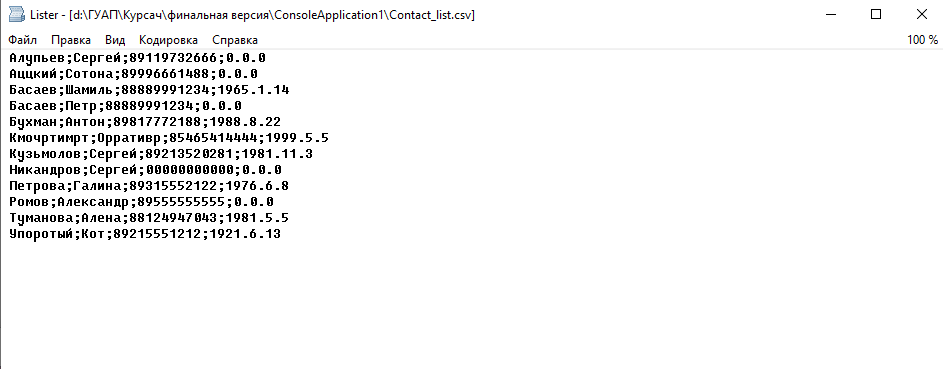


Рис 2. Пример файла с сохраненным в нем списком.

### 3.1.7 load\_list

Назначение: загрузка списка из файла формата csv.

Входные данные: отсутствуют.

Возвращаемые данные: отсутствуют.

Особенности:

- В случае отсутствия файла выдается сообщение об этом на экран. Разделителями служат «;» как стандартно используемые сейчас в «csv».

- При загрузке производится проверка на соответствие данных обозначенным рамкам.

- Фамилия и имя содержат только кириллические символы верхнего и нижнего регистра. В случае, если фамилия или имя содержат некорректные данные, то вместо некорректных данных записывается слово «Ошибка».

- Поле «дата рождения» проверяются на реальность даты, в случае некорректности даты она заполняется нулями.

- Поле «телефон» проверяется на соответствие обозначенной маске «8??????????» при несовпадении с маской поле заполняется нулями.

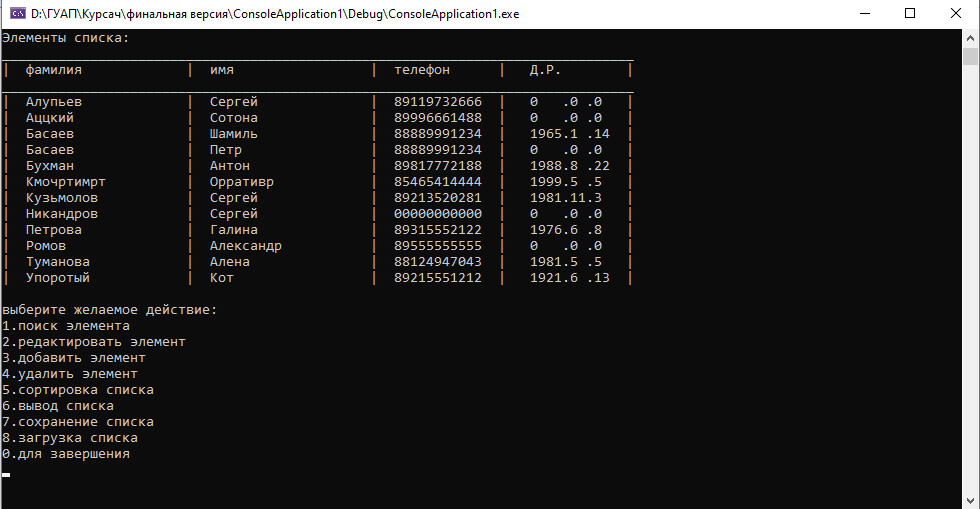


Рис 4. Список после загрузки его из файла.

### 3.1.8 is\_string\_correct

Назначение: проверка корректности введенных данных под задачи программы.

Входные данные:

test\_fam – символьный массив, несет в себе информацию для проверки корректности введенных символов.

Возвращаемые данные:

Булево, если входные данные корректны, то «true», иначе «false»

### 3.1.9 find\_family

Назначение: поиск введенной фамилии среди фамилий, имеющихся в списке.

Входные данные:

test\_fam – символьный массив, несущий фамилию для проверки ее уникальности.

Возвращаемые данные:

Булево, если фамилия найдена, то возвращается «true», иначе возвращается «false».

## Функции, вызываемые не из основного меню.

### 3.2.1 is\_unique

Назначение: проверка уникальности сочетания «фамилия-имя» в списке.

Входные данные:

tested – структура Contact\_List, ее поля и проверяются на уникальность.

Возвращаемые данные:

Булево, в случае отсутствия совпадений возвращается «true», в противном случае «false».

### 3.2.2 load\_node2

Назначение разбор входного символьного массива на части по разделителям, преобразование каждой части массива по своему механизму в соответствующие элементы списка, добавление этих элементов в узел списка, проверка этого узла на приемлемость входящих данных, добавление нового узла в список.

Входные данные:

node\_beta – символьный массив, получаемый при построчной разборке файла.

Выходные данные: отсутствуют.

Особенности:

- в случае ошибок в текстовых полях структуры они замещаются словом «ошибка»

- в случае ошибок в дате рождения и(или) телефоне они заливаются нулями соответственно.

- после разбора массива символов и заполнения структуры она проверяется на уникальность по связке «фамилия-имя», в случае уникальности элемент записывается, в противном случае пропускается.

### 3.2.3 int\_input

Назначение: ввод целочисленных данных с проверкой их корректности.

Входные данные: отсутствуют.

Возвращаемые данные: целочисленное.

Особенности: проверяет введенное значение на соответствие целочисленному типу, в случае несоответствия инициирует повторный ввод.

### 3.2.4 phone\_input

Назначение: ввод телефонного номера в соответствие с маской «8??????????»

Входные данные: отсутствуют.

Возвращаемые данные: целочисленный массив из 11 элементов, первый из которых всегда «8».

Особенности: жестко контролирует корректность вводимых данных и в случае несоответствия их маске инициирует повторный ввод.

### 3.2.5 birth\_date\_input

Назначение: ввод даты рождения с проверкой корректности этой даты.

Входные данные: отсутствуют.

Возвращаемые данные: целочисленный массив из 3 элементов.

Особенности:

Год ограничен рамками от 1920 до 2020, месяц от 1 до 12, день в зависимости от месяца и високосности года имеет специфические ограничения.

### 3.2.6 is\_date\_correct

Назначение: проверка указанной даты на корректность.

Входные данные: целочисленный массив из 3 элементов.

Возвращаемые данные:

Булево, если дата корректна, то «true», иначе «false».

Особенности:

Год ограничен рамками от 1920 до 2020, месяц от 1 до 12, день в зависимости от месяца и високосности года имеет специфические ограничения.

### 3.2.7 is\_phone\_correct

Назначение: проверка указанного телефона на корректность.

Входные данные: целочисленный массив из 11 элементов.

Возвращаемые данные:

Булево, если телефон корректен, то «true», иначе «false».

Особенности:

Первый символ массива всегда должен быть «8»

Со 2 по 11 сиволы могут быть от «0» до «9»

## Общие пояснения по исполнению основных функций приложения

- При запуске функция main() выставляет рабочую кодировку командной строки Windows-1251, выводит главное меню на экран и ожидает нажатия на кнопку клавиатуры посредством \_getch()

- Каждый пункт меню реализован через отдельную функцию.

- После выполнения любого из пунктов меню повторно происходит вывод основного меню.

# Описание пользовательского интерфейса

Главное меню приложения:

1. Поиск элемента
2. Редактировать элемент
3. Добавить элемент
4. Удалить элемент
5. Сортировка списка
6. Вывод списка
7. Сохранение элемента
8. Загрузка списка
9. Для завершения

Главное меню отображается сразу после загрузки программы.

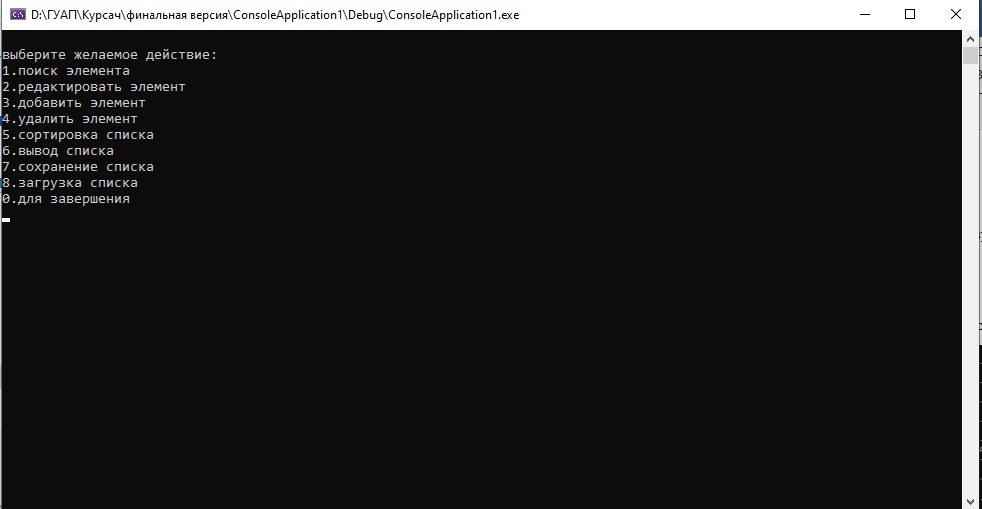


Рис 5. Главное меню.

Отображение списка записей оформлено в виде таблицы:

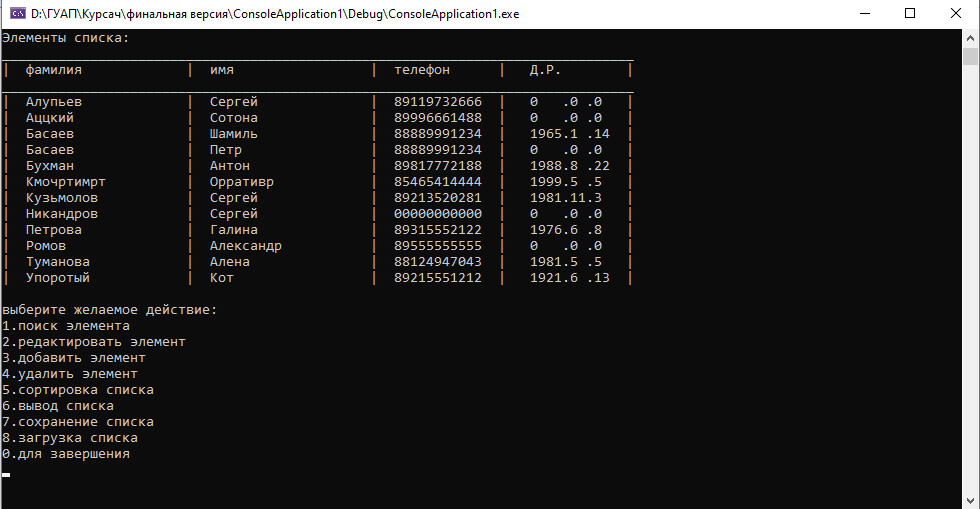


Рис 6 - Таблица записей

Операции ввода телефона и даты рождения работают по схеме вывода подсказки пользователю на ввод, затем ждут ввода.

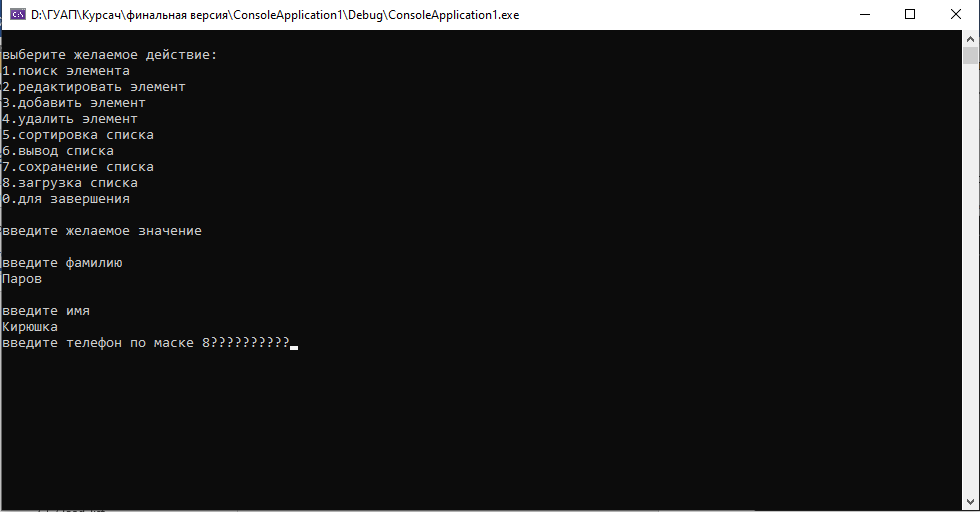


Рис 7 - Операция добавления телефона

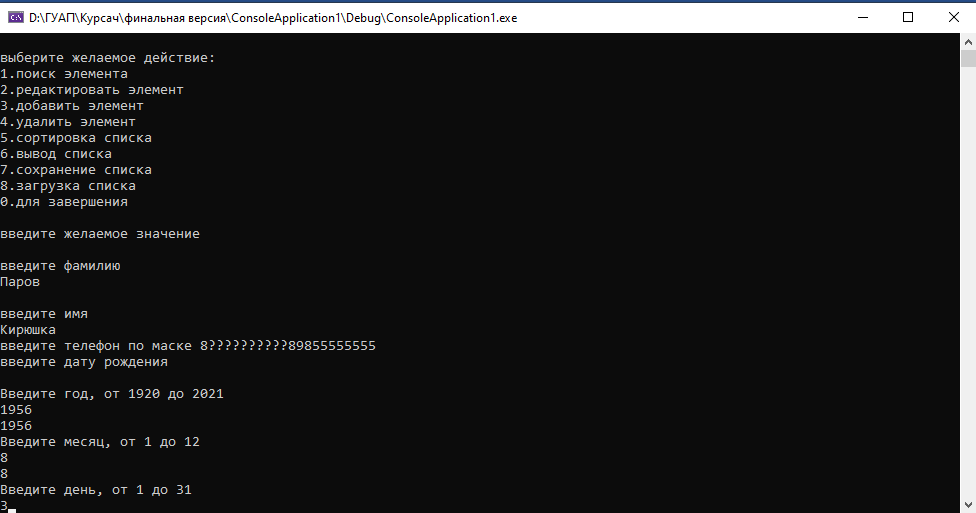


Рис 8. Операция добавления даты рождения.

# Результаты тестирования программы

## Тестирование готового приложения

Для тестирования приложения был заготовлен специально обученный файл.

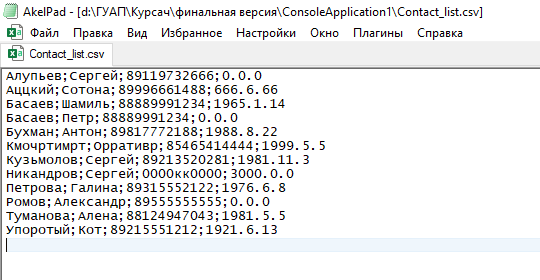


Рис 9. Файл со входными данными, некоторые из которых некорректные.

Их добавление проверялось через загрузку посредством функций list\_load и node\_load.

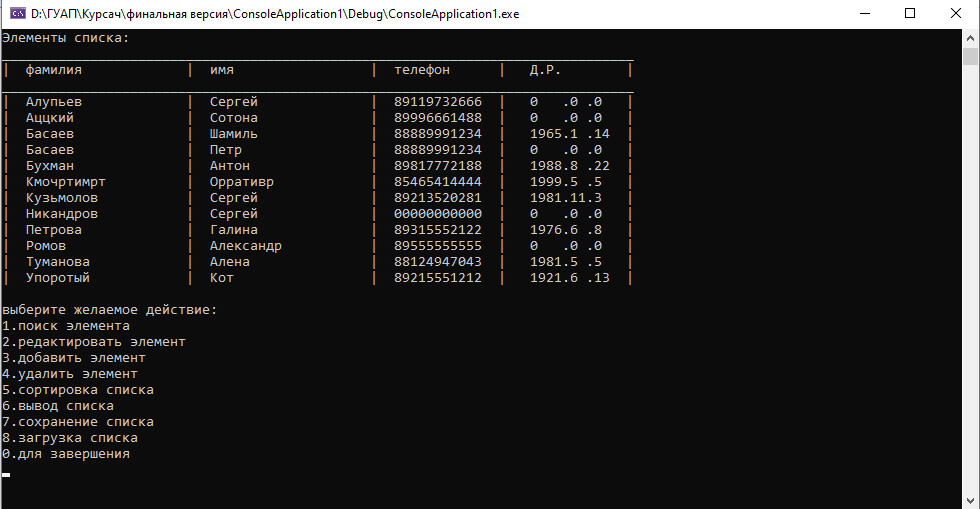


Рис 10 – Загруженный файл с внесенными программой коррективами неверных данных.

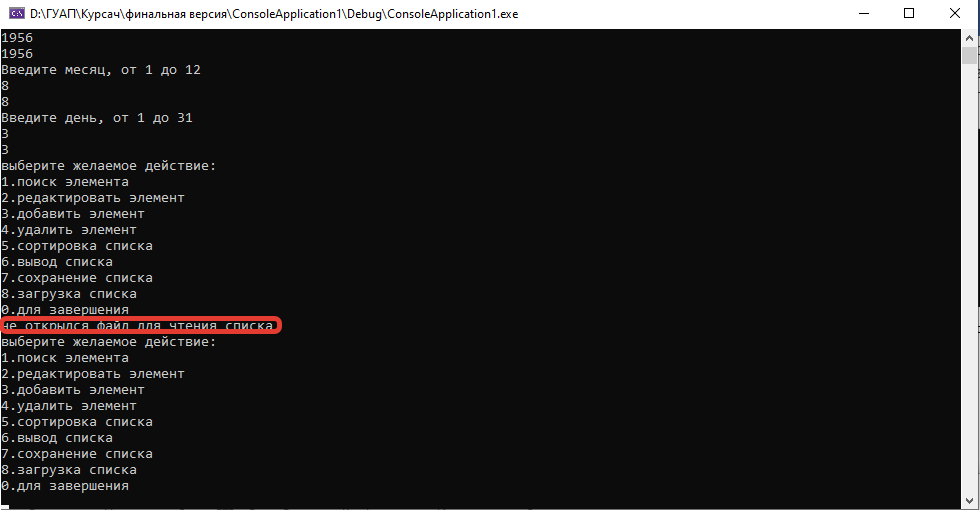


Рис 11. Извещение при недоступности файла.

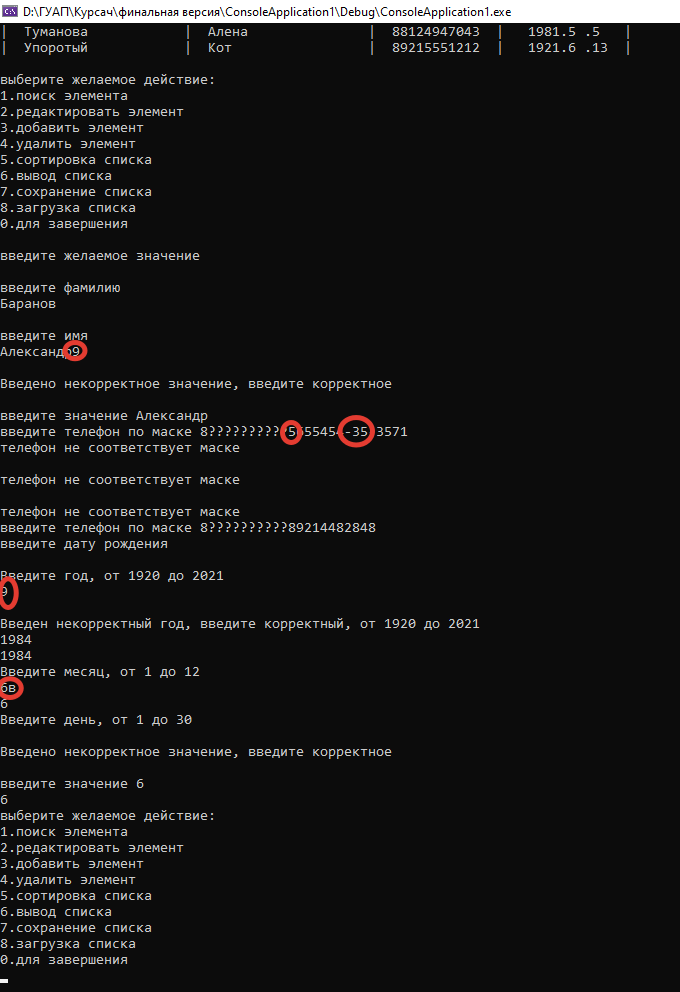


Рис 12. Проверка заполнения корректными данными полей структуры при ручном ее добавлении.

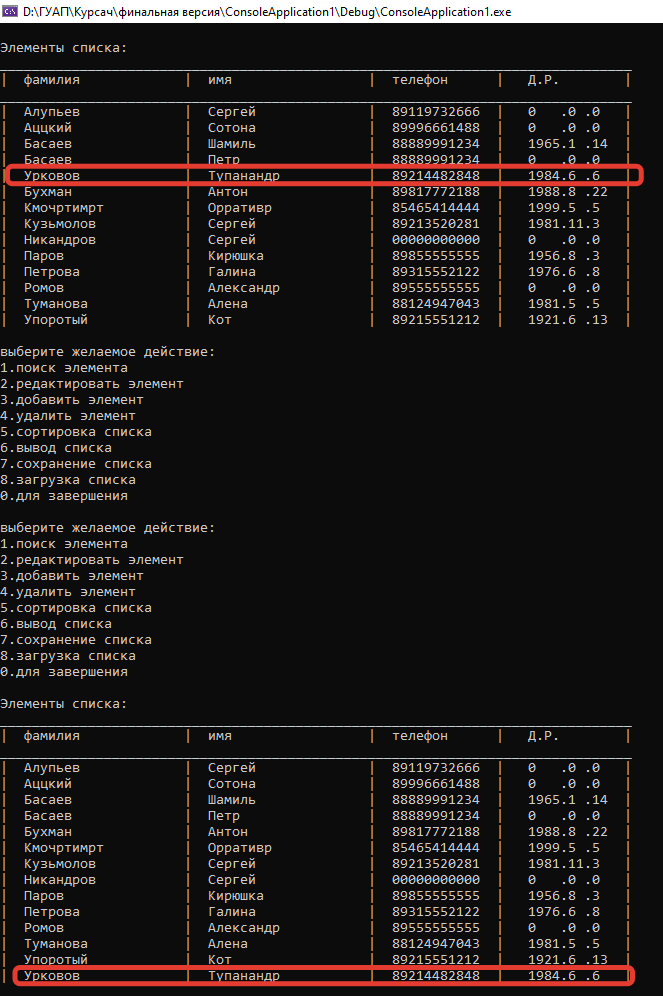


Рис 13. Проверка сортировки списка.

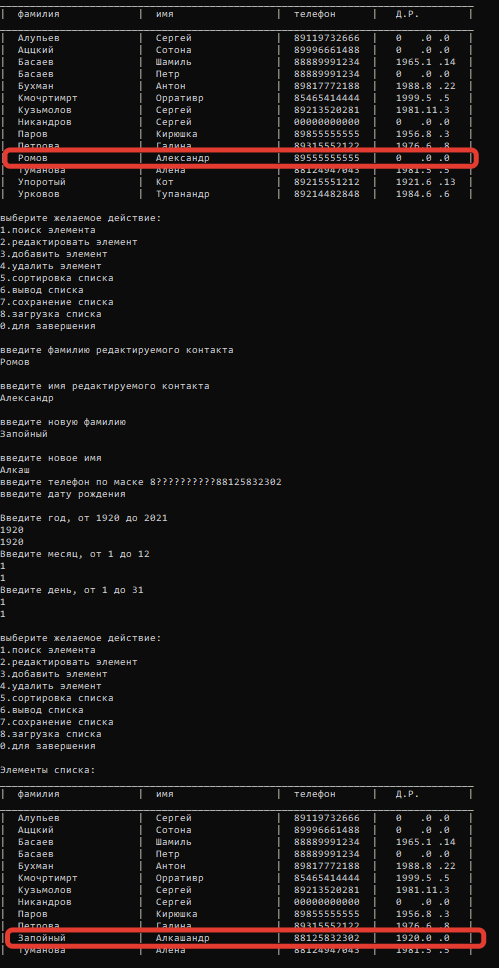


Рис 14. Проверка редактирования элемента.

# Заключение

В ходе курсовой работы было создано приложение … Приложение имеет консольный интерфейс (CLI).

Из положительных моментов можно отметить:

- Все функции реализованы отдельными функциями, что дает возможность для легкой модернизации программы

- На местах вероятных ошибок как пользователя, так и содержимого файла внедрены проверки

- Программа выполняет поставленную задачу без ошибок и в полном объеме

- Это первый опыт создания программы с таким большим количеством функций и проверок, что несомненно вселяет в меня надежды на успешное развитие моих навыков программиста

Из недостатков можно отметить

- программа реализована в скромном визуальном оформлении CLI

- программа реализована не посредством классов, ибо с ними еще не знаком

В ходе разработке программы автору удалось:

- познакомиться ближе со структурами данных, понятием списков, сортировки списков, с тем, как нужно искать вероятные места возникновения ошибок

- познакомиться с функционалом работы с памятью (memset, memcpy)

# Приложение. Листинг программы.

#include <iostream>

#include <conio.h>

#include <cmath>

#include <ctime>

#include <fstream>

#include <Windows.h>

#include <string>

#include <iomanip>

using namespace std;

struct Contact\_List

{

int birth\_date[3] = { NULL };//информационное поле дата рождения

int phone[11] = { NULL };//информационное поле номер телефона

char name[20] = { NULL }; //информационное поле имя

char family[20] = { NULL }; //информационное поле фамилия

Contact\_List\* next; //указатель на следующий элемент

Contact\_List\* prev; //указатель на предыдущий элемент

};

Contact\_List\* head;

struct load\_struct

{

int birth\_date[3] = { NULL };//информационное поле дата рождения

int phone[11] = { NULL };//информационное поле номер телефона

char name[20] = { NULL }; //информационное поле имя

char family[20] = { NULL }; //информационное поле фамилия

};

void my\_shaker();

void print\_list();

void del\_node(const char\* family\_d,const char\* name\_d);

void add\_list(const int size);

void add\_node();

int load\_node\_backup(const char\* node\_beta);

void load\_node2(const char\* node\_beta);

void find\_element(const int redact, const char\* family, const char\* name);

//блок проверок

bool is\_string\_correct(const char\* tested);

bool is\_phone\_correct(const int\* tested);

bool is\_date\_correct(const int\* tested);

bool is\_unique(const Contact\_List\* tested);

bool find\_family(const char\* searched);//возвращает факт вхождения в список этой фамилии

//блок ввода-вывода

int int\_input();

int\* phone\_input();

int\* birth\_date\_input();

int load\_list();

int save\_list();

int main()

{

setlocale(LC\_ALL, "russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

bool stopper = false;

while (!stopper)

{

int size = 0;

int i = 0;

setlocale(LC\_ALL, "rus");

cout << "\nвыберите желаемое действие:\n1.поиск элемента\n2.редактировать элемент\n3.добавить элемент\n4.удалить элемент\n5.сортировка списка\n6.вывод списка\n7.сохранение списка\n8.загрузка списка\n0.для завершения\n";

int x = 0;

while (x < 48 || x>56)

{

x = \_getch();

if (x < 48 || x>56)

{

cout << "\nвведено некорректное значение, введите корректное\n";

}

}

switch (x)

{

case 49://поиск элемента

{

char test\_fam[20] = { NULL };

char test\_nam[20] = { NULL };

cout << "\nвведите фамилию\n";

cin >> test\_fam;

while (!cin.good() || !is\_string\_correct(test\_fam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> test\_fam;

}

if (!find\_family(test\_fam))

{

cout << endl << "не найдено такой фамилии в списке" << endl;

}

else

{

cout << "\nвведите имя\n";

cin >> test\_nam;

while (!cin.good() || !is\_string\_correct(test\_nam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> test\_nam;

}

find\_element(0, test\_fam, test\_nam);

}

break;

}

case 50://редактировать элемент

{

char test\_fam[20] = { NULL };

char test\_nam[20] = { NULL };

cout << "\nвведите фамилию редактируемого контакта\n";

cin >> test\_fam;

while (!cin.good() || !is\_string\_correct(test\_fam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> test\_fam;

}

if (!find\_family(test\_fam))

{

cout << endl << "не найдено такой фамилии в списке, не удаствся ее редактировать" << endl;

}

else

{

cout << "\nвведите имя редактируемого контакта\n";

cin >> test\_nam;

while (!cin.good() || !is\_string\_correct(test\_nam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> test\_nam;

}

find\_element(1, test\_fam, test\_nam);

}

break;

}

case 51://добавить элемент

{

cout << "\nвведите желаемое значение\n";

add\_node();

break;

}

case 52://удалить элемент

{

char family\_d[20] = { NULL };

char name\_d[20] = { NULL };

cout << "\nвведите фамилию удаляемого элемента\n";

cin >> family\_d;

while (!cin.good() || !is\_string\_correct(family\_d)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> family\_d;

}

cout << "\nвведите имя удаляемого элемента\n";

cin >> name\_d;

while (!cin.good() || !is\_string\_correct(name\_d)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> name\_d;

}

//memcpy(deleting->family, family, 20);

//memcpy(deleting->name, name, 20);

del\_node(family\_d,name\_d);

break;

}

case 53://сортировка списка

{

int\* temp = new int[2];

temp[0] = 0;

temp[1] = 0;

my\_shaker();

//cout << "\nколичество сравнений = " << temp[0] << endl;

//cout << "\nколичество перестановок = " << temp[1] << endl;

break;

}

case 54://вывод списка

{

print\_list();

break;

}

case 55://сохранение в файл списка

{

save\_list();

break;

}

case 56://загрузка из файла списка

{

load\_list();

break;

}

case 48:

{

stopper = true;

break;

}

default:

break;

}

}

return 0;

}

bool find\_family(const char\* searched)//возвращает факт вхождения в список этой фамилии

{

bool find = false;

if (head == NULL)

{

cout << "\nСписок пуст\n";

return false;

}

else

{

Contact\_List\* temp = head;

do

{

int x = 0;

for (int i = 0; i < strlen(searched); i++)

{

if (temp->family[i] == searched[i])

{

x++;

}

}

if (x == strlen(searched))

find = true;

temp = temp->next;

} while (temp != head);

return find;

}

}

void find\_element(const int redact, const char\* family, const char\* name)

{

bool sovpad = true;

if (head == NULL)

{

cout << endl << "чего тут искать, список пуст" << endl;

}

else

{

Contact\_List\* temp = new Contact\_List;

temp = head;

do

{

sovpad = true;

for (int i = 0; i < strlen(family); i++)

{

if (temp->family[i] != family[i])

{

sovpad = false;

}

}

for (int i = 0; i < strlen(name); i++)

{

if (temp->name[i] != name[i])

{

sovpad = false;

}

}

if (redact == 0 && sovpad == true)

{

cout << "\nтакой элемент встречается в списке\n";

break;

}

else if (redact == 1 && sovpad == true)

{

Contact\_List\* new\_elem = new Contact\_List;

char new\_fam[20] = { NULL };

char new\_nam[20] = { NULL };

cout << "\nвведите новую фамилию\n";

cin >> new\_fam;

while (!cin.good() || !is\_string\_correct(new\_fam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> new\_fam;

}

cout << "\nвведите новое имя\n";

cin >> new\_nam;

while (!cin.good() || !is\_string\_correct(new\_nam)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> new\_nam;

}

memcpy(new\_elem->family, new\_fam, strlen(new\_fam));

memcpy(new\_elem->name, new\_nam, strlen(new\_nam));

bool sovp = true;

int len = 0;

len = strlen(temp->family);

for (int i = 0; i < len; i++)

{

if (temp->family[i] != new\_elem->family[i])

{

sovp = false;

}

}

len = strlen(temp->name);

for (int i = 0; i < len; i++)

{

if (temp->name[i] != new\_elem->name[i])

{

sovp = false;

}

}

if (is\_unique(new\_elem)||sovp)

{

int\* phone\_n = new int[11];

phone\_n = phone\_input();

cout << "\nвведите дату рождения\n";

int\* birth\_date = new int[3];

birth\_date = birth\_date\_input();

cout << endl;

for (int it = 0; it < 11; it++)

{

temp->phone[it]= phone\_n[it];

}

//memcpy(temp->phone, phone\_n, sizeof(phone\_n));

memcpy(temp->birth\_date, birth\_date, sizeof(birth\_date));

memcpy(temp->family, new\_fam, strlen(new\_fam));

memcpy(temp->name, new\_nam, strlen(new\_nam));

}

else

{

cout << "\nвведено значение, дублирующее имеющийся элемент\n";

}

}

temp = temp->next;

} while (temp != head);

}

}

bool is\_unique(const Contact\_List\* tested)

{

bool is\_correct = true;

if (head != NULL)

{

Contact\_List\* ttemp = head;

do

{

int aaa = 0;

int bbb = 0;

for (int tt = 0; tt < strlen(tested->name); tt++)

{

if (ttemp->name[tt] == tested->name[tt])

{

aaa++;

}

}

for (int tt = 0; tt < strlen(tested->family); tt++)

{

if (ttemp->family[tt] == tested->family[tt])

{

bbb++;

}

}

if (aaa == strlen(tested->name) && bbb == strlen(tested->family))

{

is\_correct = false;

}

ttemp = ttemp->next;

} while (ttemp != head);

}

return is\_correct;

}

int load\_list()

{

int amount\_errors = 0;

char str[60] = { NULL };

ifstream fin;// объект класса ifstream

fin.open("Contact\_list.csv"); // связываем объект с файлом

if (!fin)

{

cout << "не открылся файл для чтения списка";

}

else

{

while (!fin.eof())

{

fin.getline(str, 60);

for (int i = 0; i < strlen(str); i++)

cout << str[i];

load\_node2(str);

cin.sync();

}

fin.close(); // закрываем файл

}

return amount\_errors;

}

int save\_list()

{

int x = 0;

ofstream fout;// объект класса ofstream

fout.open("Contact\_list.csv"); // связываем объект с файлом

if (!fout)

{

cout << "не найдена база, верните ее в каталог";

}

else

{

Contact\_List\* temp = head;

if (head == NULL)

{

cout << "\nСписок пуст\n\n";

}

else

{

do

{

fout << temp->family;

fout << ";";

fout << temp->name;

fout << ";";

for (int i = 0; i < 11; i++)

{

fout << temp->phone[i];

}

fout << ";";

fout << temp->birth\_date[0];

fout << ".";

fout << temp->birth\_date[1];

fout << ".";

fout << temp->birth\_date[2];

fout << "\n";

temp = temp->next;

} while (temp != head);

}

fout.close(); // закрываем файл

}

return x;

}

void load\_node2(const char\* node\_beta)

{

int birth\_date[3] = { NULL };

int phone[11] = { NULL };

char family\_beta[20] = { NULL };

char name\_beta[20] = { NULL };

int family\_end = 0;

int name\_end = 0;

int phone\_end = 0;

int string\_end = strlen(node\_beta);// 1 + static\_cast<int>(node\_beta[0]);

for (int j = 1; j < string\_end; j++)//этот чудесный цикл находит разделители между хранимыми данными, не забываем о том,

{ //что в нулевом символе хранится информация о длине строки, поэтому идем от 1 а не от 0

if (node\_beta[j] == ';' && family\_end == 0)

{

family\_end = j;

}

if (node\_beta[j] == ';' && j != family\_end && family\_end != 0 && name\_end == 0)

{

name\_end = j;

}

if (node\_beta[j] == ';' && j != family\_end && name\_end != 0 && name\_end != j && phone\_end == 0)

{

phone\_end = j;

}

}

int family\_len = family\_end;

int name\_len = name\_end - family\_end;

int phone\_len = phone\_end - name\_end;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка фамилии\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int ii = 0; ii < family\_len; ii++)//выделение фамилии в отдельный массив, стартуем с единицы, ибо в нулевом шла длинна строки

{

family\_beta[ii] = node\_beta[ii];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка имени\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (int ii = family\_end + 1; ii < name\_end; ii++)//выделение имени в отдельный массив

{

name\_beta[ii - family\_end - 1] = node\_beta[ii];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка телефона\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//cout << endl << endl << "номера сегментов";

int smesh = name\_end + 1;

int stopper = 0;

while (stopper < 11)

{

//cout << node\_beta[stopper + smesh];

phone[stopper] = node\_beta[stopper + smesh] - 48;

stopper++;

}

cout << endl << endl;

if (!is\_phone\_correct(phone))

{

for (int tr = 0; tr < 11; tr++)

{

phone[tr] = 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*выделение даты рождения в отдельный массив\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ss = 0;

int ww = 1;

for (int ii = string\_end - 1; ii > phone\_end; ii--)//выделение даты рождения в отдельный массив

{

if (node\_beta[ii] == '.')//это счетчик переходов от года к месяцу и от месяца ко дню

{

ii--;

ss++;

ww = 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка года\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

if (ss == 0)

{

birth\_date[2] += ((node\_beta[ii] - 48) \* ww);

ww = ww \* 10;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка месяца\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if (ss == 1)//это обработка месяца

{

birth\_date[1] += ((node\_beta[ii] - 48) \* ww);

ww = ww \* 10;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это обработка дня\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

else if (ss == 2)//это обработка дня

{

birth\_date[0] += ((node\_beta[ii] - 48) \* ww);

ww = ww \* 10;

}

}

if (!is\_string\_correct(family\_beta))

{

family\_beta[0] = '99'; family\_beta[1] = 'О'; family\_beta[2] = 'ш'; family\_beta[3] = 'и'; family\_beta[4] = 'б'; family\_beta[5] = 'к'; family\_beta[6] = 'а';

}

if (!is\_string\_correct(name\_beta))

{

name\_beta[0] = '99'; name\_beta[1] = 'О'; name\_beta[2] = 'ш'; name\_beta[3] = 'и'; name\_beta[4] = 'б'; name\_beta[5] = 'к'; name\_beta[6] = 'а';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*это создание нового элемента\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Contact\_List\* Node = new Contact\_List;

for (int i = 0; i < strlen(family\_beta); i++)

{

Node->family[i] = family\_beta[i];

}

for (int i = 0; i < strlen(name\_beta); i++)

{

Node->name[i] = name\_beta[i];

}

for (int i = 0; i < 11; i++)

{

Node->phone[i] = phone[i];

}

if (is\_unique(Node) && is\_string\_correct(family\_beta) && is\_string\_correct(name\_beta))

{

for (int i = 0; i < strlen(family\_beta); i++)

{

Node->family[i] = family\_beta[i];

}

if (is\_date\_correct(birth\_date))//присвоение даты рождения

{

Node->birth\_date[0] = birth\_date[0];//

Node->birth\_date[1] = birth\_date[1];//присвоение даты рождения

Node->birth\_date[2] = birth\_date[2];//

}

else

{

Node->birth\_date[0] = 0;//

Node->birth\_date[1] = 0;//присвоение даты рождения

Node->birth\_date[2] = 0;//

}

if (head == NULL) //если список пуст

{

//cout << "кажется, будто голова пуста";

Node->next = Node; //установка указателя next

Node->prev = Node; //установка указателя prev

head = Node; //определяется голова списка

}

else

{

Contact\_List\* p = head;

p = p->prev;

p->next->prev = Node;

Node->next = p->next;

Node->prev = p;

p->next = Node;

}

}

}

void add\_node()

{

Contact\_List\* Node = new Contact\_List; //создание нового элемента

bool unique = false;

char family[20] = { NULL };

int fam[20] = { NULL };

char name[20] = { NULL };

int nam[20] = { NULL };

cout << "\nвведите фамилию\n";

cin >> family;

while (!cin.good() || !is\_string\_correct(family)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> family;

}

cout << "\nвведите имя\n";

cin >> name;

while (!cin.good() || !is\_string\_correct(name)) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> name;

}

memcpy(Node->family, family, 20);

memcpy(Node->name, name, 20);

if (is\_unique(Node))

{

int\* phone = new int[11];

phone = phone\_input();

cout << "\nвведите дату рождения\n";

int\* birth\_date = new int[3];

birth\_date = birth\_date\_input();

for (int i = 0; i < 11; i++)

{

Node->phone[i] = phone[i]; //присвоение телефона

}

Node->birth\_date[0] = birth\_date[0]; //присвоение даты рождения

Node->birth\_date[1] = birth\_date[1];

Node->birth\_date[2] = birth\_date[2];

if (head == NULL) //если список пуст

{

Node->next = Node; //установка указателя next

Node->prev = Node; //установка указателя prev

head = Node; //определяется голова списка

}

else

{

Contact\_List\* p = head;

p = p->prev;

p->next->prev = Node;

Node->next = p->next;

Node->prev = p;

p->next = Node;

}

}

else

{

cout << "Дублируется другой элемент, список строим на уникальных элементах";

}

}

void print\_list()

{

//system("cls");

if (head == NULL)

{

cout << "\nСписок пуст\n\n";

}

else

{

Contact\_List\* temp = head;

cout << "\nЭлементы списка: " << endl;

int line\_size = 21;

int line\_size2 = 79;

for (int i = 0; i < line\_size2; i++)

{

cout << "\_";

}

cout << endl;

cout << left << setw(line\_size + 2) << "| фамилия" << setw(line\_size + 2) << "| имя" << setw(16) << "| телефон" << setw(16) << "| Д.Р." << "|" << endl;

for (int i = 0; i < line\_size2; i++)

{

cout << "\_";

}

cout << endl;

do

{

//int line\_size = 21;

//cout <<setw(line\_size)<<"фамилия"<<setw(line\_size)<<"имя"<<setw(13)<<"телефон"<<setw(10)<<"Д.Р." <<endl;

cout << "| ";

int j = 0;

j = strlen(temp->family);

for (int i = 0; i < j; i++)

{

cout << char(temp->family[i]);

}

for (int ii = j; ii < line\_size; ii++)

{

if (ii == line\_size - 1)

{

cout << "| ";

}

else

{

cout << " ";

}

}

j = 0;

j = strlen(temp->name);

for (int i = 0; i < j; i++)

{

cout << temp->name[i];

}

for (int ii = j; ii < line\_size; ii++)

{

if (ii == line\_size - 1)

{

cout << "| ";

}

else

{

cout << " ";

}

}

for (int i = 0; i < 11; i++)

{

int xx = temp->phone[i];

cout << xx;

//zz.push\_back(int(xx));

}

cout << " | " << setw(4) << temp->birth\_date[0] << "." << setw(2) << temp->birth\_date[1] << "." << setw(2) << temp->birth\_date[2] << " |";

cout << endl;

temp = temp->next;

} while (temp != head);

}

}

void del\_node(const char\*family\_d,const char\* name\_d)//удаляет запись с указанными именем и фамилией

{

int deleted=0;

if (head == NULL)

{

cout << "\nСписок пуст\n\n";

}

else

{

Contact\_List\* temp = head;

do

{

bool sovpad = true;

int length\_f = 0;

int length\_n = 0;

if (strlen(family\_d) != strlen(temp->family)|| strlen(name\_d) != strlen(temp->name))

{

//cout << endl << "не найден такой элемент";

}

else

{

length\_f = strlen(family\_d);

for (int i = 0; i < length\_f; i++)

{

if (family\_d[i] != temp->family[i])

{

sovpad = false;

}

}

length\_n = strlen(name\_d);

for (int i = 0; i < length\_f; i++)

{

if (name\_d[i] != temp->name[i])

{

sovpad = false;

}

}

if (!sovpad)

{

cout << endl << "не найден такой элемент";

}

else

{

deleted++;

cout << "\nэлемент удален\n";

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

}

}

temp = temp->next;

} while (temp != head);

}

if (deleted == 0)

{

cout << "\nне найден такой элемент\n";

}

}

//temp->prev->next = temp->next;

//temp->next->prev = temp->prev;

int int\_input()

{

int temp = 0;

cin >> temp;

while (!cin.good()) //Контроль корректности вводимых значений

{

cin.clear();

cin.ignore();

cerr << endl << "Введено некорректное значение, введите корректное" << endl;

cout << endl << "введите значение ";

cin >> temp;

}

return temp;

}

int\* phone\_input()

{

int\* temp = new int[11];

bool correct = true;

do//Контроль корректности вводимых значений

{

correct = true;

cout << "введите телефон по маске 8??????????";

for (int i = 0; i < 11; i++)

{

int a = \_getch();

//cout << a;

temp[i] = a - 48;

cout << temp[i];

}

if (temp[0] != 8)

{

correct = false;

cout << "\nтелефон не соответствует маске\n";

}

for (int i = 1; i < 11; i++)

{

if (temp[i] < 0 || temp[i]>9)

{

cout << "\nтелефон не соответствует маске\n";

correct = false;

}

}

} while (correct == false);

return temp;

}

int\* birth\_date\_input()

{

int\* temp = new int[3];

for (int i = 0; i < 3; i++)

{

if (i == 0)

{

cout << endl << "Введите год, от 1920 до 2021" << endl;

temp[0] = int\_input();

while (temp[0] < 1920 || temp[0]>2021) //Контроль корректности вводимых значений

{

cout << endl << "Введен некорректный год, введите корректный, от 1920 до 2021" << endl;

temp[0] = int\_input();

}

}

else if (i == 1)

{

cout << endl << "Введите месяц, от 1 до 12" << endl;

temp[1] = int\_input();

while (temp[1] < 1 || temp[1]>12) //Контроль корректности вводимых значений

{

cout << endl << "Введен некорректный месяц, введите корректный, от 1 до 12" << endl;

temp[1] = int\_input();

}

}

else if (i == 2)

{

int day\_limit = 0;

if (temp[1] == 2 && temp[0] % 4 == 0)

{

day\_limit = 29;

//cout << endl << "29";

}

else if (temp[1] == 2 && temp[0] % 4 != 0)

{

day\_limit = 28;

//cout << endl << "28";

}

else if (temp[1] == 1)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 3)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 4)

{

day\_limit = 30;

//cout << endl << "31";

}

else if (temp[1] == 5)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 6)

{

day\_limit = 30;

//cout << endl << "30";

}

else if (temp[1] == 7)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 8)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 9)

{

day\_limit = 30;

//cout << endl << "30";

}

else if (temp[1] == 12)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 10)

{

day\_limit = 31;

//cout << endl << "31";

}

else if (temp[1] == 11)

{

day\_limit = 30;

//cout << endl << "30";

}

cout << endl << "Введите день, от 1 до " << day\_limit << endl;

temp[2] = int\_input();

while (temp[2] < 1 || temp[2]>day\_limit) //Контроль корректности вводимых значений

{

cout << endl << "Введен некорректный день, введите корректный, от 1 до " << day\_limit << endl;

temp[2] = int\_input();

}

}

cout << temp[i];

}

//} while (correct == false);

return temp;

}

bool is\_string\_correct(const char\* tested\_string)

{

bool is\_correct = true;

int amount\_of\_uncorrect = 0;

for (int i = 0; i < strlen(tested\_string); i++)

{

int x = static\_cast<int>(tested\_string[i]);

//cout << endl << x;

if (x < -64 || x > -1)

{

amount\_of\_uncorrect++;

}

}

if (amount\_of\_uncorrect > 0)

{

is\_correct = false;

}

//cout << "\n\n строчная проверка возвращает " << is\_correct << "\n\n";

return is\_correct;

}

bool is\_date\_correct(const int\* tested)

{

bool is\_correct = true;

for (int i = 0; i < 3; i++)

{

if (i == 0)

{

if (tested[0] < 1920 || tested[0]>2021) //Контроль корректности вводимых значений

{

is\_correct = false;

}

}

else if (i == 1)

{

if (tested[1] < 1 || tested[1]>12) //Контроль корректности вводимых значений

{

is\_correct = false;

}

}

else if (i == 2)

{

int day\_limit = 0;

if (tested[1] == 2 && tested[0] % 4 == 0)

{

day\_limit = 29;

}

else if (tested[1] == 2 && tested[0] % 4 != 0)

{

day\_limit = 28;

}

else if (tested[1] == 1)

{

day\_limit = 31;

}

else if (tested[1] == 3)

{

day\_limit = 31;

}

else if (tested[1] == 4)

{

day\_limit = 30;

}

else if (tested[1] == 5)

{

day\_limit = 31;

}

else if (tested[1] == 6)

{

day\_limit = 30;

}

else if (tested[1] == 7)

{

day\_limit = 31;

}

else if (tested[1] == 8)

{

day\_limit = 31;

}

else if (tested[1] == 9)

{

day\_limit = 30;

}

else if (tested[1] == 12)

{

day\_limit = 31;

}

else if (tested[1] == 10)

{

day\_limit = 31;

}

else if (tested[1] == 11)

{

day\_limit = 30;

}

if (tested[2] < 1 || tested[2]>day\_limit) //Контроль корректности вводимых значений

{

is\_correct = false;

}

}

}

return is\_correct;

}

bool is\_phone\_correct(const int\* tested)

{

bool is\_correct = true;

if (tested[0] != 8)

{

is\_correct = false;

}

for (int i = 1; i < 11; i++)

{

if (tested[i] < 0 || tested[i] > 9)

{

is\_correct = false;

}

}

return is\_correct;

}

void my\_shaker()

{

Contact\_List\* temp = new Contact\_List;

temp = head;

int amount\_of\_elements = 0;

do

{

amount\_of\_elements++;

temp = temp->next;

} while (temp != head);

for (int i = 0; i < pow(amount\_of\_elements, 2); i++)

{

temp = head;

do

{

int length = 0;

bool blocker = false;

if (strlen(temp->family) > strlen(temp->next->family))

{

length = strlen(temp->next->family);

}

if (temp->family[0] > temp->next->family[0])

{

swap(temp->family, temp->next->family);

swap(temp->name, temp->next->name);

swap(temp->phone, temp->next->phone);

swap(temp->birth\_date, temp->next->birth\_date);

}

if (temp->family[0] == temp->next->family[0] && (temp->family[1] > temp->next->family[1]))

{

swap(temp->family, temp->next->family);

swap(temp->name, temp->next->name);

swap(temp->phone, temp->next->phone);

swap(temp->birth\_date, temp->next->birth\_date);

}

temp = temp->next;

} while (temp->next != head);

}

}

# Использованная литература

[В Интернете] // C++ reference. - 22 01 2021 г.. - https://en.cppreference.com/w/.

**Dewhurst Stephen C.** C++ Common Knowledge: Template Template Parameters [В Интернете] // InformIT. - 12 05 2005 г.. - https://www.informit.com/articles/article.aspx?p=376878.

**O'Dwyer Arthur** Pitfalls and decision points in implementing const\_iterator [В Интернете] // Stuff mostly about C++. - 01 12 2018 г.. - 19 01 2021 г.. - https://quuxplusone.github.io/blog/2018/12/01/const-iterator-antipatterns/.

**Туманова А.В.** ОСНОВЫ ПРОГРАММИРОВАНИЯ. Методические указания к выполнению курсового проекта [Книга]. - Санкт-Петербург : ГОУ ВПО СПбГУАП, 2019.

**Туманова А.В.** Основы программирования. Методические указания по лабораторному практикуму [Книга]. - Санкт-Петербург : ГОУ ВПО СПбГУАП, 2017.